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

Urs Hengartner[†] and Peter Steenkiste[†‡]
[†]Computer Science Department
[‡]Department of Electrical and Computer Engineering
Carnegie Mellon University
{uhengart, prs}@cs.cmu.edu

Abstract

Access control to confidential information in pervasive computing environments is challenging for multiple reasons: First, a client requesting access might not know which access rights are necessary in order to be granted access to the requested information. Second, access control must support flexible access rights that include context-sensitive constraints. Third, pervasive computing environments consist of a multitude of information services, which makes simple management of access rights essential. We discuss the shortcomings of existing access-control schemes that rely on either clients presenting a proof of access to a service or services encrypting information before handing the information over to a client. We propose 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. Furthermore, we introduce an encryption-based access-control architecture that exploits hierarchical identity-based encryption in order to deal with multiple, hierarchical constraints on access rights. We present an example implementation of our proposed architectures and discuss the performance of this implementation.

1. Introduction

Access control to confidential information has been well investigated in the context of traditional distributed systems, such as a distributed file system. However, access control faces additional challenges in the context of pervasive computing environments. First, there might be *covert access requirements*. In particular, a client that wants to access information might not know which of the client's 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, the client does not know which access rights are required. Second, access rights need to be more flexible. For instance, it should be possible to issue access rights that are constrained based on a person's context, such as her location or the current time.

There are *proof-based* and *encryption-based* access-control schemes. In a proof-based scheme, a client needs to assemble some access rights in a *proof of access*, which demonstrates to a service that the client is authorized to access the requested information. This proof of access prevents the service from having to locate the required access rights, which can be an expensive task. Proof-based access control is attractive for scenarios where flexible, client-specific access rights are required. Since access rights are flexible, it is easy to include support for constraints in them. However, it is difficult to add support for covert access requirements. Existing designs [2, 12] assume that a service can inform a client of the nature of the required proof of access. When we apply this principle to our example mentioned above, we could end up with an information leak. In particular, assume that a service informs a client of the identity of the people for whose location information the client needs to present access rights. Based on this knowledge, the client can infer who the owner of the calendar entry is meeting with. A naïve solution is to have the client submit all obtained access rights to the service. This solution has privacy and bandwidth issues: the service can learn a lot about the client, and the client might have to transmit a lot of data. Instead, the service must let the client know about the nature of the required proof of access such that only clients authorized to access the information listed in the proof description can understand this description.

In an encryption-based access-control scheme, a service provides confidential information to any client, but only in an encrypted form. Clients authorized to access the information have the corresponding 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 architectures [1, 9, 14, 15]. In particular, a service encrypts information as usual, but it does not tell a client which decryption key to use. Assuming that a client has a set of decryption keys, the client now needs to search this set for a matching key. It is less straightforward to add support for constraints on access rights to the proposed architectures, especially when considering that key management should remain simple.

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

2. Access Control in Pervasive Computing

In this section, we discuss the challenges for access control and for 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, there are a lot of services that provide potentially confidential information to clients. Clients need to have *access rights* in order to be granted access to confidential 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., people location services exploiting cellphones, RFID badges, or Wi-Fi devices). 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 cellphone service.

It should be possible to constrain access rights. In this paper, we limit ourselves to constraints whose current value is publicly known (e.g., current time). Our architecture also supports constraints that involve confidential information (e.g., current location of the client or of the queried individual), but additional access control is required in order to avoid leaks of this confidential information, which is outside of the scope of this paper.

Access rights should be granularity aware. Some information (e.g., location information) is available at different levels of granularity (e.g., "CMU", "CMU Wean Hall", "CMU Wean Hall 8220"). Having an access right to fine-grained information should imply having an access right to coarse-grained information. Granularity-aware access rights also 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, an access right can be represented in different forms. For proof-based access control, it is a digital certificate issued by the policymaker, whereas for encryption-based access control, it is a decryption key. Regardless of the form, dealing with access rights should be simple for all involved entities (clients, services, and policymakers).

