Delegation in Predicate Encryption Supporting Disjunctive Queries

Dongdong Sun, Colin Boyd, and Juan Manuel González Nieto

Information Security Institute, Queensland University of Technology,
GPO Box 2434, Brisbane QLD 4001, Australia
{dongdong.sun,c.boyd,j.gonzaleznieto}@qut.edu.au

Abstract. Predicate encryption has an advantage over traditional public-key or identity-based encryption, since predicate encryption systems provide more flexible control over access to encrypted data. We focus on delegation capabilities in predicate systems. More specifically, we investigate delegatable encryption systems supporting disjunctive predicate evaluations. We present formal security definitions of delegatable predicate encryption and provide the first delegatable predicate encryption scheme which supports disjunctive predicate evaluations in the public-key setting. We analyze the security of the proposed system and give a security proof.

Keywords: Predicate Encryption, Delegation, Disjunction

1 Introduction

Traditional public-key encryption systems have been used and investigated for decades. In those systems, a user creates a public and private key pair, and the message can be encrypted with the public key and recovered with the corresponding private key. This is sufficient for many applications, where a user knows the identity of the recipient in advance. However, in other applications, a user may require more flexible control over the access to the encrypted data. Predicate encryption is a new primitive which can achieve such flexibility. In predicate-based encryption, private keys are associated with predicates f and ciphertexts are bound to attributes I. A ciphertext corresponding to I can be decrypted by a private key corresponding to f if and only if $f(I) = 1$.

There are a number of contributions to predicate encryption in the literature [1–5]. Those systems can evaluate various predicates over the ciphertext. The most expressive one is the KSW [4] construction, which can support a wide class of predicates including disjunctive predicate evaluations. The system is well suited for many applications. For example, in a bank, for the sake of security, each internal file is labeled with an attribute and both file and attribute are encrypted with the system public key. We assume that the general manager, who is the top authority, holds the master secret key. He can issue a secret key associated with a predicate, the disjunctive predicate in our case. Staffs have different privileges to access files. People in IT department are allowed to read technical

files, and those in customer service department are allowed to read all customer related files. On top of that, there are some files which the general manager wants to make ensure that everyone should read (e.g., bank policy). For each staff, the general manager will encode the predicate, which is associated with attributes in disjunctive form. For example, an IT staff member may receive a key corresponding to predicate ($x =$ **"Bank Policy"**) $\vee$ ($x =$ **"Tech Manual A"**) $\vee$ ($x =$ **"Tech Manual B"**), so he can decrypt the files whenever the attribute in the file matches any one of three attributes in the predicate. Moreover, a crucial property of the disjunctive predicate, namely *predicate indistinguishability*, is applicable to our specific application. The *predicate indistinguishability* means if an attribute in the ciphertext matches the disjunctive predicate, it is computationally infeasible to find the position of a match, only the existence of a match in at least one position is known [4]. For example, each profile of a customer is encrypted with the customer's account number (i.e., the attribute). The profile includes the general information about the customer, e.g., name, address, deposit amount, recent transactions. Now the general manager needs some statistics on those profiles. He computes a secret key associated with the predicate ($x =$ **"CustomerA's Account Number"**) $\vee$ ($x =$ **"CustomerB's Account Number"**) $\vee$ ($x =$ **"CustomerC's Account Number"**) $\vee$ ($x =$ **"CustomerD's Account Number"**), and sends the key to the staff who is responsible for doing statistics. With that key, the staff can decrypt all profiles related to the predicate. We assume that he knows the predicate, since in the public-key settings, secret keys may reveal some information about the encoded predicate. However, upon decryption, he cannot tell which customer's account number is related to the decrypted profile due to the property of *predicate indistinguishability*. For the security consideration, the property is certainly desirable if the staff is not deem to be able to identify the relationship between the account number and the profile.

The above-mentioned scenario is sufficient for a small company. If there are hundreds of thousands of staffs, it is not efficient to let only one authority (i.e., the general manager) to compute all keys. Delegation is an attractive solution to the issue. Generally speaking, this mechanism can spread the computational tasks to other entities. We stick with our bank example. Now, instead of computing all secret keys the general manager delegates the tasks to his subordinates, i.e., managers. More specifically, the general manager first computes a key associated with a predicate, e.g., ($x =$ **"Bank Policy"** $\vee$? $\vee$?), where ? is a delegatable field. He sends this key to one of the managers, e.g., a technical manager. The technical manager then creates a key for the predicate ($x =$ **"Bank Policy"**) $\vee$ ($x =$ **"Tech Manual A"**) $\vee$ ($x =$ **"Tech Manual B"**) and gives it to one of the staffs in his department. In this scenario, we have some restrictions. Only the general manager holds the master secret key so that he can initialize the key with some attributes and ensure no managers can modify those fixed attributes (the managers can only fill in the delegatable fields). We also assume the general manager and other managers are authorities who can access any files. The staffs on the lowest level can only obtain keys without delegatable fields.

The managers must ensure they give the non-delegatable keys to the staffs. One may argue that we can give the master secret key to all managers, because they have right to decrypt any files. However, as mentioned before, the general manager wants to ensure that some files (e.g., bank policies) should be decrypted by anyone in the bank. By holding the master key, he can always initialize the key with some attributes, e.g., ($x =$ **"Bank Policy"** $\vee$? $\vee$?), which ensures that anyone can access the **"Bank Policy"**. Now, all keys can be computed in an efficient way with the help of the managers. We will present a security system which can handle the above mentioned situations in this paper.

