Delegating Capabilities in Predicate Encryption Systems

Elaine Shi[1] and Brent Waters[2]

[1] Carnegie Mellon University
[2] SRI International

Abstract. In predicate encryption systems, given a capability, one can evaluate one or more predicates on the plaintext encrypted, while all other information about the plaintext remains hidden. We consider the role of delegation in such predicate encryption systems. Suppose Alice has a capability, and she wishes to delegate to Bob a more restrictive capability allowing the decryption of a subset of the information Alice can learn about the plaintext encrypted. We formally define delegation in predicate encryption systems, propose a new security definition for delegation, and give an efficient construction supporting conjunctive queries. The security of our construction can be reduced to the general 3-party Bilinear Diffie-Hellman assumption, and the Bilinear Decisional Diffie-Hellman assumption in composite order bilinear groups.

1 Introduction

In traditional public key encryption a user creates a public and private key pair where the private key is used to decrypt all messages encrypted under that public key. Traditional public key encryption allows "all-or-nothing" access to the encrypted data: the private key owner can decrypt everything; and any party without the private key learns nothing about the data encrypted. Recently, cryptographers have proposed a new notion of encryption called *predicate encryption* [5,9,8,21,1,7,17] (also referred to as *searching on encrypted data*). In predicate encryption, the private key owner can compute a capability that allows one to evaluate predicates on the encrypted data. Capabilities can be regarded as partial decryption keys that release partial information about the plaintext encrypted in a controlled manner.

For example, imagine a network audit log collection effort similar to the one mentioned in the recent work[21]. Suppose different Internet Service Providers (ISPs) contribute network audit logs to an untrusted repository. The audit logs will later be used to study network intrusions and worms. Due to privacy concerns, the ISPs encrypt their audit logs before submitting them to the repository, and only a trusted authority has the private key to search the logs. Now suppose there has been an outbreak of a new network worm. An auditor (e.g., a research institute) has been asked to study the behavior of the worm and propose countermeasures. The auditor can now request the authority for a capability that allows

L. Aceto et al. (Eds.): ICALP 2008, Part II, LNCS 5126, pp. 560–578, 2008.

the decryption of suspicious log entries, e.g., flows satisfying the following characteristic: (PORT $\in [p_1, p_2]$) $\wedge$ (TIME $\in$ LAST MONTH). Meanwhile, the privacy of all other log entries are still preserved.

In predicate encryption, it is often important for a user holding a capability (or a set of capabilities) to generate another capability that is more restrictive than the ones she currently holds. For example, suppose that Carnegie Mellon University has the capability to decrypt all log entries satisfying characteristics of the SQL Slammer worm. Now the university may ask a specific group of researchers to study the SQL Slammer worm originating from an IP address range. To do this, the head of the university can create a more restrictive capability that can decrypt all log entries having the worm characteristic, and originating from this IP range. We say that a predicate encryption system allows for delegation if a user can create capabilities more restrictive than the one she currently owns and if she can do this operation *autonomously*; that is, without interacting with an authority.

In this paper, we study delegation in predicate encryption systems. We propose new security definitions of delegation, and a delegateable predicate encryption scheme supporting conjunctive queries. In the remainder of this section, we first give an overview of related work, and then explain our approach and contributions.

1.1 Related Work

From traditional public-key encryption to predicate encryption. While traditional public-key encryption is sufficient for applications where there is a one to one association between a particular user and a public key, several applications will demand a finer-grained and more expressive decryption capabilities. Shamir [20] provided the first vision for finer-grained encryption systems by introducing the concept of Identity-Based Encryption (IBE). In an IBE system, a party encrypts a message under a particular public key and associates the ciphertext with a given string or "identity". A user can obtain a private key, that is derived from a master secret key, for a particular identity and can use it to decrypt any ciphertext that was encrypted under his identity.

Since the realization of the first Identity-Based Encryption schemes by Boneh and Franklin [6] and Cocks [13], there have been a number of new crypto-systems that provided increasing functionality and expressiveness of decryption capabilities. In Attribute-Based Encryption systems [2,12,15,18,19] a user can receive a private capability that represents complex access control policies over the attributes of an encrypted record. Other encryption systems, including keyword search (or anonymous IBE) [1,5,7,8,9,17,21] systems, allow for a capability holder to evaluate a predicate on the the encrypted data itself and learn nothing more. We henceforth refer to such encryption systems as predicate encryption. Predicate encryption represents a significant breakthrough in the sense that access to the encrypted data is no longer "all-or-nothing"; a user with a predicate capability is able to learn partial information about encrypted data.

