Attribute-Based Publishing with Hidden Credentials and Hidden Policies*

Apu Kapadia[†‡], Patrick P. Tsang[†], Sean W. Smith[†]

[†]Department of Computer Science
Dartmouth College
Hanover, NH, USA

[‡]Institute for Security Technology Studies
Dartmouth College
Hanover, NH, USA

{akapadia, patrick, sws}@cs.dartmouth.edu

Abstract

With Hidden Credentials Alice can send policy-encrypted data to Bob in such a way that he can decrypt the data only with the right combination of credentials. Alice gains no knowledge of Bob's credentials in the process, and hence the name "Hidden Credentials." Research on Hidden Credential systems has focused on messages sent to single *recipients, where the sender needs to know the recipient's pseudonym beforehand, and on Hidden Policies, where Bob learns as little information as possible about Alice's policy for decrypting the message. Current schemes provide weak policy privacy — with non-interactive schemes, the recipient can learn parts of the policy, and with interactive schemes based on secure multiparty computation, a user can try different sets of credentials as input to gain knowledge of the policy after repeated decryption attempts. Furthermore, existing schemes do not support policies with negations efficiently. For example, a policy stating "Bob is not a student" is hard to enforce since Bob can simply withhold, or not use, his student credential.*

We propose a system called PEAPOD *(Privacy-Enhanced Attribute-based Publishing Of Data) that provides the following properties: (1) Users can securely publish data protected by attribute-based policies to* multiple possible recipients *without requiring interaction between senders and receivers. This is achieved by using a semi-trusted server. (2) The plaintext message and the policy are completely hidden from the server. (3) Any recipient, intended or not, learns no other information about a message's policy beyond the number of clauses in policy that were satisfied. Furthermore the recipient is forced to use* all *of his or her issued credentials for decryption, and therefore cannot mount inference attacks by trying to decrypt the message with different subsets of credentials. (4) Lastly, since recipients are forced to use all their credentials for decryption,* PEAPOD *efficiently supports non-monotonic boolean policies by allowing senders to include negations in their policies.*

*This research was supported in part by the NSF, under grant CNS-0524695, and the Bureau of Justice Assistance, under grant 2005-DD-BX-1091. The views and conclusions do not necessarily reflect the views of the sponsors.

1. Introduction

Hidden Credentials [20, 9] were first proposed to facilitate trustworthy interaction between strangers in open environments. Using these schemes, Alice encrypts data to a specific individual Bob using an attribute-based policy such as "`Bob is a Student or Professor`." Bob can then decrypt this data if and only if he has the correct combination of credentials to satisfy Alice's policy. Since Bob decrypts the message without revealing the result to Alice, Alice gains no knowledge of Bob's credentials (hence the name "Hidden Credentials"). Hidden Credentials preserve Bob's privacy since he may consider many of his attributes to be sensitive information and would like to hide them from Alice. Current schemes [9, 15] also try to limit what Bob can learn about Alice's policy. There could be several reasons for this — Alice may want to prevent users from "gaming" the system, i.e., changing their behavior to gain access to a message, or inferring which messages are important based on their policies. For example, attackers might focus their energy on trying to decrypt messages for CIA agents if the policy is public knowledge. In some situations Alice's policy may reveal private information about herself, in which case she would like to protect her privacy against both intended and non-intended recipients of the message. Ideally, even if Bob is able to decrypt the message (i.e., he is an "intended recipient"), he should not learn anything about the structure of the policy or which of his credentials were necessary for decryption. Providing policy privacy against intended recipients, however, has proven

to be difficult. Current non-interactive schemes [20, 9] achieve partial policy privacy since Bob can learn information about sub-expressions of the policy that he satisfies. Bob is able to learn all the "satisfying sets" (where each satisfying set is a set of credentials that satisfies the policy) that are subsets of his credentials. To close this gap, Frikken et al. [15] propose an interactive scheme where each party learns only whether Bob satisfied the policy, and whatever can be inferred from that. If Bob decrypts a message by trying different combinations of credentials, however, he can still infer which credentials were necessary for decryption over several repeated decryption attempts. Similarly, other approaches in trust negotiation [37, 35, 38, 36, 39] such as oblivious attribute certificates (OACerts) [24] suffer from the same drawbacks. Ideally, a system with Hidden Policies should prevent Bob from inferring information about the policy over repeated decryption attempts.

Another drawback of previous approaches is that the sender Alice needs to know the identity (or pseudonym) of the recipient *before* sending the message, i.e., each message has a single intended recipient. In this paper, we propose a new problem in Hidden Credential systems — *securely publishing policy-encrypted data to multiple possible recipients using Hidden Credentials and Hidden Policies*, i.e., Alice should be able to securely publish messages that can be decrypted by *anybody* with the correct set of attributes. Consider the following motivating example in the context of a bulletin-board service.

Matchmaking example: Alice maintains a public profile on the bulletin-board service. She also maintains a protected profile containing more personal information such as her photograph, birth date, etc. She would like to share this information only with people who satisfy her criteria for a perfect partner and therefore encrypts it with her criteria as the policy. Alice, however, would like to keep her criteria secret. There are a couple of good reasons for this. The criteria may be embarrassing to Alice, e.g., she might be looking for a partner that also has a particular disease or disorder, or maybe she simply does not want suitors to game the system by looking at her preferences and pretending to fit the description. If Bob is interested in Alice based on her public profile, he can attempt to decrypt her protected profile. Bob is able to view this information if only if he satisfies her policy, and in the process does not learn Alice's criteria (i.e., her policy remains hidden) beyond what he can infer from the fact that he satisfies her policy. Alice, on the other hand, does not learn whether Bob tried to access her profile or not, let alone whether the decryption was successful. Bob's credentials, therefore, remain hidden from Alice.[1]

[1]If Bob chooses to inform Alice that he was able to view her protected profile, then Alice can infer that he possesses credentials that satisfy her policy. Such attacks are outside the scope of this paper, and have been addressed in the context of trust negotiation [20].

It is not obvious how current Hidden Credential systems can be modified to support multiple possible recipients. For example, previous research [20, 9, 15] has focused on single recipients by relying on *Identity-Based Encryption (IBE)* [8]. A trusted *Private Key Generator (PKG)* can issue a private key (credential) to Bob that corresponds to the public key (attribute) "`Bob is a student`." If Alice encrypts data to Bob using this public key, then Bob will be able to decrypt it if and only if he is a student. If Alice would like the data to be decryptable by *any* student, more sophisticated group-key management is needed. For example, an IBE-based scheme could require Alice to use the public key "`Student`" for encryption, and a shared private key among all students. This is undesirable because the compromise of a shared group key requires a new private key to be deployed to all students.[2] Other related approaches include "Key-Policy Attribute-Based Encryption (KP-ABE)" [17, 31], where attributes are associated with ciphertexts and keys encode decryption policies based on the data's attributes. For example, Alice can supply Bob with a key to decrypt only her data with the attributes "music video AND Metallica." Following the terminology in [17], we focus on "Ciphertext-Policy Attribute-Based Encryption (CP-ABE)," where attributes are associated with users, and policies are encoded in the ciphertexts based on users' attributes. Since KP-ABE and CP-ABE address different problems, we do not discuss KP-ABE in the remainder of the paper.

We provide a non-interactive solution that avoids shared private keys and call our system *Privacy-Enhanced Attribute-based Publishing Of Data (PEAPOD)*. As a building-block, PEAPOD uses a modified version of Khurana et al.'s Secure E-mail List Service (SELS) [23], which is a proxy encryption scheme [6] for encrypting messages to the subscribers of an email list. Briefly, when a message is sent to the list, SELS allows the "list server" to re-encrypt messages to the list's subscribers without access to the plaintext. Re-encryption ensures that each recipient can use his or her unique decryption key, thereby eliminating the need for shared keys. We show how Alice can encrypt data with attribute-based policies to multiple possible recipients by building a system on top of SELS, where the possession of a particular attribute corresponds to the user's membership in a SELS list for that attribute. Each attribute is associated with a public encryption key, and users possess *unique decryption keys* ("credentials") for their correspond-

[2]Furthermore, in an IBE-based scheme a compromised key would require rekeying *all* attribute keys since all private keys are generated from a unique secret. This is because the public-key for "`Student`" remains unchanged, and generating a new private key for "`Student`" amounts to changing the unique secret. A less drastic alternative would be to append version numbers to attributes (e.g., "`Student.v1`" and "`Student.v2`"). However, this would require a mechanism for users to acquire the current version numbers of attributes, which could be cumbersome.

ing attributes.