We now discuss how proof-based and encryption-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. Proof-Based Access Control

Proof-based access control is attractive since it offloads the assembly of a proof of access to a client. If the client does not know the nature of the required proof of access, a service will give it a description of this proof. The description lists the information for which the client needs to present access rights. However, when this description leaks confidential knowledge, the service must obscure it. Let us summarize the requirements for this case:

Indistinguishability. The service must obscure the description such that a client learns nothing about the information listed in the description, unless the client has some secret knowledge. The client has this secret knowledge only if the client has an access right to the information.

Constraints. 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 to information expires, the client should no longer be able to interpret an obscured proof description asking for this information. To support this feature, each possible value of a constraint must require separate secret knowledge. For instance, a client's secret knowledge allowing interpretation of an obscured proof description on January 1 must not allow interpretation on January 2. This requirement leads to an increase in the amount of secret knowledge to be managed by the client. The problem becomes worse when there are multiple constraints on an access right. We observe that many constraints are of a hierarchical nature. Therefore, we want an architecture that supports hierarchical constraints. For example, if a client has secret knowledge for January, the client can derive secret knowledge for January 1, January 2,... This requirement simplifies management of the secret knowledge.
Granularity awareness. Some information in pervasive computing (e.g., location information) is available at different granularities. If a client's secret knowledge allowed the client to interpret an obscured proof description asking for fine-grained information, the same knowledge should also allow the client to interpret an obscured proof description asking for coarse-grained information.
Personalization. We want obscured proof descriptions to be personalized for a client. In this way, if a client's secret knowledge required for understanding an obscured proof description asking for particular information leaked, other clients being able to understand obscured proof descriptions asking for the same information would not be affected.
Asymmetry. Service-independent access rights grant access to information independent of the service offering this information. A service generating an obscured proof description listing specific information must not be able to interpret an obscured proof description listing the same information generated by another service (unless the former service has the required access right).

2.3. Encryption-Based Access Control

If there are lots of requests for the same 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. 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:
Indistinguishability. The encrypted information must not reveal any knowledge about the used encryption key or the required decryption key.
Constraints. Each value of a constraint must require a separate key for decrypting encrypted information that should be accessible only under the given constraint/value combination. To make key management simple, we want a scheme that supports hierarchical constraints.
Granularity awareness. To simplify key management, the decryption key for coarse-grained information should be derivable from the key for fine-grained information.
Asymmetry. Service-independent access rights imply 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. We can avoid this problem by using an asymmetric cryptosystem.

We do not require personalization for encryption-based access control since it is client independent by design.

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 another service, where this service might offer the same type of information as the corrupted service. Corrupted services can collude. Attackers can also sniff, modify, or inject network traffic.

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. The architecture also 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 such environments. It is asymmetric and provides indistinguishability. Since public keys are strings, access right management and personalization are simple. In addition, a hierarchical version of identity-based encryption lends itself to the implementation of hierarchical constraints and granularity awareness. Therefore, with the help of hierarchical identity-based encryption (HIBE), we can overcome the shortcomings of existing access-control architectures for pervasive computing environments. 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). 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 [3] present one of the first practical IBE schemes. Based on this work, Gentry and Silverberg [7] 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, 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 [5]), 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 single hierarchical PKG infrastructure, we let each policymaker have a PKG. A policymaker uses the PKG for managing access rights to information. In addition, a policymaker can set up a hierarchical PKG infrastructure and control both the root PKG and any sub PKGs. In this way, a policymaker will be able to establish granularity-aware access rights with hierarchical constraints (see Section 3.4). In the rest of this paper, we use the term "policymaker" instead of PKG.

Our architecture builds on Gentry and Silverberg's HIBE scheme. This 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 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 3.4, the public key associated with some information corresponds directly to the name of the information. We discuss the advantages of a HIBE scheme in more detail in Section 3.6.

3.2. Basic Operations

Our architectures for proof-based and encryption-based access control each employ four basic, randomized operations. We discuss these operations in this section and their application in proof-based and encryption-based access control in the next two sections. Our operations are based on the operations introduced by Gentry and Silverberg, but we extend them to support multiple hierarchies. We give a detailed discussion, showing the exact cryptographic steps for each operation, in the extended version of this paper [10]. For readability reasons, we omit some of the parameters of the operations here.

In order to achieve indistinguishability, we assume that all the policymakers agree on a set of public parameters, $params$. 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 a 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 the policymaker's 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 a ciphertext, C.