Delegation. The concept of delegation was first introduced in this context by Horwitz and Lynn [16] in the form of Hierarchical Identity-Based Encryption (HIBE) [4,14,16]. In an HIBE scheme both private keys and ciphertexts are associated with ordered lists of identities. A user with a given hierarchical identity is able to decrypt any ciphertext where his identity is a prefix of the ciphertext's identity; moreover, a user is able to delegate by creating any other private key with for which his identity is a prefix. For example, a user in charge of the UC Davis domain with a private key for EDU:UCDAVIS can delegate to the computer science department a private key for EDU:UCDAVIS.CS. Since then, the introduction of HIBE the principle of delegation has been applied to other access control systems such as attribute-based encryption systems [15].

1.2 Delegation in Predicate Encryption

In this paper, we examine the problem of delegating capabilities in the more general context of predicate encryption systems [1,5,8,9,17,21,23]. Apart from the aforementioned network audit log example, delegation in predicate encryption can also be useful in other scenarios. For example, suppose Alice has the capability to decrypt all email labeled with "TO:ALICE@YAHOO.COM". If Alice plans to go on vacation over the next two weeks she might want to delegate to her assistant the ability to read all of her incoming emails, but only over this period. To do this, Alice can create a more restrictive capability that can decrypt all such messages that are sent the next two weeks. In another example, suppose Alice's email gateway has the capability to decrypt certain labels of the email and makes forwarding decisions accordingly. For example, emails label as "urgent" by her boss should be sent to her pager; emails from her family should be forwarded to her home computer, etc. The email gateway might want to install similar filtering capabilities on an upstream gateway for cost saving reasons, however, this gateway might be a less trusted device; and Alice may only wish to have the upstream gateway classify emails as "urgent" and "non-urgent" and give preference in forwarding the urgent emails.

Delegation in predicate encryption poses a unique set of challenges; and is typically harder to realize than delegation in Identity-Based Encryption (IBE). This is due to the fact that in an IBE system, a user is able to access an encrypted message if and only if his private key identity matches the ciphertext identity, but the ciphertext identity itself is not hidden. In contrast, predicate encryption systems such as anonymous IBE hides the "identity" of the ciphertext itself. In fact, one can equivalently regard the "identity" as part of the data to be encrypted; and the query predicates are directly evaluated over the encrypted data itself. In practice, this means that it is typically much more difficult to realize delegation in predicate encryption systems. For instance, in anonymous HIBE systems one needs to be careful that the delegation components themselves cannot be used to answer queries.

Another difficulty in building delegation into encryption systems is that previous definitions for security of HIBE appear to be incomplete. In the existing definitions of HIBE security, the attacker plays a game where he receives all

all of his private key queries *directly* from the HIBE authority; however this does not accurately model an adversary's view in a real system. In a real system an adversary might get the private key EDU:UCDAVIS.CS directly from an authority or it might choose to get it from a user with the key EDU:UCDAVIS. In general, private keys received directly from the authority and delegated private keys may have different distribution or forms. For example, in the Gentry and Silverberg [14] and Boneh and Boyen HIBE [3] schemes if a HIBE private key of depth ℓ is received directly from an authority, the authority will create ℓ newly random elements of $\mathbb{Z}_p^*$ in creating the key; however, if the key is generated by another user only one new degree of randomness will be added and the rest will be in common with the previous key. As a result, in the security game, we should not assume in general that delegated keys have the same distribution as keys directly computed by the authority.

Our Approach. We first set out to create a general framework and definitions for delegation in predicate encryption systems. In order to do this we create a general definition that accounts for how predicate capabilities were created. In particular, our definition allows for the adversary to make queries both for capabilities that are created by an authority and for capabilities delegated by users. The adversary may then ask for some subset of these capabilities to be revealed to him.

Using our new definition we set out to realize delegation in an expressive predicate encryption system by extending the Hidden Vector Encryption (HVE) system of Boneh and Waters [8] to allow for delegation. In order to realize security under our new definition we apply two new techniques.

First, we need to make sure that the additional delegation components do not compromise the security of our scheme. We enforce this by "tying" the delegation components of a key to the restrictions of the original key itself. Second, we have the challenge that in the previous HVE techniques of Boneh and Waters [8], the simulator typically creates key that are "completely random" in the sense that they have the same distribution as those coming directly from the authority; however our security definition demands that the keys reflect the distribution of delegation steps specified by the adversary. In order to overcome this we modify the basic scheme such that the distribution of the keys is hidden from a computationally bounded adversary. We show that no adversary can tell whether any key was delegated as he specified or came directly from the authority. After applying this hybrid step we can then proceed to use a simulation that is similar to the previous ones. We believe that our approach is novel in that it is the first instance of a computational game over the private keys in a capability oriented crypto-system.

Finally, we provide a more efficient realization of Anonymous HIBE, which can be seen as a special case of our delegateable HVE scheme. Our Anonymous HIBE scheme has the property that private keys are $O(D)$ in size for a system that allows hierarchies of depth D. Our private key space efficiency can be viewed as a direct result of our corrected definition as the previous scheme of Boyen and Waters required $O(D^2)$ to make all delegated keys have the same distribution as those that came directly from the authority.