While such an approach solves the problem of encrypting messages to multiple possible recipients without the need for shared private keys, it is not clear how the privacy of the policies can be maintained against the proxy encryption server and the recipients. PEAPOD makes use of homomorphic encryption [30] to contribute several interesting properties that are absent from previous schemes. Our system provides "clausal policy privacy" against *all* recipients, intended or not. That is, assuming a disjunctive normal form policy (a disjunction of conjunctive clauses), Bob learns no information other than the number of clauses in Alice's policy that he satisfies.[3] Even for the clauses that he does satisfy, he gains no knowledge of which of his attributes were used to satisfy those clauses. This is possible because our system forces the recipient Bob to use *all* his credentials for decrypting a message. We contrast this with the best known non-interactive scheme [9], where Bob learns the entire set of attributes for a clause that he satisfies. Furthermore, in PEAPOD, Bob cannot try to decrypt the message with different subsets of his credentials. We contrast this with the best known interactive scheme [15] where Bob can still mount inference attacks with subsets of his credentials by making repeated attempts. Finally, unlike existing hidden credential and trust negotiation schemes, our system efficiently supports non-monotonic boolean policies, i.e., negations of attributes can be included in policies, and policies *can check for the absence of an issued credential*. Bob, therefore, cannot withhold a credential that he has been issued. Existing approaches suggest that users can be issued explicit attributes such as `not a student`, thereby supporting negations in policies. This approach, however, comes at the cost of doubling the number of attributes in the system. Furthermore, users in the system must now be issued "negative credentials" for all the attributes that they do not possess, leading to a much higher computational burden for issuing and revoking attributes. For example, in a university setting, if a new attribute `is a provost` is added to the system, the negative credential `is not a provost` will have to be issued to many thousands of users. In contrast, our scheme does not require any extra attributes, and users maintain credentials for only the attributes that they possess.

Contributions We motivate the need for a system that supports attribute-based encryption of messages to multiple possible recipients using Hidden Credentials and Hidden Policies. We present our system PEAPOD that makes the following specific contributions:

1. *Offline publishing to multiple recipients:* PEAPOD decouples the sending and receiving phases, and publishers do not need to interact with recipients or know their pseudonyms beforehand. Users are able to publish information to multiple recipients and shared decryption keys are not used, which simplifies key management. Unlike other Hidden Credential systems, PEAPOD makes use of a semi-trusted server as an intermediary to achieve this property.

2. *Message confidentiality and policy privacy:* PEAPOD provides message confidentiality based on the sender's policy, and clausal policy privacy against recipients. Furthermore, the plaintext and the policy are *completely hidden* from the server, even though the server performs essential transformations on the ciphertext for each recipient.

3. *Non-monotonic boolean policies:* Unlike other approaches, PEAPOD supports policies based on negations of attributes without the need for creating explicit attributes to represent negations. Our approach leverages the server to ensure that users must use *all* of their credentials for decryption, thereby preventing users from withholding credentials.

2. Privacy-Enhanced Attribute-based Publishing of Data

Departing from previous IBE-based approaches for Hidden Credential systems, PEAPOD uses proxy encryption and splits the trusted duties of key-management between a Server and a Certification Authority (CA). Messages are "proxy-encrypted" under attribute-based policies using public attribute-keys that are set up by these entities, and are then deposited at the Server for later retrieval. We now provide a brief overview of the cryptographic tools used in PEAPOD, and formalize various notions of security that our system must satisfy.

2.1. Preliminaries

Proxy encryption As mentioned earlier, we use SELS [23] as a building block for our system. In SELS, the sender encrypts an email message and sends it to the list server. The list server plays the role of the proxy and re-encrypts the encrypted email (without access to the plaintext) for every subscriber in the list so that each subscriber can decrypt the message with his or her own unique private decryption key. Conventional proxy encryption allows only single recipients because the decryption key is known only to a single party.[4] SELS gets around this limitation by requiring the sum K of the proxy's re-encryption secret-key s_u for a subscriber u and that subscriber's

[3] and whatever can be inferred from that fact

[4] Of course, one could share that decryption key to allow group decryption. However, the difficulty of key management makes this approach very unfavorable.

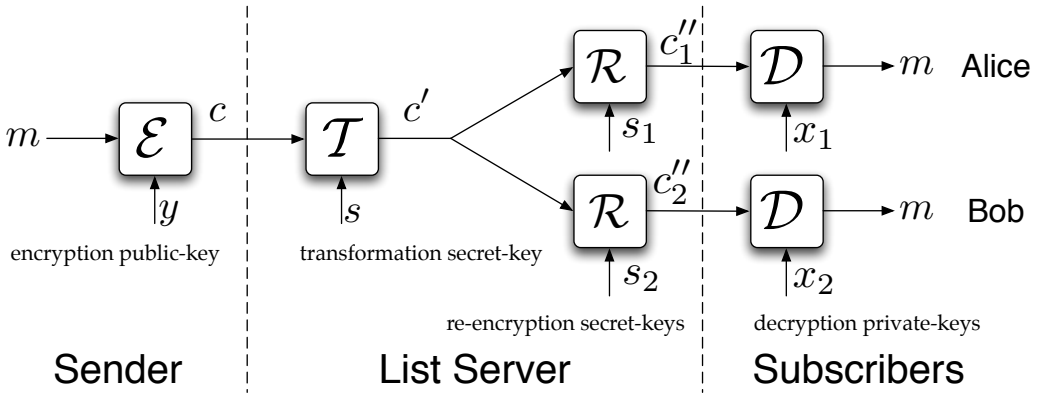

Figure 1. A block diagram illustrating the basic operations of SELS [23] that we use in our protocol

decryption private-key x_u to be the same in the underlying Elgamal proxy encryption [21] for all subscribers. That is, for all subscribers u, $x_u + s_u = K$. This property allows individual subscribers to decrypt messages with their own decryption private keys for messages encrypted with the list's public key. Moreover, SELS splits the encryption step of Elgamal proxy encryption into two operations, which we denote as $\mathcal{E}$ and $\mathcal{T}$ respectively, as explained below.

Figure 1 illustrates how SELS works under our terminology. The sender Elgamal-encrypts ($\mathcal{E}$) a message m under *encryption public-key* y, where $y = g^x$ for some group generator g and secret key x, and sends the resulting ciphertext $c = (A, B) = (g^r, my^r)$ to the list server, where r is the randomness used. The list server transforms ($\mathcal{T}$) c into another ciphertext $c' = (A', B') = (A, BA^s)$ using its *transformation secret-key* s, where $s = K - x$. Note that c' is equivalent to the Elgamal encryption of m under another encryption public key $\tilde{y}$, where $\tilde{y} = yg^s = g^{x+s} = g^K$. The list server re-encrypts ($\mathcal{R}$) ciphertext c' using its *re-encryption secret-keys* s_1 and s_2 for subscribers Alice and Bob into c''_1 and c''_2 respectively, where $c''_i = (A''_i, B''_i) = (A', B'/A'^{s_i})$. The server then sends the resulting re-encrypted ciphertexts c''_1 and c''_2 to the corresponding users. Finally each of them decrypts ($\mathcal{D}$) using his or her *decryption private-key* (x_1 for Alice and x_2 for Bob) to recover message m as $B''_i/A''^{x_i}_i$. As mentioned earlier, the two steps $\mathcal{R}$ and $\mathcal{D}$ amount to the Elgamal decryption under private key K.

To subscribe users to a list, SELS describes a protocol where the list manager interacts with the list server in such a way that K is not known to any single party. Therefore, if the list manager or a subscribed user do not collude maliciously with the list server, the list server is unable to obtain K and thereby decrypt messages sent to the list. Furthermore, the list server cannot subscribe users to the list without interacting with the list manager, which is responsible for managing list membership.

Theorem 1 in [23] states that if Elgamal is secure against chosen-plaintext attacks by any probabilistic polytime (PPT) adversary, then so is SELS. As we will see, SELS can be adapted to construct PEAPOD for policies that contain only a single attribute. It is not immediately clear, however, how SELS can be used to support complex policies based on boolean combinations of attributes, and how to support policy privacy. To achieve its goals, PEAPOD therefore also makes use of other tools such as homomorphic encryption.

Homomorphic encryption An encryption scheme ($\mathcal{K}$, $\mathcal{E}$, $\mathcal{D}$)[5] is *homomorphic* if there exists an operation $\otimes$ on ciphertexts such that for any key pair (x, y) generated by $\mathcal{K}$ and any messages m_1, m_2 in the message space $\mathcal{M}$, we have that $\mathcal{D}_x(\mathcal{E}_y(m_1) \otimes \mathcal{E}_y(m_2)) = m_1 \oplus m_2$, where $\oplus$ is an operation on messages. In other words, ciphertexts for m_1 and m_2 can be combined to obtain a ciphertext for $m_1 \oplus m_2$ *without* access to m_1 and m_2. More generally, the homomorphic property allows meaningful manipulation of ciphertexts without the knowledge of the underlying plaintexts. Applications of homomorphic encryption include electronic voting [19, 18, 2], electronic auctions [1, 34, 25, 11], Mix-nets [12, 16], and verifiable encryption [28].