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

3.3. Proof-Based Access Control

If Alice grants Bob an access right to information, she will also give him a personalized secret. When Bob receives an obscured proof description asking 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 name of the information for generating a challenge for it. In our architecture, a challenge corresponds to a ciphertext/plaintext pair and a secret corresponds to a tuple of private keys enabling the decryption of ciphertexts. To support granularity-aware, constrainable challenges and secrets, Alice defines a

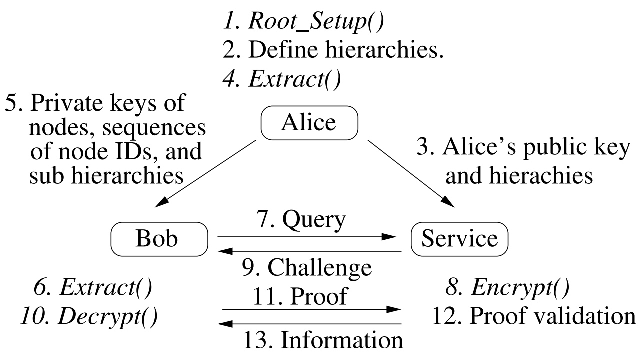

Figure 1. Architecture for proof-based access control. The service sends a challenge to Bob. Upon resolving this challenge, Bob sends a proof of access to the service.

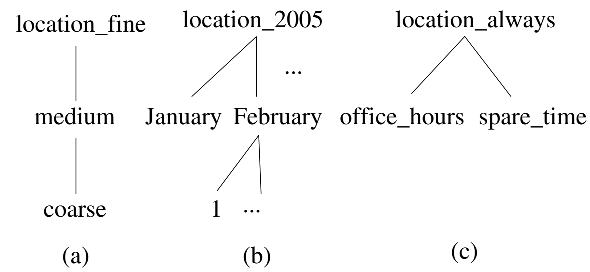

Figure 2. Hierarchies. Alice establishes hierarchies for her location information (a) and for each constraint (b, c).

set of hierarchies. We give an overview of our extended proof-based access-control architecture in Figure 1. It consists of three entities: a policymaker managing access rights to personal information ("Alice"), a service offering information with covert access requirements, and a client trying to access the information provided by the service ("Bob"). We now discuss the individual steps shown in Figure 1.

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 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 to location information (*constraint hierarchies*). Figure 2 (b) shows a hierarchy that restricts the lifetime of access rights, 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.) The root node of each hierarchy includes the name of the information to ensure that, for example, a constraint granting unlimited access to location information cannot be used for getting unlimited access to medical information.

Alice then informs the service of her public key and her hierarchies (3). Since none of this knowledge is confidential, an authenticated communication channel suffices. Instead of defining her own hierarchies, Alice can exploit predefined hierarchies that the service is already aware of. For example, we expect that there will be a widely accepted and shared hierarchy for location information.

To allow Alice to issue personalized secrets to clients, we have her 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 (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). In her information hierarchy, she chooses the node corresponding to the information to which she wants Bob to have access (e.g., "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., $Extract(\langle \text{location\_fine\_Bob} \rangle, \text{null}, params) \rightarrow S_{1,1}$ and $Extract(\langle \text{location\_fine\_Bob, medium} \rangle, S_{1,1}, params) \rightarrow S_{1,2}$) (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. The tuple of private keys returned by these calls serve as the secret.

Alice then gives the secret to Bob, together with the corresponding sequences of node IDs and the sub-hierarchies rooted in the chosen nodes (5). Transfer of the secret requires a secret communication 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 ⟨location_fine_Bob, medium⟩ and the sub-hierarchy "coarse", Bob can extract the private key for ⟨location_fine_Bob, medium, coarse⟩. It is possible for Bob to delay this step until he receives a ciphertext.

