Access Controls for Oblivious and Anonymous Systems

SCOTT E. COULL, RedJack
MATTHEW GREEN and SUSAN HOHENBERGER, Johns Hopkins University

The use of privacy-enhancing cryptographic protocols, such as anonymous credentials and oblivious transfer, could have a detrimental effect on the ability of providers to effectively implement access controls on their content. In this article, we propose a *stateful anonymous credential* system that allows the provider to implement nontrivial, real-world access controls on oblivious protocols conducted with anonymous users. Our 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 an adaptive oblivious transfer scheme to create a privacy-friendly oblivious database with strong access controls.

Categories and Subject Descriptors: E.3 [**Data**]: Data Encryption; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection

General Terms: Security

Additional Key Words and Phrases: Anonymous credentials, oblivious transfer, access controls

ACM Reference Format:
Coull, S. E., Green, M., and Hohenberger, S. 2011. Access controls for oblivious and anonymous systems. ACM Trans. Info. Syst. Sec. 14, 1, Article 10 (May 2011), 28 pages.
DOI = 10.1145/1952982.1952992 http://doi.acm.org/10.1145/1952982.1952992

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. This situation introduces a dilemma. On the one hand, users may wish to hide this meta data from the service provider. On the other hand, if the service provider does not know who is accessing which data, then how can he

S. Coull was supported in part by the U.S. Department of Homeland Security under contract FA8750-08-2-0147 and by the NSF under grant 0937060. M. Green was supported by grant 90TR0003/01 from HHS and by NSF grant CNS-1010928. S. Hohenberger was supported by NSF grant CNS-0716142, a Microsoft New Faculty Fellowship, and a Google Research Award. The contents of this publication are solely the responsibility of the authors and do not necessarily represent the official views of the HHS or any other sponsor.
Contact author's email address: scott.coull@redjack.com.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2011 ACM 1094-9224/2011/05-ART10 $10.00
DOI 10.1145/1952982.1952992 http://doi.acm.org/10.1145/1952982.1952992

implement basic security mechanisms, such as authentication, access controls, and auditing? The goal of this work is to use cryptography to preserve a user's privacy, to the extent possible, while allowing the provider to maintain control over his database.

Medical databases provide a particularly compelling motivation for this technology. 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 [Google 2009] and Microsoft HealthVault [Microsoft 2009], 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 [Chaum 1985; Camenisch and Lysyanskaya 2002] can be used to ensure that only authorized personnel gain access to the system without revealing the identity of those users. Oblivious transfer [Naor and Pinkas 1999; Camenisch et al. 2007] and private information retrieval [Chor et al. 1998] 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. A naive 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. Moreover, the previous work in this area provides no insight as to how access controls 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 oblivious and anonymous systems. These privacy-preserving access controls must conform to currently available access control models and also allow for access control methods that may have not yet been considered. To do so, we model our access policies as arbitrary state machines. We introduce *stateful anonymous credentials* to embed the user's current state within her credentials. Our stateful anonymous credentials are an extension of existing anonymous credential systems [Lysyanskaya 2002; Camenisch and Lysyanskaya 2002; 2004; Boneh and Boyen 2004], and our construction is secure in the standard model under basic assumptions, such as Strong RSA. We show how to efficiently combine these stateful credentials with the adaptive oblivious transfer protocol of Camenisch et al. [2007] to create a database system that allows users to access files anonymously (without revealing their identity) and obliviously (without revealing their choices), and yet where strong access controls can be simultaneously enforced. Finally, as a minor building block of our system, we 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 may use it to control access according to any policy that can be described as a deterministic finite state automaton. One obvious and immediate consequence of this choice is that we are able to implement a number of existing nontrivial access control models used by financial institutions and classified government systems, including the Biba [Biba 1977], Clark–Wilson [Clark and Wilson 1987], Bell-La Padula [Bell and La Padula 1988], and Brewer–Nash [Brewer and Nash 1989] models. Certainly, many of these models may be implemented without the overhead associated with a stateful scheme (e.g., interactive update protocols for retrieving new credentials), however, this stateful property enables many new and potentially exciting opportunities for controlling access that have not been explored yet, including access controls based on regular

expression matching. To provide a concrete example of these access controls, we show how to implement discretionary access controls, as well as the Bell–La Padula [Bell and La Padula 1988] and Brewer–Nash [Brewer and Nash 1989] access control models within our system. Finally, for those who may be interested in efficient stateless access controls, we briefly describe a simple method for converting our stateful credential system into a very efficient stateless system.

Related Work. Prior 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. Example limitations include double-spending prevention [Camenisch et al. 2005], authenticating anonymously at most k times [Teranishi et al. 2004; Camenisch et al. 2006], or preventing a user from exchanging too much money with a single merchant [Camenisch et al. 2006].

With respect to controls on oblivious transfer protocols, Aiello et al. [2001] proposed priced oblivious transfer, in which each user is given a declining balance that can be spent on each transfer, although user anonymity was not protected. Very recently, Camenisch et al. [2010] provided a priced oblivious transfer protocol that does protect anonymity. In the more general concept of conditional oblivious transfer [Crescenzo et al. 1999; Blake and Kolesnikov 2004], 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 et al. [2009] proposed a combination of the oblivious transfer scheme Camenisch et al. [2007] with Boneh–Boyen signatures [Boneh and Boyen 2004] 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 a strict subset of the policies enabled by our proposed scheme. In Section 3.3, we describe a simple change to our stateful credentials that enables efficient stateless access controls similar to those of Camenisch et al. without an online update procedure.

2. TECHNICAL PRELIMINARIES

In this section, we recall some basic building blocks and then introduce a new primitive called *hidden range proofs*. We'll build our constructions generically from these pieces, but we offer some recommendations for implementation choices in Section 4. For the remainder of the article, we denote the security parameter as 1^κ.

Bilinear Groups. Let 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) Nondegenerate: $e(g, g) \neq 1$.
(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 probabilistic commitment algorithm Commit, and an opening algorithm Decommit, which we describe here:

(1) CSetup(1^κ). On input a security parameter, outputs parameters ρ.
(2) Commit($\rho, v_1, \ldots, v_m$). On input values $v_1, \ldots, v_m$, outputs a commitment C and an opening O.
(3) Decommit($\rho, C, O, v_1, \ldots, v_m$). On input a commitment C, opening O and values $v_1, \ldots, v_m$, outputs 1 if (C, O) is consistent and 0 otherwise.

Pedersen [1992] provides a perfectly hiding and computationally binding commitment scheme for prime order groups under the discrete logarithm assumption, and Fujisaki and Okamoto [1997] provide a similar (though, statistically hiding) commitment scheme for composite order groups under the Strong RSA assumption.