Elgamal encryption is one example of homomorphic encryption. Consider $c_i = \mathcal{E}_y(m_i) = (g^{r_i}, m_i y^{r_i}) = (A_i, B_i)$ for $i = 1, 2$. Define $\otimes$ as $(A_1, B_1) \otimes (A_2, B_2) \mapsto (A_1 \cdot A_2, B_1 \cdot B_2)$. Then $\mathcal{D}_x(c_1 \otimes c_2) = \mathcal{D}_x(g^{r_1+r_2}, m_1 m_2 y^{r_1+r_2}) = m_1 \cdot m_2$. Paillier's cryptosystem [27] is another example.

Policies and attributes In PEAPOD, policies are non-monotonic boolean formulae in disjunctive normal form

[5] $\mathcal{K}$, $\mathcal{E}$, $\mathcal{D}$ are the key generation, encryption, and decryption algorithms respectively

(DNF), i.e. disjunctions ("ORs") of one or more clauses, which are in turn conjunctions ("ANDs") of one or more literals, where a literal is either an attribute or its negation. We argue that most policies are naturally DNF in form as senders usually have a few classes of intended recipients in mind (a disjunction of clauses), where each class is defined by a set of attributes (a clause).

To simplify the description of our system, we represent a clause within a policy using the symbols $\checkmark$, $\times$ and $*$, which denote respectively attributes in the system that are "required" (the $\checkmark$-attributes), "forbidden" (the $\times$-attributes) and "irrelevant" (the $*$-attributes). For example, if we reconsider the matchmaking example mentioned earlier and imagine the scenario in which there are altogether five attributes in the system: {`male`, `college graduate`, `age in 30's`, `smoker`, `pet lover`}, then the clause "`male` $\wedge$ `age in 30's` $\wedge \neg$`smoker`" is represented by $\langle \checkmark, *, \checkmark, \times, * \rangle$. Any clause in a non-monotonic boolean expression can be represented by such a tuple, which indicates the nature of each attribute in the clause (required, forbidden, or irrelevant). Recall that policies are a disjunction of multiple clauses. For example, the policy "(`male` $\wedge$ `age in 30's`) $\vee$ (`male` $\wedge \neg$`smoker`)" would be represented by a set of the two clauses: $\{\langle \checkmark, *, \checkmark, *, * \rangle, \langle \checkmark, *, *, \times, * \rangle\}$.

We say that an attribute set A satisfies a policy $P = \{C_1, C_2, \ldots, C_n\}$, if there exists a clause $C_i \in P$ for which A contains all the $\checkmark$-attributes and does not contain any of the $\times$-attributes in C_i. When we say a user Bob satisfies a policy, we mean the attribute set possessed by Bob satisfies the policy.

2.2. Security Notions

As hinted in Section 1, designing a system for publishing data with both policy privacy and credential privacy faces a number of security challenges. To rigorously reason about the security of PEAPOD, we need to formalize various security notions. Definitions 1 and 3 provide ideal security models for confidentiality and policy privacy under which PEAPOD is immune to adaptive chosen ciphertext attacks and user coalition attacks (where users collude by "pooling in" their credentials). Our construction is based on Elgamal encryption, and therefore is not immune to adaptive chosen ciphertext attacks. Moreover, our scheme provides security against only restricted versions of coalition attacks. Definitions 2 and 5 describe the models under which our construction of PEAPOD is secure.

Confidentiality Ideally, when given a ciphertext deposited at or retrieved from the server, individual entities (including the Server) other than the intended recipients specified by the associated policy should not be able to learn anything about the underlying plaintext message (except perhaps its size). Such a guarantee should hold even if users who are not intended recipients collude arbitrarily. Moreover, when attempting to break the confidentiality, an adversary should be given the chance to study the decryption of arbitrary retrieved ciphertexts. Formally speaking, we define the confidentiality of PEAPOD as follows.

Definition 1 (C-IND-CCA2-UCA) *A PEAPOD system has* Ciphertext Indistinguishability against Adaptive Chosen Ciphertext Attack and User Coalition Attack (C-IND-CCA2-UCA) *if no PPT adversary $\mathcal{A}$ can win the following game against the challenger $\mathcal{C}$ with probability non-negligibly greater than 1/2.*

1. *(Setup Phase.)* $\mathcal{C}$ sets up the PEAPOD system and makes all public parameters such as the attributes in the system available to $\mathcal{A}$.

2. *(Probing Phase I.)* $\mathcal{A}$ may corrupt the Server if no user has been corrupted, thereby learning its secrets and acting on behalf of it. If the Server has not been corrupted, then $\mathcal{A}$ has the ability to arbitrarily and adaptively: (a) register a new honest user into the system,[6] (b) corrupt an honest user, thereby learning his or her secrets and acting on behalf of him or her, (c) make deposit and retrieval requests to the Server, and (d) ask honest users for decrypting any retrieved ciphertexts.

3. *(Challenge Phase.)* At some point, $\mathcal{A}$ outputs two messages M_0, M_1 of equal length and a policy P^* of his choice under the following restriction:

   Restriction 1: *None of the corrupted users satisfies the policy P^* throughout the game.*

   Then $\mathcal{C}$ flips a fair coin $b \in_R \{0, 1\}$ and encrypts message M_b under policy P^* according to the sender's encryption algorithm. The resulting ciphertext C^* is returned to $\mathcal{A}$.

4. *(Probing Phase II.)* $\mathcal{A}$ may do whatever he is allowed to in Probing Phase I, except that (a) Restriction 1 applies, and (b) $\mathcal{A}$ may not ask any honest user for decrypting C^*.

5. *(End-Game Phase.)* Eventually $\mathcal{A}$ outputs his guess $\tilde{b} \in \{0, 1\}$ on b. $\mathcal{A}$ wins if and only if $\tilde{b} = b$.

□

The above definition of confidentiality is based on the well-known IND-CCA2 security model for conventional encryption schemes. It captures coalition resistance because the adversary is allowed to corrupt an arbitrary set of users as long as none of them satisfies the challenge policy.

[6] $\mathcal{A}$ also gets to decides the set of attributes possessed by the user being registered. For simplicity, we assume a user immediately acquires all the credentials for his or her attributes upon registration.

This models the real-world attack scenario when a coalition of users who individually do not satisfy the policy tries to "pool in" their credentials and gain knowledge about the message.

The PEAPOD system we propose has a weaker form of coalition-resistance, since some coalitions of users *can* pool in their credentials and decrypt the message. We say that a coalition of users "collectively satisfies policy P" if there exists a subset of users in the coalition such that the union A of their attributes satisfies the policy. In our construction, a coalition of users can decrypt a message with policy P *only if* they collectively satisfy the policy P. The following definition captures the confidentiality our system provides, which is secure against chosen plaintext attack and the restricted form of coalition attack just mentioned.

Definition 2 (C-IND-CPA-RUCA) *PEAPOD has* Ciphertext Indistinguishability against Chosen Plaintext Attack and Restricted User Coalition Attack (C-IND-CPA-RUCA) *if no PPT adversary $\mathcal{A}$ can win the following game against the challenger $\mathcal{C}$ with probability non-negligibly greater than 1/2.*

The game is the same as that in Definition 1, except that (1) Ability 2(d) is removed, i.e., $\mathcal{A}$ may not ask any honest users for decrypting any retrieved ciphertext, and (2) Restriction 1 is modified as:

Restriction 1′: *None of the coalitions of corrupted users collectively satisfies the policy P^* throughout the game.*

□

Policy-privacy Policy privacy hides the policies under which messages are encrypted from both the server and the recipients. Ideally, the server or any recipient Bob (intended or not) should not be able to gain any knowledge of the policy except that Bob knows whether he satisfies the policy. As with confidentiality, ideally no coalition of users (who individually do not satisfy the policy) should be able to gain any knowledge about the policy (except perhaps its maximum size). This notion of policy privacy is captured formally by the following definition.

Definition 3 (P-IND-CCA2-UCA) *A PEAPOD system has* Policy Indistinguishability against Adaptive Chosen Ciphertext Attack and User Coalition Attack (P-IND-CCA2-UCA) *if no PPT adversary can win the following game against the challenger $\mathcal{C}$ with probability non-negligibly greater than 1/2.*

1. *(Game Setup.)* Same as that in Definition 1.
2. *(Probing Phase I.)* Same as that in Definition 1.
3. *(Challenge Phase.)* At some point, $\mathcal{A}$ sends to $\mathcal{C}$ a message M^* and two valid policies P_0, P_1 of his choice under the following restriction:

   Restriction 2: *Either all corrupted users satisfy none of the policies P_0 and P_1 or they all satisfy both policies throughout the game.*

   Then $\mathcal{C}$ flips a fair coin $b \in_R \{0,1\}$ and encrypts the message M^* under policy P_b according to the sender's encryption algorithm. The resulting ciphertext C^* is returned to $\mathcal{A}$.
4. *(Probing Phase II.)* $\mathcal{A}$ may do whatever he is allowed to in Probing Phase I, except that (a) the Restriction 2 applies, and (b) $\mathcal{A}$ may not ask any honest user for decrypting C^*.
5. *(End Game.)* Same as that in Definition 1.

