Exploiting Hierarchical Identity-Based Encryption for Access Control to Pervasive Computing Information

Urs Hengartner and Peter Steenkiste

October 2004
CMU-CS-04-172

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Access control to sensitive information available in pervasive computing environments is challenging for multiple reasons: First, access control must support flexible access rights that include context-based constraints. Second, a client requesting access to sensitive information might not know which of its access rights are necessary in order to be granted access to the requested information. Third, pervasive computing environments consist of a multitude of information services, which makes simple management of access rights essential. Given this setting, we discuss the shortcomings of existing access control schemes that rely either on information services encrypting sensitive information before handing it over to clients or on clients presenting a proof of access to a service before being granted access. To address these shortcomings, we develop a solution based on hierarchical identity-based encryption. Namely, we present an encryption-based access control architecture that exploits hierarchical identity-based encryption in order to deal with multiple, hierarchical constraints on access rights. Furthermore, we introduce a proof-based access control architecture that employs hierarchical identity-based encryption in order to enable services to inform clients of the required proof of access in a covert way, without leaking information. We present an example implementation of our proposed schemes and discuss its performance.

This research was supported by the Army Research Office through grant number DAAD19-02-1-0389 and by the NSF under award number CNS-0411116.

Keywords: Access Control, Identity-Based Encryption, Pervasive Computing

1 Introduction

Whereas access control to sensitive information has been well investigated for traditional distributed systems (e.g., file systems), there are additional challenges for pervasive computing environments. For example, access rights need to be more flexible; it should be possible to issue access rights that depend on a person's context, such as her location or the current time. In addition, there might be *covert access requirements*. Namely, a client accessing some complex information might not know which of its access rights are required for gaining access. For instance, a person's calendar entry reveals the location of the people that the person is currently meeting with. In order to be granted access to this entry, a client should at least have access rights to each of these people's location information. However, since the client does not know who the person is meeting with, it does not know which of its access rights are required.

There are *encryption-based* and *proof-based* access control schemes. In an encryption-based scheme, a service provides sensitive information to any client, but only in an encrypted form. Only clients authorized to access the information have the required decryption key. This approach is attractive for scenarios where there are lots of queries to a service since it shields the service from having to run client-specific access control. It is straightforward to add support for covert access requirements to existing encryption-based schemes [1, 15, 20, 25, 30]. In particular, a service encrypts information as usual, but it does not tell a client which decryption key to use. Instead, the client needs to search its set of decryption keys for a matching key. However, it is less straightforward to add support for constraints on access rights to the proposed schemes, especially when considering that key management should remain simple.

In a proof-based access control scheme, a client requesting access to sensitive information needs to assemble access rights in a *proof of access*, which demonstrates to the service that the client is authorized to access the requested information. This approach is attractive for scenarios where flexible, client-specific access rights are required. A proof of access prevents a service from having to locate the required access rights itself, which can be an expensive task. Since access rights are flexible, it is easy to include support for constraints in them. When validating a proof of access, a service must also validate all the constraints on the access rights in the proof. However, it is difficult to add support for covert access requirements to proof-based access control. Existing schemes [2, 17] assume that a service can inform a client of the required proof of access. However, in our example mentioned above, a service informing a client of the identity of the people that the owner of the calendar entry is meeting with would result in an information leak. A naïve solution is to have the client transmit a proof of access for all individuals whose location it can access. This solution has privacy and bandwidth issues: a service can learn a lot about a client, and a client might have to transmit a lot of data. Therefore, a service must be able to let a client know about the required proof of access in a way such that only authorized clients can learn about the information being part of this proof description, otherwise this information will leak.

We present two novel applications of hierarchical identity-based encryption that address the above mentioned shortcomings of encryption-based and proof-based schemes in the context of pervasive computing environments (Section 3). In identity-based encryption, public keys are arbitrary strings, which simplifies management of access rights and constraints. First, we develop a hierarchical identity-based encryption scheme for encryption-based access control that supports multiple, hierarchical constraints on access rights. Second, we employ hierarchical identity-based encryption to implement covert access requirements in proof-based access control. Our contributions include extensions to an existing hierarchical identity-based encryption scheme to support constraints on access rights and novel ways of dealing with expiration of access rights in identity-based encryption. We have implemented our solutions in a pervasive computing environment (Section 4). Finally, we provide an overall evaluation and discuss the relative strengths and weaknesses of our example implementation (Section 5).

2 Access Control to Information in Pervasive Computing

In this section, we discuss the concepts of access control and access rights to information in the context of pervasive computing. We present a list of requirements and our threat model.

2.1 Overview

In pervasive computing environments, such as CMU's Aura [12], there are a lot of *services* that provide potentially sensitive information to *clients*. Clients need to have *access rights* in order to be granted access to sensitive information. An access right has an issuer, a recipient, an information item, and a set of constraints. For example, Alice grants Bob access to her location information during office hours. Multiple services may offer the same type of information (e.g., cell phone-based location information, WiFi-based location information, badge-based location information,...). To simplify management of access rights, we want service-independent access rights, that is, access rights should be about information, not about information offered by a specific service. For example, there should be access rights for Alice's location information, not for Alice's location information as offered by her cell phone service.

It should be possible to constrain access rights. In this paper, we limit ourselves to constraints whose current value is always available to a client (e.g., current time or location of the client). Having other types of constraints (e.g., location of the queried individual) requires more complex access control in order to avoid information leaks, which is out of the scope of this paper. In addition, access rights should be granularity aware. Some information (e.g., location information) is available at different levels of granularities (e.g., "CMU", "CMU Wean Hall", "CMU Wean Hall 8220"). Having an access right for fine-grained information should imply having an access right for coarse-grained information. Granularity-aware access rights simplify management of access rights.

Access rights are managed by *policymakers*. Typically, an individual is the policymaker for her own personal information. Depending on the access control scheme, access rights can be represented in different forms. For encryption-based access control, an access right is a decryption key, whereas for proof-based access control, it typically is a signed statement (i.e., a digital certificate) issued by the policymaker. Regardless of the form, it should be simple to deal with access rights for all involved entities (clients, services, and policymakers).

We now discuss how encryption-based and proof-based access control meet the requirements of granularity awareness and constraints. We also elaborate on some additional requirements, namely, indistinguishability, asymmetry, and personalization.

2.2 Encryption-based Access Control

If there are lots of requests for some information, encryption-based access control is attractive since it is independent of the individual clients issuing these requests. For example, a service can encrypt an information item once and use the ciphertext for answering multiple requests asking for this item. However, the uniform treatment of requests makes dealing with constraints on access rights and with granularity-aware access rights difficult. Covert access requirements and service-independent access rights present further challenges. Let us summarize the requirements:

Constraints. Each possible value of a constraint must require a separate decryption key for decrypting some encrypted information that should be accessible only under the given constraint/value combination.

For example, a decryption key that allows a client to access some encrypted information on “January 1” must not allow decryption on “January 2”. This requirement leads to an increase in the number of keys. The problem becomes worse when there are multiple constraints on some access right. Luckily, we observe that many constraints are of a hierarchical nature. Therefore, we want a key scheme that supports hierarchical constraints. For example, given the decryption key for “January”, we can derive the key for “January 1”, “January 2”,... Similarly, the key for (“January 1”, “Wean Hall 8220”) can be derived from the key for (“January, “Wean Hall”). This feature can drastically simplify key management.