Zero-Knowledge Protocols. In a zero-knowledge proof [Goldreich et al. 1986], a user proves a statement to another party without revealing anything other than the statement's veracity. We use standard techniques for proving statements about discrete logarithms, such as (1) a proof of knowledge of a discrete logarithm modulo a prime [Schnorr 1991] or a composite [Fujisaki and Okamoto 1997; Damgård and Fujisaki 2002]; (2) a proof of knowledge of equality of representation modulo two (possibly different) prime [Chaum and Pedersen 1992] or composite [Camenisch and Michels 1999b] moduli under the Strong RSA assumption; (3) a proof that a commitment opens to the product of two other committed values [Camenisch and Michels 1999a; Camenisch 1998; Brands 1997]; and (4) a proof of the disjunction or conjunction of any two of the previous [Cramer et al. 1994].

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 [Goldwasser et al. 1988]; (2) there exists an efficient protocol for a user to obtain a signature on the value(s) in a commitment [Pedersen 1992; Fujisaki and Okamoto 1997] without the signer learning anything about the value(s); and (3) a zero-knowledge protocol for proving knowledge of a signature. The signature scheme consists of the algorithms (KeyGen, Sign, Verify) and a protocol BlindSign, which we describe in the following.

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

(4) $\mathsf{BlindSign}(\mathcal{U}(pk, M_1, \ldots, M_n), \mathcal{S}(pk, sk)) \rightarrow (\sigma, \mathsf{nothing})$. This protocol is conducted between a signer $\mathcal{S}$ and a user $\mathcal{U}$ with message(s) to be signed. At the end of the protocol, $\mathcal{U}$ obtains the signature σ, and $\mathcal{S}$ obtains nothing.

Camenisch and Lysyanskaya designed two such schemes: one that is secure under the Strong RSA assumption [Camenisch and Lysyanskaya 2002] and another using bilinear groups under the LRSW assumption [Camenisch and Lysyanskaya 2004]. Similarly, a signature scheme with efficient protocols can be obtained by the Boneh–Boyen–Shacham signatures [Boneh et al. 2004] under the Strong Diffie–Hellman and Linear assumptions in bilinear groups [Camenisch and Lysyanskaya 2004].

Since we want to combine these signatures with an oblivious transfer protocol, we also require that BlindSign meet the definition of *selective-failure blindness*, as defined by Camenisch et al. [2007]. The definition of selective-failure blindness consists of a game where the adversary outputs a public key along with two equal-length message strings $M_{0,1}, \ldots, M_{0,n}$ and $M_{1,1}, \ldots, M_{1,n}$. The adversary is then allowed to interact with two oracles running $\mathcal{U}$'s portion of the BlindSign protocol, $\mathcal{U}(pk, M_{b,1}, \ldots, M_{b,n})$ and $\mathcal{U}(pk, M_{b-1,1}, \ldots, M_{b-1,N})$ for $b \xleftarrow{\$} \{0, 1\}$, where the adversary acts as the signer for her chosen public key. The output from these oracles 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 probabilistic, polynomial time (p.p.t.) adversary has non-negligible advantage in guessing the bit b. The original constructions of Camenisch–Lysyanskaya [2002; 2004] and Boneh–Boyen–Shacham [2004] were not proved secure under this stronger definition of blindness. However, recent work by Fischlin and Schröder [2009] 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 [Goldreich et al. 1986] 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) [Dodis and Yampolskiy 2005] 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 [Jarecki and Liu 2009]. Naor and Reingold describe another PRF that is secure under the Decisional Diffie–Hellman assumption in prime and composite order groups [Naor and Reingold 1997].

Verifiable Encryption. A verifiable encryption scheme allows the encryptor to prove that a ciphertext contains a message without revealing the message to the verifier. Specifically, the verifiable encryption algorithm takes as input a public key pk, a set of commitment parameters ρ, and a message m. It produces a ciphertext $C = \mathsf{Encrypt}_{pk}(m)$ and a commitment $A = \mathsf{Commit}(m)$. In the schemes we employ, 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 contained in the ciphertext. More formally, the encryptor must prove that it knows an opening O and message m such that $\mathsf{Decommit}(\rho, C, O, m) = \mathsf{true}$, and simultaneously that $m \in \mathsf{Decrypt}(sk, C)$. 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 [Camenisch and Damgård 2000].

Hidden-Range Proofs. Standard techniques [Chan et al. 1998; Camenisch and Michels 1999a, 1999b; Boudot 2000] 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 ($\geq$0) is now public. The prover first selects a composite-order group $\mathbf{G} = \langle \mathbf{g} \rangle$, with special RSA modulus n and quadratic residue $\mathbf{g}$, along with a random value $\mathbf{h} \in \mathbf{G}$. 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 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 or any other information about themselves. A stateful anonymous credential system adds the notion of credential *state*, which is embedded as an attribute within the credential. Intuitively, this state captures a user's history including her past actions. A user may update the state in her credential only according to some well-defined *policy* dictated by the credential provider.

At a high level, the stateful anonymous credential system, which is defined by the tuple (Setup, ObtainCred, UpdateCred, ProveCred), operates as follows. First, the user and credential provider obtain the system parameters and their private keys during Setup and negotiate the use of a specified policy using the 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 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. 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

(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 pid she is following is the same as what is provided in the tag and that 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 nontrivial, 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; we describe one 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 now describe their input/output behavior and intended functionality.

(1) 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 $(pk_{\mathcal{U}}, sk_{\mathcal{U}})$, and requests that it be certified by a trusted CA.

(2) ObtainCred($\mathcal{U}(pk_{\mathcal{P}}, sk_{\mathcal{U}}, \Pi_i^{\mathcal{C}}, S)$, $\mathcal{P}(pk_{\mathcal{U}}, sk_{\mathcal{P}}, \Pi_i^{\mathcal{C}}, S)$)**:** $\mathcal{U}$ identifies herself to $\mathcal{P}$ and then receives her credential, Cred, which binds her to a policy graph Π_i and starting state S.

(3) UpdateCred($\mathcal{U}(pk_{\mathcal{P}}, sk_{\mathcal{U}}$, 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 embedded in Cred. Simultaneously, $\mathcal{P}$ should not learn $\mathcal{U}$'s identity, attributes, or current state. To prevent replay attacks, $\mathcal{P}$ maintains a database D containing information about prior executions of UpdateCred, which it updates as a result of the protocol.

(4) ProveCred($\mathcal{U}(pk_{\mathcal{P}}, sk_{\mathcal{U}}$, Cred), $\mathcal{P}(pk_{\mathcal{P}}, E)$)**:** $\mathcal{U}$ proves possession of a credential Cred in a particular state. To prevent reuse of credentials, $\mathcal{P}$ maintains a database E

containing information about prior executions of ProveCred, which it updates as a result of the protocol.