□

We now introduce a weaker form of policy privacy, which we call *clausal policy privacy*. We believe that other constructions of PEAPOD, may be able to provide clausal policy privacy at best, and therefore emphasize this weaker intermediate model before we describe the security model for our construction in Definition 5. It is weaker because an intended recipient is able to infer information more than merely whether he or she satisfies the policy. Precisely, that piece of extra information is the number of clauses a recipient satisfies. This makes two policies with a different number of satisfying clauses distinguishable by an intended recipient. Hence policy indistinguishability is only among those policies with the same number of satisfying clauses. The following is a formal definition of this notion.

Definition 4 (C-P-IND-CCA2-UCA) *A PEAPOD system has* Clausal Policy Indistinguishability against Adaptive Chosen Ciphertext Attack and User Coalition Attack (C-P-IND-CCA2-UCA) *if no PPT adversary can win the following game against the challenger $\mathcal{C}$ with probability non-negligibly greater than 1/2.*

The game is the same as that in Definition 3, except that Restriction 2 is modified as:

Restriction 2′: *All corrupted users satisfy the same number of clauses in both policies P_0 and P_1 throughout the game.*

□

In our construction, an *individual recipient* of a message cannot gain any knowledge about the policy other than the number of clauses that he or she satisfies and whatever can be inferred from that fact. Therefore our construction provides clausal policy privacy if the users do not collude. We do not, however, provide coalition resistance for policy privacy. This is because we try to provide policy privacy even when the message can be decrypted. Recall that in some cases coalitions of users can pool in their credentials to decrypt ciphertexts. In that case, the coalition will be able to gain knowledge about the policy. Our construction, therefore, is secure under the following (non-CCA2 and non-coalition-resistant) model.

Definition 5 (C-P-IND-CPA) *A PEAPOD system has* Clausal Policy Indistinguishability against Chosen Plaintext Attack (C-P-IND-CPA) *if no PPT adversary can win the following game against the challenger $\mathcal{C}$ with probability non-negligibly greater than 1/2.*

The game is the same as that in Definition 4, except that the adversary $\mathcal{A}$ (1) may not ask any honest users for decrypting any retrieved ciphertext, and (2) may corrupt at most 1 honest user. □

We discuss the implications of coalition and inference attacks on confidentiality and policy privacy in Section 4.

Credential privacy Credential privacy protects the privacy of the recipient from the sender. Specifically, credential privacy hides from the sender the credentials that the recipient possesses and thus uses when attempting to decrypt the sender's encrypted messages, even if the sender is able to observe the system. More formally, what can be observed by a sender (or a coalition of senders) constitutes to the protocol view of the sender (or the coalition), which includes secrets, inputs and randomness of the sender (or the coalition) during the executions of the encryption algorithm and the deposit protocol, protocol transcripts of all deposit and retrieval protocol runs. A PEAPOD system has credential privacy if given any protocol view of the system, an adversary cannot decide non-negligibly better than random guessing if a recipient has a certain attribute for any recipient and attribute in the system.

Achieving credential privacy is generally a big challenge for interactive schemes such as policy-based access control. Trust negotiation allows users to specify "release policies" that explicitly allow certain credentials to be "leaked" [15]. Nevertheless, credential privacy comes as a natural guarantee in PEAPOD due to the existence of a server as an intermediary. In particular, the offline nature of message delivery from the sender to the recipient through an intermediate server ensures that the sender never directly interacts with recipients. In fact, the sender of an encrypted message might never know who has attempted to decrypt the message.

3. Our Construction

For simplicity, we describe the construction of PEAPOD in stages. First we show that it is straightforward to construct PEAPOD using an adapted version of SELS as a building block if only single-attribute policies are allowed. Then, we demonstrate a construction that supports policies based on a conjunction of attributes and their negations (a conjunctive clause). Finally, we describe how complex policies in the form of a disjunction of conjunctive clauses can also be supported. In the remainder of the paper we will use the terms "conjunctive clauses" and "clauses" interchangeably.

3.1. PEAPOD for single-attribute policies

Earlier in this paper we reviewed how SELS achieves its goal of securely sending email messages to a list of subscribers. If we think of a mailing list as the list of users who possess a certain attribute, then SELS can immediately be used to securely disseminate data to users who possess that attribute. Therefore, if all polices contain only a single attribute, then all we need to do to construct PEAPOD is to instantiate a SELS list for each of the attributes in the system.

The list managers who manage list membership in these SELS lists now collectively act like the Certification Authority (CA) in PEAPOD, which is the entity responsible for authenticating users according to their attributes before giving out credentials to them. The list servers become the PEAPOD Server which handles deposits of encrypted messages. However, instead of actively broadcasting the encrypted messages to the intended recipients as in SELS, the PEAPOD Server waits for individual retrieval requests from users in the system.

Now the SELS instantiation in Figure 1 can be thought of as being associated with an attribute a and re-interpreted as follows. Key y is the *encryption public key* for encrypting data under attribute a for deposit. Secret s is the server's *transformation secret* with respect to attribute a for transforming ciphertexts before storage. Secrets s_A, s_B are the *re-encryption secrets* the server keeps for re-encrypting stored ciphertexts to Alice and Bob, the two users who possess attribute a, respectively. Secrets x_A and x_B are respectively the decryption secrets of Alice and Bob that enable them to decrypt retrieved ciphertexts and recover the plaintext message encrypted under the attribute a.

We call the above construction Simple-PEAPOD, which supports only single-attribute policies by adapting SELS in a rather straightforward fashion. We do not analyze the security of Simple-PEAPOD as it is only an intermediate step towards our full construction. However, it is easy to see that Simple-PEAPOD is confidential as the security of SELS implies an adversary who does not possess an attribute cannot decrypt a ciphertext encrypted under that attribute's public key.

3.2. PEAPOD for single-clause policies

We now demonstrate how to construct a PEAPOD system that supports policies that are conjunctions of attributes and their negations (a single "conjunctive clause"). We present the construction by walking through an example. This example is generic enough to illustrate the actual construction, which we refer to as Clausal-PEAPOD.

The example We continue with the matchmaking scenario we have been using. Recall that there are five attributes: {males, college graduates, in their 30's, smokers, pet lovers} in the system. Let us assume the criteria user Alice has for her perfect partner is a "male in his 30's who does not smoke." Her policy is thus

$$\langle \checkmark, *, \checkmark, \times, * \rangle. \quad (1)$$

Bob is a 32 year-old non-smoking gentleman who loves pets and has no college degree. Bob thus satisfies Alice's criteria and hence her policy. In the following steps, Alice encrypts and deposits at the server her protected profile (in form of a message string) under her policy, which is then retrieved and decrypted by Bob.

Encryption To leave a message M for her potential perfect partners, Alice encrypts M with a secure symmetric encryption under key $k \in \mathbb{Z}_p$ generated uniformly at random. Let the resulting ciphertext be ψ. Alice then randomly picks a "sub-key" for each $\checkmark$-attribute in Eqn. (1) such that the sub-keys multiply to $k \pmod p$, i.e. $k_1, k_3 \in \mathbb{Z}_p$ such that $k_1 k_3 \equiv k \pmod p$. She also picks $r_4 \in \mathbb{Z}_p$ for the $\times$-attribute uniformly at random. For the remaining $*$-attributes, she picks the identity element 1. These result in the tuple:

$$\langle k_1, 1, k_3, r_4, 1 \rangle. \quad (2)$$

The basic idea behind these values is that Bob will eventually receive the subset of values that correspond to his attributes. As long as Bob does not possess any of the $\times$-attributes, these values can be multiplied together to retrieve the key – the 1's corresponding to the $*$-attributes do not affect the overall product thereby preserving the key, and the random numbers for the $\times$-attribute will "destroy" the key if used. What remains to be shown is how Bob is *forced* to multiply all these values to maintain the requisite security and privacy properties of the message and the policy.