1.1 Our Results

In this paper, we present formal security definitions of delegatable predicate encryption supporting disjunctive predicate evaluations. We also present a delegatable predicate encryption scheme in the public-key setting. Our scheme is based on the KSW [4] construction but, unlike their scheme, we achieve delegation. A formal security proof of our scheme is also provided. The required security assumptions have all been introduced in prior works. Our systems are based on a group whose order is the product of four primes.

1.2 Related Work

Identity-Based Encryption and Attribute-Based Encryption. To address the issue of certificate overhead in the PKI system, Shamir [6] introduced the notion of identity-based cryptography. The first practical ID-based encryption (IBE) was proposed by Boneh *et al* [7]. Thereafter, many efficient ID-based schemes have been proposed [8–10]. Because of the efficiency of the system, ID-based cryptography is now flourishing in the cryptographic community. In attribute-based encryption (ABE) [11–13], a user can receive a secret key associated with an access control policy over the attributes of an encrypted data.

Predicate-Based Encryption. Recently, Boneh and Waters [3] proposed the first encryption system possessing the properties of conjunctive, subset and range queries in the public-key setting. Concurrent work by Shi *et al.* [1] also achieves a similar function. However, they achieve match-revealing instead of match-concealing. How to construct a system supporting disjunctive predicate was left as an open problem until the work by Katz *et al* [4]. The KSW system can be regarded as a significant improvement in the theory of predicate-based encryptions. Their work also implies all the results of the BW construction [3]. Based on the KSW system, Shen *et al* [5] proposed a similar system in the symmetric-key setting. Their system achieves predicate privacy as well as data privacy.

Delegation in Predicate Encryption. The notion of delegation was first introduced in this context by Horwitz and Lynn [14]. Subsequently, a number of works address delegation issues in Hierarchical Identity-Based Encryption (HIBE) [15, 16]. The most related context of delegation in predicate encryption

appeared in the work of Shi and Waters [2]. They constructed a delegatable predicate encryption supporting conjunctive queries. However, to construct a system supporting disjunctive queries was left as an open problem, which motivates us to investigate the new system in this paper.

2 Definitions

We describe definitions in our specific settings, where the class of predicates is $P = \{OR_{x_1,\ldots,x_l}(x) | (x_1, \ldots, x_l) \in \mathbb{Z}_N^l\}$ such that $OR_{x_1,\ldots,x_l}(x) = 1$ iff $x = x_1$ or $x = x_2$ or $\ldots$ or $x = x_l$. The above-mentioned disjunctive predicate evaluation is based on inner product predicate evaluation which was specified in the work of Katz *et al* [4]. When we have a message M for the attribute $w \in \mathbb{Z}_N$ to encrypt, we set $\boldsymbol{w} := (w^l \mod N, \ldots, w^0 \mod N) \in \mathbb{Z}_N^n$, where $n = l + 1$, then encrypt $\boldsymbol{w}$ with M. To compute a secret key for predicate $OR_{x_1,\ldots,x_l}(x)$, we compute a polynomial $p(x) = \prod_{i\in[l]}(x - x_i) \mod N$ and set $\boldsymbol{p} := (a_l, ..., a_0)$, where $a_l, \ldots, a_0$ are the coefficients of $p(x)$. We then compute the secret key on $\boldsymbol{p}$. The actual evaluation is based on the class of predicates $F = \{f_{\boldsymbol{x}} | \boldsymbol{x} \in \mathbb{Z}_N^n\}$, where $f_{\boldsymbol{x}}(\boldsymbol{y}) = 1$ iff $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = 0 \mod N$. As shown by Katz *et al* [4], $OR_{x_1,\ldots,x_l}(w) = 1$ iff $f_{\boldsymbol{p}}(\boldsymbol{w}) = 1$, which forms the bases for our systems. In all our definitions, we let Ω denote a finite set of $\mathbb{Z}_N$ and $\Omega_? = \Omega \cup \{?\}$, where ? is a delegatable field.

Definition 1. *A* Delegatable Predicate Encryption Scheme *comprises of the following algorithms:*

- $\mathsf{Setup}(1^\lambda)$ *The* Setup *algorithm takes as input a security parameter* 1^λ *and outputs a public key* PK *and a master secret key* MSK*.*
- $\mathsf{Encrypt}(PK, w, M)$ *The* Encrypt *algorithm takes as input a public key* PK*, an attribute* $w \in \Omega$*, and a message* M *in some associated message space. It outputs a ciphertext* C*.*
- $\mathsf{GenKey}(MSK, X)$ *The* GenKey *algorithm takes as input a master secret key* MSK *and an attribute vector* $X = (x_1, \ldots, x_l) \in \Omega_?^l$ *(*l *is fixed in the system), which corresponds to predicate* $OR_X(x)$*. It outputs a secret key* SK_X *for evaluating* $OR_X(x)$ *over a ciphertext* C*.*
- $\mathsf{Delegate}(SK_X, \hat{x})$ *The* Delegate *algorithm takes as input a secret key* SK_X *for* $OR_X(x)$ *and an attribute* $\hat{x}$*. It outputs a delegated secret key* $SK_{X'}$ *for the predicate* $OR_{X'}(x)$*, where* X' *is obtained by fixing one of the delegatable fields of* X *with the attribute* $\hat{x}$*.*
- $\mathsf{Query}\ (SK_X, C)$ *The* Query *algorithm takes as input a secret key* SK_X *and a ciphertext* C*. It outputs either a message* M *or the distinguished symbol* $\perp$*.*