Security Definitions. To capture the security requirements needed for our applications, we employ 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 that she holds a credential in a particular state without first obtaining that credential and being placed in that state, or arriving at that state via an admissible sequence of state updates under the agreed-upon policy.

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 Camenisch et al. [2007]. 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 the adversary would 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. 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 up to 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, the policy graphs for each user $\Pi_1, \ldots, \Pi_\eta$ are provided to all parties. The users are also provided with an adaptive strategy Σ that, on input of the user's identity, current graph state and the protocol transcript thus far (including selections and responses), outputs the next action to be taken by the user.

At the beginning of the experiment, the provider conducts the Setup procedure to obtain *params* (which it distributes to the users via broadcast), and subsequently conducts the ObtainCred procedure with each user. At the end of this step, $\hat{\mathcal{P}}$ produces an initial output P_1 containing all of the provider's state information, and each user $\hat{\mathcal{U}}_i$ outputs a similar value $U_{1,i}$. For each subsequent round $j \in [2, k]$, each user may interact with $\hat{\mathcal{P}}$ via either the UpdateCred or ProveCred protocols as indicated 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 $(P_k, U_{1,k}, \ldots, U_{\eta,k})$.

For corrupted parties, those outputs are unrestricted. For honest parties, we define the real-world *honest* provider $\mathcal{P}$ as one that honestly runs its portion of Setup and ObtainCred in the first round, honestly runs its side of the UpdateCred and ProveCred protocols when requested by a user at round $j > 1$, and outputs $P_k = params$ after the final round. Similarly, an honest user $\mathcal{U}_i$ runs the Setup and ObtainCred protocols honestly in the first round, executes the user's side of the UpdateCred and 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 $\hat{\mathcal{P}}'$, trusted party $\mathcal{T}$, and possibly cheating users are provided with the graphs $\Pi_1, \ldots, \Pi_\eta$. Each user is provided with the selection strategy Σ. 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 from state S_i to state T_i using the UpdateCred protocol and a transaction id t_{id}, or (prove, i, S_i) to prove her state using the ProveCred protocol according to the strategy Σ. Here, the use of a transaction id models the user's ability to restart an aborted protocol instance with the caveat of linking the two instances, which we describe shortly.

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, $\mathcal{T}$ 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 $\hat{\mathcal{U}}'$. (The user could reveal the state directly to the provider if the overall system does not hide the user's actions. The definition can accommodate this by relaying the state S_i to $\hat{\mathcal{P}}'$.) 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 $(P_k, V_j, U_{1,k}, \ldots, U_{\eta,k})$.

For corrupted parties, those outputs are unrestricted. For honest parties, we define the ideal-world *honest* provider $\mathcal{P}'$ as one that 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 an empty string in P_k. An honest user $\mathcal{U}'_i$ sends (join, i) in the first round, and update, prove messages according to the selection policy in rounds $2, \ldots, k$, finally outputing all received credentials in round k.

Let $\ell(\cdot), c(\cdot)$ be polynomially-bounded functions in the security parameter κ. 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 probabilistic, polynomial-time (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:

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

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:

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

3.2. Basic Construction for Stateful Credentials

We now describe how to realize stateful anonymous credentials. The state records information about the user's attributes and prior access history. The core innovations are a protocol for performing state updates and a technique for "translating" a history-dependent update policy into a suitable cryptographic representation.

The setup, credential granting, and credential update protocols are presented in Figure 1. We 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 shows 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 Camenisch and Lysyanskaya [2001, 2002] 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

$\mathsf{Setup}(\mathcal{U}(1^k), \mathcal{P}(1^k, \Pi_1, \ldots, \Pi_n))$**:** The provider $\mathcal{P}$ generates parameters for the blind signature protocol, including among other things, a group $\mathbb{G} = \langle g \rangle$ of prime order q. The provider also generates parameters ρ for the commitment scheme.

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 a random $u \in \mathbb{Z}_q$ 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.

$\mathsf{ObtainCred}(\mathcal{U}(pk_{\mathcal{P}}, sk_{\mathcal{U}}, \Pi_i^{\mathcal{C}}, S), \mathcal{P}(pk_{\mathcal{U}}, sk_{\mathcal{P}}, \Pi_i^{\mathcal{C}}, S))$**:** On input a cryptographic representation of a policy $\Pi_i^{\mathcal{C}}$ and initial state S, $\mathcal{U}$ and $\mathcal{P}$ conduct the following protocol:

(1) $\mathcal{U}$ picks random usage and update nonces $N_s, N_u \in \mathbb{Z}_q$ and parses $\Pi_i^{\mathcal{C}}$ to obtain pid. It then computes $A \leftarrow \mathrm{Commit}(\rho, sk_{\mathcal{U}}, N_s, N_u, \mathsf{pid}, S)$.
(2) $\mathcal{U}$ sends A to $\mathcal{P}$ and interactively proves knowledge of a valid opening w.r.t. the public values A, pid, S and $pk_{\mathcal{U}} = g^{sk_{\mathcal{U}}}$.
(3) $\mathcal{U}$ and $\mathcal{P}$ run the blind signing protocol on the committed values in A such that $\mathcal{U}$ obtains the signature $\sigma_{\mathsf{state}} \leftarrow \mathsf{Sign}(ssk_{\mathcal{P}}, (sk_{\mathcal{U}}, N_s, N_u, \mathsf{pid}, S))$.
(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 parses $\Pi_i^{\mathcal{C}}$ to obtain pid. It 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 random $N_s', N_u' \in \mathbb{Z}_q$ and computes $A \leftarrow \mathrm{Commit}(\rho, sk_{\mathcal{U}}, N_s', N_u', \mathsf{pid}, T)$.
(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 $(sk_{\mathcal{U}}, \mathsf{pid}, S, T, N_s', N_u', N_s, \sigma_{\mathsf{state}}, \sigma_{S \to T})$ s.t.:
  (a) $A = \mathrm{Commit}(\rho, sk_{\mathcal{U}}, N_s', N_u', \mathsf{pid}, T)$.
  (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 this proof does not verify, $\mathcal{P}$ aborts. Otherwise $\mathcal{U}$ and $\mathcal{P}$ run the blind signing protocol on the committed values in A to provide $\mathcal{U}$ with $\sigma'_{\mathsf{state}} \leftarrow \mathsf{Sign}(ssk_{\mathcal{P}}, (sk_{\mathcal{U}}, N_s', N_u', \mathsf{pid}, T))$.
(6) $\mathcal{U}$ stores the updated credential $\mathsf{Cred}' = (\Pi_i^{\mathcal{C}}, T, \sigma'_{\mathsf{state}}, N_s', N_u')$.