To each entry in Eqn. (2), Alice applies the Elgamal encryption algorithm $\mathcal{E}$ under the public key y_i for the corresponding attribute, resulting in tuple below,[7] which is then sent along with the ciphertext ψ to the server for deposit.

$$\langle \{k_1\}^{\mathcal{E}}_{y_1},\ \{1\}^{\mathcal{E}}_{y_2},\ \{k_3\}^{\mathcal{E}}_{y_3}, \{r_4\}^{\mathcal{E}}_{y_4},\ \{1\}^{\mathcal{E}}_{y_5} \rangle. \quad (3)$$

Deposit Upon receiving the tuple in Eqn. (3) and ψ from Alice, the server applies the transformation function $\mathcal{T}$ on the entries in Eqn. (3) using the transformation secrets for the corresponding attributes, thereby obtaining $\langle \{\{k_1\}^{\mathcal{E}}_{y_1}\}^{\mathcal{T}}_{s_1}, \{\{1\}^{\mathcal{E}}_{y_2}\}^{\mathcal{T}}_{s_2}, \{\{k_3\}^{\mathcal{E}}_{y_3}\}^{\mathcal{T}}_{s_3}, \{\{r_4\}^{\mathcal{E}}_{y_4}\}^{\mathcal{T}}_{s_4}, \{\{1\}^{\mathcal{E}}_{y_5}\}^{\mathcal{T}}_{s_5} \rangle$, which, as explained earlier, is equivalent to the Elgamal encryptions of entries in Eqn. (2) under $\tilde{y}_i$'s, i.e.

$$\langle \{k_1\}^{\mathcal{E}}_{\tilde{y}_1},\ \{1\}^{\mathcal{E}}_{\tilde{y}_2},\ \{k_3\}^{\mathcal{E}}_{\tilde{y}_3},\ \{r_4\}^{\mathcal{E}}_{\tilde{y}_4},\ \{1\}^{\mathcal{E}}_{\tilde{y}_5} \rangle. \quad (4)$$

[7] We adopt the notation of $\{x\}^{\mathcal{A}}_k$ to mean the output of a (possibly randomized) algorithm $\mathcal{A}$ under key k when given the input x.

The server stores in its database the tuple in Eqn. (4) along with ψ.

Retrieval Now Bob comes to the server and asks for the ciphertext Alice just deposited. The server reads the tuple in Eqn. (4) from its database and then operates on it as follows. It first strips off the entries that correspond to the attributes Bob does not have (the second and fourth in this case),[8] resulting in the tuple

$$\langle \{k_1\}^{\mathcal{E}}_{\tilde{y}_1}, \{k_3\}^{\mathcal{E}}_{\tilde{y}_3}, \{1\}^{\mathcal{E}}_{\tilde{y}_5} \rangle. \quad (5)$$

For each of the entries in Eqn. (5), the server randomly picks a blinding factor from $\mathbb{Z}_p$, such that the product of these factors equals 1 $(\mathrm{mod}\ p)$, i.e. $b_1, b_3, b_5 \in \mathbb{Z}_p$ such that $b_1 b_3 b_5 \equiv 1 (\mathrm{mod}\ p)$. The server then applies Elgamal encryption $\mathcal{E}$ on these blinding factors under the corresponding public keys $\tilde{y}_i$'s, resulting in the tuple $\langle \{b_1\}^{\mathcal{E}}_{\tilde{y}_1}, \{b_3\}^{\mathcal{E}}_{\tilde{y}_3}, \{b_5\}^{\mathcal{E}}_{\tilde{y}_5} \rangle$, which is then "homomorphically" multiplied into Eqn. (5) in a pairwise fashion, giving rise to:

$$\langle \{k_1\}^{\mathcal{E}}_{\tilde{y}_1} \otimes \{b_1\}^{\mathcal{E}}_{\tilde{y}_1},\ \{k_3\}^{\mathcal{E}}_{\tilde{y}_3} \otimes \{b_3\}^{\mathcal{E}}_{\tilde{y}_3},\ \{1\}^{\mathcal{E}}_{\tilde{y}_5} \otimes \{b_5\}^{\mathcal{E}}_{\tilde{y}_5} \rangle, \quad (6)$$

which is equivalent to:

$$\langle \{k_1 b_1\}^{\mathcal{E}}_{\tilde{y}_1},\ \{k_3 b_3\}^{\mathcal{E}}_{\tilde{y}_3}, \{b_5\}^{\mathcal{E}}_{\tilde{y}_5} \rangle. \quad (7)$$

These blinding factors ensure that the receiver is forced to multiply all values, and does not have access to any of the individual values. Finally, the server uses its re-encryption secret-keys for Bob ($s_{1,B}$, $s_{3,B}$ and $s_{5,B}$) and re-encrypts the ciphertexts into the tuple below, which is sent[9] to Bob along with the ciphertext ψ:

$$\langle \{\{k_1 b_1\}^{\mathcal{E}}_{\tilde{y}_1}\}^{\mathcal{R}}_{s_{1,B}},\ \{\{k_3 b_3\}^{\mathcal{E}}_{\tilde{y}_3}\}^{\mathcal{R}}_{s_{3,B}},\ \{\{b_5\}^{\mathcal{E}}_{\tilde{y}_5}\}^{\mathcal{R}}_{s_{5,B}} \rangle. \quad (8)$$

Decryption Now Bob uses his decryption secret-keys ($x_{1,B}$, $x_{3,B}$ and $x_{5,B}$) to decrypt the entries in Eqn. (8) in order to get the tuple:

$$\langle k_1 b_1, k_3 b_3, b_5 \rangle \quad (9)$$

Bob multiplies all the entries in Eqn. (9) together, which gives him back k, the symmetric key for decrypting the ciphertext ψ, thus enabling him to recover the original message M. Note that if Bob was actually a smoker and thus possessed a forbidden attribute, Eqn (9) would look like $\langle k_1 b'_1, k_3 b'_3, r_4 b'_4, b'_5 \rangle$. The existence of r_4 would make it impossible for Bob to recover k.

[8] The server knows the set of attributes every user in the system has as it is involved in the procedure during which a user obtains a credential for an attribute from the CA. For details, consult [23].

[9] The size of the overall ciphertext leaks information on the number of attributes Bob possesses. To cope with this, we assume that the Server packs the overall ciphertext into a fixed size with bogus sub-ciphertexts of which the bogusness is only identifiable by Bob.

We assume the existence of a mechanism for the users in the system to tell if they satisfy the policies associated with the ciphertexts and thus have correctly decrypted those ciphertexts.[10]

We state without proof that Clausal-PEAPOD is confidential in the C-IND-CPA-RUCA model, enjoys policy privacy in the C-P-IND-CPA model, and also guarantees credential privacy. The proofs can easily be inferred as special cases from the theorems for our full construction of PEAPOD, which supports complex policies.

3.3. Complete version of PEAPOD

PEAPOD for a single conjunctive clause can be generalized to support complex policies that contain more than one clause (interpreted as a disjunction of multiple conjunctive clauses). To recover an encrypted message, it suffices to reconstruct the associated symmetric key k. The sender can encrypt the message for each of the clauses in a complex policy, such that the same k is used for each clause. Note that for each clause, new sub-keys are generated for k. In this way, anyone who satisfies at least one clause will be able to recover k, and thus the encrypted message.

However, care must be taken in order not to leak (too much) information about the policy. First, if a ciphertext contains only those legitimate clauses in the policy, anyone would be able to tell how many clauses are there in the policy that was used in the encryption. To keep the actual size of the policy secret, one could pad the policy with "bogus" clauses so that ciphertexts will always be of the same size, irrespective of the actual size of the policy. This can easily be done by assuming a random symmetric key $k' \neq k$ in the bogus clauses. We assume that the system picks a parameter n, which is the maximum number of clauses for each policy, and requires that all ciphertexts deposited have policies padded to n clauses.

The sender should also compute a random permutation of the original clauses and the bogus ones (call this a "shuffle"). This makes the recipients learn only the number of clauses they satisfy. All the remaining clauses may or may not be legitimate clauses. For example, if the clauses were not shuffled and Bob satisfied only the third clause, then Bob would know for sure that the first and second clauses are not bogus clauses and hence the policy contains *at least three clauses*. If the clauses are shuffled, however, Bob cannot infer whether the first and second clauses are bogus and only knows that there is *at least one clause* in the policy.

We call such a construction Full-PEAPOD.

Security theorems We now list the security theorems for Full-PEAPOD. Their proofs can be found in the Appendix.

[10]For example, we can require the senders to first encode the plaintext messages, such as padding them with a string of zeros at the front.

Theorem 1 (Confidentiality) *If the DDH assumption holds for $\mathbb{Z}_p^*$, then Full-PEAPOD has* Ciphertext Indistinguishability against Chosen Plaintext Attack and Restricted User Coalition Attack (C-IND-CPA-RUCA).

Theorem 2 (Policy privacy) *If the DDH assumption holds for $\mathbb{Z}_p^*$, then Full-PEAPOD has* Clausal Policy Indistinguishability against Chosen Plaintext Attack (C-P-IND-CPA).

