Patient Controlled Encryption: Ensuring Privacy of Electronic Medical Records

Josh Benaloh, Melissa Chase, Eric Horvitz, and Kristin Lauter
Microsoft Research
Redmond, WA, USA
{benaloh,melissac,horvitz,klauter}@microsoft.com

ABSTRACT

We explore the challenge of preserving patients' privacy in electronic health record systems. We argue that security in such systems should be enforced via encryption as well as access control. Furthermore, we argue for approaches that enable patients to generate and store encryption keys, so that the patients' privacy is protected should the host data center be compromised. The standard argument against such an approach is that encryption would interfere with the functionality of the system. However, we show that we can build an efficient system that allows patients both to share partial access rights with others, and to perform searches over their records. We formalize the requirements of a Patient Controlled Encryption scheme, and give several instantiations, based on existing cryptographic primitives and protocols, each achieving a different set of properties.

Categories and Subject Descriptors

E.3 [**Data Encryption**]: Public key cryptosystems

General Terms

Security, Algorithms, Design

1. INTRODUCTION

On February 13, 2009, U.S. President Barack Obama signed into law the American Recovery and Reinvestment Act of 2009, which contained provisions authorizing the federal government to spend 19 billion dollars to digitize U.S. health records. President Obama stated, "We will make the immediate investments necessary to ensure that within five years all of America's medical records are computerized," and that digital medical records could prevent medical errors, save lives, and create hundreds of thousands of jobs.[1] Moving to electronic health records is important to the modernization and revamping of our healthcare system, but solving the great challenges of ensuring the safety, security, and privacy of patients is equally critical, of which the federal government is acutely aware.[2]

Computerized medical records are open to potential abuses and threats. Some have pointed out that large amounts of sensitive healthcare information held in data centers is vulnerable to loss, leakage, or theft.[3] "In the last few years, personal health information on hundreds of thousands of people has been compromised because of security lapses at hospitals, insurance companies and government agencies."[4] Medical data is also susceptible to misuse by those seeking to profit from it. For example, some companies make a business of buying and selling doctors' prescribing habits to pharmaceutical companies.[5] There are also intrinsic legal results. World Privacy Forum warns that sensitive electronic data, especially when stored by a third party, is vulnerable to blind subpoena or change in user agreements. Businesses such as hospitals and law firms, which are required by law to respect users' privacy, "may be at risk of a lawsuit simply for using a cloud computing service, even if information is not leaked."[6] Furthermore, disputes among lawmakers seeking to regulate the computerization of medical records and lobbyists from pharmaceutical companies and insurance companies may delay or derail effective privacy safeguards from being put in place to protect patients' rights and safety.

Given the plan to widely deploy electronic medical records systems, challenges of interoperability and standardization come to the fore. In the absence of regulations which assure patient privacy, standards for interoperability may impose designs that limit that privacy. On the other hand, the focus of attention on potential standards and interoperability provides a rich context of developing systems that can protect privacy as data flows across multiple systems.

Today, many organizations provide electronic medical

[1]"Privacy Issue Complicates Push to Link Medical Data", Robert Pear, New York Times, January 17, 2009.

CCSW'09, November 13, 2009, Chicago, Illinois, USA.

[2]Quoting from the ARR Act of 2009, p. 117. Specific objectives include "(iii) The incorporation of privacy and security protections for the electronic exchange of an individual's individually identifiable health information. (iv) Ensuring security methods to ensure appropriate authorization and electronic authentication of health information and specifying technologies or methodologies for rendering health information unusable, unreadable, or indecipherable."

[3]Data Hemorrhages in the Health-Care Sector, M. Eric Johnson, Forthcoming in Financial Cryptography and Data Security, February 22-25, 2009.

[4]"Privacy Issue", Robert Pear, New York Times, January 17, 2009.

[5]ibid.

[6]"Does Cloud Computing Mean More Risks to Privacy?", Saul Hansell, New York Times, February 23, 2009.

records (EMR) solutions. The primary method of guaranteeing privacy in today's systems is access control. In a system which relies solely on access control, the servers that store data run an access control program, which verifies that any party accessing a patient's healthcare record has appropriate permissions. These access control systems keep a log of all accesses, and communications are securely encrypted. Overall, this has been a fairly effective approach. However, patients must trust the third party storing their data with their private health record. If a data center is compromised, patients' private data may be revealed.

Given the weaknesses of access control, we propose that an electronic health record system should encrypt records in addition to restricting access. But who should hold the decryption keys in such systems? If the server itself holds the key, then it will be similarly vulnerable to the theft and compromise issues that can challenge access-centric systems, as the key might be stolen along with the encrypted data. Similarly, such a storage design will be vulnerable to privacy and trust issues. Thus, we propose instead that each patient should generate her own decryption key, and use that key to encrypt her records.

Encryption schemes with strong security properties will guarantee that the patient's privacy is protected (assuming that the patient stores the key safely; see section 3.1 for discussion). However, adherence to a simple encryption scheme can interfere with the desired functionality of health record systems. In particular, we would like to employ encryption, yet support such desirable functions as allowing users to share partial access rights with others and to perform various searches over their records. In what follows, we consider encryption schemes that enable patients to delegate partial decryption rights, and that allow patients (and their delegates) to search over their health data.

We shall propose a design that we refer to as Patient Controlled Encryption (PCE) as a solution to secure and private storage of patients' medical records. PCE allows the patient to selectively share records among doctors and healthcare providers. The design of the system is based on a hierarchical encryption system. The patient's record is partitioned into a hierarchical structure, each portion of which is encrypted with a corresponding key. The patient is required to store a root secret key, from which a tree of subkeys is derived. The patient can selectively distribute subkeys for decryption of various portions of the record. The patient can also generate and distribute trapdoors for selectively searching portions of the record. Our design prevents unauthorized access to patients' medical data by data storage providers, healthcare providers, pharmaceutical companies, insurance companies, or others who have not been given the appropriate decryption keys.

Related work.

We note that hierarchical access control via encryption is not a new idea. Akl and Taylor [2] and later Sadhu [16] propose constructions based on one-way functions which are very similar to what we describe in Section 4.2. More recently, Hengartner and Steenkiste proposed a scheme for encryption-based access control based on hierarchical IBE [13]. Atallah et al present a symmetric key scheme which allows not only for tree-based hierarchies, but also for arbitrary acyclic graphs [3]. Finally, recent work on attribute-based encryption [15, 12] enables encryption-based access control with more expressive policies; however it is not known how to incorporate searchability in these schemes, so we do not discuss them here.

A related problem is that of cryptographic storage file systems (CSFS) introduced by Blaze [5], in which files are encrypted before being stored on an untrusted file server. Fu [10] presents a CSFS system which allows for sharing of access rights. Our approach of using a hierarchical encryption scheme differs from the traditional CSFS approach in that in our case the encryption scheme itself enforces the access structure. This reduces the number of keys we need to store/retrieve, and guarantees consistent access rights when different parties write to the same portion of a record. On the other hand, it limits the types of access structures one can have (in that the access structure must be hierarchical).

Other works consider improving ease of use of hierarchical encryption schemes. Miklau and Suciu describe a policy language for cryptographic access control [14]. Di Vimercati et gal consider a model where the data is encrypted twice: once by the data owner to reflect initial permissions, and again by the server to accommodate changes in the access permissions [9]. These reflect important questions for a practical system, but are outside the scope of this paper.

Organization.

The rest of the paper flows as follows: In Section 2, we describe the high-level outline of a PCE scheme. Then, we present in Section 3 an example of how such a system might be implemented in practice and discuss several practical issues. In Section 4, we describe two solutions that fit into the overall PCE framework . Section 5 presents an alternative construction with somewhat different properties.

2. PATIENT CONTROLLED ENCRYPTION

In an electronic health record system, patients, healthcare providers, and medical devices can upload health records and retrieve and view them at a later time. Furthermore, patients may delegate access rights and allow family, friends, and designated healthcare providers to view or to edit parts of their record. Patients and their delegates may wish to efficiently perform searches in an efficient manner over part or all of the record.

Let us now consider the challenges that arise in a naive attempt to add security to such a system. We argued above that we want to allow the patient to generate her own decryption key. But in this case, how can she allow others to access her record? Clearly she does not want to give out her entire key, as that would allow the recipient to read and modify all parts of her record.

In what follows, we describe the PCE proposal.

At a high level, a system using PCE allows the patient to use her decryption key to generate subkeys which will allow her delegates to search and access only certain parts of her record.