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

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 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, nonoblivious

$\mathsf{ProveCred}(\mathcal{U}(pk_{\mathcal{P}}, sk_{\mathcal{U}}, \mathsf{Cred}), \mathcal{P}(sk_{\mathcal{P}}, E))$**:** 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 $\mathcal{P}$. $\mathcal{P}$ aborts if N_s is already recorded in database E.
(2) If $N_s \notin E$, $\mathcal{U}$ continues by sending either:
  (a) The current credential state and policy identifier (S, pid) to $\mathcal{P}$ in the clear when the user's actions are not hidden.
  (b) A commitment on the current credential state and policy identifier $A \leftarrow \mathrm{Commit}(\rho, S, \mathsf{pid})$ to $\mathcal{P}$ when the user's actions are hidden.
(3) $\mathcal{U}$ then conducts an interactive proof of knowledge of $(\sigma_{\mathsf{state}}, S, \mathsf{pid}, sk_{\mathcal{U}}, O)$ s.t.:
  (a) $\mathsf{Verify}(spk_{\mathcal{P}}, \sigma_{\mathsf{state}}, (sk_{\mathcal{U}}, N_s, \cdot, \mathsf{pid}, S)) = 1$.
  (b) $\mathsf{Decommit}(\rho, A, O, S, \mathsf{pid}) = 1$. (When $\mathcal{U}$'s actions are hidden.)
(4) $\mathcal{P}$ adds N_s to the database E.

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

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 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) assumption when instantiated with the RSA-based Camensich–Lysyanskaya (respectively, bilinear Camenisch–Lysyanskaya) signature scheme.*

Due to space constraints, we omit a proof of this theorem. For related proof techniques, we refer the reader to the proof of Theorem 5.2 which addresses the security of the above credential system when combined with the adaptive oblivious transfer protocol of Camenisch et al. [2007].

3.3. From Stateful to Stateless Credentials

One of the shortcomings of our system is that it requires an interactive update procedure to ensure that the user may only show a credential based on her current state. If, however, the access graph does not use state (i.e., the graph only contains a single vertex), then there exists a simple procedure to convert the stateful credential protocols shown in Figures 1 and 2 into an efficient stateless system without the need for interactive updates. To do so, we simply remove the nonces from the credential and discontinue the check for the usage nonce in step two of the ProveCred protocol. Without this check, the user may continue to use her given credential without updating, and the system remains secure since there is no other state for the user to transition to. This approach is equivalent to the stateless credential system of Camenisch et al. [2009]. Later on, we will require stateful credentials to implement history-based access control policies.

3.4. Multishow 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 Camenisch et al. [2006]. To do so, the user keeps a counter $0 < i \leq k$ and reveals the output of a pseudorandom function with seed s evaluated on the counter, $f_s(i)$. In this model, the per-state nonces can be used as the seed s.

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 value 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 [Chan et al. 1998; Camenisch and Michels 1999a, 1999b; Boudot 2000]. 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 either the Dodis–Yampolskiy [Dodis and Yampolskiy 2005] or Naor–Reingold [Naor and Reingold 1997] 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 each user to follow k distinct access control strategies simultaneously, 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 $(\mathsf{pid}, S \to T, k_{S \to T})$ are created, where $k_{S \to T}$ indicates the number of times the user will be allowed to reuse the credential when they use the tag associated with edge $S \to T$. When the user creates the commitment for her new credential in the UpdateCred protocol, she embeds the value $k_{S \to T}$, 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 to limit the update and usage behavior of the resultant credential according to the user's current state and her access history.

3.5. 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 (noninteractive) 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 at a given state. With a small extension to the protocol, *all* nonces can be derived from a single user-specific seed s, which would allow the auditor to trace any transaction conducted by the user.[1]

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 et al. [2005]. In this auditing scheme, the user generates a keypair for a verifiable encryption algorithm and publishes the public key. The provider is given a verifiable encryption C of the user's pseudorandom function (PRF) seed during Setup. The user will be forced by the ProveCred and UpdateCred protocols to reveal some output of the PRF using this seed during each show or update of the credential. The novel feature is that if the user ever attempts to use the same credential "too many" times, the decryption key for the ciphertext C is revealed. The provider can then decrypt C, recover the user's PRF seed, and then use this value to trace all of the user's past transactions.

In more detail, suppose a user may show a credential at most k times. The user chooses two random PRF seeds s, t and gives the provider a verifiable encryption of PRF seed s under the public key associated with $sk_{\mathcal{U}}$. During each transaction, the user sends a token containing the values $S = f_s(i)$ and $T = sk_{\mathcal{U}} \cdot f_t(i)^R$, where $f_{(\cdot)}(\cdot)$ is a pseudorandom function; 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. The user will prove in zero knowledge that she is sending the correct values and that $1 \leq i \leq k$. Notice that if the user wishes to reuse the credential more than k times, then she must provide two tokens where $S_1 = S_2$, which allows the provider to easily detect the abuse. Moreover, the user will have provided two the values 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:

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

[1]In this case, the nonces take the form $f_s(S||i)$ where s is established on a per-user basis, S represents the state, and i indicates how many times the credential has been used in state S.

Once the provider has retrieved the secret key, the verifiable encryption can be decrypted and the provider will then be able to recover the seed s and recompute the $S = f_s(i)$ values for all possible k shows of this credential by this user.

3.6. Efficiency

Since our stateful anonymous credential system is intended to model 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 requires $O(n_i)$ space, which is dominated by the policy graph i. Of course, if the user had access to a reliable policy graph server from which she could anonymously retrieve tags, then her storage requirements would be constant. With regard to the time complexity, each protocol 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, although the constants remain moderate. Thus, the overall time complexity for the user is $O(1)$ to execute each protocol in the 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 $O(\sum_{\forall i} n_i)$, 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 each polynomially-bounded user contributes at most a polynomial number of entries to each database. 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 protocol, the provider need only do $O(1)$ work to issue a fixed-size credential. For the UpdateCred protocol, the provider needs to do one database lookup and then only $O(1)$ work to update one of our fixed-size credentials. Finally, in the ProveCred protocol, the provider needs to do one database lookup and then only $O(1)$ work to check a zero-knowledge proof of knowledge of a fixed size.

Practically speaking, our system is very efficient to execute after the initial Setup protocol where the provider to signs 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 resign those tags (i.e., edges) that have been altered. Unfortunately, the complete removal of individual edges from the graph would not be possible in this model.

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. We now 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 based on her identity, current state, and past history of actions. We describe some example access control models that

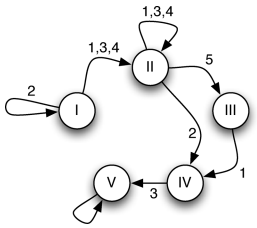

Fig. 3. Sample access policy for a small oblivious database. The labels on each transition correspond to the database item indices that can be requested when a user traverses the edge, with null transitions represented by unlabeled edges.

can be supported by our system. In Section 5, we provide concrete examples of oblivious transfer protocols that have been extended to incorporate these access controls.