Correctness. *We require the following correctness property. For all* λ*, all* $(w, M) \in \Omega \times \mathcal{M}$*, and all* $OR_X \in P$*, let* $(PK, MSK) \xleftarrow{R} \mathsf{Setup}(1^\lambda)$*,* $C \xleftarrow{R} Encrypt(PK, w, M)$*, and* $SK_X \xleftarrow{R} \mathsf{GenKey}(MSK, X)$*.*

- *If* $OR_X(w) = 1$ *then* $\mathsf{Query}(SK_X, C) = M$*.*
- *If* $OR_X(w) = 0$ *then* $\mathsf{Query}(SK_X, C) = \perp$ *with all but negligible probability.*

The same property holds, if SK_X is computed from Delegate *algorithm.*

Selective Security. We will prove the selective security of our scheme. Our security definition is similar to that of KSW [4], except that there are extra *create delegated secret key* queries. The formal definition of selective security is provided below.

Definition 2. *A delegatable predicate encryption scheme is* ***selective secure*** *if for all PPT adversaries $\mathcal{A}$, the advantage* $\mathsf{Adv}_{\mathcal{A}}$ *of $\mathcal{A}$ in the following game is a negligible function of security parameter λ:*

Setup(1^λ). *A challenger $\mathcal{C}$ runs the* Setup *algorithm to generate public key PK and master secret key MSK, a public key value N is given to $\mathcal{A}$.*

Init. *$\mathcal{A}$ outputs vectors $\boldsymbol{a}, \boldsymbol{b} \in \Omega^n$, which correspond to attributes A, $B \in \Omega$, and is then given PK.*

Query phase 1. *$\mathcal{A}$ adaptively makes a polynomial number of the following queries:*

- *Create secret key. $\mathcal{A}$ requests a secret key for a vector $\boldsymbol{p} \in \Omega^n$ corresponding to predicate $OR_X(x)$, where $X = (x_1, \ldots, x_l) \in \Omega^l$. $\mathcal{C}$ creates the secret key and gives it to $\mathcal{A}$.*
- *Create delegated secret key. $\mathcal{A}$ requests a secret key for a vector $\boldsymbol{p} \in \Omega^n$ corresponding to predicate $OR_X(x)$, where $X = (x_1, \ldots, x_l) \in \Omega^l$. $\mathcal{C}$ chooses a random number i, where $1 \leq i \leq l$ and creates the secret key for $OR_{X_i}(x)$, where $X_i = (x_1, \ldots, x_i)$. Using that key as the parent key, $\mathcal{C}$ creates the key for $OR_X(x)$ in delegatable way.*

Any key revealed to $\mathcal{A}$ are subject to the restriction such that $OR_X(A) = OR_X(B)$, which is equivalent to $f_{\boldsymbol{p}}(\boldsymbol{a}) = f_{\boldsymbol{p}}(\boldsymbol{b})$.

Challenge. *$\mathcal{A}$ outputs two equal-length messages M_0 and M_1. If there is any $OR_X(A) = OR_X(B) = 1$, then it is required that $M_0 = M_1$. $\mathcal{C}$ flips a random coin b. If $b = 0$ then $\mathcal{A}$ is given $C =$* $\mathsf{Encrypt}(M_b, A)$ *and if $b = 1$ then $\mathcal{A}$ is given $C =$* $\mathsf{Encrypt}(M_b, B)$.

Query phase 2. *Repeat the* **Query phase 1** *subject to the restrictions as before.*

Guess. *$\mathcal{A}$ outputs a guess b' of b, and succeeds if $b' = b$.*

The advantage of $\mathcal{A}$ is defined to be $Adv_{\mathcal{A}} = |Pr[b = b'] - 1/2|$.

3 Background on Pairings and Complexity Assumptions

We review the notions about groups of composite order and bilinear maps. Let $\mathcal{G}$ be an algorithm that takes as input a security parameter 1^λ and outputs a tuple $(p, q, r, s, \mathbb{G}, \mathbb{G}_T, e)$ where p, q, r, s are distinct primes, $\mathbb{G}$ and $\mathbb{G}_T$ are two cyclic groups of order $N = pqrs$, and e is a non-degenerate bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ satisfying the following properties:

1. Bilinearity: $\forall u, v \in \mathbb{G}, \forall a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$.
2. Non-degeneracy: if g generates $\mathbb{G}$ then $e(g, g)$ generates $\mathbb{G}_T$

We assume the group computation in $\mathbb{G}$ and $\mathbb{G}_T$, as well as the bilinear map e, are all computable in time polynomial in λ. Furthermore, we assume that the description of $\mathbb{G}$ and $\mathbb{G}_T$ includes generators of $\mathbb{G}$ and $\mathbb{G}_T$ respectively.