The goals of PCE are:

Guarantee Strong Security In particular, PCE will (1) guarantee the patient's privacy: the patient should have confidence that the administrators of the health data server will not learn anything about the patient's record[7], (2) guarantee security in the case of server compromise: even if the server is compromised or stolen, the patient should be certain that his data has not been

[7]with the exception of trivial information like the existence and size of the record.

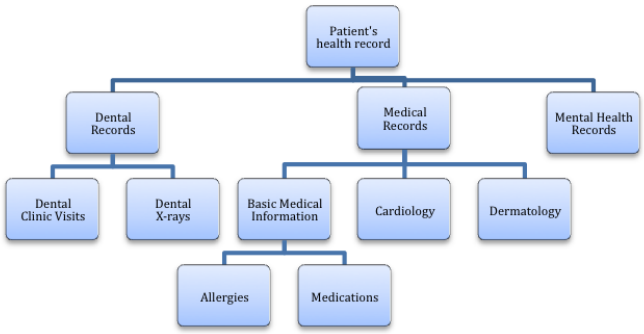

Figure 1: A hierarchical health record

leaked, and (3) guarantee correctness of the health record: the patient should be able to verify that no one has tampered with his record.

Maintain Functionality We want to guarantee the above security without compromising the functionality of the server. This means the system should guarantee (1) efficient access to patient health records, (2) easy sharing of parts of the record, and (3) efficient searching over records.

2.1 Patient Health Structures

We assume that a patient's record is organized into a hierarchical data structure. There are multiple ways to decompose medical data into a hierarchical representation based on the use of different ontologies. For example, a record may be decomposed at the top level into a set of mutually exclusive high-level categories such as dental records, medical records, mental health data, and a category representing the set of all lab results accrued by the patient. The medical records category might be further broken down into subcategories for basic medical information (containing such subcategories as prescribed medications, and known allergies), cardiologic data, dermatologic data, etc. These categories in turn might be subdivided according to which clinic provided the care, then further according to doctor or date. We note that while we have given an example of a record partitioned according to a single hierarchy, our constructions can be extended easily to the case where there are several different hierarchies which can be used to organize the patient's records.

The key design criterion for PCE is that patients should be able to delegate access to any subset of these categories to their doctor, dentist, pharmacist, spouse, etc. We note that while a patient might wish to share her entire record with her doctor, she might not want to allow pharmacists, billing staff, or lab technicians to see any more information than is necessary. Thus, we propose a system in which a patient can grant access to specific potions of the data. As described above, the patient will generate and store her own secret key, which we will call the *root key*. Then she can use this root key to generate subkeys for various categories or subcategories. Data in each subcategory will be encrypted under the corresponding subkey. To delegate rights to read a particular category, she will generate the corresponding subkey, and send it to her doctor, dentist, etc.

Let us consider the sample hierarchy displayed in Figure 1. Here, the patient might decide to grant her dentist access to both the "Dental Records" category and the "Basic Medical Info" category. This would allow the dentist to read all data concerning dental clinic visits, dental x-rays, allergies, and medications. However, the dentist would have no way of decrypting any of the information in the patient's mental health records, or her cardiologic data, for example.

Note that the server that stores the health information will not have access to the secret key, or any of the subkeys given to the doctor, and thus will be unable to decrypt any of the data.

One advantage of our hierarchical structure is that it is easily extendable, in that the patient (and potentially other parties to whom the patient grants the appropriate permissions) can add additional subcategories within any existing category. Thus, within the "Medications" category, Alice's dentist might add a new category for "Anaesthetics", or "visit 4/2/09". We will present a solution so that once the patient gives her doctor access to her medications, if her dentist adds a new subcategory, all documents in that subcategory will automatically be accessible to the doctor.

Advantages The patient can easily grant access to a category, without knowing all the types of files that might eventually be included in it.

Similarly, doctors can add subcategories with arbitrary names, without assistance from the patient. This will be particularly useful if we can't predict the names of all possible subcategories, i.e., if a doctor needs to add a category for a new type of test, or if categories are labeled by visit dates.

Disadvantages The hierarchy is fixed in that there is only one way in which we can partition the record. If we want to give out access rights based on something else (e.g. based on document type or sensitivity of data) we will have to look at all the low-level categories involved, and give a separate decryption key for each. (e.g. in our example, giving a lab access to all X-rays would require giving separate keys for "Dental X-rays", "Cardiologic X-rays", and "Mental Health X-rays".)

(This might be partially avoided if we have several fixed hierarchies, and we encrypt each file under each hierarchy.)

2.2 Preliminaries

Before we describe our proposal in more detail, we make the following assumptions about the format of the patient's record. We assume that the patient's record is stored as a collection of entries, where each entry contains the name of a file, the name of the smallest category containing that file, a "locator tag" which the patient can use to refer to the file, and an encrypted version of the file itself. (In our proposed construction, the file name and category name will be encrypted, and the tag will not reveal any information about the contents of the file. See Section 2.5 for details.)

We also summarize some terminology we will use later: For a hierarchical representation of data, we say that a given category cat_1 is an *ancestor* of another category cat_2 if cat_2 is contained within cat_1. For example, in Figure 1, "Medical Records" is an ancestor of "Allergies". We call a category a *leaf category* if it does not contain any other category. In Figure 1, "Dental Clinic Visits", "Dental X-Rays", "Allergies", ... are all leaf categories. We sometimes refer to the name of a category as its label.

Finally, we give some notation: We will use $cat_{(i_1,\ldots,i_\ell)}$ to specify a category in our hierarchy, where (i_1) specifies

that top level ancestor of the category, (i_1, i_2) specifies the next ancestor down the chain, and so on. For example, for the hierarchy in Figure 1, the "Allergies" category would be specified by $cat_{(MedicalRecords, BasicMedicalInformation, Allergies)}$. Similarly, we will use $sk_{(i_1,\ldots,i_\ell)}$ to represent the decryption key for category $cat_{(i_1,\ldots,i_\ell)}$.

2.3 Basic PCE

Above, we described the high level functionality of our PCE system. Here we will give a more formal description of the required properties.

We say a PCE system consists of four algorithms:

- A key generation algorithm PCEKeyGen which generates a root secret key and (in a public key system) public key for the patient.
- A key derivation algorithm PCEKeyDer which takes a secret key for a category, and the name of one of its subcategories, and generates the secret key for that subcategory.
- An encryption algorithm Enc which takes a public key and/or a secret key for a category, the name of that category, and a file, and encrypts that file for that category.
- A decryption algorithm Dec which takes the name of a category and the secret key for that category, and a ciphertext encrypted for that category, and produces the decrypted file.

The two properties that we require are **Correctness**, which asserts that any correctly encrypted document will be successfully decrypted given the appropriate decryption key, and **Security**, which says that an encryption of a document in a given category will reveal no information about that document as long as an adversary has not been given the decryption key for an ancestor category, even if she obtains arbitrarily many other decryption keys for other categories, and has access to encryption and decryption oracles for all categories (with the restriction that she cannot query the decryption oracle on the challenge document).

Thus, the health data server will store only encrypted files. When a patient wishes to grant access to a category to her doctor, she will run PCEKeyDer to generate the appropriate subkey and send it to the doctor. Then the doctor can retrieve all the encrypted files in that category from the server and decrypt them using this subkey. (For more details on how this works in practice, see Section 3.)

2.4 Searchability of Encrypted Records

As mentioned above, our scheme includes an efficient searching mechanism. This mechanism should satisfy the following properties: *Searchability*, which means the health server correctly returns the records which match the query, and *Privacy*, which means the patient can perform the search without revealing any information to the server (in this way, security is still guaranteed even if server has been compromised). Thus, we require that the server learn nothing about what query is being made or about the documents or keywords in the record. The server should only learn which encrypted documents must be returned.

We will now show how to combine PCE with two existing schemes for searchable encryption. Both of these schemes search for exact matches. This means our database consists of encrypted strings, we can choose a string, compute a corresponding encrypted query, and send it to the server. The server can then determine if the query string matches any string in the database. It does this without seeing the strings—instead we have a test algorithm, which takes an encrypted query and a set of encrypted strings, and returns true if there is a match, without revealing any other information. Thus, we can guarantee that, even given this encrypted query, the server will not learn the string being queried for, or any other information besides which ciphertexts were matches. In a practical implementation, one could combine this approach with some software for clustering related terms, so that a search would also return documents containing words that are similar to the query string.