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 $(\mathsf{pid}, S \rightarrow T, j)$, where pid is the policy graph identity; $S \rightarrow T$ 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 the action identity are based on the cryptographic system that we are combining our credential system with, such as 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 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 prove 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 [Biba 1977], Clark–Wilson [Clark and Wilson 1987], Bell-La Padula [Bell and La Padula 1988], and Brewer–Nash [Brewer and Nash 1989]. 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.

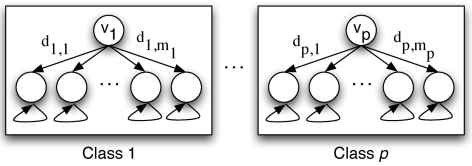

Fig. 4. Example access graphs for the Brewer–Nash model. The user receives one access graph per class, where each graph allows access to at most one of the datasets $d_{i,j}$ for the associated conflict of interest class i.

Discretionary Access Controls. One of the most widely used forms of access controls are discretionary access controls [Department of Defense 1985], 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 [Lampson 1969]. 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.

Brewer–Nash Model. The Brewer–Nash model [Brewer and Nash 1989], otherwise known as the Chinese Wall, is a mandatory access control model that is widely used in the financial services industry to prevent 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 consists of disjoint components for each of the p conflict of interest classes within a single policy graph.[2] Each component 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 conflict of interest classes and is allowed to move within each component independently, thereby allowing

[2]These classes may be split among independent graphs to enforce more granular control over the conflict of interest classes allowed to each user.

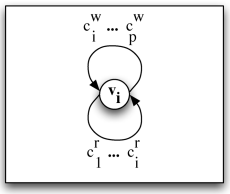

Fig. 5. Example access graph for a user with security level i in the Bell–Padula model. The graph allows read access to all resources in classes c_1^r through c_i^r and write access to all objects in classes c_i^w to c_p^w.

the user access to exactly one dataset (and its respective objects) within each conflict of interest class.

Bell–Padula Model. Another well-known mandatory access control model is the Bell–Padula model [Bell and La Padula 1988], also known as the Multilevel Security model. The Bell–Padula model is designed with the intent of maintaining data confidentiality in a classified or highly-sensitive computer system. 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–Padula 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 of 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 et al. [2007]. Similar methods can be used to couple our system with other oblivious transfer schemes, such as those of Green and Hohenberger [2007], in addition to many other oblivious and anonymous protocols, such as keyword search.

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 now describe the protocol specifications.

(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) $\mathsf{ObtainCred}(\mathcal{U}(pk_{\mathcal{D}}, sk_{\mathcal{U}}, \Pi_i^{\mathcal{C}}, S), \mathcal{D}(pk_{\mathcal{U}}, sk_{\mathcal{D}}, \Pi_i^{\mathcal{C}}, S))$**.** $\mathcal{U}$ 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 $(\mathsf{pid}, S \rightarrow T, j)$ 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 [Camenisch

et al. 2007]. 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 which, 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 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 ObtainCred in the first round; honestly runs its side of the 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 ObtainCred protocol honestly in the first round; executes the user's side of the 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 $\hat{\mathcal{U}}'_i$ send 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, $S_i = \perp$. 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 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 $(\mathsf{access\_req}, t_{id}, b_{\mathcal{T}})$ 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 $\perp$ 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 define the *honest* database $\mathcal{D}'$ as one that sends $M_1, \ldots, M_{N+1}$, with $M_{N+1} = \perp$, in the first round, returns $b_{\mathcal{D}'} = 1$ in all subsequent rounds, and outputs the empty string for D_k. Similarly, an honest user $\mathcal{U}'_i$ runs the Setup and 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:

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

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:

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

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 Camenisch et al. [2007]. 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 ObtainCred protocol, and agrees to be bound by a policy that will control her ability to access the database.

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 her identity or the nature of her request. Figures 6 and 7 describe the protocols in detail.

THEOREM 5.2. *When instantiated with the RSA-based Camensich-Lysyanskaya signature scheme, the protocol described above satisfies database and user security (as defined in Definition 5.1) under the q-PDDH (alternatively, q-BDHE [Camenisch et al. 2009]), 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 Camenisch et al. [2007]. 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:

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

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 $\Pi_1^{\mathcal{C}}, \ldots, \Pi_\eta^{\mathcal{C}}$ and ciphertexts $C_1, \ldots, C_{N+1}$. The simulator

$\mathsf{Setup}(\mathcal{U}(1^k), \mathcal{D}(1^k, \Pi_1, \ldots, \Pi_n, M_1, \ldots, M_N))$**:** 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 $\gamma = (q, \mathbb{G}, \mathbb{G}_T, e, g) \leftarrow \mathsf{BMsetup}(1^\kappa)$, random values $h \in \mathbb{G}$ and $x \in \mathbb{Z}_q$, and sets $H = e(g, h)$. $\mathcal{D}$ 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 $\sigma_{S \to T,j} \leftarrow \mathsf{Sign}(gsk_\mathcal{P}, (\mathsf{pid}, S, T, j))$. 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 $sk_\mathcal{D} = (ssk_\mathcal{D}, gsk_\mathcal{D}, h, x)$. $\mathcal{D}$ then publishes each $\Pi_i^\mathcal{C}$ and $pk_\mathcal{D}$ via an authenticated channel.

$\mathsf{ObtainCred}(\mathcal{U}(pk_\mathcal{D}, sk_\mathcal{U}, \Pi_i^\mathcal{C}, S), \mathcal{D}(pk_\mathcal{U}, sk_\mathcal{D}, \Pi_i^\mathcal{C}, S))$**:** 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 $A \leftarrow \mathrm{Commit}(sk_\mathcal{U}, N_s, \mathsf{pid}, S)$.
(2) $\mathcal{U}$ conducts an interactive proof to convince $\mathcal{D}$ that A contains $sk_\mathcal{U}$ correlated with $pk_\mathcal{U}$ as well as the public values pid, S. $\mathcal{D}$ 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 the committed values in A so that $\mathcal{U}$ obtains the valid state signature σ_state signed under key $ssk_\mathcal{P}$.
(4) $\mathcal{U}$ stores the credential $\mathsf{Cred} = (\Pi_i^\mathcal{C}, S, \sigma_\mathsf{state}, N_s)$.

Fig. 6. Setup and user registration algorithms for an access controlled oblivious database based on the oblivious transfer protocol Camenisch et al. [2007]. The database operator and users first run the Setup portion of the protocol. Each user subsequently registers with the database using ObtainCred.