Theorem 3 (Credential privacy) *If the DDH assumption holds for $\mathbb{Z}_p^*$, then PEAPOD has Credential Indistinguishability.*

4. Discussion

Trust relationships In PEAPOD, the procedure for distributing credentials is split between the CA and the Server, and each credential for a user is generated with the honest cooperation between these two entities. This trust assumption is similar to that of SELS since our system uses a modified version of SELS as a building block. Neither the Server nor the CA can decrypt messages intended for a particular recipient. If they collude, however, the credentials of each recipient can be computed and the Server can decrypt all messages in the system and infer their policies. We observe that in current approaches for Hidden Credentials based on IBE, the PKG already has the ability to decrypt all the messages in the system. PEAPOD splits this trusted functionality between the CA and the Server and therefore the security of published messages (e.g., at a bulletin-board) with respect to the trusted entities is at least as good as that in previous schemes. We now examine what damage a corrupt CA or Server can do individually.

A corrupt CA by itself cannot decrypt messages in the system without having the correct credentials. Therefore, a CA can simply issue unlimited credentials to itself. However, we assume that the Server does not collude with the CA and will only allow legitimate users in the system (verified using, e.g., PKI [3]) to obtain the credential for an attribute. Therefore, the CA cannot pose as a user in the system. The policy privacy with respect to the CA is maintained under these circumstances, which can only be broken with the help of a malicious Server.

Analogous to a corrupt CA, a corrupt Server cannot decrypt any messages without having the correct credentials. The Server cannot issue itself any credentials since this requires the cooperation of the CA, which we assume does not collude with the Server. Nonetheless, the server can be malicious when serving retrieval requests by not following the algorithm, causing ciphertexts to be decrypted into "garbage" even by intended recipients. In this kind of denial of service attack (DoS), a recipient who cannot decrypt a ciphertext has no way to tell if he or she does not satisfy

the associated policy, the server is malicious, or the sender encrypted garbage. An interesting area for future work is to devise a protocol that will detect a misbehaving Server and let the receiver determine whether he or she satisfied the policy.

Our system provides policy privacy with respect to the Server and clausal policy-indistinguishability for all recipients. If the Server and recipient collude, however, both the recipient and the Server can learn the policy of the sender. Therefore, some amount of trust must be placed in the Server to not collude with users. One option is to involve the CA in the protocol so that cooperation with the CA would also be needed to break policy privacy. Involving the CA, however, raises issues such as performance, offline vs. online CAs, and so on.

Inference attacks on policy privacy We now examine the implications of the information that a recipient Bob learns about the policy. As discussed earlier, if Bob satisfies $\ell \geq 0$ clauses, he can infer only that the policy is one among a set of policies for which he satisfies ℓ clauses. We call this set Bob's "inference set," within which all policies are indistinguishable to Bob. Since Bob's goal is to figure out the exact nature of the ℓ clauses that he does satisfy, Bob can focus on policies with ℓ clauses and try to infer what they may be. We will refer to this set of policies as Bob's "inference set restricted to ℓ clauses." The size of the inference set will vary for different receivers. In systems that support only monotonic boolean formulae for policies, consider the trivial example: Bob has only one credential "`is a smoker`." If he is able to decrypt the message, he can infer that the policy contains the clause "`is a smoker`." The size of the inference set (restricted to the satisfied clause) is 1. PEAPOD, however, provides much better guarantees since it supports non-monotonic boolean policies. For example, the inference set in PEAPOD would also include a vast number of other possibilities such as "`not in their 30's`," "`not a college graduate` $\wedge$ `is a smoker`," and so on.

Coalition attacks We now discuss some of the coalition attacks where several recipients "pool in" their credentials. As a first line of defense, users cannot simply share the pieces of their key shares together since they are forced to compute the product of shares over their entire set of attributes. Therefore the simple pooling in of credentials will not succeed in general. Consider the case when two colluding receivers obtain k_1k_2 and k_2k_3 respectively, and let $k = k_1k_2k_3$. In such as case k cannot be retrieved since the individual pieces k_1, k_2 and k_3 are not known to the colluding users. In certain cases, however, these products can be combined meaningfully. Say $k = k_1k_2$. Bob may possess attribute a_1 but not a_2 and Charlie may possess attribute a_2 but not a_1. Bob will thus recover k_1 and Charlie will recover k_2. They can collude to expose k. Therefore, even though coalition attacks are not straightforward, PEAPOD is not secure against coalition attacks in general. Furthermore, different recipients can compute the intersection of their inference sets for a particular message and try to narrow down the set of possible policies for that message.

Sender and receiver anonymity In PEAPOD, any party can retrieve from the Server ciphertexts for any user, say, Bob. Confidentiality guarantees that only Bob can decrypt the ciphertexts and therefore it does not matter if the Server gives away the re-encrypted ciphertexts to anyone unauthenticated or even anonymous. As a consequence, anyone could have retrieved Bob's ciphertexts and this allows Bob to deny that he requested the ciphertext. Therefore PEAPOD supports a weak form of anonymity for receivers called *plausible deniability*, which means that no user can be implicated with overwhelming probability. A detailed discussion on how receivers can protect their anonymity is outside the scope of this paper, but in general users can access the server using an anonymizing network such as Tor [13]. This approach, however, opens up the possibility of DoS attacks where malicious users can bog the Server down with repeated requests for ciphertexts. Authentication of receivers can alleviate this problem, but most authentication schemes will destroy the property of plausible deniability. The system can employ *deniable authentication* [14, 22, 29, 26] to provide plausible deniability,[11] while maintaining DoS resistance. The server may also choose to employ authentication only when it is under DoS attack, and forego authentication under normal operation.

Senders can post messages anonymously to the Server, and therefore the authenticity (the identity of the sender) of messages cannot be guaranteed unless the sender digitally signs the message. If the sender desires anonymity, he or she may choose to use a group signature scheme [10, 4, 5] to maintain anonymity (within the group) while guaranteeing to the receiver that the message was signed by somebody in the group.

Dynamism We assume a static set of attributes that remains unchanged throughout the lifetime of the system. It would be useful, however, to support the addition and removal of attributes, both to the system, and to the individual users' attribute sets. While it is quite possible to adapt our system to support dynamism without losing confidentiality, maintaining policy privacy in a dynamic environment is not straightforward. For example, a user who does not satisfy a policy with respect to attribute set A might satisfy the same policy after A has been updated to some different attribute set A'. By studying the difference between A and A', the

[11] With such an authentication scheme, the Server cannot prove to any third party that Bob requested the ciphertext, even though the Server knows that Bob requested it.

user can possibly infer the nature of some attributes in the policy. One simple countermeasure would be to prevent users from retrieving two versions of the same ciphertext under different attribute sets. We plan to address the effects of dynamism on policy privacy and suitable countermeasures in future work.

Efficiency It is worth looking at the message expansion imposed by Full-PEAPOD as a consequence of achieving the various desirable properties on top of sole confidentiality. Let m be the number of established attributes and n be the maximum number of clauses in a policy. Also let λ be the security parameter, which equals the bit-length of the size of the group $\mathbb{Z}_p^*$. Observe that a ciphertext has a $(2\lambda mn)$-bit space overhead in addition to the symmetric encryption of the plaintext message. When the size of the plaintext message is big enough, the expansion is insignificant. For example, in the case when $n = 8$, $m = 50$ and $\lambda = 1024$, the overhead is 100 kilobytes. Both depositing a ciphertext and retrieving a ciphertext have space and time complexities of $O(mn)$ if we ignore the symmetric encryption and decryption. In particular, when Bob retrieves a message, the Server must perform $O(m_B n)$ operations, where m_B is the number of attributes that Bob has been issued credentials for. In the worst case, $m_B = m$ and the Server has to perform mn re-encryption and homomorphic encryption steps. We stress that Full-PEAPOD is very scalable in terms of number of users in the system because the time and space complexities of all algorithms are independent of the number of users.

The main computational bottleneck is the ciphertext retrieval step at the Server. To handle a retrieval request, the Server has to do 3 modular exponentiations and 4 modular multiplications per attribute for each clause.[12] On a reasonably fast server machine such as Sun Fire T2000 [33], this step takes less than 0.1s if $n = 8$ and $m = 50$.[13] Therefore, the system can handle at least 600 message retrieval per minute, which would be sufficient for organizational networks. For example, a college or a university could be easily serviced without noticeable delays. Furthermore, since the retrieval operation is naturally parallelizable, Field Programmable Gate Arrays (FPGAs) can be used to significantly reduce the amount of time for retrieving messages. As FPGAs get faster and cheaper, one could fit several re-encryption engines onto a single FPGA. For example, using current technology, one could fit seven 1024-bit exp. architecture due to Blum and Paar [7] in Xilinx's Virtex-5 XC5VLX330.[14]

In future work we plan to address various tradeoffs between policy privacy and the system parameters to improve scalability with respect to the number of attributes n in the system. For example, users can pick a subset of attributes within which the policies are private. This would reduce the overhead, and yet still provide sufficient policy privacy in systems with a large number of attributes.