Granularity awareness. To enforce that access rights to coarse-grained information do not grant access to fine-grained information, we require separate decryption keys for the two cases. Similar to constraints, a naïve implementation of granularity-aware access rights can lead to an increase in the number of keys. With a hierarchical key scheme, we can avoid this increase. In particular, the decryption key for coarse-grained information should be derivable from the decryption key for fine-grained information.

Indistinguishability. To implement covert access requirements, encrypted information returned by a service must not reveal any knowledge about the used encryption key or the required decryption key. Only a client having this decryption key should be able to gain this knowledge.

Asymmetry. Service-independent access rights grant access to some information independent of the service offering this information. This concept implies that if multiple services offer the same information, this information will be decryptable with the same decryption key. Therefore, in a symmetric cryptosystem, a service encrypting information would be able to access the same information offered by some other service. For example, a cell phone service offering Alice’s cell phone-based location information would be able to access her WiFi-based location information as offered by some other service. We can avoid this problem by using an asymmetric cryptosystem.

2.3 Proof-based Access Control

Proof-based access control is attractive since it offloads the assembly of the proof of access to a client. If the client does not know about the required proof of access, a service will give it a description of this proof. However, when this proof description contains sensitive information, the service must obscure the proof description. Let us summarize the requirements for this case:

Indistinguishability. The service must obscure the description such that only clients authorized to access the sensitive information can learn about the policymaker responsible for this information and the nature of the information.

Granularity awareness. Understanding obscured proof description should be granularity aware: Being able to interpret an obscured proof description for fine-grained information should imply being able to interpret an obscured proof description for coarse-grained information.

Constraints. Since access rights can have constraints on them, these constraints should also apply to a client’s ability to interpret an obscured proof description. For example, when a client’s access right for some information expires, the client should no longer be able to interpret obscured proof descriptions asking for a proof of access for this information.

Personalization. We want obscured proof descriptions to be personalized for a client. In this way, if a client leaked its secret knowledge required for understanding an obscured proof description for some information, other clients being able to understand a proof description for the same information would not be affected. Clients do not have to be malicious to leak their secret knowledge; since it has no value for them (as opposed to their private key), they might neglect keeping it secret. (We do not require personalization for encryption-based access control since it is is client independent by design.)

Asymmetry. Based on the same argument as in the case of encryption-based access control, a service generating an obscured description of the required proof of access for some information must not be able to identify an obscured proof description for the same information generated by some other service.

2.4 Threat Model

In our threat model, an attacker can corrupt clients or services, but not policymakers. Corrupted clients try to gain non-authorized access to information provided by a service, that is, information to which a client does not have any access rights. Corrupted clients can collude. A corrupted service tries to gain non-authorized access to information provided by some other service, the information can be of the same type as the one that it offers. Corrupted services can collude. Attackers can also sniff, modify, or inject traffic between clients, services, and policymakers. We do not explicitly address denial of service attacks, though we try to limit load on services.

3 Access Control based on Hierarchical Identity-based Encryption

We want an access control architecture where access rights are simple to manage, aware of granularity, and constrainable. In addition, the architecture has to be asymmetric, provide indistinguishability, and be personalizable in the case of proof-based access control. Identity-based encryption (IBE) is a good fit for environments that have these requirements. It is asymmetric and provides indistinguishability. Since public keys are arbitrary strings, key and access right management are simple. In addition, a hierarchical version of identity-based encryption lends itself to the implementation of hierarchical constraints and granularity awareness. Some modifications also allow for the support of personalization. Therefore, we propose an access control architecture for pervasive computing environments that is based on hierarchical identity-based encryption (HIBE). In this section, we review HIBE and discuss how we extend it to build an access control architecture satisfying our requirements.

3.1 Hierarchical Identity-based Encryption

In an IBE scheme, the public key of an individual is an arbitrary string, typically corresponding to her ID (e.g., her email address) [21]. The individual gets her private key from a third party, called a Private Key Generator (PKG). The third party also provides additional, public parameters required for the cryptographic operations. Boneh and Franklin [4] present one of the first practical IBE schemes. Based on this work, Gentry and Silverberg [13] introduce a HIBE scheme. In this scheme, a root PKG gives out private keys to sub PKGs, which in turn give out private keys to individuals in their domains (or further sub PKGs). The public key of an individual corresponds to the IDs associated with the root PKG, any sub PKGs on the path from the root PKG to the individual, and the individual. For encrypting messages, additional public parameters are required only from the root PKG.

The limited success of PKI has lead to the development of simpler public key infrastructures (e.g,. SPKI [9]), that do not require (hierarchical) certification authorities. In SPKI, a user's public key is her identity, and not her name as certified by an authority. In our work, we pursue a similar approach. Instead of requiring the existence of a hierarchical PKG infrastructure, we let each policymaker have its own PKG. The policymaker uses its PKG for managing access rights to its information. A policymaker can set up a hierarchical PKG infrastructure, where it controls both the root PKG and any sub PKGs. In this way, a policymaker will be able to establish granularity-aware access rights and hierarchical constraints (see

Section 3.3). Boneh and Franklin [4] also suggest a deployment scenario where individuals become PKGs. In the rest of this paper, we refrain from talking about PKGs and use the term "policymaker" instead.

Our architecture builds on the HIBE scheme proposed by Gentry and Silverberg. Their proposed scheme supports only a single hierarchy for a root PKG, which is too limiting for our application scenarios, where we might have multiple, hierarchical constraints on some access rights. Therefore, we extend the scheme to support multiple hierarchies.

A HIBE scheme has the advantage that it reduces the amount of required storage and the complexity of the access right management. As we will see in Section 3.3, the public key of some information corresponds directly to the identification string of the information. There is no need for storing a separate public key (obtained from a conventional cryptosystem such as RSA) or some other public value, as suggested in earlier work [19], for each information item. Maintaining the mappings from some information to a key or a public value would also make access right management more difficult. We discuss the advantages of using a HIBE scheme in more detail in Section 3.6.

3.2 Basic Operations

Our architectures for encryption-based and proof-based access control each employ four basic, randomized operations. We discuss these operations in this section and their application in encryption-based and proof-based access control in the next two sections. Our operations are based on the operations introduced by Gentry and Silverberg [13], we extend them to support multiple hierarchies. A detailed discussion, giving the exact cryptographic steps for each operation, is in Appendix A. For readability reasons, we omit some of the parameters of the operations here.

We assume that all the policymakers agree on a set of public parameters, $params$. We require this agreement in order to achieve indistinguishability. The basic operations are $Root\_Setup()$, $Extract()$, $Encrypt()$, and $Decrypt()$.

- $Root\_Setup(params) \rightarrow Q_0$:
  A policymaker runs this operation in order to generate the policymaker's master secret. In addition, the operation returns the policymaker's public key, Q_0.

- $Extract(\langle ID_{i,1}, ..., ID_{i,t_i}\rangle, S_{i,t_i-1}, params) \rightarrow S_{i,t_i}$ with $t_i \geq 1$:
  This operation returns the private key, S_{i,t_i}, of a node at level t_i in hierarchy i. Unless $t_i = 1$, this key is derived from the private key of the ancestor node, S_{i,t_i-1}. If $t_i = 1$, this operation needs to be run by a policymaker, since it requires its master secret. $\langle ID_{i,1}, ..., ID_{i,t_i}\rangle$ is the sequence of node IDs along the path from the root node of hierarchy i to the node in question.

- $Encrypt(\langle ID_{1,1}, ..., ID_{1,t_1}\rangle, ..., \langle ID_{h,1}, ..., ID_{h,t_h}\rangle, M, Q_0, params) \rightarrow C$:
  After choosing a node in each hierarchy, a service uses this operation to encrypt a message M using the nodes' public keys. For each of the h hierarchies, the operation accepts a sequence of node IDs, $\langle ID_{i,1}, ..., ID_{i,t_i}\rangle$, from the root node to the chosen node. The operation returns ciphertext C.