Our design combines the idea of searchable encryption with our patient controlled encryption. We will consider that within a health record, each leaf category contains a set of files (test records, records of a particular checkup, medication listings, individual readouts from a medical device, etc). When a file is written we will assume that the doctor also includes a list of keywords that a patient might want to search for. This could be anything from a few tags to all meaningful words in the file. These keywords will be encrypted with a searchable encryption scheme, using the appropriate subkey for that category, to form an encrypted index. Then a doctor with decryption permissions for a particular category will also be able to search within that category for files which contain various keywords.

For simplicity our system as we present it here has separate sets of keys to handle encrypting keywords and generating search trapdoors. (This way we can describe the searching algorithms independent of the basic PCE implementation.) These keys will be generated, delegated, and distributed whenever the PCE keys are. In many cases, there can be just one set of keys for both purposes; we chose this separation only to simplify the description of the constructions.

We extend our notation to include four additional algorithms:

- A key generation algorithm SrchKeyGen that generates a root secret key and (in a public key system) a public key for generating trapdoors for searching the patient's record.
- A key derivation algorithm SrchKeyDer that takes a searching secret key for a category and the name of one of its subcategories, and generates the secret key for that subcategory.
- An index generation algorithm IndexGen that takes a list of keywords and a public and/or secret key and generates an encrypted index.
- A trapdoor generation algorithm Trap which takes the decryption key for a category and a keyword, and generates a corresponding trapdoor.
- A search algorithm Search which takes an encrypted index and a trapdoor, and returns any matching items. In particular, if the trapdoor was generated for a given category and keyword, the search should return locator tags for all items within that category which include that keyword.

2.5 Hiding the Category Labels

In some cases, the category labels themselves reveal sensitive information. For example, if a doctor uploads a large amount of data under the category "Cancer", that might reveal that a patient is being tested or treated for cancer, even if the data itself is encrypted.

Thus, we want to be able to extend the above schemes so that the labels themselves are encrypted. This brings up 3 problems.

1. We need some way of communicating the category names to the doctor, in order to allow her to generate the appropriate subkeys and decrypt the record. As described above, the doctor's key allows her to decrypt documents in many categories. However, in order to do this decryption she must first derive a key for each appropriate leaf category. Doing this requires that the doctor know the name of these subcategories. (This is particularly an issue when new subcategories can be added and category names can be chosen at encryption time.)
2. The doctor and the health server (and the encryptor who uploaded the file) must share some way of referring to that file or set of files without revealing anything about their contents. In order to maintain efficiency, we do not want to require that the doctor download the patient's entire record in order to view one file; ideally the doctor would only have to download the file she is interested in. If the labels are public, parties can upload data labeled by the name of the category in which it belongs. Then the doctor can ask the server which of the categories she has access to actually contain data, and then request all encrypted files in the desired categories. Now, we must find some way to accomplish the same thing without revealing the category labels to the server.
3. Finally, as described in section 2.1, we want to ensure if a doctor has been given a key for a given category, then he will be able to read all documents in the category. In particular, if a new file or a new subcategory is added, the doctor should automatically find it the next time he views the patient's record. We do not want to require that the patient send the doctor the file names or category names for all new documents. And we must do this without revealing this information to the health data server.

In what follows, we will discuss how to extend our schemes to this case. First, we will require that each file be labeled by a random "locator tag". These tags will be generated by the encryptor (e.g. the patient or his doctor), and the health data server will store the tags together with the corresponding encrypted files. However, these tags will not reveal any information about the data in the associated files.

We shall now describe how we can build an encrypted "directory" that lists all of the data files. A doctor will be able to decrypt some portions of this directory (using the key she was given by the patient), which will allow her to identify the locator tags for the files she wants to download. Finally, she will send these tags to the server, which will send back the corresponding encrypted files.[8] The key is that this directory must be built in a distributed fashion, with each encryptor adding entries encrypting the new tags that she has generated.

This process works as follows: when a party uploads a file, he will also create a directory entry, which includes the locator tag, category name, and file name. This entry will be encrypted in such a way that only those parties allowed to access the given category will be able to decrypt the message, but the decrypting parties need not know the exact name of the category in order to decrypt.

When a doctor wishes to view some files in a category, she will download the entire directory and attempt to decrypt each entry. For every file she is allowed to read, she will thus discover the filename, category name, and locator tag. She will decide which files she wants to download, send the corresponding locator tags to the server, and receive the associated encrypted files. Finally, she will use the category names to generate the appropriate decryption keys and will thus decrypt the desired files.

Thus, the key aspects we have to cover when describing such a scheme are (1) how the directory entries are generated, and (2) how they are decrypted. As above, for simplicity we will have a separate set of keys to handle generating and decrypting the directory entries. Again, in many cases these can be combined with the basic PCE keys. We add an additional four algorithms:

- A key generation algorithm DirKeyGen that generates a root secret key (and possibly a root public key) for the directory information for the patient's record.
- A key derivation algorithm DirKeyDer which takes a directory secret key for a category and the name of one of its subcategories, and generates a directory secret key for that subcategory.
- An algorithm for generating encrypted directory entries, FormDirEntry that takes a secret key for a category and possibly a public key, a category name, a file name, and a locator tag, and outputs an encrypted directory entry.
- An algorithm for decrypting directory entries, DecDirEntry, that takes a secret key and an encrypted directory entry for some category. If the key is supposed to give access to that category, the algorithm will output the category name, file name, and locator tag. If not, it will output "failure".

2.6 Public Key and Symmetric Key Schemes

In what follows, we will present both symmetric key and public key schemes. Each has advantages and disadvantages, so which is used will depend on the particular scenario in which the health record system is used.

Public Key PCE.

In a public key scheme, anyone can encrypt data without any secret information. Thus, in a public key PCE system, we can allow anyone to encrypt documents for the patient's file, and upload rights do not imply the ability to read other files in the same category. In practical terms, this means that doctors, devices, etc, will be able to upload to a patient's record without receiving any secret key (and without getting the associated decryption permissions). See Section 3.1 for a discussion of how one might limit upload rights in this situation. However public key schemes tend to be slower, and when we also require searchability or hidden labels, they seem to have inherent privacy weaknesses.

Advantages

Public Key: As described above, a public key scheme allows a doctor or medical device to upload data given only some basic public information, without obtaining any decryption capabilities.

[8]If the patient's record is small, we can simply require the doctor to download the entire encrypted record, avoiding this second round of communication. However, we assume that patients' records will be fairly large (as they may contain large files such as test results, x-rays, genetic data, etc.), so the above approach will be much more efficient.

Disadvantages

Efficiency: In general public key operations are much slower than symmetric key primitives.

Efficiency of searchability: This will be the slowest option. First, the search time will be linear: the server must test each tag of each document in the appropriate category.

Privacy loss in searchability: There is also a significant loss in privacy in the public key option: since anyone can encrypt to any search term, any party with access to the patient's public key and history of query trapdoors can use these trapdoors to determine what keyword was being searched for: he simply encrypts each possible keyword (using the patient's public key), and tests to see which one returns a match. (This is true in the basic PEKS of Boneh et al [6], and seems to be inherent in the public key setting.)

Privacy loss when hiding labels: Our scheme for hiding the labels will incur some privacy loss against a malicious health server administrator who can edit the patient's record: A malicious party can encrypt fake locator tags under possible category names and see if the doctor requests them. If the doctor requests one of the fake tags, then it has determined which category the doctor is requesting. However, we note that this is an active attack, in that the adversary must interact with the patient. An adversary who only has read access to the record and log files of the server will not be able to learn anything.

Symmetric Key PCE.

In a symmetric key scheme, one must know the decryption key in order to encrypt data. Thus, in a symmetric key PCE system, anyone who can encrypt for a given category can also decrypt any files in that category. In practical terms, this means that the patient will have to issue an appropriate decryption key to a doctor or a device for a given category before they will be able to upload to this category. Furthermore, the doctor or device will also be able to read any data in that category. On the other hand, these schemes tend to be much more efficient and have stronger privacy guarantees.

Advantages

Efficiency: These constructions can rely primarily on symmetric key primitives for which we have very efficient instantiations.

Efficiency of Searching: This scheme allows for fairly efficient searches: the search time is proportional to the number of documents returned and the number of categories searched.

Secure Searchability: We can guarantee that the only thing that an adversary learns is which documents are returned in each query. (This requires a slight variation on the scheme we present, see remark 4.1 for details.) Besides that, the only parties who can learn anything about the encrypted data or the searches the parties perform will be those who have already have access to the relevant files.