2 Definitions

In this section, we introduce the notion of delegation in predicate encryption systems and provide a formal definition of security.

In a predicate encryption system, some user, Alice, creates a public key and a corresponding master key. Using her master key, Alice can compute and hand out a token to Bob, such that Bob is able to evaluate some function[1], f, on the plaintext that has been encrypted. Meanwhile, Bob cannot learn any more information about the plaintext, apart from the output of the function f.

In this paper, we consider the role of delegation in predicate encryption systems. Suppose Alice (the master key owner) has given Bob tokens to evaluate a set of functions $f_1, f_2, \ldots, f_m$ over ciphertexts. Now Bob wishes to delegate to Charles the ability to evaluate the functions $\{f_1 + f_2, f_3, f_4\}$ over the ciphertext. Charles should not be able to learn more information about the plaintext apart from the output of the functions $\{f_1 + f_2, f_3, f_4\}$. For example, although Charles can evaluate $f_1 + f_2$, he should not be able to learn f_1 or f_2 separately. In general, Bob may be interested in delegating any set of functions that is more *restrictive* than what he is able to evaluate with his tokens. Delegation can also happen more than a single level. For example, after obtaining a token from Bob for functions $\{f_1 + f_2, f_3, f_4\}$, Charles may now decide to delegate to his friend David a token to evaluate $f_3 \cdot f_4$.

2.1 Definition

We now formally define delegation in predicate encryption systems that captures the above notion.

Let $X = (x_1, x_2, \ldots, x_\ell) \in \{0,1\}^\ell$ denote a plaintext. Without loss of generality, assume that we would like to evaluate from the ciphertext boolean functions (a.k.a. predicates) on X. Functions that output multiple bits can be regarded as concatenation of boolean functions. Let $\mathcal{F}$ denote the set of all boolean functions from $\{0,1\}^\ell$ to $\{0,1\}$, i.e., $\mathcal{F} := \{f \mid f : \{0,1\}^\ell \to \{0,1\}\}$.

A token allows one to evaluate from the ciphertext a set of functions on X. Let $\mathcal{G} = \{g_1, g_2, \ldots, g_m\} \subseteq \mathcal{F}$ denote a collection of functions (also referred to as a *function family*). We use the notation $\mathsf{closure}(\mathcal{G})$ to denote the set of functions mapping $\{0,1\}^\ell$ to $\{0,1\}$ that can be evaluated from $\{g_1(X), g_2(X), \ldots, g_m(X)\}$, i.e.,

$$\mathsf{closure}(\mathcal{G}) = \left\{ f' : \{0,1\}^\ell \to \{0,1\} \;\middle|\; f'(X) = h(g_1(X), g_2(X), \ldots, g_m(X)), \quad \text{where } h : \{0,1\}^m \to \{0,1\} \right\}$$

Given a token to evaluate a function family $\mathcal{G} \subseteq \mathcal{F}$ from a ciphertext, we have sufficient information to evaluate any function in $\mathsf{closure}(\mathcal{G})$ (assuming unrestricted computational power). A party with a token for a function family $\mathcal{G}$ may be interested in delegating to a friend the ability to evaluate a subset of $\mathsf{closure}(\mathcal{G})$. In other words, any subset of $\mathsf{closure}(\mathcal{G})$ can be used to define a token more *restrictive* than a token for the function family $\mathcal{G}$.

[1] Although we focus on functions that are predicates in our solutions, we use the more general term of functions in this discussion and our formal definitions.

A Delegateable Predicate Encryption (DPE) scheme consists of the following (possibly randomized) algorithms.

Setup(1^λ)**.** The *Setup* algorithm takes as input a security parameter 1^λ, and outputs a public key PK and a master secret key MSK.

Encrypt(PK, X)**.** The *Encrypt* algorithm takes as input a public key PK and a plaintext $X = (x_1, x_2, \ldots, x_\ell) \in \{0,1\}^\ell$; and outputs a ciphertext CT.

GenToken$(\mathsf{PK}, \mathsf{MSK}, \mathcal{G})$**.** The *GenToken* algorithm takes as input a public key PK, master secret key MSK, and a set of boolean functions $\mathcal{G} \subseteq \mathcal{F}$. It outputs a token for evaluating the set of functions $\mathcal{G}$ from a ciphertext.

Query$(\mathsf{PK}, \mathsf{TK}_\mathcal{G}, \mathsf{CT}, f)$**.** The *Query* algorithm takes as input a public key PK, a token $\mathsf{TK}_\mathcal{G}$ for the function family $\mathcal{G}$, a function $f \in \mathcal{G}$, and a ciphertext CT. Suppose CT is an encryption of the plaintext X; the algorithm outputs $f(X)$.