- $Decrypt(\langle S_{1,t_1}, ..., S_{h,t_h}\rangle, C, params) \rightarrow M$:
  A client uses this operation to decrypt ciphertext C. The operation requires the private key of each node chosen by the service in its call to $Encrypt()$ and the ciphertext.

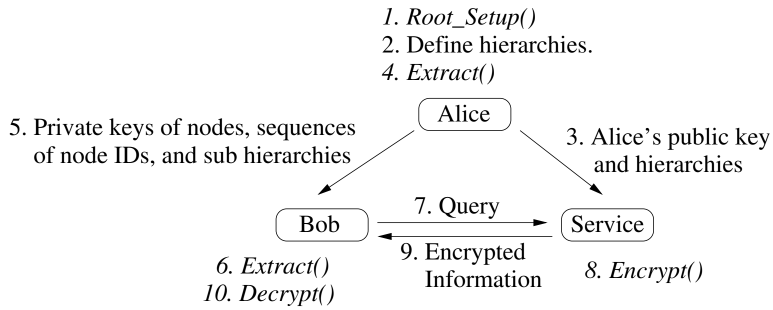

Figure 1: *Architecture for encryption-based access control. Alice sets up her IBE scheme and hierarchies, informs the service, and grants access to Bob. Bob issues a query to the service.*

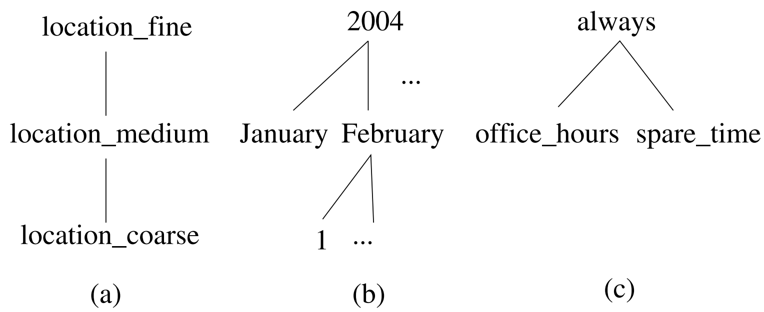

Figure 2: *Hierarchies. Alice establishes hierarchies for her location information (a) and for each of her constraints (b, c).*

3.3 Encryption-Based Access Control

Figure 1 gives an overview of the architecture for encryption-based access control, which consists of three entities: the policymaker managing access rights to her personal information ("Alice"), the client trying to access this information ("Bob"), and the service offering this information. In our architecture, we keep key management simple by using the identification string of the information as its public key. To support granularity-aware access rights and constraints on them, we let Alice define a set of hierarchies that reflect the granularity properties of her information and her constraints. We now discuss the individual steps shown in Figure 1 in detail.

Setup. Alice runs $Root\_Setup()$ to set up her IBE scheme (1) and to retrieve her public key. She also establishes multiple hierarchies (2): She first defines a hierarchy resembling the granularity properties of some information about her (*information hierarchy*). Figure 2 (a) gives an example hierarchy for location information. The rule for a hierarchy is that anyone who has access to information covered by a node should also have access to information covered by a child node. Alice then establishes another hierarchy for each of the constraints that she wants to include in her access rights (*constraint hierarchies*). Figure 2 (b) shows a hierarchy that restricts the lifetime of an access right, and Figure 2 (c) presents a hierarchy for limiting access based on time of the day. (Non-hierarchical constraints are dealt with similarly; there, the hierarchy has only one level and lists all possible values.) Alice then informs the service of her public key and her hierarchies (3). Since none of this information is confidential, there is need only for an authenticated channel. Instead of defining her own hierarchies and submitting them to the service, Alice can exploit predefined hierarchies that the service is already aware of. For example, we expect that there will be a widely accepted hierarchy for location information, which is commonly used by location services and to which Alice can refer.

Alice grants Bob access to some of her information. In her information hierarchy, she chooses the node corresponding to the information to which she wants Bob to have access (e.g., "location_medium"). She then walks the path from the root node to this node. In particular, she keeps a sequence of node IDs and, for each node on the path, she calls $Extract()$ with the current sequence (e.g., $Extract(\langle$location_fine$\rangle$, null, $params)$ $\rightarrow S_{1,1}$ and $Extract(\langle$location_fine, location_medium$\rangle, S_{1,1}, params) \rightarrow S_{1,2})$ (4). Ultimately, this process will return the private key of the chosen node. Similarly, for each type of constraint, she picks the appropriate node in the corresponding constraint hierarchy and derives the private key by repeated calls to $Extract()$. For each hierarchy, Alice will end up with a private key. She then gives the tuple of private keys to Bob, together with the corresponding sequences of node IDs and the sub hierarchies rooted in the chosen nodes (5). Transfer of the private keys requires a secret channel.

Given the tuple of private keys and the sub hierarchies from Alice, Bob can derive additional tuples of private keys for nodes in the sub hierarchies by (repeatedly) calling $Extract()$ (6). For example, given the private key for $\langle$location_fine, location_medium$\rangle$ and the sub hierarchy "location_coarse", Bob can extract the private key for $\langle$location_fine, location_medium,location_coarse$\rangle$. It is possible for Bob to delay this step till he receives encrypted information from a service requiring a particular tuple of private keys derivable by Bob.

Access Control. When queried by Bob for information about Alice (7), the service encrypts the information (8) and returns the encrypted information to Bob (9). Namely, the service splits up the information based on its granularity properties and encrypts each piece separately. For example, the information "CMU Wean Hall 8220" is split up into "CMU", "Wean Hall", and "8220". Then, for each piece, the service locates the node in Alice's information hierarchy that describes the piece and gathers the IDs of all the nodes along the path from the root node to this node. In our example, the ID sequences are $\langle$location_fine, location_medium, location_coarse$\rangle$, $\langle$location_fine, location_medium$\rangle$, and $\langle$location_fine$\rangle$, respectively. Similarly, for each of the constraint hierarchies, the service chooses the leaf node that contains the current value of the constraint and gathers the IDs along the path from the root node. The service then calls $Encrypt()$ with the gathered sequences of node IDs (e.g., $Encrypt(\langle$location_fine, location_medium, location_coarse$\rangle$, $\langle$2004, February, 2$\rangle$, $\langle$always, office_hours$\rangle$, "CMU", $Q_0, params))$. Note that the public keys used for encryption correspond directly to the identification strings of nodes. Bob decrypts the received ciphertexts by calling $Decrypt()$ with the required tuple of private keys (10) for each ciphertext. He can decrypt a ciphertext only if the encrypted information is of a granularity that he has access to.

Discussion. Bob typically has multiple tuples of private keys, either by deriving them or because he has been given multiple tuples by a or multiple policymakers. As explained in Section 1, he might not know which tuple to use for decryption, and the service cannot tell him in order to avoid information leaks. In this case, Bob has to search his tuples till he finds a tuple that allows successful decryption. We discuss ways to limit the search space in Section 3.5.

Our IBE-based scheme fulfills the requirements of being asymmetric and hierarchical and supporting multiple, hierarchical constraints. Using the identification string of some information or of a constraint directly as its public key drastically simplifies key management. Compared to a previous approach for dealing with expiration [4], which makes the current date part of the identification string of some information, our approach does not require handing out separate private keys for each possible date.