Secure Label Hiding: Our label hiding scheme is secure against active attackers – the only parties who can determine what category a particular encrypted file belongs to will be those who have permissions to access it.

Disadvantages

Symmetric key: As described above, anyone who can upload to a given category can also decrypt all documents in that category.

Particularly for the case of an untrusted device or piece of software, this might be undesirable. Alternatively there might be a situation where a doctor's assistant is expected to upload the doctor's notes, but should not be able to see anything else in the patient's record. If this is undesirable, one possible option is to subdivide each leaf category to have a separate subcategory for each party who will be uploading. Then the doctor's assistant or the medical device will be able to decrypt any information that they upload, but not information provided by others.

Symmetric Key Searchability: As this is a symmetric key scheme, the underlying SSE requires that the party who forms the encrypted index must know the associated symmetric key, and thus must be able to perform arbitrary searches.

3. WORK FLOW IN PRACTICE

We now describe a concrete instantiation of PCE. We will consider interactions between a patient Bob, his doctor Alice, and the server which stores Bob's health records.

Creating User Accounts When patient Bob wants to sign up for an account, he registers as he would in a traditional EMR system. He connects to the server (via an SSL connection) and generates an account username and password, to be used in the future to identify his account and to authenticate to the server.

Next, an application will be downloaded to his machine. This application will run locally and will perform the following tasks: first it will generate a public key pair (pk_B, sk_B), which will be used to secure communications with other parties in the system. Then it will generate a root secret key sk_R under which all Bob's health records will be encrypted. Finally, it will upload (via SSL) the public key pk_B, which will be stored as part of Bob's health record. The sk_B and sk_R will be stored locally on Bob's machine.

At this point Bob may be offered the opportunity to upload some basic information (e.g., birthdate, gender, height, etc.). If he so chooses, Bob may enter this information into the application. The application, which as above runs locally, derives the appropriate subkeys and from sk_R, and choose random locator tags for each file. It uses the subkeys to encrypt Bob's information, and generate an encrypted index. It also uses the subkeys to generate the encrypted dictionary entry for each (file name, category name, locator tag) tuple. Finally, it will send the tags and the encrypted information and encrypted index to the server (via SSL). The server will store each encrypted file and encrypted index labeled by the corresponding tag.

Creating Provider Accounts When a provider organization wants to sign up for an account, it will register as in a traditional EMR system: it will connect to the server via an SSL connection and establish an account username and credentials, which will be used in the future to identify the account and to authenticate to the server. (A large organization will also have the opportunity to dynamically upload public keys corresponding to different parties within its staff.)

Before an Appointment When Bob makes an appointment to see his provider, they exchange user names. (And if the provider is a large organization, Bob may also learn the identifier for the specific doctor he will be seeing.)

Next, the doctor requests access to any necessary information as follows: The doctor logs in to the health server and enters Bob's user name, and her application downloads Bob's public key. Then the doctor encrypts the request for information under Bob's public key and upload the result to the server. Similarly, the doctor can request permissions to upload files to certain portions of the record.

Bob logs in to the server and his application downloads and decrypts this request and then presents it to him along with the doctor's username (and identifier). It also downloads the doctor's public key. If Bob agrees to allow access to the requested information, the application uses his root key sk_R to locally generate subkeys for the appropriate categories. It encrypts these under the doctor's public key, and uploads them to the server.

The doctor logs in to the server, and her application downloads the encrypted subkeys and decrypts them. It also downloads the patient's encrypted directory, and using the subkeys it decrypts the (file name, category name, locator tag) tuples for all the files that the doctor has access to. It presents these file names to the doctor (organized using the category names), who then selects the categories or individual files she wants to download. The application sends the corresponding locator tags to the server, and obtains the associated encrypted files. Finally, it decrypts the received files using the previously decrypted subkeys, and displays them to the doctor. When the doctor is done accessing this record, the application deletes all subkeys. (They can be retrieved again from the server when they are needed.)

After an appointment The doctor can request permissions to read and write additional categories as above.

When the doctor wants to upload data to Bob's health record, she logs in to the health server. Her application downloads the necessary encrypted subkeys from the server. If no such keys have been uploaded, she requests them from Bob as above. The application decrypts the necessary subkeys, encrypts the data under those subkeys, and generates a corresponding encrypted index. For each file it also chooses a random locator tag and generates an encrypted directory entry. Finally, the application uploads the encrypted data, the encrypted index, the locator tag, and the directory entry for each file. When the doctor is done accessing this record, the application deletes all subkeys. (They can be retrieved again from the server when they are needed.)

The doctor can also choose to search over the data that she has access to, without downloading all of it. In this case she logs in to the server and her application downloads the encrypted subkeys. Then it decrypts and identifies the appropriate subkey(s). It also downloads the encrypted directory, and decrypts entries for the files the doctor has access to. It presents the resulting file names to the doctor who selects those over which the search will be conducted. Then the application uses his subkeys to generate the appropriate trapdoors, which it sends to the server along with the locator tags for all files being searched. The server applies the search trapdoors to the indexes associated with the given tags, and returns any encrypted files for which a match is detected. Once again, all subkeys are deleted afterwards.

Finally, the doctor can delegate decryption or searching rights to any of the data that she has access to (for instance if she wants to allow the office admin to see visit information for billing). In this case she determines the categories to which she wishes to grant access. Then she logs in to the server, and her application downloads her encrypted subkeys and decrypts them. It then generates keys for the categories that are to be delegated, which the doctor can then encrypt under the admin's public key and transfer via the health server as above.

3.1 Key Management and related issues

Key Revocation A patient always has the option of changing (essentially revoking) keys by decrypting portions of her record locally and re-encrypting with new subkeys. This might be desirable if a patient suffers a key compromise or wants to discontinue access to her health record for a particular provider, or family member, or other proxy.

Emergency response Patients might be given the option to wear or carry an enhanced medic-alert bracelet or similar device which might function like a barrier that one must break for engaging a fire alarm: it would contain a tamper-evident seal which could be broken to obtain access to the patient's medical records.

Patient Key Management Escrowing of keys should be recommended to patients when setting up their accounts. Escrowing can be done formally through a professional service or informally by sharing keys with family members or via a threshold scheme. In addition or as an alternative, patients could keep a hardware device that stores a back-up of their root secret key: sk_R. In some cases, third party escrow agents could also serve emergency response requirements.

Doctor/Device Key Management In the PCE system, doctors could potentially have to store, manage, and protect local copies of secret keys for each their patients. A hierarchically organized system has the advantage that in many situations, this would only be a single secret key from each patient. However, doctors could avoid even this burden by simply downloading encrypted keys from the health records server (encrypted by the patient under the doctor's public key) whenever needing access to a patient's record, and then deleting the locally decrypted copy of the secret key once the record is decrypted.

Usability While the PCE system aims to give the patient full control over who can access her record, it puts the burden on the patient to properly decide which providers should have access to which parts of her record. This burden is the same in any existing health records system which uses access control as a means to patient privacy. To help the patient easily navigate such choices, we suggest that the system might be preset with several different options defining default hierarchies and sets of keys to issue to doctors, family members, devices, etc. For example, one default option could be to provide access to the Basic Medical Information category to all of the patient's doctors. A

patient can choose to accept these defaults or to make her own choices. Even if she chooses a custom setting, she might choose to set some standard policies for what to release to different types of parties. Or, she could assume full control, and decide whether or not to grant access each time she is contacted by a doctor, family member, device, etc. Thus, the PCE system can accommodate both the basic user and a privacy-concerned user who wants full control.

4. BUILDING PCE

4.1 Solution 1: Public Key PCE

Here we show a construction that satisfies the properties in section 2, and that allows for public key encryption.