Delegate$(\mathsf{PK}, \mathsf{TK}_\mathcal{G}, \mathcal{G}')$**.** The *Delegate* algorithm takes as input a public key PK, a token for the function family $\mathcal{G} \subseteq \mathcal{F}$, and $\mathcal{G}' \subseteq \mathsf{closure}(\mathcal{G})$. It computes a token for evaluating the function family $\mathcal{G}'$ on a ciphertext.

Remark 1. We note that the above definition captures delegation in predicate encryption systems in the broadest sense. In a predicate encryption system, we would like to maximize the expressiveness of delegation; however, one should not be able to delegate beyond what she can learn with her own tokens. Otherwise, the security of predicate encryption would be broken.

Since we care about being able to perform expressive delegations, we can judge a system by its expressiveness, e.g., what types of functions one can evaluate over the ciphertext, and what types of delegations one can perform. Our vision is to design a predicate encryption system that supports a rich set of queries and delegations. As an initial step, we restrict ourselves to some special classes of functions. At the time of writing this paper, the most expressive predicate encryption system (without delegation) we know of supports conjunctive queries [8]. However, soon after this writing, Katz, Sahai and Waters propose a novel predicate encryption system supporting inner product queries [17].

2.2 Security

To define the security for delegation in predicate encryption, we describe a query security game between a challenger and an adversary. This game formally captures the notion that the tokens reveal no unintended information about the plaintext. In this game, the adversary asks the challenger for a number of tokens. For each queried token, the adversary gets to specify its path of derivation: whether the token is directly generated by the root authority, or delegated from another token. If the token is delegated, the adversary also gets to specify from which token it is delegated. The game proceeds as follows:

Setup. The challenger runs the *Setup* algorithm, and gives the adversary the public key PK.

Query 1. The adversary adaptively makes a polynomial number of queries of the following types:

- *Create token.* The adversary asks the challenger to create a token for a set functions $\mathcal{G} \subseteq \mathcal{F}$. The challenger creates a token for $\mathcal{G}$ without giving it the adversary.
- *Create delegated token.* The adversary specifies a token for function family $\mathcal{G}$ that has already been created, and asks the challenger to perform a delegation operation to create a child token for $\mathcal{G}' \subseteq \mathsf{closure}(\mathcal{G})$. The challenger computes the child token without releasing it to the adversary.
- *Reveal token.* The adversary asks the challenger to reveal an already created token for function family $\mathcal{G}$.

Note that when token creation requests are made, the adversary does not automatically see the created token. The adversary only sees a token when it makes a reveal token query.

Challenge. The adversary outputs two strings $X_0^*, X_1^* \in \{0,1\}^\ell$ subject to the following constraint:

For any token revealed to the adversary in the **Query 1** stage, let $\mathcal{G}$ denote the function family corresponding to this token. For all $f \in \mathcal{G}, f(X_0^*) = f(X_1^*)$.

Next, the challenger flips a random coin b, and encrypts X_b^*. It returns the ciphertext to the adversary.

Query 2. Repeat the **Query 1** stage. All tokens revealed in this stage must satisfy the same condition as above.

Guess. The adversary outputs a guess b' of b. The advantage of an adversary $\mathcal{A}$ in the above game is defined to be $\mathsf{Adv}_{\mathcal{A}} = |\Pr[b = b'] - 1/2|$.

Definition 1. *We say that a delegateable predicate encryption system is* secure *if for all polynomial-time adversaries $\mathcal{A}$ attacking the system, its advantage $\mathsf{Adv}_{\mathcal{A}}$ is a negligible function of λ.*

Selective Security. We also define a weaker security notion called *selective security*. In the selective security game, instead of submitting two strings X_0^*, X_1^* in the **Challenge** stage, the adversary first commits to two strings at the beginning of the security game. The rest of the security game proceeds exactly as before. The selective security model has appeared in various constructions in the literature [10,11,3,8,9,21], since it is often easier to prove security in the selective model.

We say that a delegateable predicate encryption system is *selectively secure* if all polynomial time adversaries $\mathcal{A}$ have negligible advantage in the selective security game.

Remark 2. We note that our security definition is complete in the sense that in the query phase, the adversary gets to specify, for each queried token, its path of derivation: whether the token is generated by the root authority, or from whom the token has been delegated. Previously when researchers studied delegation in identity-based encryption systems, (e.g., Hierarchical Identity-Based Encryption

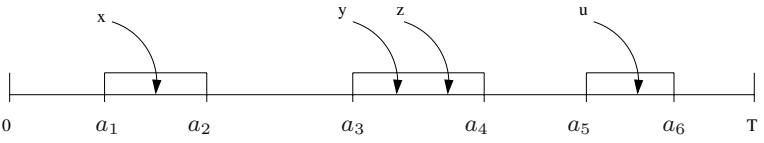

Fig. 1. A simple example of predicate encryption similar to the one described in BW06 [8]

(HIBE) [4], Anonymous Hierarchical Identity-Based Encryption (AHIBE) [9]), the security game was under-specified: the adversary does not get to specify from whom each queried token is delegated. One way to interpret this is to assume that all tokens are generated from the same probability distribution. For example, the AHIBE [9] work uses this approach. While this allows us to prove the security of these systems, it is in fact an overkill. This motivates our new security definition for delegation. Under the new security definition, the delegated token need not be picked from the same probability distribution as the non-delegated tokens. In fact, we show that the ability to capture such nuances in our security definition allows us to construct a simpler AHIBE scheme with smaller private key size.