$\mathsf{AccessAndUpdate}(\mathcal{U}(pk_\mathcal{D}, sk_\mathcal{U}, \mathsf{Cred}, j), \mathcal{D}(pk_\mathcal{D}, E))$**:** 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 a random $N_s' \in \mathbb{Z}_q$ and computes $A \leftarrow \mathrm{Commit}(sk_\mathcal{U}, N_s', \mathsf{pid}, T)$.
(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 $v \in \mathbb{Z}_q$ and sets $V = (A_j)^v$. 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) $A = \mathrm{Commit}(sk_\mathcal{U}, N_s', \mathsf{pid}, T)$.
  (c) $\mathsf{Verify}(spk_\mathcal{P}, \sigma_\mathsf{state}, (sk_\mathcal{U}, N_s, \mathsf{pid}, S)) = 1$.
  (d) $\mathsf{Verify}(gpk_\mathcal{P}, \sigma_{S \to T,j}, (\mathsf{pid}, S, T, j)) = 1$.
(5) If these proofs verify, $\mathcal{U}$ and $\mathcal{D}$ run the CL signing protocol on the committed values in A such that $\mathcal{U}$ obtains σ'_state under signing key $ssk_\mathcal{P}$. $\mathcal{U}$ 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 [Camenisch et al. 2007]). $\mathcal{U}$ computes the message $M_j = B_j / U^{1/v}$.

Fig. 7. Database access protocol for an access-controlled oblivious database based on the adaptive oblivious transfer protocol [Camenisch et al. 2007].

$\hat{\mathcal{D}}'$ checks that the parameters, signatures, and ciphertexts are well-formed and valid. If any check fails, the simulator aborts. Otherwise, $\hat{\mathcal{D}}'$ generates a user keypair for a randomly chosen user and performs the 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 $(\mathsf{join}, i, b_{\mathcal{T}})$ 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 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 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 $(\mathsf{access\_req}, t_{id}, b_{\mathcal{T}})$ message from $\mathcal{T}$ in rounds $2, \ldots, k$, it first checks to see if the value t_{id} has been seen previously. If not, $\hat{\mathcal{D}}'$ 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 AccessAndUpdate protocol with $\hat{\mathcal{D}}$, acting as the selected user on an arbitrary (valid) transition and associated message index. $\hat{\mathcal{D}}'$ keeps a log of all transactions it conducts, recording all of the private values (e.g., decommitments) used to conduct the ZK proofs. If t_{id} has been seen before, $\hat{\mathcal{D}}'$ simply examines its log to recover the commitment, decommitment and nonce used during that earlier transaction, then reinitiates the protocol using the recorded commitment.

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 $\mathbf{Game\ 0} = \mathbf{Real}_{\hat{\mathcal{D}},\mathcal{U}_1,\ldots,\mathcal{U}_\eta}$, 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 $\hat{\mathcal{D}}'$ aborts (and outputs error) whenever the knowledge extractor fails to obtain the value h in ObtainCred and during the initialization phase (or this extracted value is inconsistent). This abort will occur with probability equal to the (negligible) knowledge error of the zero-knowledge protocol. We define the probability that $\hat{\mathcal{D}}'$ aborts after $\eta+1$ extractions by the negligible value $\nu_1(\kappa)$.[3] Thus 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 all of the zero-knowledge proof instances conducted by $\hat{\mathcal{D}}'$ with *simulated* proofs. By the zero-knowledge property we are guaranteed that $Pr[\mathbf{Game\ 2}] - Pr[\mathbf{Game\ 1}] < \nu_2(\kappa)$ for some negligible $\nu_2(\kappa)$.[4]

[3]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.

[4]Many perfect zero-knowledge proof protocols exist, such as Cramer et al. [2000].

Game 3 operates exactly as Game 2, except that $\hat{\mathcal{D}}'$ selects random users and transitions as described above, and modifies the transcript to embed these choices into the commitments. By the hiding property of the commitment scheme, the distribution of these commitments will be indistinguishable from any other set of user/transition choices.[5] Note that this does not effect any zero-knowledge proofs related to these commitments, since those proofs are simulated. For some negligible function $\nu_3(\kappa)$, we have $Pr[\textbf{Game 3}] - Pr[\textbf{Game 2}] \leq \nu_3(\kappa)$.

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 during the blind signing protocol 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 and Lysyanskaya 2002] or bilinear [Camenisch and Lysyanskaya 2004] blind signature schemes, and apply the transformations described by Fischlin and Schröder [2009] to make them selective-failure blind signatures. These signatures are secure under the Strong RSA and LRSW assumptions, respectively. Therefore, we have $Pr[\textbf{Game 4}] - Pr[\textbf{Game 3}] < \nu_4(\kappa)$.

Our simulation incorporates all of the changes outlined above. We can bound D's advantage in distinguishing the simulated interaction from the real protocol to the following negligible value:

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

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 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 $\Pi^{\mathcal{C}}_1, \ldots, \Pi^{\mathcal{C}}_\eta$. Additionally, $\hat{\mathcal{U}}'_i$ generates dummy ciphertext 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 ciphertext C_{N+1} for the null message is created as defined in AccessAndUpdate. $\hat{\mathcal{U}}'_i$ sends the parameters, policy graphs, and ciphertexts 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 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 $(\mathsf{access}, i, S_i, T_i, \alpha, t_{id})$

[5]When using Pedersen [1992] commitments, the commitment values are perfectly hiding.

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

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

Game 1 is the same as Game 0, except that our simulator replaces the zero-knowledge proofs conducted in the normal protocol with *simulated* zero-knowledge proofs. If $Pr[\textbf{Game 1}] - Pr[\textbf{Game 0}]$ 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 $\nu_1(\kappa)$ as a negligible value, and bound $Pr[\textbf{Game 1}] - Pr[\textbf{Game 0}] \leq \nu_1(\kappa)$.

Game 2 is identical to Game 1, except that our simulator generates $B_1, \ldots, B_{N+1}$ as 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 $\nu_2(\kappa)$. We refer the reader to the proof of Claim (2) from Camenisch et al. [2007] for a full proof of this proposition. As such, we have $Pr[\textbf{Game 2}] - Pr[\textbf{Game 1}] < \nu_2(\kappa)$.