In our basic scheme we need the following four algorithms:

- A key generation algorithm $\mathsf{PCEKeyGen}(1^k) \rightarrow (PK_{root}, SK_{root})$ which takes as input the security parameter k and generates a root decryption key SK_{root} and a public key PK_{root} for the patient's record.
- The key derivation algorithm $\mathsf{PCEKeyDer}(sk_{(i_i,\ldots,i_{\ell-1})}, (i_1,\ldots,i_\ell)) \rightarrow sk_{(i_1,\ldots,i_\ell)}$ takes as input the name of a category (specified as a hierarchical list $(i_1,\ldots,i_\ell)$), and the decryption key $sk_{(i_1,\ldots,i_{\ell-1})}$ for the parent category $cat_{(i_1,\ldots,i_{\ell-1})}$ (or SK_{root} for $\ell = 1$). It outputs a decryption key $sk_{(i_1,\ldots,i_\ell)}$ for category $cat_{(i_1,\ldots,i_\ell)}$.
- The encryption algorithm $\mathsf{Enc}(PK_{root}, (i_1,\ldots,i_\ell), m) \rightarrow c$ takes as input a public key and a category specified as a list $(i_1,\ldots,i_\ell)$ and a message m. It outputs an encryption of m for category $cat_{(i_1,\ldots,i_\ell)}$.
- The decryption algorithm $\mathsf{Dec}(sk_{(i_1,\ldots,i_\ell)}, c) \rightarrow m$ takes as input a category name $(i_1,\ldots,i_\ell)$, a corresponding decryption key $sk_{(i_1,\ldots,i_\ell)}$ and a ciphertext c. It outputs decrypted message m if if the ciphertext was formed correctly for category $cat_{(i_1,\ldots,i_\ell)}$.

4.1.1 Basic Approach

A basic solution in this case is to use a CCA-secure Hierarchical Identity Based Encryption (HIBE) scheme. Shamir [18] proposed the concept of IBE as an encryption scheme in which any string (e.g. a name or email address) could be used as a public key. In the traditional IBE setting, there would also be a trusted authority who would issue the corresponding decryption key to the appropriate parties. HIBE, introduced by Gentry and Silverberg [11], allows for hierarchical identities : an encryptor encrypts a message using a list of strings $id_1,\ldots,id_L$, and a party who had obtained the decryption key for id_1 could then delegate a key for id_1, id_2 for any string id_2, and so on.

The key innovation with the hierarchical structuring in PCE is that the patient plays the role of the trusted party. We replace the "identities" with the categories in hierarchical health record. Thus, in order to delegate access rights for a given category, the patient will generate the appropriate HIBE decryption key using that category as the identity, and give the resulting key to the doctor or friend or family member. From this key, the recipient will be able to generate keys for all subcategories using the HIBE derivation algorithm.

Thus, we assume we are given a CCA secure HIBE scheme[9] consisting of algorithms $\mathsf{HIBESetup}(1^k)$, which generates parameters *params* and a master secret key mk for the authority, $\mathsf{HIBEKeyGen}(sk_{(id_1,\ldots,id_{\ell-1})}, (id_1,\ldots,id_\ell))$, which uses decryption key $sk_{(id_1,\ldots,id_{\ell-1})}$ to generate decryption key $sk_{(id_1,\ldots,id_\ell)}$, $\mathsf{HIBEEnc}(m, params, (id_1,\ldots,id_\ell))$, which encrypts a message m to identity $(id_1,\ldots,id_\ell)$, and $\mathsf{HIBEDec}(c, sk_{(id_1,\ldots,id_\ell)}, (id_1,\ldots,id_\ell))$, which uses decryption key $sk_{(id_1,\ldots,id_\ell)}$ to decrypt a ciphertext intended for identity $(id_1,\ldots,id_\ell)$.

We build a PCE scheme as follows:

$\mathsf{PCEKeyGen}(1^k)$**:** Run $\mathsf{HIBESetup}(1^k)$ to obtain $params, mk$. Output $PK_{root} = params$, $SK_{root} = mk$.

$\mathsf{PCEKeyDer}(sk_{(i_1,\ldots,i_{\ell-1})}, (i_1,\ldots,i_\ell))$**:** Run $\mathsf{HIBEKeyGen}(sk_{(i_1,\ldots,i_{\ell-1})}, (i_1,\ldots,i_\ell))$ and output the resulting $sk_{(i_1,\ldots,i_\ell)}$.

$\mathsf{Enc}(PK_{root}, (i_1,\ldots,i_\ell), m)$**:** Run $\mathsf{HIBEEnc}(m, PK_{root}, (i_1,\ldots,i_\ell))$, and output the resulting ciphertext c.

$\mathsf{Dec}(sk_{(i_1,\ldots,i_\ell)}, (i_1,\ldots,i_\ell), c)$**:** Run $\mathsf{HIBEDec}(c, sk_{(i_1,\ldots,i_\ell)}, (i_1,\ldots,i_\ell))$, and output the resulting message m.

Security The security properties of PCE follow directly from the security properties of the underlying HIBE.

4.1.2 Public Key Searchable Encryption

We can extend the public key solution suggested in Section 4.1, so as to combine it with a searchable encryption scheme that allows for public key encryption. For this, we propose using a scheme developed by Boneh et al. referred to as PEKS [6]. The high-level idea is that when the document is uploaded, each keyword is also encrypted, using the PEKS scheme, and the encryption is stored along with the encrypted document as a tag. Then the owner of the corresponding decryption key can generate a trapdoor for a particular keyword, which will allow the server to test whether a particular tag is a match (without learning the search keyword).

We need to combine the PEKS scheme with the hierarchical encryption scheme used for PCE. This means that the PEKS keyword encryption will use the appropriate encryption key for the given category, and the corresponding decryption key will be required to generate the trapdoor. The result is that anyone who can decrypt a given category can also perform any search over that category. This is essentially the hierarchical identity based PEKS scheme as described in [1].

The resulting scheme is as follows:

$\mathsf{SrchKeyGen}(1^k)$**:** Run $\mathsf{HIBEPEKSSetup}(1^k)$ to obtain $Srchparams, Srchmk$. Output $SrchPK_{root} = Srchparams$, and $SrchSK_{root} = Srchmk$.

$\mathsf{SrchKeyDer}(Srchsk_{(i_1,\ldots,i_{\ell-1})}, (i_1,\ldots,i_\ell))$**:** Run $\mathsf{HIBEPEKSKeyGen}(Srchsk_{(i_1,\ldots,i_{\ell-1})}, (i_1,\ldots,i_\ell))$ and output the resulting $Srchsk_{(i_1,\ldots,i_\ell)}$.

$\mathsf{IndexGen}(PK_{root}, (i_1,\ldots,i_\ell), word_1,\ldots,word_N)$**:** where $PK_{root} = Srchparams$. For each keyword w_i run $\mathsf{HIBEPEKS}(w_i, Srchparams, (i_1,\ldots,i_\ell))$ to obtain tag t_i. Output $(t_1,\ldots t_N)$.

$\mathsf{Trap}(Srchsk_{(i_1,\ldots,i_\ell)}, (i_1,\ldots,i_\ell), word)$**:** Run $\mathsf{HIBEPEKSTrap}(Srchsk_{(i_1,\ldots,i_\ell)}, (i_1,\ldots,i_\ell), word)$, and output the resulting trapdoor T.

$\mathsf{Search}(Index, T)$**:** For each tag t of each document in the index, run $\mathsf{HIBEPEKSTest}(t, T)$. Return the documents which are paired with a tag t for which this procedure outputs 1.

Security The weaker security property described in 2.6 follows directly from the properties of the HIBEPEKS scheme.

[9]We can obtain an efficient CCA secure HIBE by applying the techniques of Canetti, Halevi, and Katz [7] to the Gentry-Silverberg HIBE [11].

4.1.3 Hiding Labels in a Public Key scheme

If we want to hide the category labels, we must first ensure that the PCE encryption or searchable encryption does not leak any information on the category being used. Luckily, we can simply replace the HIBE with an anonymous HIBE [1], and we get the necessary guarantee.

We will also use the anonymous HIBE to form the encrypted directory as follows:

A party who wants to upload a file chooses a random string as a locator tag. It then uses the HIBE to encrypt the tag, filename, and category under identity i_1 and (i_1, i_2), ..., and $(i_1, \ldots, i_\ell)$. It uploads all of these to the server.

A party who wants to find all files in a given category first generates the key for that category. It then downloads the directory and uses the HIBE decryption to attempt to decrypt each entry in the directory. For every element which decrypts successfully to an element of the right form, it outputs the filename and locator tag.

Thus, we have the following algorithms:

$\mathsf{DirKeyGen}(1^k)$**:** Run $\mathsf{HIBESetup}(1^k)$ to obtain $Dirparams$, $Dirmk$. Output $DirPK_{root} = Dirparams$, and $DirSK_{root} = Dirmk$.

$\mathsf{DirKeyDer}(Dirsk_{(i_1,\ldots,i_{\ell-1})}, (i_1, \ldots, i_\ell))$**:** Run $\mathsf{HIBEPEKS\text{-}KeyGen}(Dirsk_{(i_1,\ldots,i_{\ell-1})}, (i_1, \ldots, i_\ell))$ and output the resulting $Dirsk_{(i_1,\ldots,i_\ell)}$.