2.3 A Simple Example

To help understand the above definition, we give a simple example similar to that in the BW06 paper [8]. As shown by Figure 1, suppose the point X encrypted takes on integer values between 0 and T. Given $a, b \in [0, T]$, let $f_{a,b}$ denote the function that decides whether $X \in [a, b]$:

$$f_{a,b}(X) = \begin{cases} 1 & X \in [a, b] \\ 0 & \text{o.w.} \end{cases}$$

In Figure 1, we mark three disjoint segments $[a_1, a_2]$, $[a_3, a_4]$ and $[a_5, a_6]$; and four points x, y, z, u. Suppose Alice has a token for functions $\{f_{a_1,a_2}, f_{a_3,a_4}, f_{a_5,a_6}\}$. This allows her to evaluate the following three predicates: whether $a_1 \leq X \leq a_2$, $a_3 \leq X \leq a_4$, and $a_5 \leq X \leq a_6$. Alice can now distinguish between ciphertexts $\mathit{Encrypt}(\mathsf{PK}, x)$ and $\mathit{Encrypt}(\mathsf{PK}, y)$; but she cannot distinguish between ciphertexts $\mathit{Encrypt}(\mathsf{PK}, y)$ and $\mathit{Encrypt}(\mathsf{PK}, z)$.

Suppose now Alice performs a delegation, and computes a child token for the function $g(X) = f_{a_1,a_2}(X) \vee f_{a_3,a_4}(X)$. Suppose that Bob receives this delegated token from Alice. Now Bob is able to decide whether $(a_1 \leq X \leq a_2) \vee (a_3 \leq X \leq a_4)$; this is a subset of information allowed by Alice's token. Given this new token, Bob can decide whether X falls inside these two ranges, but he cannot decide between the cases whether $X \in [a_1, a_2]$ or $X \in [a_3, a_4]$. For example, Bob can distinguish between the ciphertexts $\mathit{Encrypt}(\mathsf{PK}, x)$ and $\mathit{Encrypt}(\mathsf{PK}, u)$, but he cannot distinguish between the ciphertexts $\mathit{Encrypt}(\mathsf{PK}, x)$ and $\mathit{Encrypt}(\mathsf{PK}, y)$.

3 Delegateable Hidden Vector Encryption (dHVE)

We propose a primitive called delegateable hidden vector encryption (dHVE), where we add delegation to the HVE construction proposed in BW06 [8]. This is an interesting special case to the general definition given in Section 2.1, and represents an initial step towards our bigger vision of enabling expressive queries and delegations in predicate encryption systems.

3.1 Delegateable HVE Overview (dHVE)

In our dHVE system, plaintexts consists of multiple "fields". For example, a plaintext can be the tuple (IP, PORT, TIME, LENGTH). A token corresponds to a conjunction of a subset of these fields: we can fix a field to a specific value, make a field "delegateable", or choose not to include a field in a query. For example, the query $(\text{IP} = ?) \wedge (\text{PORT} = 80) \wedge (\text{TIME} = 02/10/08)$ fixes the values of the PORT and TIME fields, and makes the IP field delegateable. The LENGTH field is not included in the query. A party in possession of this token can fill in any appropriate value for the delegateable field IP, however, she cannot change the values of the fixed field or delete them from the query, nor can she add in the missing field LENGTH to the query. We now give formal definitions for the above notions.

Let Σ denote a finite alphabet and let $?, \perp$ denote two special symbols not in Σ. Define $\Sigma_{?,\perp} := \Sigma \cup \{?, \perp\}$. The symbol $?$ denotes a delegateable field, i.e., a field where one is allowed to fill in an arbitrary value and perform delegation. The symbol $\perp$ denotes a "don't care" field, i.e., a field not involved in some query. Typically, if a query predicate does not concern a specific field, we call this field a "don't care" field. In the aforementioned example, $(\text{IP} = ?) \wedge (\text{PORT} = 80) \wedge (\text{TIME} = 02/10/08)$, the IP field is a delegateable field, LENGTH is a "don't care" field, and the remaining are fixed fields.

Plaintext. In dHVE, our plaintext is composed of a message $\mathsf{M} \in \{0,1\}^*$ and ℓ fields, denoted by $X = (x_1, x_2, \ldots, x_\ell) \in \Sigma^\ell$. The *Encrypt* algorithm takes as input a public key PK, a pair $(X, \mathsf{M}) \in \{0,1\}^* \times \Sigma^\ell$, and outputs a ciphertext CT.