Game 3 is defined exactly as Game 2, except that during the AccessAndUpdate protocol our simulator employs the knowledge extractor for the ZK proof system to obtain the user's public key, state, and requested item. If the extractor cannot extract the values, the simulator aborts and outputs error. We note that the probability of failure for our extractor is bounded by the knowledge error of the proof of knowledge (which is conducted $(k\eta)$ times in this protocol). We define the total probability of abort as $\nu_3(\kappa)$.[6] Thus the statistical difference is bounded by

$$Pr[\textbf{Game 3}] - Pr[\textbf{Game 2}] \leq \nu_3(\kappa)$$

In Game 4 we operate as in Game 3, but consider the ability of a cheating user to output (or prove knowledge of) an invalid signature (i.e., a signature that our simulator does not output, either directly or via the blind signature protocol). If the adversary does produce such a signature, we abort the protocol and output error. In practice, a cheating user who produces such a signature can be converted into an existential forger for the blind signature scheme. By the security definition for the blind signature scheme, this abort will occur with at most negligible probability $\nu_4(\kappa)$.[7] We define the total probability that the the simulator aborts (after $(k\eta)$ executions) by the value ν_4. Thus, $Pr[\textbf{Game 4}] - Pr[\textbf{Game 3}] \leq \nu_4(\kappa)$.

Our simulation incorporates all of the changes outlined above. We can bound D's advantage in distinguishing the simulated interaction from the real protocol to the following negligible value:

$$Pr\left[D(X) = 1 : X \xleftarrow{\$} \mathbf{Ideal}_{\hat{\mathcal{P}}',\mathcal{U}_1',\ldots,\mathcal{U}_\eta'}\right] - Pr\left[D(X) = 1 : X \xleftarrow{\$} \mathbf{Real}_{\hat{\mathcal{P}},\mathcal{U}_1,\ldots,\mathcal{U}_\eta}\right]$$
$$\leq \nu_1(\kappa) + \nu_2(\kappa) + \nu_3(\kappa) + \nu_4(\kappa) \quad \square$$

[6]For the Schnorr technique in a group of prime order q, this error is $1/q$ after one extraction.

[7]This property holds under the discrete logarithm assumption for Pedersen [1992] commitments and the factoring assumption for Fujisaki and Okamoto [1997] commitments, and the Strong RSA assumption for the RSA-based CL signatures.

6. CONCLUSION

In this article, 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 [Camenisch and Lysyanskaya 2002, 2004; Boneh and Boyen 2004; Waters 2005]; identity-based key extraction [Green and Hohenberger 2007]; and keyword search protocols [Ogata and Kurosawa 2004]. This work takes a significant step forward in the important balance of user privacy and provider control.

ACKNOWLEDGMENTS

The authors are grateful to the ACM TISSEC anonymous reviewers for their helpful comments.

REFERENCES

AIELLO, W., ISHAI, Y., AND REINGOLD, O. 2001. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology* (*EUROCRYPT'01*), Lecture Notes in Computer Science, vol. 2045, Springer, Berlin, 119–135.

BELL, D. E. AND PADULA, L. J. L. 1988. Secure computer system: Unified exposition and Multics interpretation. MITRE Corp., Bedford, MA.

BIBA, K. J. 1977. Integrity considerations for secure computer systems. Tech. rep. ESD-TR-76-372, MITRE Corp. Bedford, MA.

BLAKE, I. F. AND KOLESNIKOV, V. 2004. Strong Conditional Oblivious Transfer and Computing on Intervals. In *Advances in Cryptology* (*ASIACRYPT'04*), Lecture Notes in Computer Science, vol. 3329, Springer, Berlin, 515–529.

BONEH, D. AND BOYEN, X. 2004. Short signatures without random oracles. In *Advances in Cryptology* (*EUROCRYPT'04*), Lecture Notes in Computer Science, vol. 3027, Springer, Berlin, 56–73.

BONEH, D., BOYEN, X., AND SHACHAM, H. 2004. Short group signatures. In *Advances in Cryptology* (*CRYPTO'04*), Lecture Notes in Computer Science, vol. 3152, Springer, Berlin, 227–242.

BOUDOT, F. 2000. Efficient proofs that a committed number lies in an interval. In *Advances in Cryptology* (*EUROCRYPT'00*), Lecture Notes in Computer Science, vol. 1807, Springer, Berlin, 431–444.

BRANDS, S. 1997. Rapid demonstration of linear relations connected by boolean operators. In *Advances in Cryptology* (*EUROCRYPT'97*), Lecture Notes in Computer Science, vol. 1233. Springer, Berlin, 318–333.

BREWER, D. F. C. AND NASH, M. J. 1989. The Chinese Wall security policy. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, Los Alamitos, CA, 206–214.

CAMENISCH, J. AND DAMGARD, I. 2000. Verifiable encryption, group encryption, and their applications to separable group signatures and signature sharing schemes. In *Advances in Cryptology* (*ASIACRYPT '00*), Lecture Notes in Computer Science, vol. 1976, 331–345.

CAMENISCH, J., DUBOVITSKAYA, M., AND NEVEN, G. 2009. Oblivious transfer with access controls. In *Proceedings of the 16th ACM Conference on Computer and Communications Security* (*CCS'09*). ACM, New York, 131–140.

CAMENISCH, J., DUBOVITSKAYA, M., AND NEVEN, G. 2010. Unlinkable priced oblivious transfer with rechargeable wallets. In *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, vol. 6052, Springer, Berlin.

CAMENISCH, J., HOHENBERGER, S., KOHLWEISS, M., LYSYANSKAYA, A., AND MEYEROVICH, M. 2006. How to win the clone wars: Efficient periodic n-times anonymous authentication. In *Proceedings of the 13th ACM Conference on Computer and Communications Security* (*CCS '06*). ACM, New York, 201–210.

CAMENISCH, J., HOHENBERGER, S., AND LYSYANSKAYA, A. 2005. Compact e-cash. In *Advances in Cryptology (EUROCRYPT'05*), Lecture Notes in Computer Science, vol. 3494, Springer, Berlin, 302–321.

CAMENISCH, J., HOHENBERGER, S., AND LYSYANSKAYA, A. 2006. Balancing accountability and privacy using e-cash. In *Security and Cryptography for Networks*, Lecture Notes in Computer Science, vol. 4116, Springer, Berlin, 141–155.

CAMENISCH, J. AND LYSYANSKAYA, A. 2001. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In *Advances in Cryptology* (*EUROCRYPT'01*), Lecture Notes in Computer Science, vol. 2045, Springer, Berlin, 93–118.

CAMENISCH, J. AND LYSYANSKAYA, A. 2003. A signature scheme with efficient protocols. In *Security in Communication Networks,* Lecture Notes in Computer Science, vol. 2576, Springer, Berlin, 268–289.

CAMENISCH, J. AND LYSYANSKAYA, A. 2004. Signature schemes and anonymous credentials from bilinear maps. In *Advances in Cryptology* (*CRYPTO'04*), Lecture Notes in Computer Science, vol. 3152, Springer, Berlin, 56–72.