We will use the notation $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_s$ to denote the respective subgroups of order p, order q, order r and order s of $\mathbb{G}$ and we will use the notation $\mathbb{G}_{T,p}, \mathbb{G}_{T,q}, \mathbb{G}_{T,r}, \mathbb{G}_{T,s}$ to denote the respective subgroups of order p, order q, order r and order s of $\mathbb{G}_T$. There is a crucial property in composite order bilinear map: if $h_p \in \mathbb{G}_p$ and $h_q \in \mathbb{G}_q$, then $e(h_p, h_q) = 1$. This property holds whenever e is applied to elements in any two distinct subgroups.

3.1 The composite 3-party Diffie-Hellman assumption

The composite 3-party Diffie-Hellman assumption (C3DH) was first introduced by Boneh and Waters [3]. For a given generator $\mathcal{G}$ define the following distribution $P(1^\lambda)$:
$(p, q, r, s, \mathbb{G}, \mathbb{G}_T, e) \xleftarrow{R} \mathcal{G}(1^\lambda),\ N \leftarrow pqrs,\ g_p \xleftarrow{R} \mathbb{G}_p,\ g_q \xleftarrow{R} \mathbb{G}_q,\ g_r \xleftarrow{R} \mathbb{G}_r,\ g_s \xleftarrow{R} \mathbb{G}_s$
$R_1, R_2, R_3 \xleftarrow{R} \mathbb{G}_q, \quad a, b, c \xleftarrow{R} \mathbb{Z}_N$
$\bar{Z} \leftarrow ((N, \mathbb{G}, \mathbb{G}_T, e),\ g_q,\ g_p,\ g_r,\ g_s,\ g_p^a,\ g_p^b,\ g_p^{ab} R_1,\ g_p^{abc} R_2),\ T \leftarrow g_p^c R_3$
$Output(\bar{Z}, T)$

For an algorithm $\mathcal{A}$, define $\mathcal{A}$'s advantage in solving the composite 3-party Diffie-Hellman problem for $\mathcal{G}$ as:

$$C3DHAdv_{\mathcal{G},\mathcal{A}}(1^\lambda) := |Pr[\mathcal{A}(\bar{Z}, T) = 1] - Pr[\mathcal{A}(\bar{Z}, R) = 1]|$$

where $(\bar{Z}, T) \xleftarrow{R} P(1^\lambda)$ and $R \xleftarrow{R} \mathbb{G}_{pq}$.

Definition 3. *We say that $\mathcal{G}$ satisfies the composite 3-party Diffie-Hellman assumption (C3DH) if for any polynomial time algorithm $\mathcal{A}$, and its advantage $C3DHAdv_{\mathcal{G},\mathcal{A}}(1^\lambda)$ is negligible in the security parameter λ.*

The assumption is formed around the intuition that it is hard to test for Diffie-Hellman tuples in the subgroup $\mathbb{G}_p$ if the elements have a random $\mathbb{G}_q$ subgroup component.

3.2 Other Assumptions

Assumption 1. For a given generator $\mathcal{G}$ define the following distribution $P(1^\lambda)$:
$(p, q, r, s, \mathbb{G}, \mathbb{G}_T, e) \xleftarrow{R} \mathcal{G}(1^\lambda),\ N \leftarrow pqrs,\ g_p \xleftarrow{R} \mathbb{G}_p,\ g_q \xleftarrow{R} \mathbb{G}_q,\ g_r \xleftarrow{R} \mathbb{G}_r,\ g_s \xleftarrow{R} \mathbb{G}_s$
$Q_1, Q_2 \xleftarrow{R} \mathbb{G}_q, \quad R_1, R_2, R_3 \xleftarrow{R} \mathbb{G}_r, \quad a, b, c \xleftarrow{R} \mathbb{Z}_p$
$\bar{Z} \leftarrow ((N, \mathbb{G}, \mathbb{G}_T, e),\ g_p,\ g_r,\ g_s,\ g_q R_1,\ g_p^b,\ g_p^{b^2},\ g_p^a g_q,\ g_p^{ab} Q_1,\ g_p^c,\ g_p^{bc} Q_2 R_2)$
$\beta \xleftarrow{R} \mathbb{Z}_q,\ T \leftarrow g_p^{b^2 c} g_q^\beta R_3$
$Output(\bar{Z}, T)$

For an algorithm $\mathcal{A}$, define $\mathcal{A}$'s advantage in the above experiment for $\mathcal{G}$ as:

$$A1Adv_{\mathcal{G},\mathcal{A}}(1^\lambda) := |Pr[\mathcal{A}(\bar{Z}, g_p^{b^2c}R_3) = 1] - Pr[\mathcal{A}(\bar{Z}, g_p^{b^2c}g_q^{\beta}R_3) = 1]|$$

Definition 4. *We say that $\mathcal{G}$ satisfies the assumption 1 if for any polynomial time algorithm $\mathcal{A}$, its advantage $A1Adv_{\mathcal{G},\mathcal{A}}(1^\lambda)$ is negligible in the security parameter λ.*