Access Control. Bob issues a query to the service and fails to submit a proof of access (7). Since the requested informa-

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

tion (e.g., calendar information) has covert access requirements, the service needs to computes a challenge (8). In particular, the service calls $Encrypt()$ to encrypt a random plaintext, M. The public keys required for this operation come from the information and constraint hierarchies of the policymaker responsible for the information for which the client needs to present an access right. The service locates the corresponding node in Alice's information hierarchy. The service then gathers the IDs of all the nodes along the path from the root node to this node. For example, if an access right to fine-grained information is required, the ID sequence is ⟨location_fine_Bob⟩. 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., $Encrypt($⟨location_fine_Bob⟩, ⟨location_2005_Bob, February, 2⟩, ⟨location_always_Bob, office_hours⟩, $M, Q_0,$ $params$)). Note that the public keys used for encryption correspond directly to the node IDs.

The plaintext, M, and the obtained ciphertext, C, serve as the 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 communication channel, since a challenge is personalized to a client and useless to other clients.

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) need a secret communication channel.

Discussion. The benefits of our architecture are secrets that support constraints and that are personalized and granularity aware. Because a challenge for information is based on the name 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. As opposed to a previous approach for dealing with expiration [3], which makes the current date part of an ID, our approach does not require handing out separate private keys for each possible date.

A client resolves a challenge before submitting the required proof of access to a service. However, for some scenarios, this second step can be omitted since resolving the challenge(s) already gives the client all the information that the client is asking for. For example, if the client asks for

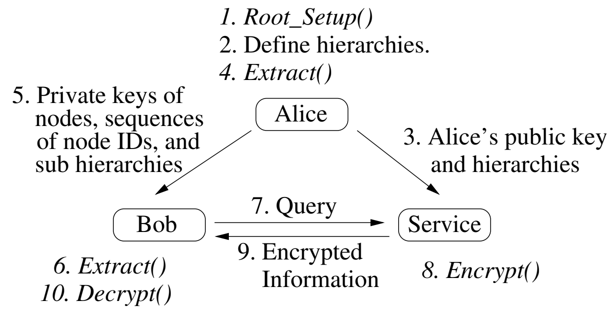

Figure 3. Architecture for encryption-based access control. Alice sets up her IBE scheme and hierarchies, informs the service, and grants access to Bob. Bob issues a query.

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 about all the people 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 a random plaintext (as suggested by Holt et al. [11]). We refrain from adapting this model because, as we will see in Section 5, the decryption operation is expensive. 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 ciphertexts on them for every request to confidential information.

Security Analysis. The security of the scheme is based on the hardness of the Bilinear Diffie-Hellman problem. (Please refer to the extended version [10] for details.) Given this assumption, Gentry and Silverberg [7] 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 traffic sniffers cannot decrypt ciphertexts without having the required decryption key or modify ciphertexts. We need to ensure that a client cannot learn from the ciphertext which public key was used to produce this ciphertext (indistinguishability). Holt et al. [11] 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 secure against collusion of clients or services, since keys are personalized.

3.4. Encryption-Based Access Control

Figure 3 gives an overview of our encryption-based access-control architecture; the architecture is similar to the architecture for proof-based access control with challenges given in Figure 1. We now review the changes. We assume that the service provides location information.

Setup. There is no need for Alice to personalize her information and constraint hierarchies, since encryption-based access control is not client-specific.

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 ⟨location_fine, medium, coarse⟩, ⟨location_fine, medium⟩, and ⟨location_fine⟩, 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., $Encrypt($⟨location_fine, medium, coarse⟩, ⟨location_2005, February, 2⟩, ⟨location_always, office_hours⟩, "CMU", $Q_0, params$)). 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. Our solution fulfills the requirements of being asymmetric and hierarchical and supporting multiple, hierarchical constraints. Using the name of information or of a constraint directly as its public key drastically simplifies key management.

Security Analysis. The 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⟩, ⟨location_2005⟩, ⟨location_always, office_hours⟩) and that Carol has the tuple for (⟨location_fine⟩, ⟨location_2005, January⟩, ⟨location_always⟩). If Bob and Carol colluded, they could determine the tuple for (⟨location_fine⟩, ⟨location_2005⟩, ⟨location_always⟩). Yao et al. [16] propose a collusion-resistant HIBE scheme, which we could also adopt. 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 $\mathcal{O}(mn)$.