Security Analysis. The security of the scheme is based on the hardness of the Bilinear Diffie-Hellman problem. (Please refer to Appendix A for details.) Given this assumption, Gentry and Silverberg [13] show that their HIBE scheme has adaptive chosen ciphertext security in the random oracle model. It is straightforward to adapt their proof for multiple hierarchies. Therefore, corrupted clients and services and

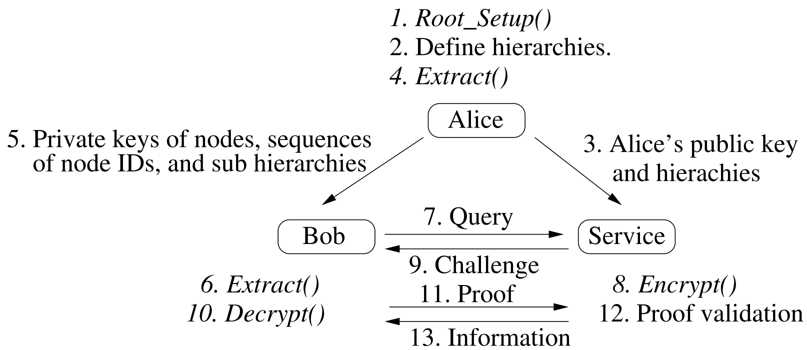

Figure 3: *Architecture for proof-based access control. The service sends a challenge to Bob. Upon resolving this challenge, Bob sends a proof of access to the service.*

traffic sniffers cannot decrypt encrypted information without having the required decryption key or modify encrypted information. In the case where a client (or traffic sniffer) does not know which decryption key to apply, we need to ensure that the client cannot learn from the ciphertext which public key of which policymaker was used to produce this ciphertext (indistinguishability). Holt et al. [16] prove this property for the scenario where all the policymakers share the same set of public parameters, as assumed in our model.

Our scheme is not secure if a client is given access rights to different types of information by the same policymaker. For example, for the hierarchies given in Figure 2, assume that Bob has the tuple of private keys for (⟨location_fine⟩, ⟨2004⟩, ⟨always⟩). In addition, Bob has access to some information other than location information, for example, he has the tuple of private keys for (⟨medical⟩, ⟨2004, January⟩, ⟨always, office_hours⟩). This setup allows Bob to derive the tuple for (⟨medical⟩, ⟨2004⟩, ⟨always⟩). We can fix this problem by including the ID of the root node of an information hierarchy in the root nodes of the corresponding constraint hierarchies. For example, for the constraint hierarchies in Figure 2, their root nodes would become "2004_location_fine" and "always_location_fine". This fix has the drawback that it makes key management for the policymaker more difficult. The policymaker can no longer reuse private keys of a constraint hierarchy when it wants to grant access to different types of information under the same constraint.

Our scheme is not secure against collusion. For example, for the hierarchies given in Figure 2, assume that Bob has the tuple of private keys for (⟨location_fine⟩, ⟨2004⟩, ⟨always, office_hours⟩) and that Carol has the tuple for (⟨location_fine⟩, ⟨2004, January⟩, ⟨always⟩). If Bob and Carol colluded, they could determine the tuple for (⟨location_fine⟩, ⟨2004⟩, ⟨always⟩). Yao et al. [28] propose a collision resistant HIBE scheme. However, the complexity of the $Encrypt()$ and $Decrypt()$ operations in their scheme is $\mathcal{O}(n^m)$, where n is the depth of a hierarchy and m is the number of hierarchies. As we will see in Section 5, the complexity of the operations in our scheme is $\mathcal{O}(mn)$.

3.4 Proof-Based Access Control

If Alice hands over an access right for some information to Bob, she will also give him a personalized secret. When Bob receives an obscured proof description for this information from a service, this secret will allow him to interpret the description. In the rest of this paper, we use the term *challenge* for such an obscured proof description. We keep management of the challenges simple by using the identification string of some information for generating a challenge for it. In our architecture, a challenge corresponds to a ciphertext and a secret corresponds to a tuple of private keys enabling the decryption of ciphertexts.

To support granularity-aware, constrainable challenges and secrets, Alice also defines a set of hierarchies. The architecture for proof-based access control is given in Figure 3; it is similar to the architecture for encryption-based access control given in Figure 1. We now review the changes.

Setup. Alice defines an information hierarchy and constraint hierarchies (2) and submits them to the service (3). To allow Alice to issue personalized secrets to clients, she could define another hierarchy listing all the clients. However, as we will see in Section 5, the cost for some of the cryptographic operations is proportional to the number of hierarchies. Therefore, we refrain from introducing another hierarchy. Instead, we have Alice personalize the information hierarchy by adding the identity of a client to its root node. For example, for the hierarchy given in Figure 2 (a), the root node becomes "location_fine_Bob.[1] Since this personalization is done in the same way for each client, there is no need for Alice to submit each personalized information hierarchy to the service. To avoid collusion attacks between clients, Alice should also personalize each of her constraint hierarchies.

When issuing an access right to Bob for some information (e.g., in the form of a digital certificate), Alice also gives Bob a personalized secret, corresponding to the information in the access right and limited to the same constraints (5). She generates this secret by calling $Extract()$ for the information hierarchy and for each of the constraint hierarchies (4). The tuple of private keys returned by these calls serve as the secret.

Access Control. Bob issues a query to the service and fails to submit a proof of access (7). Assuming that the requested information requires covert access requirements, the service computes a challenge for it (8). In particular, the service calls $Encrypt()$ to encrypt a random plaintext, M. The public keys required for this operation come from the information hierarchy and the constraint hierarchies of the policymaker responsible for the requested information (e.g., $Encrypt(\langle$location_fine_Bob$\rangle$, $\langle$2004_location_fine_Bob, February, 2$\rangle$, $\langle$always_location_fine_Bob, office_hours$\rangle, M, Q_0, params))$. Plaintext M and the obtained ciphertext, C, serve as challenge, and the service sends them to Bob (9). If the requested information covers multiple individuals, there will be multiple challenges. Sending a challenge to Bob requires only an authenticated channel, since a challenge is personalized to a client and useless to other clients (without knowing the corresponding personalized secret).

To resolve challenge (M, C), Bob needs to find a tuple of private keys that makes ciphertext C decrypt to plaintext M. In particular, Bob calls $Decrypt()$ for each of his (potentially derived) tuples of private keys given to him by Alice (and other policymakers) (10). He stops when the returned plaintext is identical to M. We discuss ways to limit the search space in Section 3.5. If Bob successfully resolves the challenge(s), he will resubmit the query, together with the required proof of access (11). The service will validate the proof (12) and return the requested information (13). Steps (11) and (13) require a secret channel.

Discussion. The benefits of our scheme are secrets that are personalized, support constraints, and are granularity aware. Because the challenge for some information is based on the identification string of the information, challenges are simple to manage. Since all the policymakers use the same set of public parameters, the challenges generated by a service are indistinguishable.

A client uses its secrets to resolve a challenge before submitting the required proof of access to the service. However, for some scenarios, this second step can be omitted since resolving the challenge(s) already gives the client all the information it is asking for. For example, if the client asks for the people in a room, the client will require access to all these people's location information. The service thus sends a challenge for each person's location information to the client. After resolving these challenges, the client knows all the policymakers in the room and thus all the originally requested information and can skip submission of a proof of access. An obvious question is why not skip this second step all the time and stop using proofs of access? In this model, the service would encrypt the requested information instead of

[1] In the actual implementation, Bob is identified by his public key.

a random plaintext (as suggested by Holt et al. [16]). We refrain from adapting this model because, as we will see in Section 5, especially decryption of information is an expensive operation. We view covert access requirements as a special case. For most queries, we expect clients to know what they need to deliver a proof of access for. Therefore, we do not place the burden of decrypting information on them for every request to sensitive information.