Assumption 2. For a given generator $\mathcal{G}$ define the following distribution $P(1^\lambda)$:
$(p, q, r, s, \mathbb{G}, \mathbb{G}_T, e) \xleftarrow{R} \mathcal{G}(1^\lambda),\ N \leftarrow pqrs,\ g_p \xleftarrow{R} \mathbb{G}_p,\ g_q \xleftarrow{R} \mathbb{G}_q,\ g_r \xleftarrow{R} \mathbb{G}_r,\ g_s \xleftarrow{R} \mathbb{G}_s$
$h \xleftarrow{R} \mathbb{G}_p, \quad Q_1, Q_2 \xleftarrow{R} \mathbb{G}_q, \quad c, \gamma \xleftarrow{R} \mathbb{Z}_q$
$\bar{Z} \leftarrow ((N, \mathbb{G}, \mathbb{G}_T, e), g_p, g_q, g_r, g_s, h, g_p^c, h^c Q_1, g_p^{\gamma} Q_2, e(g_p, h)^{\gamma}),\ T \leftarrow e(g_p, h)^{\gamma c}$
$Output(\bar{Z}, T)$

For an algorithm $\mathcal{A}$, define $\mathcal{A}$'s advantage in the above experiment for $\mathcal{G}$ as:

$$A2Adv_{\mathcal{G},\mathcal{A}}(1^\lambda) := |Pr[\mathcal{A}(\bar{Z}, T) = 1] - Pr[\mathcal{A}(\bar{Z}, R) = 1]|$$

where $(\bar{Z}, T) \xleftarrow{R} P(1^\lambda)$ and $R \xleftarrow{R} \mathbb{G}_T$.

Definition 5. *We say that $\mathcal{G}$ satisfies the assumption 2 if for any polynomial time algorithm $\mathcal{A}$, its advantage $A2Adv_{\mathcal{G},\mathcal{A}}(1^\lambda)$ is negligible in the security parameter λ.*

The above two assumptions imply the hardness of finding any non-trivial factor of N. They are proven to hold in the generic group by Katz *et al* [4].

4 Our Scheme

We construct our delegatable predicate encryption scheme by extending the KSW system [4]. Our scheme possesses all the properties of the KSW system. In our construction, we require that the fixed attributes associated with the disjunctive predicate cannot be modified by anyone. Only the delegatable fields can be filled in. More specifically, it is hard to obtain the parent key by carrying out computations on the child keys. On the technical level, our construction is based on the following observations. Assume that we have a key for the predicate $((x = a) \vee (x = b) \vee \ ?\ )$, where ? is a delegatable field. The predicate can be rewritten in polynomial equation $p(x) = (x - a) \cdot (x - b)$. If we fill in $x = c$ in the third field, then the equation is $p'(x) = (x - a) \cdot (x - b) \cdot (x - c) = p(x) \cdot x + p(x) \cdot (-c)$. As specified in **Section 2**, we know that the coefficients of the above polynomials will be encoded into secret keys. Our secret key SK consists of two components, a decryption key component DK and a delegation component DL. Assume coefficients of $p(x)$ is encoded in secret key SK_Z. We can shift elements in DK_Z to obtain the elements associated with $p(x) \cdot x$ and raise elements in DL_Z to the power of $(-c)$ to obtain the elements associated with $p(x) \cdot (-c)$. We combine DL_Z and DK_Z to obtain keys corresponding to $p'(x)$. More details can be found in the following scheme.

$\mathsf{Setup}(1^\lambda)$ The setup algorithm first picks random large primes p, q, r, s and creates groups $\mathbb{G}$ and $\mathbb{G}_T$ of composite order $N = pqrs$. It then computes g_p, g_q, g_r and g_s as generators of group $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r$ and $\mathbb{G}_s$, respectively. Next, it randomly chooses $R_{1,i}, R_{2,i} \in \mathbb{G}_r$ and $h_1, h_2 \in \mathbb{G}_p$, where $i = 1$ to n. It also chooses $R_0 \in \mathbb{G}_r, \gamma \in \mathbb{Z}_p$ and $h \in \mathbb{G}_p$ at random, and sets $l = n - 1$, which is the size of the attribute vector. It publishes $(N = pqrs, \mathbb{G}, \mathbb{G}_T)$ and the values $PK = (\ g_p,\ \ g_r,\ \ g_s,\ \ Q = g_q \cdot R_0,\ \ P = e(g_p, h)^\gamma,\ \ l,\ \ \{H_{1,i} = h_1 \cdot R_{1,i},\ H_{2,i} = h_2 \cdot R_{2,i}\}_{i=1}^n\ )$. The master secret key MSK is $(p, q, r, s, g_q, h^{-\gamma}, h_1, h_2)$.

$\mathsf{Encrypt}(PK, w \in \Omega, M \in \mathcal{M} \subseteq \mathbb{G}_\mathcal{T})$ Assume that $\Omega \subseteq \mathbb{Z}_N$. $\mathcal{M}$ is some efficiently-recognizable subgroup of $\mathbb{G}_T$. To encrypt a message M for the attribute w, the algorithm computes $(w_1 = w^0 \mod N, \ldots, w_n = w^{n-1} \mod N)$. Then, it chooses random $\delta, \alpha, \beta \in \mathbb{Z}_N$ and $R_{3,i}, R_{4,i} \in \mathbb{G}_r$ for $i = 1$ to n. The ciphertext is

$$C = \big(C' = M \cdot P^\delta,\ C_0 = g_p^\delta,\ \{C_{1,i} = H_{1,i}^\delta \cdot Q^{\alpha \cdot w_i} \cdot R_{3,i},\ C_{2,i} = H_{2,i}^\delta \cdot Q^{\beta \cdot w_i} \cdot R_{4,i}\}_{i=1}^n\big).$$