5. Conclusions

We present PEAPOD, a system where publishers can disseminate information securely to multiple possible recipients using attribute-based policies. Unlike previous approaches that require online interaction or knowledge of the recipient's identity or pseudonym beforehand, in our approach messages are securely deposited at a server for offline retrieval by multiple possible recipients unknown to the sender. Users can decrypt these messages if and only if their credentials satisfy the publisher's policy, and the publisher does not gain any knowledge of the users "Hidden Credentials." Our system uses SELS as a building block to solve the problem of shared decryption keys between users with the same attribute and extends this technique with homomorphic encryption to provide message confidentiality and clausal policy-indistinguishability against *all* recipients, intended or not, and complete policy-indistinguishability against the server. In the context of the problem of securely publishing messages to multiple possible recipients, the policy privacy properties provided by PEAPOD surpass those provided by all previously known Hidden Credential schemes. Unlike previous approaches, PEAPOD is also able to efficiently support non-monotonic boolean policies, i.e., policies that contain negations of attributes.

6. Acknowledgments

We would like to thank Alexander Iliev, Chris Masone, Peter Johnson, Nikos Triandopoulos, and Nihal D'Cunha for their helpful comments.

References

[1] M. Abe and K. Suzuki. M+1-st price auction using homomorphic encryption. In D. Naccache and P. Paillier, editors, *Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 115–124. Springer, 2002.

[12] Specifically, 1 modular exponentiation (exp.) and 1 modular multiplication (mul.) for re-encryption, 2 exp. and 1 mul. for encrypting the blinding factor and 2 mul. for homomorphic operation.

[13] This machine can do 17,023 1024-bit DSA-signing per second [32]. The 0.1s estimate is based on two assumptions: one DSA-signing takes the same time as one exp.; and four mul. take the same time as one exp. These are conservative assumptions and therefore the estimate is a loose upper bound. In practice, one should be able to achieve much better performance.

[14] Blum and Paar's architecture occupies 6633 Configurable Logic Blocks (CLBS) and can do one exp. in 11.95ms. XC5VLX330 has 51840 slices.

[2] A. Acquisti. Receipt-free homomorphic elections and write-in ballots. Cryptology ePrint Archive, Report 2004/105, 2004. http://eprint.iacr.org/.

[3] C. Adams and S.Farrell. Internet X.509 Public Key Infrastructure Certificate Management Protocols. Internet Engineering Task Force: RFC 2510, 1999.

[4] G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A practical and provably secure coalition-resistant group signature scheme. In M. Bellare, editor, *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2000.

[5] M. Bellare, H. Shi, and C. Zhang. Foundations of group signatures: The case of dynamic groups. In A. Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 136–153. Springer, 2005.

[6] M. Blaze, G. Bleumer, and M. Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*, pages 127–144, 1998.

[7] T. Blum and C. Paar. High-radix montgomery modular exponentiation on reconfigurable hardware. *IEEE Trans. Comput.*, 50(7):759–764, 2001.

[8] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *Lecture Notes in Computer Science*, 2139:213–229, 2001.

[9] R. W. Bradshaw, J. E. Holt, and K. E. Seamons. Concealing complex policies with hidden credentials. In *Eleventh ACM Conference on Computer and Communications Security, Washington, DC*, pages 146–157, oct 2004.

[10] D. Chaum and E. van Heyst. Group signatures. In *EUROCRYPT*, pages 257–265, 1991.

[11] X. Chen, B. Lee, and K. Kim. Receipt-free electronic auction schemes using homomorphic encryption. In J. I. Lim and D. H. Lee, editors, *ICISC*, volume 2971 of *Lecture Notes in Computer Science*, pages 259–273. Springer, 2003.

[12] I. Damgård and M. Jurik. A length-flexible threshold cryptosystem with applications. In R. Safavi-Naini and J. Seberry, editors, *ACISP*, volume 2727 of *Lecture Notes in Computer Science*, pages 350–364. Springer, 2003.

[13] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The Second-Generation Onion Router. In *Usenix Security*, aug 2004.

[14] C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.

[15] K. Frikken, J. Li, and M. Atallah. Trust negotiation with hidden credentials, hidden policies, and policy cycles. In *13th Annual Network and Distributed System Security Symposium*, pages 157–172, feb 2006.

[16] P. Golle, M. Jakobsson, A. Juels, and P. F. Syverson. Universal re-encryption for mixnets. In T. Okamoto, editor, *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 163–178. Springer, 2004.

[17] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98, New York, NY, USA, 2006. ACM Press.

[18] J. Groth. A verifiable secret shuffle of homomorphic encryptions. In Y. Desmedt, editor, *Public Key Cryptography*, volume 2567 of *Lecture Notes in Computer Science*, pages 145–160. Springer, 2003.

[19] M. Hirt and K. Sako. Efficient receipt-free voting based on homomorphic encryption. In *EUROCRYPT*, pages 539–556, 2000.

[20] J. E. Holt, R. W. Bradshaw, K. E. Seamons, and H. Orman. Hidden credentials. In *2nd ACM Workshop on Privacy in the Electronic Society, Washington, DC*, pages 1–8, oct 2003.

[21] A. Ivan and Y. Dodis. Proxy cryptography revisited. In *NDSS*. The Internet Society, 2003.

[22] J. Katz. Efficient and non-malleable proofs of plaintext knowledge and applications. In E. Biham, editor, *EUROCRYPT*, volume 2656 of *Lecture Notes in Computer Science*, pages 211–228. Springer, 2003.

[23] H. Khurana, A. J. Slagell, and R. Bonilla. Sels: a secure e-mail list service. In H. Haddad, L. M. Liebrock, A. Omicini, and R. L. Wainwright, editors, *SAC*, pages 306–313. ACM, 2005.

[24] J. Li and N. Li. Policy-hiding access control in open environment. In *Proceedings of ACM Symposium on Principles of Distributed Computing (PODC)*, pages 29–38, jul 2005.

[25] H. Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In C.-S. Laih, editor, *ASIACRYPT*, volume 2894 of *Lecture Notes in Computer Science*, pages 416–433. Springer, 2003.

[26] M. Naor. Deniable ring authentication. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 481–498. Springer-Verlag, 2002.

[27] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[28] G. Poupard and J. Stern. Fair encryption of rsa keys. In *EUROCRYPT*, pages 172–189, 2000.

[29] M. D. Raimondo and R. Gennaro. New approaches for deniable authentication. In V. Atluri, C. Meadows, and A. Juels, editors, *ACM Conference on Computer and Communications Security*, pages 112–121. ACM, 2005.

[30] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. In R. DeMillo, D. Dobkin, A. Jones, and R. Lipton, editors, *Foundations of Secure Computation*, pages 169–180. Academic Press, 1978.

[31] A. Sahai and B. Waters. Fuzzy identity based encryption. in advances. In *Advances in Cryptology – Eurocrypt volume 3494 of LNCS*, pages 457–473, 2005.

[32] SunMicrosystems. Sun fire T1000 and T2000 Servers Benchmarks, 2006. http://www.sun.com/servers/coolthreads/t1000/benchmarks.jsp#j.

[33] SunMicrosystems. Sun Fire T2000 Server, 2006. http://www.sun.com/servers/coolthreads/t2000/.

[34] K. Suzuki and M. Yokoo. Secure generalized vickrey auction using homomorphic encryption. In R. N. Wright, editor, *Financial Cryptography*, volume 2742 of *Lecture Notes in Computer Science*, pages 239–249. Springer, 2003.

[35] W. H. Winsborough and N. Li. Towards practical automated trust negotiation. In *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, pages 92–103, June 2002.

[36] W. H. Winsborough and N. Li. Safety in automated trust negotiation. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy*, pages 147–160, Oakland, CA, May 2004. IEEE Press.

[37] W. H. Winsborough, K. E. Seamons, and V. E. Jones. Automated trust negotiation. In *DARPA Information Survivability Conference and Exposition (DISCEX)*, pages 88–102, Jan. 2000.

[38] T. Yu and M. Winslett. A unified scheme for resource protection in automated trust negotiation. In *Proceedings of the IEEE Symposium on Security and privacy*, pages 110–122, May 2003.

[39] T. Yu, M. Winslett, and K. E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. Inf. Syst. Secur.*, 6(1):1–42, 2003.