Security Analysis. As mentioned in Section 3.3, the security of the scheme is based on the hardness of the Bilinear Diffie-Hellman problem. (Please refer to Appendix A for details.) When choosing a random plaintext, a service should choose it long enough to make the probability of the client seeing a false positive while resolving the challenge small. (For a plaintext of length l, the probability of a false positive when using random tuples of private keys is $1/2^l$.) A false positive will make the client send a wrong access right to the service. If the access right contained private information, this information would leak to the service.

Internally, the $Encrypt()/Decrypt()$ operations compute a random value, use it as an exponent in a modular exponentiation, hash the resulting value into the domain of the plaintext, and XOR the hashed value with the plaintext. In proof-based access control, instead of using $Encrypt()/Decrypt()$ on a known, random plaintext, we could omit the XORing step and directly use the result of the exponentiation step as a challenge. This approach relies on the hardness of the Decision Bilinear Diffie-Hellman problem. (Please refer to Appendix A for details.) We choose an approach based on $Encrypt()/Decrypt$ since it allows us to use the same basic routines for both encryption-based and proof-based access control and since the Bilinear Diffie-Hellman problem is at least as hard as the Decision Bilinear Diffie-Hellman problem.

3.5 Limiting the Search Space

For covert access requirements, Bob does not know which of his (potentially derived) tuples of private keys to use for the $Decrypt()$ operation, and he has to search through his tuples. We discuss some optimization strategies in this section.

We first concentrate on the scenario where the challenge or the encrypted information returned by a service cover a single individual only, that is, Bob needs to find only one tuple of private keys. As described in Section 3.3, when a policymaker gives a tuple of private keys to Bob granting him access to some information under some constraints, Bob can potentially derive additional tuples from this tuple. We argue that among the original tuple and the derived tuples, at most one tuple is of relevance for the search. In particular, since we assume that Bob is aware of the current value of a constraint, Bob knows which private key is relevant for each constraint hierarchy, and he can throw out all the tuples not having this private key within them. In practice, we expect that Bob can also limit the search space for the information hierarchy. In many cases, it is safe for the service to inform Bob of the nature and the granularity of the information for which he needs to resolve a challenge, but not about the identity of the policymaker the information is about. This observation exploits the fact that the composition of most types of information is well known. For example, calendar information is composed of fine-grained location information and activity information, but not of medical information. Therefore, when Bob asks the service for calendar information, the service can safely inform him that a challenge involves fine-grained location information. In summary, for all tuples of private keys given to Bob by a single policymaker and all tuples derivable from these tuples, we expect at most one tuple to be relevant for a search. In summary, the number of tuples that Bob needs to search is at most one per policymaker.

If the information returned by a service covers multiple individuals (e.g., a service encrypts information multiple times or returns multiple challenges), Bob will have to locate multiple tuples of private keys. Therefore, Bob's search cost is proportional to the number of tuples of private keys given to him by policymakers multiplied by the number of individuals covered by the information returned by the service. While this sounds expensive, Bradshaw et al. [5] present an optimization that requires the client to perform the most expensive cryptographic operation in this search only once for each tuple of private keys and not for each combination of a tuple of private keys and covered individuals.

3.6 Discussion

A motivation for the design of IBE was to simplify certificate management in email systems [21]. For example, IBE allows Bob to encrypt email to Alice simply by using her email address as public key. There is no need for Bob to contact Alice beforehand to acquire a separate public key. In our IBE scheme, we seem to lose this advantage: Alice needs to inform a service of her hierarchies and her public key. However, as mentioned in Section 3.3, we do not expect each policymaker to define its own set of hierarchies. Instead, there can be a shared set of hierarchies, which a service is aware of. In addition, we argue that a setup step is also necessary for IBE in an email system: First, IBE schemes typically require a set of public parameters for encryption. Bob must acquire these parameters before he can encrypt email for Alice. Second, Bob should ensure that the email address he is going to use to encrypt sensitive information destined for Alice really belongs to Alice. He should use this address only if he was given it directly by Alice (or a trusted third entity) in a setup step.

In our HIBE scheme, the public key of some information corresponds to its identification string in the hierarchy. An alternative design approach is to have the policymaker assign a public key of a conventional asymmetric cryptosystem (e.g., RSA) to each node in the hierarchy. When handing out the hierarchy to services, the policymaker also gives them all the public keys in the hierarchy. Similarly, when handing out a sub hierarchy to clients, the policymakers also gives them all the corresponding private keys. This approach has the drawback that it increases the key material that needs to be stored and transferred. Instead of assigning a key to each node in a hierarchy, Ray et al. [19] suggest a more sophisticated scheme in which the public key of a node can be derived from the public key of the parent node and in which the private key of a node can be used to decrypt information encrypted with the public key of its child node. However, this scheme (and similar algorithms [1, 15, 25, 20]) still requires more key material to be stored and transferred, since for each node, we would have to keep not only its ID, but also an additional, public value required by the algorithm. Moreover, this scheme makes management of access rights more difficult when a policymaker uses a shared hierarchy instead of defining its own. Namely, the policymaker would still have to generate its own set of public values for all the nodes in the shared hierarchy and submit these values to individual information services. Our HIBE scheme does not require any such public values for each node.

The access control mechanism proposed in this paper is also suitable to environments other than pervasive computing. In general, the mechanism targets scenarios where multiple information services, run by different organizations, need to distribute the same kind of information to the same set of clients. For example, another deployment scenario is in the context of medical information, where multiple hospitals, run by different HMOs, need to grant the same set of researchers access to the same kind of sensitive statistical information gathered in a hospital.

As we will see in Section 5, our proposed HIBE scheme can be expensive in terms of performance. This could become a problem in a pervasive computing environment, where clients might employ computationally weak devices for accessing information (e.g., a cell phone). A common architecture for such environments is to have agents perform tasks on behalf of clients [7, 11]. We could have this agent decrypt information for its client. For performance and availability reasons, it makes sense to run this agent on a more powerful processing platform and to run only a lightweight proxy on a client's personal device.

4 Prototype Implementation

The Aura ubiquitous computing environment [12] serves as a testbed for the implementation and deployment of our proposed access control mechanisms. Because the environment is mostly Java, we implemented our discussed HIBE scheme in Java. In particular, we ported a C implementation of IBE [14] to Java and added support for (multiple) hierarchies. The operations introduced in Sections 3.3 and 3.4 require expensive cryptographic computations. This is especially troublesome for the $Encrypt()$ operation since it is performed by a service. However, upon closer examination of this operation, we realize that a service can precompute most of the expensive computations. We refer to Appendix B for details. We employ a hybrid encryption scheme, that is, we symmetrically encrypt information with a session key and encrypt only this session key with $Encrypt()$.

We also implemented a few sample information services that require access control. There are several location services, each exploiting a different approach for locating people. They can run either proof-based access control or encryption-based access control. There is also a service that provides calendar information and has covert access requirements. In proof-based access control, we use SPKI/SDSI [9] certificates for expressing access rights. Alice provides the public parameters of her identity-based cryptographic scheme and her hierarchies in SPKI/SDSI "auto-certificates" [10], whose purpose is to make information about their issuer available in an authentic way. Alice also uses auto-certificates for handing out private keys. Obviously, recipients of such an auto-certificate should keep it secret. There is a command line tool for issuing certificates, setting up IBE schemes, and extracting private keys.