CAMENISCH, J. AND MICHELS, M. 1999a. Proving in zero-knowledge that a number n is the product of two safe primes. In *Advances in Cryptology* (*EUROCRYPT'99*), Lecture Notes in Computer Science, vol. 1592, Springer, Berlin, 107–122.

CAMENISCH, J. AND MICHELS, M. 1999b. Seperability and efficiency for generic group signature schemes. In *Advances in Cryptology* (*CRYPTO'99*), Lecture Notes in Computer Science, vol. 1666, Springer, Berlin, 413–430.

CAMENISCH, J., NEVEN, G., AND SHELAT, A. 2007. Simulatable adaptive oblivious transfer. In *Advances in Cryptology* (*EUROCRYPT'07*), Lecture Notes in Computer Science, vol. 4515, Springer, Berlin, 573–590.

CAMENISCH, J. 1998. Group signature schemes and payment systems based on the discrete logarithm problem. Ph.D. dissertation, ETH Zurich.

CHAN, A., FRANKEL, Y., AND TSIOUNIS, Y. 1998. Easy come-easy go divisible cash. In *Advances in Cryptology* (*EUROCRYPT'98*), Lecture Notes in Computer Science, vol. 1403, Springer, Berlin, 561–575.

CHAUM, D. 1985. Security without identification: Transaction systems to make big brother obsolete. *Comm. ACM 28*,10, 1030–1044.

CHAUM, D. AND PEDERSEN, T. P. 1992. Wallet databases with observers. In *Advances in Cryptology* (*CRYPTO'92*), Lecture Notes in Computer Science, vol. 740, Springer, Berlin, 89–105.

CHOR, B., KUSHILEVITZ, E., GOLDREICH, O., AND SUDAN, M. 1998. Private information retrieval. *J. ACM 45*, 6, 965–981.

CLARK, D. D. AND WILSON, D. R. 1987. A Comparison of commercial and military computer security policies. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE, Washington, D.C., 27–29.

CRAMER, R., DAMGARD, I., AND MACKENZIE, P. 2000. Efficient zero-knowledge proofs of knowledge without intractability assumptions. In *Public Key Cryptography*, Lecture Notes in Computer Science, vol. 1751, 354–372.

CRAMER, R., DAMGARD, I., AND SCHOENMAKERS, B. 1994. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology* (*CRYPTO'94*), Lecture Notes in Computer Science, vol. 839, Springer, Berlin, 174–187.

CRESCENZO, G. D., OSTROVSKY, R., AND RAJAGOPOLAN, S. 1999. Conditional oblivious transfer and time released encryption. In *Advances in Cryptology* (*EUROCRYPT'99*), Lecture Notes in Computer Science, vol. 1592, Springer, Berlin, 74–89.

DAMGARD, I. AND FUJISAKI, E. 2002. An integer commitment scheme based on groups with hidden order. In *Advances in Cryptology* (*ASIACRYPT'02*), Lecture Notes in Computer Science, vol. 2501, Springer, Berlin, 125–142.

DEPARTMENT OF DEFENSE. 1985. Trusted computer system evaluation criteria. Tech. rep. DoD 5200.28-STD.

DODIS, Y. AND YAMPOLSKIY, A. 2005. A verifiable random function with short proofs and keys. In *Public Key Cryptology* (*PKC'05*), Lecture Notes in Computer Science, vol. 3386. Springer, Berlin, 416–431.

FISCHLIN, M. AND SCHRODER, D. 2009. Security of blind signatures under aborts. In *Public Key Cryptography* (*PKC'09*), Lecture Notes in Computer Science, vol. 5443, Springer, Berlin, 297–316.

FUJISAKI, E. AND OKAMOTO, T. 1997. Statistical zero knowledge protocols to prove modular polynomial relations. In *Advances in Cryptology* (*CRYPTO'97*), Lecture Notes in Computer Science, vol. 1294, Springer, Berlin, 16–30.

GOLDREICH, O., GOLDWASSER, S., AND MICALI, S. 1986. How to construct random functions. *J. ACM 33*, 4, 792–807.

GOLDREICH, O., MICALI, S., AND WIGDERSON, A. 1986. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proceeding of the 27th Annual Symposium on Foundations of Computer Science* (*FOCS'86*), IEEE, Los Alamitos, CA, 174–187.

GOLDWASSER, S., MICALI, S., AND RIVEST, R. L. 1988. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput. 17*, 2, 281–308.

GOOGLE. 2009. Google Health. https://www.google.com/health.

GREEN, M. AND HOHENBERGER, S. 2007. Blind identity-based encryption and simulatable oblivious transfer. In *Advances in Cryptology* (*ASIACRYPT'07*), Lecture Notes in Computer Science, vol. 4833, Springer, Berlin, 265–282.

JARECKI, S. AND LIU, X. 2009. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *Theory of Cryptography*, Lecture Notes in Computer Science, vol. 5444, Springer, Berlin, 577–594.

LAMPSON, B. W. 1969. Dynamic protection structures. In *Proceedings of the AFIPS '69 Fall Joint Computer Conference*. ACM, New York, 27–38.

LYSYANSKAYA, A. 2002. Signature schemes and applications to cryptographic protocol design. Ph.D. dissertation, MIT, Cambridge, MA.

MICROSOFT. 2009. Microsoft HealthVault. http://www.healthvault.com/.

NAOR, M. AND PINKAS, B. 1999. Oblivious transfer with adaptive queries. In *Advances in Cryptology* (*CRYPTO'99*), Lecture Notes in Computer Science, vol. 1666, Springer, Berlin, 573–590.

NAOR, M. AND REINGOLD, O. 1997. Number-theoretic constructions of efficient pseudo-random functions. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science* (*FOCS'97*). IEEE, Los Alamitos, CA, 458–467.

OGATA, W. AND KUROSAWA, K. 2004. Oblivious keyword search. Special issue on coding and cryptography. *J. Complexity 20*, 2–3, 356–371.

PEDERSEN, T. P. 1992. Non-interactive and information-theoretic secure verifiable secret sharing. In *Advances in Cryptology* (*CRYPTO'92*). Lecture Notes in Computer Science, vol. 576, Springer, Berlin, 129–140.

SCHNORR, C.-P. 1991. Efficient signature generation for smart cards. *J. Cryptology 4*, 3, 239–252.

TERANISHI, I., FURUKAWA, J., AND SAKO, K. 2004. k-times anonymous authentication. In *Advances in Cryptology* (*ASIACRYPT'04*), Lecture Notes in Computer Science, vol. 3329, Springer, Berlin, 308–322.

WATERS, B. 2005. Efficient identity-based encryption without random oracles. In *Advances in Cryptology* (*EUROCRYPT'05*), Lecture Notes in Computer Science, vol. 3494, Springer, Berlin, 114–127.

Received January 2010; revised September 2010; accepted October 2010