A. Security Proofs

A.1. Lemmas

We need the following lemmas for the proof of Full-PEAPOD's confidentiality and policy privacy. The implication of Lemma 1 with respect to Full-PEAPOD is that if a randomness is added to the set of sub-keys of the symmetric key and then the whole set is blinded, the knowledge of such a set is useless in recovering the symmetric key. The lemma can easily be generalized so that there could be more than one randomness and they can be at arbitrary positions. Lemma 2 says the blinding operation on a tuple retains only the product of the entries, but individual blinded values reveal nothing about the underlying values. Recall that in Full-PEAPOD, the plaintext behind each of the Elgamal ciphertexts is either a sub-key, a random value or the identity element. With this lemma, these three types of values are no longer distinguishable after blinding.

We write $a \xleftarrow{\$} S$ to mean that a is drawn from set S using fresh coin-flips according to a uniform distribution.

Lemma 1 *$\forall m \in \mathbb{N}$, $\forall x_1, x_2, \ldots, x_m \in \mathbb{Z}_p^*$, the random variables $X = (b_1 x_1,\ b_2 x_2,\ \ldots,\ b_m x_m,\ (\Pi_{i=1}^m b_i)^{-1} r)$, where $b_1, b_2, \ldots, b_m, r \xleftarrow{\$} \mathbb{Z}_p^*$, and $R = (r_1\ r_2,\ \ldots,\ r_m,\ r_{m+1})$, where $r_1, r_2, \ldots, r_{m+1} \xleftarrow{\$} \mathbb{Z}_p^*$, have the same distribution.*

Lemma 2 *$\forall m \in \mathbb{N}$, $\forall x_1, x_2, \ldots, x_{m+1} \in \mathbb{Z}_p^*$, the random variables $X = (b_1 x_1,\ b_2 x_2,\ \ldots,\ b_m x_m,\ (\Pi_{i=1}^m b_i)^{-1} x_{m+1})$, where $b_1, b_2, \ldots, b_m \xleftarrow{\$} \mathbb{Z}_p^*$, and $R = (r_1,\ r_2,\ \ldots,\ r_m,\ \Pi_{i=1}^{m+1} x_i \cdot (\Pi_{i=1}^m)^{-1} r_i)$, where $r_1, r_2, \ldots, r_m \xleftarrow{\$} \mathbb{Z}_p^*$, have the same distribution.*

The proofs for both lemmas are straightforward and are thus omitted.

A.2. Proofs

Proof 1 (Theorem 1) *(Sketch.)* Recall that a ciphertext returned by the encryption algorithm executed by a sender in Full-PEAPOD consists of the symmetric encryption ψ of the plaintext message M under a randomly chosen key k and an array of Elgamal encryption of either a sub-key, a random value or the identity element. The Elgamal ciphertext at the i-th row and j-th column in the array is an encryption under the public key of attribute j. The underlying plaintext is a sub-key, a random value or the identity element if the j-th attribute is respectively a required, forbidden or irrelevant attribute for the i-th clause in the policy.

Now, assume there exists a PPT adversary $\mathcal{A}$ who can tell which message among M_0 and M_1 was used in the encryption that resulted in the challenge ciphertext C^* with probability non-negligibly greater than 1/2. The semantic security of the symmetric encryption implies that $\mathcal{A}$ knows the symmetric key k. The knowledge of k implies that there is at least one clause in the challenge policy P^* such that $\mathcal{A}$ knows the product of all the sub-keys for that clause. We claim that this implies that $\mathcal{A}$ knows all the individual sub-keys in the product for that clause. Such knowledge means that $\mathcal{A}$ knows the set K^* of Elgamal decryption keys to the encryption of all those sub-keys, as otherwise Elgamal encryption would not be IND-CPA secure, contradicting to the assumption that the DDH problem is hard in $\mathbb{Z}_p^*$.

We need to consider both cases when $\mathcal{A}$ corrupted the Server and when $\mathcal{A}$ did not. In the former case, the model requires that $\mathcal{A}$ did not corrupt any of the users. The Server itself does not know any of the Elgamal decryption keys. The Server knows all the transformation secret keys, but without the help of the CA or any user in the system, the transformation secret keys have statistical zero-knowledge on any of the Elgamal decryption keys. This contradicts to the fact that $\mathcal{A}$ knows K^*.

In the latter case, $\mathcal{A}$ might have corrupted some subset of users. However, Restriction $1'$ guarantees that any coalition of corrupted users does not collectively satisfy the challenge policy P^*, which means that for any union of corrupted users' attribute set and any clause in P^*, the union does not satisfy the clause, which is so either because at least one required attribute is missing from the union, or one forbidden attribute is present in the union. The former possibility contradicts to the fact that $\mathcal{A}$ knows K^*. The latter possibility also contradicts to the very same fact, due to Lemma 1.

Therefore, there does not exist a PPT adversary $\mathcal{A}$ who can win the game with probability non-negligibly greater than 1/2. Finally, we note that the above is only a proof sketch. In a full proof, one would have to simulate the deposit and retrieval queries, which could be achieved by following the protocol specification. The contradiction could be derived by embedding a DDH problem instance into one of the array entries of Elgamal ciphertexts in the challenge ciphertext. Such a challenge ciphertext can be simulated correctly without the knowledge of sub-key behind that particular Elgamal ciphertext. □

Proof 2 (Theorem 2) *(Sketch.)* Observe that during encryption, the policy affects only the array of plaintexts that are to be Elgamal-encrypted. The symmetric encryption of the message is totally independent of the policy. To prove that Full-PEAPOD has Clausal Policy Indistinguishability, it suffices to show the two ciphertexts resulted from any message encrypted with two different policies with the same number of satisfying clauses with respect to an adversary are statistically indistinguishable, thereby leaving the adversary no chance in making a correct guess on which policy was used any better than pure guessing, i.e. it can't win with probability non-negligibly greater than 1/2.

Assume there exists a PPT adversary $\mathcal{A}$ who can win the game with probability non-negligibly greater than 1/2. Let's first consider the possibility that $\mathcal{A}$ corrupted the Server during the game. In such a case, $\mathcal{A}$ did not corrupt any of the users or otherwise could not have won the game. Since the server does not possess the decryption keys of any of the attributes, all the Elgamal ciphertexts within a deposited ciphertext have computational zero knowledge in their underlying plaintexts, the knowledge of which is required to tell whether an attribute is required, forbidden or irrelevant for each clause. Thus, a ciphertext deposited at the server has zero knowledge in the policy under which it was created, meaning that an adversary who corrupted the server in order to attack policy privacy of Full-PEAPOD cannot do any better then pure guessing which one of the two challenge policies was used.

Therefore, $\mathcal{A}$ must have won without corrupting the Server, in which case $\mathcal{A}$ might have corrupted up to one honest user. If $\mathcal{A}$ did not corrupt any user, then by arguments similar to the above, a ciphertext deposited at the server has zero knowledge in the policy under which it was created and thus $\mathcal{A}$ could only win with probability negligibly greater than 1/2. The only possibility left is thus that $\mathcal{A}$ corrupted exactly one honest user. Note that the game specification implies that this single corrupted user satisfies the same number of clauses in both challenge policies. The following derives a contradiction.

The ciphertexts the adversary retrieves from the server have all gone through the blinding step through the homomorphic operation on the individual Elgamal ciphertexts. Lemma 2 implies that the decryption of any of these Elgamal ciphertexts could have been a sub-key, a random value or the identity element before blinding and the adversary has statistically zero knowledge to tell which is the case. This means any attribute in any clause in the policy could be a $\checkmark$-attribute, $\times$-attribute, or $*$-attribute. The only information a retrieved ciphertext carry is thus the product of the plaintexts for each clause, which is either equal to the symmetric key if the clause is satisfied by the adversary, or a group element distributed over $\mathbb{Z}_p^*$ uniformly at random if the clause is not satisfied by the adversary or if it is a bogus one, due to Lemma 1. Finally, due to the uniformly random shuffling executed at the encryption step, two polices with the same number of satisfying clauses have the same statistical distribution of the plaintext products. Therefore, the two challenge policies are indistinguishable to $\mathcal{A}$ and thus $\mathcal{A}$ could not have won the game with probability non-negligibly greater than 1/2 in this case.

Combining all cases leads to the conclusion that there does not exist a PPT adversary $\mathcal{A}$ who can win the game with probability non-negligibly greater than 1/2. □

Proof 3 (Theorem 3) *(Sketch.)* As discussed, Full-PEAPOD is an offline system in which a sender only interacts once with the *Server* during a deposit and the receiver only interacts once with the *Server* during a retrieval. Receivers never contact with senders during retrieval or decryption. The transcripts for retrieval protocol runs consist of retrieved ciphertexts. The size of the ciphertext retrieved by any user is independent of the attribute set possessed by the user. Furthermore, by arguments similar to the proof sketch for policy privacy, the IND-CPA security of Elgamal encryption implies that the sender who does not have any credentials to decrypt a retrieved ciphertext is unable to learn non-negligible information from the Elgamal ciphertexts about their underlying plaintexts. The result follows. □