Tokens. In dHVE, a token allows one to evaluate a special class of boolean functions on the fields $X \in \Sigma^\ell$. We use a vector $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_\ell) \in (\Sigma_{?,\perp})^\ell$ to specify a set of functions being queried. Given σ, let $\mathcal{W}(\sigma)$ denote the indices of all delegateable fields, let $\mathcal{D}(\sigma)$ denote the indices of all "don't care" fields, and let $\mathcal{S}(\sigma)$ denote the indices of the remaining fixed fields. In the following, we use the notation $[\ell]$ to denote the set $\{1, 2, \ldots, \ell\}$.

$$\mathcal{W}(\sigma) := \{i \mid \sigma_i = ?\}, \qquad \mathcal{D}(\sigma) := \{i \mid \sigma_i = \perp\}$$
$$\mathcal{S}(\sigma) := \{i \mid \sigma_i \in \Sigma\} = [\ell] \setminus (\mathcal{W}(\sigma) \cup \mathcal{D}(\sigma))$$

Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_\ell) \in (\Sigma_{?,\perp})^\ell$, σ specifies the following function family $\mathcal{C}_\sigma$ on the point $X = (x_1, \ldots, x_\ell)$ encrypted:

$$\mathcal{C}_\sigma := \left\{ \left(\bigwedge_{i \in W'} (x_i = a_i) \right) \wedge \left(\bigwedge_{j \in \mathcal{S}(\sigma)} (x_j = \sigma_j) \right) \mid W' \subseteq \mathcal{W}(\sigma), \forall i \in W', a_i \in \Sigma \right\} \tag{1}$$

In other words, given a token for σ, the family $\mathcal{C}_\sigma$ denotes the set of functions we can evaluate from a ciphertext. For the delegateable fields, we can fill in any appropriate value, but we cannot change or delete any of the fixed fields or add a "don't care" field to the query. In addition, if any function in $\mathcal{C}_\sigma$ evaluates to 1, one would also be able to decrypt the payload message M.

Remark 3. The family $\mathcal{C}_\sigma$ is a set of conjunctive equality tests, where we can fill in every delegateable field in σ with a value in Σ or "don't care". In particular, we fill in fields in W' with appropriate values in σ, and for the remaining delegateable fields $\mathcal{W}(\sigma) - W'$, we fill them with "don't care". If σ has no delegateable field, then the set $\mathcal{C}_\sigma$ contains a single function. This is exactly the case considered by the original HVE construction, where each token allows one to evaluate a single function from a ciphertext.

Delegation. In dHVE, Alice, who has a token for σ, can delegate to Bob a subset of the functions she can evaluate: 1) Alice can fill in delegateable fields (i.e., $\mathcal{W}(\sigma)$) with a value in Σ or with the "don't care" symbol $\perp$; 2) Alice can also leave a delegateable field unchanged (with the ? symbol). In this case, Bob will be able to perform further delegation on that field.

Definition 2. *Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_\ell), \sigma' = (\sigma'_1, \sigma'_2, \ldots, \sigma'_\ell) \in \Sigma^\ell_{?,\perp}$. We say that $\sigma' \prec \sigma$, if for all $i \in \mathcal{S}(\sigma) \cup \mathcal{D}(\sigma)$, $\sigma'_i = \sigma_i$.*

Note that $\sigma' \prec \sigma$ means that from TK_σ we can perform a delegation operation and compute $\mathsf{TK}_{\sigma'}$. In addition, if $\sigma' \prec \sigma$, then $\mathcal{C}_{\sigma'} \subseteq \mathcal{C}_\sigma$, i.e., $\mathsf{TK}_{\sigma'}$ allows one to evaluate a subset of the functions allowed by TK_σ.

In summary, we introduce delegateable fields to the original HVE construction. We use the notation $\sigma \in \Sigma^\ell_{?,\perp}$ to specify a function family. Given TK_σ, one can perform a set of conjunctive equality tests (defined by Equation (1)) from the ciphertext. One may also fill in the delegateable fields in σ with any value in $\Sigma \cup \{\perp\}$ and compute a child token for the resulting vector. The child token allows one to evaluate a subset of the functions allowed by the parent token.