$\mathsf{FormDirEntry}(DirPK_{root}, (i_1, \ldots, i_\ell), locatortag, filename)$**:** where $PK_{root} = Dirparams$. For each $j \in \{1, \ldots, \ell\}$ run $\mathsf{HIBEEnc}((locatortag, (i_1, \ldots, i_\ell), filename), Dirparams, (i_1, \ldots, i_j))$ to obtain c_j. Output $(c_1, \ldots c_\ell)$.

$\mathsf{DecDirEntry}(Dirsk_{(i_1,\ldots,i_L)}, (c_1, \ldots, c_\ell))$**:** For each $j \in 1, \ldots, \ell$, run $\mathsf{HIBEDec}(Dirsk_{(i_1,\ldots,i_L)}, c_j)$. If any of the decryptions succeeds[10] we output the resulting $locatortag$, $(i_1, \ldots, i_\ell), filename$.

Security The security (against passive servers) follows from the anonymity property of the anonymous HIBE.

4.2 Solution 2: Symmetric key PCE

Here we show a construction for a similar, hierarchical set of categories, but which only allows for secret key encryption. The result is a construction built primarily from simple symmetric key primitives, which is much more efficient than the previous solution, but which does not allow public key encryption.

In our basic scheme we need the following four algorithms:

- The key generation algorithm $\mathsf{PCEKeyGen}(1^k) \rightarrow SK_{root}$ which takes as input the security parameter k and generates a root decryption key for the patient SK_{root}.
- The key derivation algorithm $\mathsf{PCEKeyDer}(sk_{(i_1,\ldots,i_{\ell-1})}, (i_1, \ldots, i_\ell)) \rightarrow sk_{(i_1,\ldots,\ell)}$ takes as input the name of a category (specified as a hierarchical list $(i_1, \ldots, i_\ell)$), and the decryption key $sk_{(i_1,\ldots,i_{\ell-1})}$ for the parent category $cat_{(i_1,\ldots,i_{\ell-1})}$ (or SK_{root} for $\ell = 1$). It outputs a decryption key $sk_{(i_1,\ldots,i_\ell)}$ for category $cat_{(i_1,\ldots,i_\ell)}$.
- The encryption algorithm $\mathsf{Enc}(sk_{(i_1,\ldots,i_\ell)}, (i_1, \ldots, i_\ell), m) \rightarrow c$ takes as input a public key, a message m, a category name specified as $(i_1, \ldots, i_\ell)$, and the corresponding decryption key $sk_{(i_1,\ldots,i_\ell)}$. It outputs an encryption of m for category $cat_{(i_1,\ldots,i_\ell)}$.
- A decryption algorithm $\mathsf{Dec}(sk_{(i_1,\ldots,i_\ell)}, (i_1, \ldots, i_\ell), c) \rightarrow m$ takes as input the name of a category $((i_1, \ldots, i_\ell))$, a corresponding decryption key $sk_{(i_1,\ldots,i_\ell)}$ and a ciphertext c. It outputs decrypted message m if the ciphertext was formed correctly for category $cat_{(i_1,\ldots,i_\ell)}$.

[10] We can easily modify the HIBE so that it rejects messages encrypted under the wrong identity by appending some fixed string to the front of each message encrypted – a ciphertext decrypted with the wrong key will have the wrong form.

4.2.1 Basic Approach

We will construct our PCE system from a pseudorandom function (or a keyed block cipher) $F : \{0,1\}^s(k) \times \{0,1\}^{p(k)} \rightarrow \{0,1\}^{p(k)}$ for some polynomials s, p, and a CCA-secure encryption scheme, $(\mathsf{CCAEnc}, \mathsf{CCADec})$, with keyspace $\{0,1\}^{p(k)}$.

$\mathsf{PCEKeyGen}(1^k)$**:** Choose $SK_{root} \leftarrow \{0,1\}^{p(k)}$.

$\mathsf{PCEKeyDer}(sk_{(i_1,\ldots,i_{\ell-1})}, (i_1, \ldots, i_\ell))$**:** Compute and output $sk_{(i_1,\ldots,i_\ell)} = F_{sk_{(i_1,\ldots,i_{\ell-1})}}(i_\ell)$.

$\mathsf{Enc}(sk_{(i_1,\ldots,i_\ell)}, (i_1, \ldots, i_\ell), m)$**:** Compute and output $c = \mathsf{CCAEnc}(sk_{(i_1,\ldots,i_\ell)}, m)$.

$\mathsf{Dec}(sk_{(i_1,\ldots,i_\ell)}, (i_1, \ldots, i_\ell), c)$**:** Compute and output $m = \mathsf{CCADec}(sk_{(i_1,\ldots,i_\ell)}, c)$.

Security The security of the scheme can be trivially derived from the pseudorandom properties of the PRF and the CCA security of the encryption. (Recall also that we only encrypt files in leaf categories, so each key is used either for delegation or for encryption, but not for both.)

4.2.2 Symmetric Key Searchable Encryption with Preprocessing

Curtmola et al [8] presented a very efficient scheme for symmetric key searchable encryption (SSE) with preprocessing. In their scenario there is a single user, who encrypts a set of documents along with an index specifying in which documents each keyword appears. Both encrypted index and encrypted documents are then sent to the server. The user can use her secret key to generate search trapdoors for particular keywords. The trapdoors do not leak any information about the corresponding keywords, but they can be combined with the encrypted index to determine which documents contain those keywords.

Such a system can be combined with the symmetric key PCE construction described in section 4.2. The main idea is that we store an index for each category. Then, anyone with the decryption key for that category can generate the appropriate trapdoors.

Thus, the construction proceeds as follows:

$\mathsf{SrchKeyGen}(1^k)$**:** Choose $SrchSK_{root} \leftarrow \{0,1\}^{p(k)}$.

$\mathsf{SrchKeyDer}(Srchsk_{(i_1,\ldots,i_{\ell-1})}, (i_1, \ldots, i_\ell))$**:** Compute and output $Srchsk_{(i_1,\ldots,i_\ell)} = F_{Srchsk_{(i_1,\ldots,i_{\ell-1})}}(i_\ell)$.

$\mathsf{IndexGen}(Srchsk_{(i_1,\ldots,i_\ell)}, (i_1, \ldots, i_\ell), word_1, \ldots, word_N)$**:** Generate an index I from $word_1, \ldots, word_N$. Compute $\mathsf{SSEEnc}(Srchsk_{(i_1,\ldots,i_\ell)}, I)$ to form a new searchable encrypted index C_I. Output C_I.

$\mathsf{Trap}(Srchsk_{(i_1,\ldots,i_\ell)}, (i_1, \ldots, i_\ell), word)$**:** Run $\mathsf{SSETrap}(Srchsk_{(i_1,\ldots,i_\ell)}, (i_1, \ldots, i_\ell), word)$, and output the resulting trapdoor T.

$\mathsf{Search}(C_I, T)$**:** Find the appropriate encrypted index C_I for this category. Run $\mathsf{SSESearch}(C_I, T)$ to find the identifiers for the appropriate encrypted documents, and return those ciphertexts.

REMARK 4.1. We note that by reusing the same key to generate many indexes for the same category, we obtain somewhat weaker security properties than those guaranteed by Curtmola et al. (For example, the server may be able to tell when two files within the same category share a keyword, although it will not know what that keyword is.) If we want to obtain the full security, there are two options: one is that we require that each time a party wants to upload to a category, she downloads and decrypts the entire index for that category, updates it with the new documents, and then uploads the encrypted index and new encrypted documents to the server. The second option is that the uploader does not directly add information to the index. Instead she encrypts the document identifier and keyword list directly to the patient, who periodically downloads and decrypts all new document information in each category, downloads and decrypts the indexes for those categories, updates the indexes and reencrypts them, and uploads the new encrypted indexes to the server. An advantage to this approach is that the patient has control of all updates. The disadvantage is that there will be a delay between the time when documents are uploaded, and the time when they are available for searching.

Security With the above modification, the security follows from the pseudorandomness of the PRF and the security of the SSE.

4.2.3 Hiding category labels

First we note that the PCE and searchable encryption described above completely hides the category used—as the category is only used as input to the PRF F. However, creating the directory itself is somewhat more difficult. Here, we will need to make use of some public key primitives as well.