$\mathsf{GenKey}(MSK, X \in \Omega_?^l)$ Assume $\Omega_? = \Omega \cup \{?\}$, where ? is a delegatable field. Let $X = (x_1, \ldots, x_l) \in \Omega_?^l$. $\mathcal{I}(X)$ denotes the set of all indices u where $x_u \in \Omega$. This algorithm encodes X as a univariate polynomial $p(x) = \prod_{u \in \mathcal{I}(X)}(x - x_u) \mod N$, and then extends the equation to obtain $p(x) = a_{I+1}x^I + \cdots + a_1 x^0 \mod N$, where $a_{I+1}, \ldots, a_1$ are the coefficients of the resulting polynomial and I is the number of all fixed fields. We set $a_i = 0$ for $i > I + 1$. The secret key for X consists of two parts: a decryption key component DK and a delegation component DL.

- DK: Choose random $r_{1,i}, r_{2,i} \in \mathbb{Z}_p$ and $Y, Y_{1,i}, Y_{2,i} \in \mathbb{G}_s$ for $i = 1$ to n $(n = l + 1)$, random $R_5 \in \mathbb{G}_r$, random $f_1, f_2 \in \mathbb{Z}_q$ and random $Q_6 \in \mathbb{G}_q$. The decryption key is

$$DK = \begin{pmatrix} K = R_5 \cdot Q_6 \cdot h^{-\gamma} \cdot \prod_{i=1}^n h_1^{-r_{1,i}} \cdot h_2^{-r_{2,i}} \cdot Y, \\ \{K_{1,i} = g_p^{r_{1,i}} \cdot g_q^{f_1 \cdot a_i} \cdot Y_{1,i},\ K_{2,i} = g_p^{r_{2,i}} \cdot g_q^{f_2 \cdot a_i} \cdot Y_{2,i}\}_{i=1}^n \end{pmatrix}$$

- DL: Let w denotes the number of delegatable fields. The algorithm computes w parallel components. They have similar structures with the decryption key component. The main difference is that only the decryption key component contains the master secret $h^{-\gamma}$. Let $\mathcal{W} = \{1, \ldots, w\}$. For each $v \in \mathcal{W}$, for $i = 1$ to n, choose random $r_{1,i,v}, r_{2,i,v} \in \mathbb{Z}_p$ and $Y_{1,i,v}, Y_{2,i,v} \in \mathbb{G}_s$. For each $v \in \mathcal{W}$, choose random $R_{5,v} \in \mathbb{G}_r$, random $Y_v \in \mathbb{G}_s$ and random $Q_{6,v} \in \mathbb{G}_q$. The delegation component is

$$DL_v = \begin{pmatrix} L_v = R_{5,v} \cdot Q_{6,v} \cdot \prod_{i=1}^n h_1^{-r_{1,i,v}} \cdot h_2^{-r_{2,i,v}} \cdot Y_v, \\ \{L_{1,i,v} = g_p^{r_{1,i,v}} \cdot g_q^{f_1 \cdot a_i} \cdot Y_{1,i,v},\ L_{2,i,v} = g_p^{r_{2,i,v}} \cdot g_q^{f_2 \cdot a_i} \cdot Y_{2,i,v}\}_{i=1}^n \end{pmatrix}$$

  where $v \in \mathcal{W}$.

$\mathsf{Delegate}(SK_{X\in\Omega_\gamma^l}, \hat{x}\in\Omega)$ Given a secret key for X and an attribute $\hat{x}$, this algorithm fixes one of the delegatable fields of X with $\hat{x}$ to obtain X', and computes the secret key for X'. Clearly, if we can perform delegation on one field, then we can perform delegation on multiple fields. If there is no delegatable field, the algorithm simply aborts.

Step 1: Let (DK, DL) denote the secret key for X with w delegatable fields. Pick a random $\mu \in \mathbb{Z}_N$ and rerandomize the w^{th} delegation component DL_w by raising every element in DL_w to μ:

$$\hat{DL} = \left(\hat{L} = L_w^{\mu},\ \{\hat{L}_{1,i} = L_{1,i,w}^{\mu},\ \hat{L}_{2,i} = L_{2,i,w}^{\mu}\}_{i=1}^{n}\right)$$

Step 2: Multiply the decryption key component DK by $\hat{DL}$:

$$\hat{DK} = \left(\ \hat{K} = K\cdot\hat{L},\ \{\hat{K}_{1,i} = K_{1,i}\cdot\hat{L}_{1,i},\ \hat{K}_{2,i} = K_{2,i}\cdot\hat{L}_{2,i}\}_{i=1}^{n}\right)$$

Step 3: Multiply the delegation component DL_v by $\hat{DL}$ for all $v \in W'$, where $W' = \{1,\ldots,w-1\}$. For all $v \in \mathcal{W}'$, we compute following:

$$\hat{DL_v} = \left(\hat{L}_v = L_v\cdot\hat{L},\ \{\hat{L}_{1,i,v} = L_{1,i,v}\cdot\hat{L}_{1,i},\ \hat{L}_{2,i,v} = L_{2,i,v}\cdot\hat{L}_{2,i}\}_{i=1}^{n}\right)$$