3.5. Limiting the Search Space

Both for proof-based and encryption-based access control, if there are covert access requirements, Bob will not know which of his (potentially derived) tuples of private keys to use for the $Decrypt()$ operation, and he will have 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 covers only a single individual, that is, Bob needs to find only one tuple of private keys. As described in Section 3.4, when a policymaker gives a tuple of private keys to Bob granting him access to 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. For each constraint hierarchy, Bob knows the current value of the constraint and can throw out all the tuples that do not include the corresponding private key. In practice, 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. For example, it is well known that calendar information is composed of fine-grained location information, but not of medical information. Therefore, the service can safely inform Bob 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. Overall, 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 (i.e., a service returns multiple challenges or encrypts information multiple times), Bob will have to locate multiple tuples of private keys. Therefore, Bob's search cost is proportional to the number of policymakers multiplied by the number of individuals covered by the information returned by the service. While this sounds expensive, Bradshaw et al. [4] present an optimization that requires the client to perform the most expensive cryptographic operation in this search only once for each policymaker and not for each combination of a policymaker and a covered individual.

3.6. Discussion

IBE simplifies key management. For example, in an email system, IBE allows Bob to encrypt email to Alice simply by using her email address as public key. Bob does not need to contact Alice beforehand to acquire a separate public key. We seem to lose this advantage: Alice needs to inform a service of her hierarchies and her public key. However, as mentioned in Section 3.4, we do not expect each policymaker to define her own hierarchies. Instead, there can be a shared set of hierarchies, which a service is

	Personal hierarchy		Shared hierarchy	
	Public values	Private keys	Public values	Private keys
Conventional cryptosystem	$2n$	1	n	1
HIBE scheme	n	1	0	1

Table 1. Key management demand. For a hierarchy of n nodes, we show the number of public values (including public keys) and private keys that a policymaker needs to define and give to a service and to a client, respectively.

aware of. In addition, we observe that a setup step is also necessary for IBE in an email system: First, IBE schemes 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 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.

Instead of using a HIBE scheme, it is possible to make a conventional asymmetric cryptosystem, such as ElGamal or RSA, hierarchy aware [14]. The drawback of this approach is increased demand in key management and transfer. We summarize this demand in Table 1. (Both conventional and HIBE schemes typically also require storage and transfer of a constant amount of additional information, which is not shown in the table.) If a policymaker defines a personal set of hierarchies, the policymaker will have to transfer at least the ID of each node to a service in order to inform the service of the node's meaning, regardless of the employed hierarchical cryptosystem. For a HIBE scheme, only this ID is required. For a conventional cryptosystem, a separate public key needs to be generated and transferred for each node. If a policymaker uses a shared information or constraint hierarchy and employs a conventional cryptosystem, the policymaker will still have to generate a set of public keys for all the nodes in the shared hierarchy and submit these values to individual services. This is not necessary for a HIBE scheme.

As we will see in Section 5, our proposed HIBE scheme can be expensive in terms of performance. This could become a problem when a client employs a computationally weak device for accessing information (e.g., a cellphone). A common architecture for pervasive computing is to have agents perform tasks on behalf of clients. 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 pervasive computing environment [6] serves as a testbed for the implementation and deployment of our proposed access-control architectures. Because the environment is mostly Java, we implemented our HIBE scheme in Java. We ported a C implementation of IBE [8] to Java and added support for hierarchies. We employ a hybrid encryption scheme, that is, we symmetrically encrypt information with a session key and encrypt only this key with $Encrypt()$.

We also implemented a few sample information services that require access control. There is a service that provides calendar information. This service runs proof-based access control and has covert access requirements. There are also several location services, each exploiting a different approach for locating people. They run either proof-based or encryption-based access control. These services do not have covert access requirements, so the proof-based versions do not employ HIBE. The encryption-based versions always use HIBE. While it is possible to switch to a different asymmetric cryptosystem if, for example, no constraints are used or information is not granularity aware, key management would become difficult. In proof-based access control, we express access rights in SPKI/SDSI certificates [5]. An individual provides the public parameters of her IBE scheme, her hierarchies, and her tuples of private keys in self-signed certificates. There is a command line tool for issuing certificates, setting up IBE schemes, and extracting private keys.

We use SSL for communication between entities, which gives us authentication of peers and confidentiality and integrity of the transmitted data. We employ client authentication only for proof-based access control.