We will construct the directory as follows:

First, our key derivation generates two additional keys. One is a MAC key. The other is used to generate a public/private key pair $Dirpk_{(i_1,\ldots,i_\ell)}$. Furthermore, each uploader receives, as part of their decryption key for category $cat_{(i_1,\ldots,i_\ell)}$, the public keys for all ancestor categories: $Dirpk_{(i_1)},\ldots,Dirpk_{(i_1,\ldots,i_\ell)}$.

Now when a party wants to upload a file, he first chooses a random locator tag. Next he computes a MAC on the tag, category name, and file name. Then for each of $Dirpk_{(i_1)},\ldots,Dirpk_{(i_1,\ldots,i_\ell)}$ he will encrypt the tag, category name, file name, and the MAC under $Dirpk_{(i_1,\ldots,i_j)}$. The directory entry includes all of these ciphertexts.

A party who wants to find all files in a given category first generates the key for that category. She then downloads the directory and uses the public key decryption to attempt to decrypt each entry in the directory. For every element which decrypts successfully to an element of the right form, it will output the filename, category name, and locator tag.

Thus, we have the following algorithms:

$\mathsf{DirKeyGen}(1^k)$**:** Choose $SrchSK_{root} \leftarrow \{0,1\}^{p(k)}$.

$\mathsf{DirKeyDer}(Dirsk_{(i_1,\ldots,i_{\ell-1})},(i_1,\ldots,i_\ell))$**:** Parse $Dirsk_{(i_1,\ldots,i_{\ell-1})}$ as $(pk_{(i_1)},\ldots,pk_{(i_1,\ldots,i_{\ell-1})},s_{(i_1,\ldots,i_{\ell-1})})$. Compute $s_{(i_1,\ldots,i_\ell)} = F_{s_{(i_1,\ldots,i_{\ell-1})}}(\text{"Del"}||i_\ell)$.

Compute $t = F_{s_{(i_1,\ldots,i_\ell)}}(\text{"PK"})$, and $(pk_{(i_1,\ldots,i_\ell)}, dk_{(i_1,\ldots,i_\ell)}) \leftarrow \mathsf{PKKeyGen}(t)$.

Output $sk_{(i_1,\ldots,i_\ell)} = (pk_{(i_1)}\ldots,pk_{(i_1,\ldots,i_\ell)},s_{(i_1,\ldots,i_\ell)})$

$\mathsf{FormDirEntry}(Dirsk_{(i_1,\ldots,i_\ell)},(i_1,\ldots,i_\ell),locatortag,filename)$**:** Parse $Dirsk_{(i_1,\ldots,i_\ell)}$ as $(pk_{(i_1)},\ldots,pk_{(i_1,\ldots,i_\ell)},s_{(i_1,\ldots,i_\ell)})$.

Compute $MACkey = F_{s_{(i_1,\ldots,i_\ell)}}(\text{"MAC"})$. Compute $\sigma \leftarrow \mathsf{MAC}(MACkey, locatortag, filename,(i_1,\ldots,i_\ell))$.

For each $j \in \{1,\ldots,\ell\}$ run $\mathsf{PKEnc}(pk_{(i_1,\ldots,i_j)}(locatortag, (i_1,\ldots,i_\ell), filename, \sigma)$ to obtain c_j. Output $(c_1,\ldots c_\ell)$.

$\mathsf{DecDirEntry}(Dirsk_{(i_1,\ldots,i_L)},(c_1,\ldots,c_\ell))$**:** Parse $Dirsk_{(i_1,\ldots,i_L)}$ as $(pk_{(i_1)},\ldots,pk_{(i_1,\ldots,i_L)},s_{(i_1,\ldots,i_L)})$.

Compute $t = F_{s_{(i_1,\ldots,i_L)}}(\text{"PK"})$, and $(pk_{(i_1,\ldots,i_L)}, dk_{(i_1,\ldots,i_L)}) \leftarrow \mathsf{PKKeyGen}(t)$.

For each $j \in \{1,\ldots,\ell\}$ run $\mathsf{PKDec}(dk_{(i_1,\ldots,i_L)},c_j)$. If decryption succeeds, parse the result as $(locatortag, (i_1,\ldots,i_\ell), filename, \sigma)$. Compute $MACkey = F_{s_{(i_1,\ldots,i_\ell)}}(\text{"MAC"})$. Compute $\mathsf{MACVer}(MACkey,\sigma, (locatortag,(i_1,\ldots,i_\ell),filename))$. If it succeeds, output $(locatortag,(i_1,\ldots,i_\ell),filename))$.

If there is no j for which decryption and MAC verification succeeds, output "Failure".

Security The security of the scheme relies on the pseudorandomness of the PRF, and the CPA security of the public key encryption, and the unforgeability of the MAC. Here we get security even against active attackers because no party who does not have access to the decryption key for the leaf category will be able to generate a valid signature on a new string. There is no way for the adversary to distinguish an invalid ciphertext from a non-match, so he will not learn anything by creating extra entries, or modifying existing entries.

5. A DIFFERENT RECORD STRUCTURE

Here we present a different approach, which provides more flexibility in choosing which hierarchy to use. In the previous two constructions, the hierarchy was fixed from the beginning. However, there are potentially many different ways to organize health data in a hierarchy. One is by medical area as described above. However, one might imagine another organization, by document type, where one category would contain all the lab test results, one would contain notes from doctors, and a third readings from various devices. This might for instance make it easier to give a medical lab write access for all necessary files. Similarly, one might organize a record by sensitivity level, so that it would be easy to give decryption rights to all non-sensitive data at once.

Here we will see a scheme which allows a patient to choose a different hierarchy each time she gives out a decryption key. Essentially, we have a standard set of lowest level categories. For each, we define a fixed public value, which is the same for all patients. Each patient chooses a root key as usual, which defines a decryption key for each of these categories. When the patient wishes to give access rights to her doctor, she can choose any subset of these categories and issue a single key, from which keys for all these categories can be computed. Thus, we can essentially use any hierarchy we choose.

We note that it would not be hard to extend the previous scheme to allow for several fixed hierarchies. However, there will be an efficiency loss for each additional hierarchy, so there is a limit to how much flexibility an efficient system could provide. On the other hand, with the approach described in this section, one could have an unlimited number of hierarchies with no efficiency loss.

With this approach, the health record is divided into a set of leaf categories. When Alice wants to grant access rights to her doctor, she can choose to allow access to any subset of those categories. As there might be many of these, our goal is to allow Alice to compute one concise key that will allow her physician to access to documents in any of the desired categories.

Advantages This has the advantage that it allows for very flexible structures; Alice can partition her record in any arbitrary way and she will still be able to give her doctor a concise key. Furthermore, she can choose a different partitioning each time she gives out a key.

Disadvantages Alice must know exactly which files (or category names) she wishes to grant others access to before she can generate the decryption key. For example, if we have a separate category for each visit date, she will have to know all appropriate dates before she can issue a decryption key.

Efficiency: Our constructions rely on RSA group operations, which are much less efficient than symmetric key operations.

Notation.

We use cat_{i_j} to denote the leaf category with name i_j. Here keys correspond to sets of leaf categories. If $S = \{i_1, \ldots, i_n\}$ is a set of leaf categories, then we denote the corresponding secret key sk_S. We also sometimes use the notation cat_S to denote $cat_{i_1}, \cup, \ldots, \cup, cat_{i_\ell}$.

Thus, in our basic scheme we need the following four algorithms:

- The key generation algorithm $\mathsf{PCEKeyGen}(1^k) \rightarrow SK_{root}$ which takes as input the security parameter k and generates a root decryption key for the patient SK_{root}.
- The key derivation algorithm $\mathsf{PCEKeyDer}(sk_S, S, S') \rightarrow sk_{S'}$ takes as input a key, the of names of the corresponding set of categories (specified by the set $S = \{i_1, \ldots, i_n\}$), and a subset $S' \in S$ of those categories. It outputs a decryption key $sk_{S'}$ for $cat_{S'}$.
- The encryption algorithm $\mathsf{Enc}(sk_j, j, m) \rightarrow c$ takes as input a public key, a message m, a category name j,[11] and the corresponding decryption key $sk_{\{j\}}$. It outputs an encryption of m for category $cat_{\{j\}}$.
- A decryption algorithm $\mathsf{Dec}(sk_{\{j\}}, j, c) \rightarrow m$ takes as input the name of a category j, a corresponding decryption key $sk_{\{j\}}$, and a ciphertext c. It outputs decrypted message m if the ciphertext was formed correctly for category cat_j.