We use the SSL protocol for communication between entities [24], which gives us authentication of peers and confidentiality and integrity of the transmitted data. Strictly speaking, we do not require confidentiality of the (already encrypted) information returned by a service in encryption-based access control and of a challenge in proof-based access control. Server authentication and query confidentiality and integrity are required to deal with attackers listening, modifying, or injecting traffic. For encryption-based access control, we require data integrity since our IBE implementation is only semantically secure, but does not provide chosen-ciphertext security. A similar argument holds for challenges in proof-based access control. We decided against implementing the required features, together with IBE, in a protocol of our own, since, as history has shown, correctly implementing protocols is hard. SSL has been well researched, and the overhead caused by the redundant, symmetric encryption is low. We employ client authentication only for proof-based access control.

While not being part of our threat model, a deployed system needs to be able to deal with attackers learning private keys or, worse, the compromise of a policymaker's master secret. We can exploit mechanisms proposed earlier [4, 23] for this purpose, that is, adding a salt to the naming scheme, including a time-to-live value with configuration information, and storing secrets in a distributed way.

5 Evaluation

In our evaluation, we concentrate on encryption-based access control.We run our experiments on an unloaded Pentium IV/2.5 GHz with 1.5 GB of memory, Linux 2.4.20, and Java 1.4.2. An experiment consists of ten runs, we report both the mean and the standard deviation (in parentheses).

We have an Aura client contact an Aura service. The service provides encrypted people location information, which is split into three levels of granularity and encrypted using a three-level information hierarchy. There are no constraints. We look only at the case where information about a single individual is provided. In addition, we assume that the client knows which decryption key to use. It takes 1091ms (42ms) for the

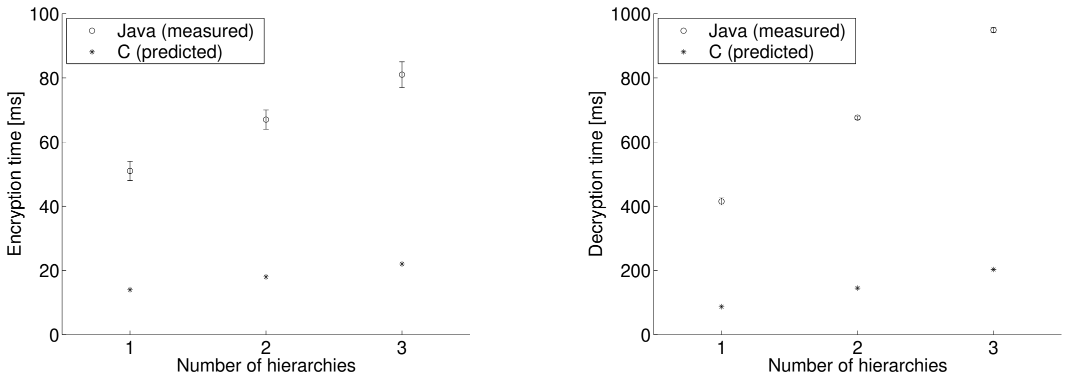

Figure 4: *Performance of encryption/decryption. We encrypt/decrypt a single message using a variable number of two-level hierarchies, whereas the first hierarchy has three levels. (Note that the two graphs are differently scaled.)*

client to retrieve and decrypt the information. Let us examine this cost in more detail. (Detailed results are in Appendix C.) For the service, there is a cost of 25ms (2ms) for an $Encrypt()$ operation that exploits only the root level of a hierarchy. (Remember that our service has to perform three $Encrypt()$ operations.) In addition, there is a cost of 14ms (1ms) per additional level used in an $Encrypt()$ operation (i.e., $3 * 14$ms in our experiment). Therefore, the overall cost of encryption is about 117ms. The overall processing time of the service is 253ms (31ms); 46% of the cost is due to encryption. The rest of the cost is caused by fingering a person's desktop computer in order to locate her and by (de)marshalling of the request and the response. For the client, there is a cost of 136ms (2ms) per level used in a $Decrypt()$ operation. Our client runs three such operations, operating at 1, 2, or 3 levels. Therefore, the overall decryption cost is about 816ms or 75% of the overall processing time.

In our second experiment, we investigate the influence of the number of hierarchies on processing time. We encrypt and decrypt a random message using a variable number of hierarchies, whereas we exploit all the levels in each hierarchy. Similar to the first experiment, the first hierarchy has three levels. All the additional hierarchies have two levels. Figure 4 presents the results. The cost for encryption and decryption increases linearly with the number of hierarchies. This observation is consistent with the characteristics of the $Encrypt()$ and $Decrypt()$ operations (see Appendix C). Taking these characteristics into account, if there are m hierarchies having n_i levels ($1 \leq i \leq m$), the cost of an $Encrypt()$ operation exploiting all the levels in each hierarchy is 25ms + $\sum_{i=1}^{m}(n_i - 1) * 14$ms. For a $Decrypt()$ operation, the cost is $\sum_{i=1}^{m} n_i * 136$ms.

The performance numbers heavily depend on the underlying implementation. Our implementation is in Java and uses Java's standard mathematical package for its cryptographic routines. While we currently do not have a C-based implementation of HIBE, there is a more optimized, publicly available C-based implementation of standard IBE [18]. Since hierarchical IBE exploits the same basic mathematical routines as standard IBE, we can predict the performance of a C-based implementation of hierarchical IBE based on this implementation. Figure 4 shows our predictions. (More detailed results are in Appendix C). In summary, the performance of a C-based, more optimized implementation would be at least 3.5 (encryption) or 4.5 (decryption) times better.

The presented results allow us to judge the relative benefit, performance-wise, of encryption-based and proof-based access control. In our implementation of proof-based access control, it takes a service about 3ms to validate the 1024 bit RSA signature of a SPKI/SDSI certificate. Assuming a single-level information hierarchy and no constraint hierarchies, it takes the service 25ms to encrypt a piece of information. However, this operation does not need to be executed for every client, the service can reuse an encrypted piece of information to answer requests from multiple clients. Therefore, it pays off for the service to use encryption-based access control if there are more than 8 requests for some information during the lifetime of the information. If there are constraints on access rights, this number will become correspondingly larger.

In the case of covert access requirements, the overall cost for proof-based access control will be larger than for encryption-based access control. The performance of the operations for identity-based encryption will be similar for both cases. However, proof-based access control requires two round trips, client authentication, and validation of the proof of access.

6 Related Work

Identity-based cryptography has been used for different types of applications, such as searchable audit logs [27] or secure email and IPsec [23]. All these applications, including our proposed one, exploit the Boneh and Franklin IBE scheme [4]. While there are other schemes (e.g., by Cocks [8], Boneh and Boyen [3], Yao et al. [28], and Waters [26]), we choose this scheme because of the existence of publicly available implementations [14, 18].

There has been previous work about access control in a hierarchy, where information items are classified into partially ordered security classes depending on their sensitivity and users are assigned to classes depending on their clearance. Each class has an encryption (decryption) key, which is used for encrypting (decrypting) information in the class. Given the encryption (decryption) key for a class, it is possible to derive the encryption (decryption) key for a class of a lower security level. None of the proposed hierarchical schemes fulfills our requirements of asymmetry and easy access rights management: Akl and Taylor [1], Harn and Yin [15], and Tzeng [25] present symmetric schemes, Sandhu [20] and Zheng et al. [30] propose symmetric schemes exploiting strings for key generation, and Ray et al. [19] discuss an asymmetric scheme that does not exploit strings for key generation. Our scheme supports only tree-based and not arbitrary hierarchies. However, tree-based hierarchies are sufficient for expressing granularity-aware access rights and hierarchical constraints on them. Similar to our scheme, Briscoe [6] uses a hierarchy for managing time-based access.