Example. Suppose the trusted authority T issues to A a token for $\sigma_A = (\mathcal{I}_1, \mathcal{I}_2, ?, ?, \perp, \perp, \ldots, \perp)$. This token allows A to evaluate the following functions from the ciphertext:

- $(x_1 = \mathcal{I}_1) \wedge (x_2 = \mathcal{I}_2)$
- $\forall \mathcal{I}_3 \in \Sigma: \quad (x_1 = \mathcal{I}_1) \wedge (x_2 = \mathcal{I}_2) \wedge (x_3 = \mathcal{I}_3)$
- $\forall \mathcal{I}_4 \in \Sigma: \quad (x_1 = \mathcal{I}_1) \wedge (x_2 = \mathcal{I}_2) \wedge (x_4 = \mathcal{I}_4)$
- $\forall \mathcal{I}_3, \mathcal{I}_4 \in \Sigma: \quad (x_1 = \mathcal{I}_1) \wedge (x_2 = \mathcal{I}_2) \wedge (x_3 = \mathcal{I}_3) \wedge (x_4 = \mathcal{I}_4)$

Later, suppose A delegates to B the following token: $\sigma_B = (\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_3, ?, \perp, \perp, \ldots, \perp)$, where $\mathcal{I}_3 \in \Sigma$. Note that this allow B to evaluate the following functions:

- $(x_1 = \mathcal{I}_1) \wedge (x_2 = \mathcal{I}_2) \wedge (x_3 = \mathcal{I}_3)$
- $\forall \mathcal{I}_4 \in \Sigma: \quad (x_1 = \mathcal{I}_1) \wedge (x_2 = \mathcal{I}_2) \wedge (x_3 = \mathcal{I}_3) \wedge (x_4 = \mathcal{I}_4)$

Clearly, a token for σ_B releases a subset of information allowed by σ_A. Meanwhile, B is able to further delegate on the x_4 field.

3.2 dHVE Definition

We now give a formal definition of dHVE.

Setup(1^λ)**.** The *Setup* algorithm takes as input a security parameter 1^λ, and outputs a public key PK and a master secret key MSK.

Encrypt$(\mathsf{PK}, X, \mathsf{M})$**.** The *Encrypt* algorithm takes as input a public key PK, a pair $(X, \mathsf{M}) \in \Sigma^\ell \times \{0,1\}^*$; and outputs a ciphertext CT.

GenToken$(\mathsf{PK}, \mathsf{MSK}, \sigma)$**.** The *GenToken* algorithm takes as input a public key PK, master secret key MSK, and a vector $\sigma \in (\Sigma_{?,\perp})^\ell$. It outputs a token for evaluating the set of conjunctive queries $\mathcal{C}_\sigma$ from a ciphertext.

Delegate$(\mathsf{PK}, \mathsf{TK}_\sigma, \sigma')$**.** The *Delegate* algorithm takes as input a public key PK, a token TK_σ for the vector σ, and another vector $\sigma' \prec \sigma$. It outputs a delegated token $\mathsf{TK}_{\sigma'}$ for the new vector σ'.

Query$(\mathsf{PK}, \mathsf{TK}_\sigma, \mathsf{CT}, \sigma')$**.** The *Query* algorithm takes as input a public key PK, a token TK_σ for the vector σ, a ciphertext CT, and a new vector σ' satisfying the following conditions: (1) $\sigma' \prec \sigma$; (2) σ' does not contain delegatable fields, that is, such a σ' specifies a single conjunctive query (denoted $f_{\sigma'}$) over the point X encrypted. The algorithm outputs $f_{\sigma'}(X)$; in addition, if $f_{\sigma'}(X) = 1$, it also outputs the message M.

Remark 4. We note that in comparison to the general definition given in Section 2, in dHVE, we add a payload message $\mathsf{M} \in \{0,1\}^*$ to the plaintext. Meanwhile, the conjunctive queries in dHVE are functions on the attributes $X \in \Sigma^\ell$, but not the payload M. Additionally, if a query matches a point X encrypted, one can successfully decrypt the payload message using the corresponding token. It is not hard to show that the above formalization for dHVE is captured by the general definition given in Section 2: We can regard (M, X) as an entire bit-string, and decrypting the payload M can be regarded as evaluating a concatenation of bits from the bit-string (M, X). We choose to define dHVE with a payload message to be consistent with the HVE definition in BW06 [8].

Selective security of dHVE. We will prove the selective security of our dHVE construction. We give the formal selective security definition below. The full security definition for dHVE can be found in the Appendix.

- **Init.** The adversary commits to two strings $X_0^*, X_1^* \in \Sigma^\ell$.
- **Setup.** The challenger runs the *Setup* algorithm and gives the adversary the public key PK.

- **Query 1.** The adversary adaptively makes a polynomial number of "create token", "create delegated token" or "reveal token" queries. The queries must satisfy the following constraint: For any token σ revealed to the adversary, let $\mathcal{C}_\sigma$ denote the set of conjunctive queries corresponding to this token.

$$\forall \mathsf{TK}_\sigma \text{ revealed}, \quad \forall f \in \mathcal{C}_\sigma : \quad f(X_0^*) = f(X_1^*) \tag{2}$$

- **Challenge.** The adversary outputs two equal length messages M_0 and M_1 subject to the following constraint:
  For any token σ revealed to the adversary in the **Query 1** stage, let $\mathcal{C}_\sigma$ denote the set of conjunctive queries corresponding to this token.

$$\forall \mathsf{TK}_\sigma \text{ revealed} : \quad \text{if } \exists f \in \mathcal{C}_\sigma, f(X_0^*) = f(X_1^*) = 1, \text{then } \mathsf{M}_0 = \mathsf{M}_1 \tag{3}$$

  The challenger flips a random coin b and returns an encryption of (M_b, X_b) to the adversary.
- **Query 2.** Repeat the **Query 1** stage. All tokens revealed in this stage must satisfy constraints (2) and (3).
- **Guess.** The adversary outputs a guess b' of b.