5.1 Basic Approach

Our construction is based on a scheme originally proposed for concisely transmitting large numbers of keys in broadcast scenarios [4].

The PCE scheme works as follows: Let h be a collision resistant hash function that maps strings to primes.

Let $H : \{0,1\}^* \rightarrow \{0,1\}^{f(k)}$ be a publicly available hash function which we will model as a random oracle.

Finally, let $\mathsf{CCAEnc}, \mathsf{CCADec}$ be a CCA-secure encryption scheme, with keyspace $\{0,1\}^{f(k)}$.

[11]Here we require that documents are contained in a single category. Our scheme generalizes in a straightforward way to the case where documents can be in several categories, and a party must have permissions for all categories in order to access the document.

$\mathsf{PCEKeyGen}(1^k)$**:** Generate RSA modulus $N = pq$ for primes p, q. Choose random $y \leftarrow Z_N^*$. Output $SK_{root} = (p, q, y)$.

$\mathsf{PCEKeyDer}(sk_S, S, S')$**:** Parse S, S' as sets of strings, and parse $sk_S = (N, y_S)$ (or $SK_{root} = (p, q, y_S)$)

First we will compute $e_j = h(j)$ for each $j \in S$. Then $sk_{S'}$ should be $(N, y^{\frac{1}{\prod_{j \in S'} e_j}} \mod N)$. Note that this can be computed in two ways: Using $SK_{root} = (p, q)$, we can compute it directly. (This is what the patient will do.) Alternatively, if we know $y_S = y^{\frac{1}{\prod_{j \in S} e_j}} \mod N$, and $S' \subset S$, then we can compute $sk_{S'} = (N, y_S^{\prod_{j \in S \setminus S'}} \mod N)$. (This is how all those who have been delegated access rights will compute their encryption and decryption keys.)

$\mathsf{Enc}(sk_{\{j\}}, j, m)$**:** where $sk_S = (N, y^{e_j} \mod N)$ for $e_j = h(j)$. Compute and output $c = \mathsf{CCAEnc}(H(sk_S), m)$.

$\mathsf{Dec}(sk_{\{j\}}, j, c)$**:** where $sk_{\{j\}} = (N, y^{e_j} \mod N)$ for $e_j = h(j)$. Compute and output $m = \mathsf{CCADec}(H(sk_S), c)$.

The security of this scheme under the strong RSA assumption can be trivially derived from the security of the construction in [4], which is based on a lemma by Shamir [17].

5.2 Adding Searchability

We note that this scheme can be easily extended to allow efficient searching over using the same techniques as in Section 4.2.

5.3 Hiding Category Labels

We use roughly the same approach as that presented in Section 4.2. However, here we assume that when the patient sends a subkey to the doctor, he also sends the name of all the leaf categories that it corresponds to. (Note that, unlike the scenario in Section 4, here we already require that the patient determine all leaf categories when the subkeys are generated.)

Now, when generating the encrypted directory, we do not encrypt under keys for all possible ancestor categories, since there may be many of these. Instead, we only encrypt under the key for the given leaf category. We also add a pseudorandom identifier for the category, generated using the category subkey.

Then, when the doctor wants to download files in a given category, he can generate the appropriate identifier and decrypt the entries which are listed with that identifier. He will identify the files that he wishes to view, and send the corresponding locator tags to the server to obtain the encrypted files.

Thus, we have the following algorithms:

$\mathsf{DirKeyGen}(1^k)$**:** Generate RSA modulus $N = pq$ for primes p, q. Choose random $y \leftarrow Z_N^*$. Output $DirSK_{root} = (p, q, y)$.

$\mathsf{DirKeyDer}(sk_S, S, S')i_\ell))$**:** First we will compute $e_j = h(j)$ for each $j \in S$. Then compute and output $sk_{S'} = (N, y^{\frac{1}{\prod_{j \in S'} e_j}} \mod N)$ as described in the $\mathsf{PCEKeyDer}$ algorithm above.

$\mathsf{FormDirEntry}(Dirsk_{\{j\}}, j, locatortag, filename)$**:** Compute $s = H(Dirsk_S)$. Compute $MACkey = F_s(\text{“MAC”})$. Compute $\sigma \leftarrow \mathsf{MAC}(MACkey, locatortag, filename, j)$.

Compute $dk = F_s(\text{“Enc”})$, and run $\mathsf{CCAEnc}(dk, (locatortag, filename, j, \sigma))$ to obtain c.

Properties	Public Key PCE Sec 4.1	Symmetric Key PCE Sec 4.2	Flexible PCE Sec 5
Upload without Key Distribution	Yes	No	No
Flexible Hierarchies	No	No	Yes
High Efficiency	No	Yes	No
Easy to Add Categories	Yes	Yes	No

Table 1: Properties of the schemes presented in Sections 4 and 5.

Compute $identifier = F_s(\text{“Identifier”}||\text{j})$

Output $(identifier, c)$.

$\mathsf{DecDirEntry}(Dirsk_{\{j\}}, j, (identifier', c))$**:** Compute $MACkey$, dk, $identifier$ for category cat_j as above. If $identifier \neq identifier'$, output “Failure”.

Otherwise, run $\mathsf{PKDec}(dk_{(i_1,\ldots,i_L)}, c_j)$. If decryption succeeds, parse the result as $(locatortag, filename, j, \sigma)$. Compute $\mathsf{MACVer}(MACkey, \sigma,$ $(locatortag, (i_1, \ldots, i_\ell), filename))$. If it succeeds, output $(locatortag, filename, j)$. If not, output “Failure”.

6. CONCLUSIONS

We have presented several schemes for Patient Controlled Encryption, each appropriate for a different setting. Table 1 summarizes the advantages and disadvantages of these schemes. For a concrete design, we suggest that one follow the set-up described in Section 3. We conclude that it is possible and practical to achieve secure and private EMR while maintaining efficiency and functionality, including searchability and delegation.

Acknowledgements.
We thank Mihir Bellare for valuable discussions.

7. REFERENCES

[1] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous ibe, and extensions. *J. Cryptology*, 21(3):350–391, 2008.

[2] Selim G. Akl and Peter D. Taylor. Cryptographic solution to a problem of access control in a hierarchy. *ACM Trans. Comput. Syst.*, 1(3):239–248, 1983.

[3] Mikhail J. Atallah, Marina Blanton, Nelly Fazio, and Keith B. Frikken. Dynamic and efficient key management for access hierarchies. *ACM Trans. Inf. Syst. Secur.*, 12(3):1–43, 2009.

[4] Josh Benaloh. Key compression and its application to digital fingerprinting. Technical Report Technical Report, Microsoft Research, 2009.

[5] Matt Blaze. A cryptographic file system for UNIX. In *ACM Conference on Computer and Communications Security*, pages 158–165, 1993.

[6] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *EUROCRYPT*, pages 506–522, 2004.

[7] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-ciphertext security from identity-based encryption. In *EUROCRYPT*, pages 207–222, 2004.

[8] Reza Curtmola, Juan A. Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *ACM Conference on Computer and Communications Security*, pages 79–88, 2006.

[9] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Over-encryption: management of access control evolution on outsourced data. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 123–134. VLDB Endowment, 2007.

[10] Kevin Fu. Group sharing and random access in cryptographic storage file systems. Master's thesis, Massachusetts Institute of Technology, June 1999.

[11] Craig Gentry and Alice Silverberg. Hierarchical ID-based cryptography. In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 548–566. Springer Verlag, 2002.

[12] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.

[13] Urs Hengartner and Peter Steenkiste. Exploiting hierarchical identity-based encryption for access control to pervasive computing information. In *SECURECOMM '05: Proceedings of the First International Conference on Security and Privacy for Emerging Areas in Communications Networks*, pages 384–396, Washington, DC, USA, 2005. IEEE Computer Society.

[14] Gerome Miklau and Dan Suciu. Controlling access to published data using cryptography. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 898–909. VLDB Endowment, 2003.

[15] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[16] Ravi S. Sandhu. Cryptographic implementation of a tree hierarchy for access control. *Inf. Process. Lett.*, 27(2):95–98, 1988.

[17] Adi Shamir. On the generation of cryptographically strong pseudorandom sequences. In *ACM Transaction on Computer Systems*, volume 1, pages 38–44, 1983.

[18] Adi Shamir. Identity-based cryptosystems and signature schemes. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology — CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer Verlag, 1985.