Step 4: Perform a circular shift on the randomized decryption key component $\hat{DK}$:

$$pDK = \begin{pmatrix} pK = \hat{K},\ pK_{1,1} = \hat{K}_{1,n},\ pK_{2,1} = \hat{K}_{2,n}, \\ \{pK_{1,i} = \hat{K}_{1,i-1}, pK_{2,i} = \hat{K}_{2,i-1}\}_{i=2}^{n} \end{pmatrix}$$

Step 5: Compute decryption key component DK' for X'. DK' is computed from two components: 1) pDK, the shifted decryption key component of secret key for X. 2) $\hat{DL}_1$, the randomized delegation component for X. First randomly select $Y', Y'_{1,i}, Y'_{2,i} \in \mathbb{G}_s$ for $i = 1$ to n, then raise every element in $\hat{DL}_1$ to $-\hat{x}$, output the following DK':

$$DK' = \begin{pmatrix} K' = \hat{L}_1^{-\hat{x}}\cdot pK\cdot Y', \\ \{K'_{1,i} = \hat{L}_{1,i,1}^{-\hat{x}}\cdot pK_{1,i}\cdot Y'_{1,i},\quad K'_{2,i} = \hat{L}_{2,i,1}^{-\hat{x}}\cdot pK_{2,i}\cdot Y'_{2,i}\}_{i=1}^{n} \end{pmatrix}$$

Step 6: Compute delegation component DL' of X'. DL' is computed from the randomized delegation component $\hat{DL}_v$ for all $v \in W'$, where $W' = \{1,\ldots,w-1\}$. Generally speaking, each time the algorithm performs some computations on two of the $\hat{DL}_v$ components to obtain a DL' component. The resulting DL' will consists of $w-1$ components. For example, choose $\hat{DL}_1$ and $\hat{DL}_2$ to compute DL'_1, then choose $\hat{DL}_2$ and $\hat{DL}_3$ to compute DL'_2, etc. To compute the last component DL'_{w-1}, choose $\hat{DL}_{w-1}$ and $\hat{DL}_1$.

Now we describe how to compute on two of $\hat{DL}_v$ components to obtain a DL' component. Assume two components are $\hat{DL}_1$ and $\hat{DL}_2$. First perform a circular shift on $\hat{DL}_1$:

$$pDL'_1 = \begin{pmatrix} pL'_1 = \hat{L}_1, \;\; pL'_{1,1,1} = \hat{L}_{1,n,1}, \;\; pL'_{2,1,1} = \hat{L}_{2,n,1} \\ \{pL'_{1,i,1} = \hat{L}_{1,i-1,1}, \;\; pL'_{2,i,1} = \hat{L}_{2,i-1,1}\}_{i=2}^{n} \end{pmatrix}$$

Next, choose random $Y'', Y''_{1,i}, Y''_{2,i} \in \mathbb{G}_s$ for $i = 1$ to n, then raise every element in $\hat{DL}_2$ to $-\hat{x}$ and output the following DL'_1:

$$DL'_1 = \begin{pmatrix} L'_1 = \hat{L}_2^{-\hat{x}} \cdot pL'_1 \cdot Y'' \\ \{L'_{1,i,1} = \hat{L}_{1,i,2}^{-\hat{x}} \cdot pL'_{1,i,1} \cdot Y''_{1,i}, \;\; L'_{2,i,1} = \hat{L}_{2,i,2}^{-\hat{x}} \cdot pL'_{2,i,1} \cdot Y''_{2,i}\}_{i=1}^{n} \end{pmatrix}$$

In this way, we will be able to compute DL'_v for all $v \in W'$.

NB: If there is only one delegatable field in X, we have one delegation component DL_1. In **Step 1 − 3**, we use DL_1 to randomize DK and DL_1, all other computations are the same.

$\mathsf{Query}(SK_X, C)$ Given a ciphertext and a secret key for X, where $X \in \Omega_?^l$, compute the following:

$$C' \cdot e(C_0, K) \cdot \prod_{i=1}^{n} e(C_{1,i}, K_{1,i}) e(C_{2,i}, K_{2,i})$$

It returns M, if $M \in \mathcal{M}$. Otherwise, it returns an error.

Correctness. To verify that the correctness holds for the scheme, we first show that the secret keys obtained by Delegate algorithm have the same structures as those created directly by GenKey algorithm. We focus on the decryption key component and the delegation component, respectively. The notation is the same as in the scheme. In the decryption components DK, for $i = 1$ to n, the substitutions:

$$r'_{1,i} = \mu \cdot r_{1,i,w} + r_{1,i}, \; r'_{2,i} = \mu \cdot r_{2,i,w} + r_{2,i}$$
$$f'_1 = \mu \cdot f_1 + f_1, \qquad f'_2 = \mu \cdot f_2 + f_2$$

can be rewritten in the Delegate algorithm as

$$DK' = \begin{pmatrix} K' = R'_5 \cdot Q'_6 \cdot h^{-\gamma} \cdot \prod_{i=1}^{n} h_1^{-r'_{1,i}} \cdot h_2^{-r'_{2,i}} \cdot Y', \\ \{K'_{1,i} = g_p^{r'_{1,i}} \cdot g_q^{f'_1 \cdot a'_i} \cdot Y'_{1,i}, \;\; K'_{2,i} = g_p^{r'_{2,i}} \cdot g_q^{f'_2 \cdot a'_i} \cdot Y'_{2,i}\}_{i=1}^{n} \end{pmatrix}$$

where a'_i, for $i = 1$ to n, are coefficients of polynomial after the delegating operations. It is not hard to see that the decryption key component is correctly distributed. Similarly, it can be shown that the delegation component DL is also correctly distributed. Hence, we establish that the secret keys computed from GenKey and Delegate have the correct distributions.