5. Evaluation

In our evaluation, we concentrate on encryption-based access control. We run our experiments on a 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 a client contact a service that 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 client to retrieve and decrypt the information. Let us examine this cost in more detail. For the service, there is a cost of

25ms (2ms) for an $Encrypt()$ operation that exploits only the root level of a hierarchy. 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, 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 encryption and decryption 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. As shown in Figure 4, the cost increases linearly with the number of hierarchies.

The performance numbers heavily depend on the underlying implementation. Our implementation 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 [13]. 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 also shows our predictions. 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 proof-based and encryption-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 information during the lifetime of the information. If there are constraints on access rights, this number will become correspondingly larger.

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

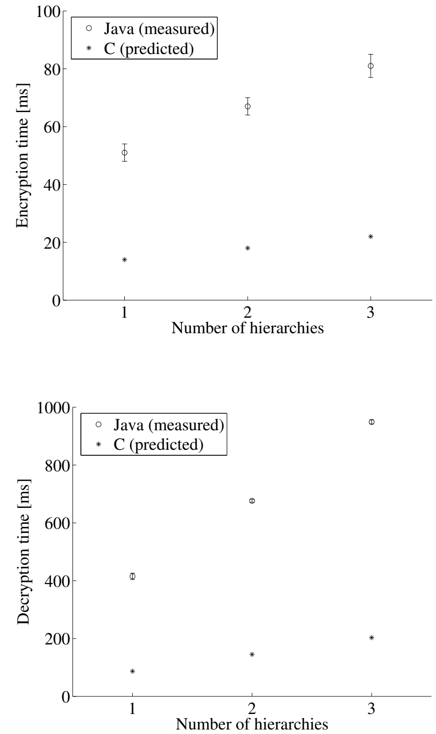

Figure 4. Performance of encryption/decryption. We encrypt/decrypt a message using a variable number of two-level hierarchies, whereas the first hierarchy has three levels. (The two graphs are differently scaled.)

6. Related Work

Automated trust negotiation explores issues related to covert access requirements. For example, Yu and Winslett [17] 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 access rights of a client 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 architecture [11], 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 architecture, Holt et al.'s work is based on the

Boneh and Franklin IBE scheme. However, due to reasons outlined in Section 3.3, 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.

There has been previous work about access control in a hierarchy [1, 9, 14, 15], 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 a key, which is used for encrypting (decrypting) information in the class. Given the key for a class, it is possible to derive the key for a class of a lower security level. None of the proposed hierarchical architectures fulfills our requirements of asymmetry and easy access rights management. Our architecture supports only tree-based hierarchies. However, tree-based hierarchies are sufficient for expressing granularity-aware access rights and hierarchical constraints on them.

7. Conclusions and Future Work

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

We implemented our proposed architectures in the context of a pervasive computing environment. Our evaluation shows that identity-based encryption is expensive. However, the overhead can be significantly lowered using a more optimized implementation. Furthermore, our design gives us the convenience of being able to use the name of the information or of a constraint as public key.

A weakness of our architecture is that all the policy-makers need to share the same parameters for their HIBE schemes, which could be difficult to achieve. A topic for further investigation is whether we can weaken this assumption without significantly compromising on security.

Acknowledgments

We are grateful to Nick Hopper for pointing out the application of IBE to proof-based access control. We thank the anonymous reviewers for their comments. 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.

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

[4] R. Bradshaw, J. Holt, and K. E. Seamons. Concealing Complex Policies with Hidden Credentials. In *Proceedings of 11th ACM conference on Computer and Communications Security (CCS 2004)*, pages 146–157, October 2004.

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

[6] 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.

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

[8] S. A. C. Group. IBE Secure E-mail. `http://crypto.stanford.edu/ibe`.

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

[10] U. Hengartner and P. Steenkiste. Exploiting Hierarchical Identity-Based Encryption for Access Control to Pervasive Computing Information. Technical Report CMU-CS-04-172, Computer Science Department, Carnegie Mellon University, October 2004.

[11] 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.

[12] 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.

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

[14] 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.

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

[16] 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)*, pages 354–363, October 2004.

[17] 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.