Automated trust negotiation explores issues related to covert access requirements. In particular, Yu and Winslett [29] study the scenario where (parts of) a service's access policy is confidential. (An access policy lists the required access rights.) The authors suggest two strategies, neither of them applicable to our scenario. The first strategy transmits all the client's access rights to a service, even if they are not required. The second one transmits only access rights that the service asks for by revealing (parts of) its access policy. However, this strategy fails if access rights whose corresponding access policy cannot be revealed are required. In Holt et al.'s scheme [16], a service encrypts information in a client-specific way, and the client needs to find the corresponding decryption key(s) in its set of keys. Similar to our scheme, Holt et al.'s work is based on the Boneh and Franklin IBE scheme. However, due to reasons outlined in Section 3.4, we do not have a service encrypt information for proof-based access control. Holt et al. do not investigate constraints on access rights and expiration of access rights. Bradshaw et al. [5] extend Holt et al.'s scheme to support complex access policies expressed as monotonic boolean functions. They apply a secret splitting system in order to conceal the structure of such policies. Smart [22] also examines how to support complex access policies in IBE, but he assumes that the policies are not concealed.

7 Conclusions and Future Work

When running access control to sensitive information in a pervasive computing environment, we need to be able to deal with constraints on access rights and avert information leaks. We showed how hierarchical identity-based encryption can be employed to address these challenges.

We implemented our proposed architecture in the context of the Aura pervasive computing environment. Our evaluation shows that identity-based encryption is expensive (though the overhead can be significantly lowered using a more optimized implementation), but it gives us the convenience of being able to use the identification string of some information or of a constraint as public key.

A weakness of our proposed architecture is that it relies on all the policymakers agreeing on a set of parameters, which could be difficult to achieve in practice. A topic for further investigation is whether we can weaken this assumption without significantly compromising on security. Another area of future research involves the delegation of personalized secrets used for covert access requirements in proof-based access control: Whereas a recipient of an access right can delegate this right (e.g., by issuing another digital certificate), the recipient currently cannot delegate the corresponding secret, since this delegation requires knowledge of the policymaker's master secret.

Acknowledgments

We are grateful to Nick Hopper for pointing out the application of IBE for proof-based access control.

References

[1] S. G. Akl and P. D. Taylor. Cryptographic Solution to a Problem of Access Control in a Hierarchy. *ACM Transactions on Computer Systems*, 1(3):293–248, 1983.

[2] L. Bauer, M. A. Schneider, and E.W. Felten. A General and Flexible Access-Control System for the Web. In *Proceedings of 11th Usenix Security Symposium*, pages 93–108, August 2002.

[3] D. Boneh and X. Boyen. Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles. In *Proceedings of EUROCRYPT 2004*, pages 223–238, May 2004.

[4] D. Boneh and M. Franklin. Identity-Based Encryption from the Weil Pairing. *SIAM J. of Computing*, 32(3):586–615, 2003. Extended Abstract in Proc. of Crypto 2001, pp. 213-229, 2001.

[5] R. Bradshaw, J. Holt, and K. E. Seamons. Concealing Complex Policies with Hidden Credentials. `http://eprint.iacr.org/2004/109`, May 2004.

[6] B. Briscoe. MARKS: Zero Side Effect Multicast Key Management Using Arbitrarily Revealed Key Sequences. In *Proceedings of First International Workshop on Networked Group Communication*, pages 301–320, November 1999.

[7] H. Chen, T. Finin, and A. Joshi. Semantic Web in the Context Broker Architecture. In *Proceedings of PerCom 2004*, pages 277–286, March 2004.

[8] C. Cocks. An Identity Based Encryption Scheme Based on Quadratic Residues. In *Proceedings of 8th IMA International Conference on Cryptography and Coding*, pages 360–363, December 2001.

[9] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. SPKI Certificate Theory. RFC 2693, September 1999.

[10] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen. SPKI Examples. Internet Draft, March 1998. `http://theworld.com/~cme/examples.txt`.

[11] F. Gandon and N. Sadeh. A Semantic eWallet to Reconcile Privacy and Context Awareness. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, October 2003.

[12] D. Garlan, D. Siewiorek, A. Smailagic, and P. Steenkiste. Project Aura: Towards Distraction-Free Pervasive Computing. *IEEE Pervasive Computing*, 1(2):22–31, April-June 2002.

[13] C. Gentry and A. Silverberg. Hierarchical ID-Based Cryptograph. In *Proceedings of Asiacrypt 2002*, pages 548–566, December 2002.

[14] Stanford Applied Crypto Group. IBE Secure E-mail. `http://crypto.stanford.edu/ibe`.

[15] L. Harn and H. Y. Lin. A Cryptographic Key Generation Scheme for Multi-level Data Security. *Computer & Security*, 9(6):539–546, 1990.

[16] J. Holt, R. W. Bradshaw, K. E. Seamons, and H. Orman. Hidden Credentials. In *Proceedings of 2nd ACM Workshop on Privacy in the Electronic Society*, October 2003.

[17] J. Howell and D. Kotz. End-to-end authorization. In *Proceedings of 4th Symposium on Operating System Design & Implementation (OSDI 2000)*, pages 151–164, October 2000.

[18] Shamus Software Ltd. Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL). `http://indigo.ie/~mscott/`.

[19] I. Ray, I. Ray, and N. Narasimhamurthi. A Cryptographic Solution to Implement Access Control in a Hierarchy and More. In *Proceedings of 7th ACM Symposium on Access Control Models and Technologies (SACMAT'02)*, pages 65–73, June 2002.

[20] R. S. Sandhu. Cryptographic Implementation of a Tree Hierarchy for Access Control. *Information Processing Letters*, 27(2):95–98, 1988.

[21] A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In *Proceedings of Crypto '84*, pages 47–53, 1984.

[22] N. P. Smart. Access Control Using Paring Based Cryptography. In *Proceedings of The Cryptographer's Track at RSA Conference (CT-RSA 2003)*, pages 111–121, April 2003.

[23] D. K. Smetters and G. Durfee. Domain-Based Administration of Identity-Based Cryptosystems for Secure Email and IPSEC. In *Proceedings of 12th Usenix Security Symposium*, August 2003.

[24] Claymore Systems. PureTLS. `http://www.rtfm.com/puretls/`.

[25] W.-G. Tzeng. A Time-Bound Cryptographic Key Assignment Scheme for Access Control in a Hierarchy. *IEEE Transactions on Knowledge and Data Engineering*, 14(1):182–188, 2002.

[26] B. R. Waters. Efficient Identity-Based Encryption Without Random Oracles. `http://eprint.iacr.org/2004/180`, July 2004.

[27] B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters. Building an Encrypted and Searchable Audit Log. In *Proceedings of 11th Annual Network and Distributed System Security Symposium (NDSS 2004)*, February 2004.

[28] D. Yao, Y. Dodis, N. Fazio, and A. Lysyanskaya. ID-Based Encryption for Complex Hierarchies with Applications to Forward Security and Broadcast Encryption. In *Proceedings of 11th ACM Conference on Computer and Communications Security (CCS 2004)*, October 2004.