Now we need to prove that the scheme is consistent with respect to the Encrypt, GenKey, Delegate and Query algorithms. Since we have already proven that the secret keys produced by GenKey and Delegate have the correct distributions, we will only consider GenKey for consistency. The rest of the proof is exactly the same argument shown in the work of Katz *et al* [4]. We omit it in here.

5 Proof of Security

Theorem 1. *If $\mathcal{G}$ satisfies the composite 3-party Diffie-Hellman assumption, assumption 1 and 2, then the delegatable predicate encryption scheme is selectively secure.*

Our scheme is secure against adversaries, who are not allowed to perform delegation computation. In our security games, the adversaries can only request secret keys for full attribute vectors, in which there are no delegatable fields. The secret keys can be computed from either the GenKey or the Delegate algorithm. We note that delegated secret keys have some correlations with their parent secret keys. As a result, the distributions of delegated keys differ from freshly generated keys. Our proof is inspired by the technique used in Shi and Waters' work [2], which is called "key indistinguishability". Although delegated keys have correlations with their parent keys, they are computationally indistinguishable from freshly generated keys computed by the GenKey algorithm. This is a crucial step in our proof and simplifies our simulation. Now, instead of answering the adversary's query honestly, the simulator can give a freshly generated key to the adversary whenever the adversary makes a fresh key query or a delegated key query. Intuitively, the notion of "key indistinguishability" relies on the C3DH assumption: if we use a random hiding factor from $\mathbb{G}_r$ to randomize each term in the key, then Decisional Diffie-Hellman problem is hard in the subgroup $\mathbb{G}_p$.

After "key indistinguishability" is established, the rest of the proof will be very similar to the proof in Katz's work [4]. It is not hard to see that by "key indistinguishability", we effectively reduce our security game to Katz's original game, because the delegation key queries are treated as fresh key queries. We provide a proof in the full version of this paper.

6 Conclusion

In this paper, we study delegation techniques in predicate encryption systems, which support disjunctive predicate evaluation. We first provide security definitions for delegatable predicate encryption and then give a delegatable scheme in the public-key setting supporting disjunctive predicate evaluation. In the future, we will focus on delegatable predicate encryption supporting disjunctive and conjunctive normal form. Our aim is to find a delegatable system which can support arbitrary combinations of disjunctive and conjunctive predicate evaluations. The ultimate goal is to achieve delegation in all predicate systems.

References

1. Shi, E., Bethencourt, J., Chan, H.T., Song, D., Perrig, A.: Multi-Dimensional Range Query over Encrypted Data. In: IEEE Symposium on Security and Privacy, pp. 350–364. IEEE Press, Los Alamitos (2007)
2. Shi, E., Waters, B.: Delegating Capability in Predicate Encryption Systems. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 560–578. Springer, Heidelberg (2008)
3. Boneh, D., Waters, B.: Conjunctive, Subset, and Range Queries on Encrypted Data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 535–554. Springer, Heidelberg (2007)
4. Katz, J., Sahai, A., Waters, B.: Predicate Encryption Supporting Disjunctions, Polynomial Equations, and Inner Products. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 146–162. Springer, Heidelberg (2008)
5. Shen, E., Shi, E., Waters, B.: Predicate Privacy in Encryption Systems. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 457–473. Springer, Heidelberg (2009)
6. Shamir, A.: Identity-Based Cryptosystems and Signature Schemes. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO'84. LNCS, vol. 196, pp. 47–53. Springer, Heidelberg (1984)
7. Boneh, D., Franklin, M.: Identity-Based Encryption From the Weil Pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
8. Waters, B.: Efficient Identity-Based Encryption Without Random Oracles. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 114–127. Springer, Heidelberg (2005)
9. Boneh, D., Boyen, X.: Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 223–238. Springer, Heidelberg (2004)
10. Gentry, C.: Practical Identity-Based Encryption Without Random Oracles. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 445–464. Springer, Heidelberg (2006)
11. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-Based Encryption for Fine-grained Access Control of Encrypted Data. In: ACM Conference on Computer and Communication Security 2006, pp. 89–98. ACM, New York (2006)
12. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-Policy Attribute-Based Encryption. In: 2007 IEEE Symposium on Security and Privacy, pp. 321–334. IEEE Press, Los Alamitos (2007)
13. Sahai, A., Waters, B.: Fuzzy Identity-Based Encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)
14. Horwitz, J., Lynn, B.: Towards Hierarchical Identity-Based Encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 466–481. Springer, Heidelberg (2002)
15. Boyen, X., Waters, B.: Anonymous Hierarchical Identity-Based Encryption (Without Random Oracles). In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 290–307. Springer, Heidelberg (2006)
16. Boneh, D., Boyen, X., Goh, E.: Hierarchical Identity Based Encryption with Constant Size Ciphertext. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005)