[29] T. Yu and M. Winslett. A Unified Scheme for Resource Protection in Automated Trust Negotiation. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 110–122, May 2003.

[30] Y. Zheng, T. Hardjono, and J. Seberry. New Solutions to the Problem of Access Control in a Hierarchy. Technical Report Preprint No.93-2, Department of Computer Science, University of Wollongong, Australia, 1993.

A Operations

In this section, we describe the operations introduced in Section 3 in detail. Our operations are based on the operations proposed by Gentry and Silverberg [13], we extend these operations to support multiple hierarchies. (We merge the $Lower\_Level\_Setup()$ and $Extract()$ operations.) We also assume that there is a global set of public parameters. Gentry and Silverberg present two encryption schemes, a semantically secure one and a scheme secure against adaptive chosen ciphertext attacks in the random oracle model. For presentation purposes, we base our discussion on the semantically secure scheme. It is straightforward to generalize our scheme to a scheme secure against chosen ciphertext attacks.

There is a set of public parameters $params = (\mathbb{G}_1, \mathbb{G}_2, q, \hat{e}, P_0, H_1, H_2)$, where $\mathbb{G}_1$ and $\mathbb{G}_2$ are groups of some prime order q, $\hat{e}$ is an admissible paring: $\mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$, P_0 is an arbitrary generator of $\mathbb{G}_1$, and

H_1 and H_2 are cryptographic hash functions $\{0,1\}^* \rightarrow \mathbb{G}_1$ and $\mathbb{G}_2 \rightarrow \{0,1\}^n$ for some n, respectively. One of the properties of an admissible pairing is bilinearity: $\hat{e}(aQ, bR) = \hat{e}(Q, R)^{ab}$ for all $Q, R \in \mathbb{G}_1$ and all $a, b \in \mathbb{Z}$.

- $Root\_Setup(params) \rightarrow Q_0$:
  Choose a random master secret, $s_0 \in \mathbb{Z}/q\mathbb{Z}$, and return $Q_0 = s_0 P_0$.

- $Extract(\langle ID_{i,1}, ..., ID_{i,t_i}\rangle, S_{i,t_i-1}, params) \rightarrow S_{i,t_i}, \langle Q_{i,1}, ..., Q_{i,t_i-1}\rangle$ with $t_i \geq 1$:

  1. If $t_i > 1$, pick random, secret $s_{i,t_i-1} \in \mathbb{Z}/q\mathbb{Z}$. Otherwise $s_{i,0} = s_0$.
  2. Compute $P_{i,t_i} = H_1(ID_{i,1}, ..., ID_{i,t_i}) \in \mathbb{G}_1$.
  3. Compute secret point $S_{i,t_i} = S_{i,t_i-1} + s_{i,t_i-1} P_{i,t_i} = \sum_{j=1}^{t_i} s_{i,j-1} P_{i,j}$. ($S_{i,0}$ is the identity element of $\mathbb{G}_1$.)
  4. Compute (non-secret) $Q_{i,j} = s_{i,j} P_0$ for $1 \leq j \leq t_i - 1$.

- $Encrypt(\langle ID_{1,1}, ..., ID_{1,t_1}\rangle, ..., \langle ID_{h,1}, ..., ID_{h,t_h}\rangle, M, Q_0, params) \rightarrow C$:

  1. Compute $P_{i,j} = H_1(ID_{i,1}, ..., ID_{i,j}) \in \mathbb{G}_1$ for $1 \leq j \leq t_i$ and $1 \leq i \leq h$.
  2. Choose a random $r \in \mathbb{Z}/q\mathbb{Z}$.
  3. Set the ciphertext to be:

  $$C = [rP_0, \langle rP_{1,2}, ..., rP_{1,t_1}\rangle, ...\langle rP_{h,2}, ..., rP_{h,t_h}\rangle, M \oplus H_2(g^r)]$$

  where

  $$g = \prod_{i=1}^{h} \hat{e}(Q_0, P_{i,1}) \in \mathbb{G}_2.$$

- $Decrypt(\langle S_{1,t_1}, ..., S_{h,t_h}\rangle, \langle Q_{1,1}, ..., Q_{1,t_1-1}\rangle, ..., \langle Q_{h,1}, ..., Q_{h,t_h-1}\rangle, C, params) \rightarrow M$:
  Let $C = [U_0, \langle U_{1,2}, ..., U_{1,t_1}\rangle, ..., \langle U_{h,2}, ..., U_{h,t_h}\rangle, V]$ be the ciphertext encrypted using the sequences of IDs $\langle ID_{1,1}, ..., ID_{1,t_1}\rangle, ..., \langle ID_{h,1}, ..., ID_{h,t_h}\rangle$. To decrypt C, compute

  $$V \oplus H_2\left(\frac{\prod_{i=1}^{h} \hat{e}(U_0, S_{i,t_i})}{\prod_{i=1}^{h} \prod_{j=2}^{t_i} \hat{e}(Q_{i,j-1}, U_{i,j})}\right) = M.$$

The security of the scheme is based on the hardness of the Bilinear Diffie-Hellman problem: Given a randomly chosen $P \in \mathbb{G}_1$, as well as aP, bP, and cP (for unknown randomly chosen $a, b, c \in \mathbb{Z}/q\mathbb{Z}$), compute $\hat{e}(P, P)^{abc}$. For proving security of our scheme in the random oracle model, we have to modify Gentry and Silverberg's proof (Appendix A.3 of [13]). This modification is straightforward, it exploits the same ideas that we have exploited for extending Gentry and Silverberg's scheme. We give only an outline of the modifications here: There needs to be a separate H_1^{list} for each hierarchy, and the challenge operation needs to take the additional hierarchies into account. For the latter modification, we rely on the symmetry of $\hat{e}$.

The alternative solution for proof-based access control suggested in Section 3.4 relies on the hardness of the Decision Bilinear Diffie-Hellman problem: Given a randomly chosen $P \in \mathbb{G}_1$, as well as aP, bP, cP, and r (for some $a, b, c, r \in \mathbb{Z}/q\mathbb{Z}$), return true if $r = \hat{e}(P, P)^{abc}$.

Operation	Java		C	
	μ	(σ)	μ	(σ)
$rP_{i,j}$	14	(1)	4	(0)
Exponentiation	11	(1)	2	(0)
Pairing	136	(2)	29	(0)

Table 1: *Processing times. Mean and standard deviation of elapsed time for expensive cryptographic operations in our Java-based implementation and in MIRACL's C-based implementation [18] [ms].*

B Optimizations

In this section, we discuss a few optimizations that we apply in our implementation of the operations outlined in Appendix A. We concentrate on encryption-based access control, the argument is similar for proof-based access control.

$Encrypt()$ is a time-critical operation since it is run by a service. To speed up this operation, a service can precompute the $P_{i,j}$ in step 1 and the pairings in step 3. In addition, the service can precompute sliding windows for multiplications on $P_{i,j}$ and use them for the computation of $rP_{i,j}$.

C Evaluation

Our Java-based implementation, which is based on a C implementation [14], exploits Tate pairings over super-singular elliptic curves. We use a 160 bit prime for q and a 512 bit prime for p, where $\mathbb{G}_1$ is an order-q subgroup of $E(\mathbb{F}_p)$ and $\mathbb{G}_2$ is an order-q subgroup of $\mathbb{F}_{p^2}$. For the configuration given in Section 5, the performance of the expensive cryptographic operations is given in Table 1. For the C-based implementation [18], we use the same set of parameters and configuration as for the Java-based version (gcc 3.2.3 instead of Java 1.4.2).