The advantage of an adversary $\mathcal{A}$ in the above game is defined to be $\mathsf{Adv}_\mathcal{A} = |\Pr[b = b'] - 1/2|$. We say that a dHVE construction is *selectively secure* if for all polynomial time adversaries, its advantage in the above game is a negligible function of λ.

Observation 1. *Anonymous Hierarchical Identity-Based Encryption (AHIBE) is a special case of the above defined dHVE scheme.*

AHIBE is very similar to the dHVE definition given above. The only difference is that in AHIBE, the function family queried is $\mathcal{C}_\sigma$, where σ has the special structure such that $\mathcal{S}(\sigma) = [d]$ where $d \in [\ell]$, $\mathcal{W}(\sigma) = [d+1, \ell]$, and $\mathcal{D}(\sigma) = \emptyset$. In fact, we show that the new security definition and the techniques we use to construct dHVE can be directly applied to give *an AHIBE scheme with shorter private key size*. While the previous AHIBE scheme by Boyen and Waters require $O(D^2)$ private key size, our new construction has $O(D)$ private key size, where D is the maximum depth of the hierarchy. We refer readers to the Appendix for details of the construction.

4 Background on Pairings and Complexity Assumptions

Our construction relies on bilinear groups of composite order $n = pqr$, where p, q and r are distinct large primes. We assume that the reader is familiar with bilinear groups. More background on composite order bilinear groups can be found in the Appendix.

Our construction relies on two complexity assumptions: the bilinear Diffie-Hellman assumption (BDH) and the generalized composite 3-party Diffie-Hellman assumption (C3DH). *Although our construction only requires bilinear groups whose*

order is the product of three primes $n = pqr$, we state our assumptions more generally for bilinear groups of order n where n is the product of three or more primes.

We begin by defining some notation. We use the notation GG to denote the *group generator* algorithm that takes as input a security parameter $\lambda \in \mathbb{Z}^{>0}$, a number $k \in \mathbb{Z}^{>0}$, and outputs a tuple $(p, q, r_1, r_2, \ldots, r_k, \mathbb{G}, \mathbb{G}_T, \mathsf{e})$ where $p, q, r_1, r_2, \ldots, r_k$ are $k+2$ distinct primes, $\mathbb{G}$ and $\mathbb{G}_T$ are two cyclic groups of order $n = pq \prod_{i=1}^{k} r_i$, and $\mathsf{e} : \mathbb{G}^2 \to \mathbb{G}_T$ is the bilinear mapping function. We use the notation $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_{r_1}, \ldots, \mathbb{G}_{r_k}$ to denote the respective subgroups of order p, q, $r_1, \ldots, r_k$ of $\mathbb{G}$. Similarly, we use the notation $\mathbb{G}_{T,p}, \mathbb{G}_{T,q}, \mathbb{G}_{T,r_1}, \ldots, \mathbb{G}_{T,r_k}$ to denote the respective subgroups of order $p, q, r_1, \ldots, r_k$ of $\mathbb{G}_T$.

The bilinear Diffie-Hellman assumption. We review the standard Bilinear Diffie-Hellman assumption, but in groups of composite order. For a given group generator GG define the following distribution $P(\lambda)$:

$$
\begin{array}{l}
(p, q, r_1, \ldots, r_k, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda, k), \quad n \leftarrow pq \prod_{i=1}^{k} r_i, \\
g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_q, \quad h_1 \xleftarrow{R} \mathbb{G}_{r_1}, \quad \ldots, \quad h_k \xleftarrow{R} \mathbb{G}_{r_k} \\
a, b, c \xleftarrow{R} \mathbb{Z}_n \\
\bar{Z} \leftarrow \left((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}),\ g_q,\ g_p,\ h_1,\ h_2,\ \ldots,\ h_k,\ g_p^a,\ g_p^b,\ g_p^c\right) \\
T \leftarrow \mathsf{e}(g_p,\ g_p)^{abc} \\
\text{Output } (\bar{Z},\ T)
\end{array}
$$

Define algorithm $\mathcal{A}$'s advantage in solving the composite bilinear Diffie-Hellman problem as $\mathsf{cBDH\ Adv}_{\mathsf{GG},\mathcal{A}}(\lambda) := \left|\Pr[\mathcal{A}(\bar{Z}, T) = 1] - \Pr[\mathcal{A}(\bar{Z}, R) = 1]\right|$, where $(\bar{Z}, T) \xleftarrow{R} P(\lambda)$ and $R \xleftarrow{R} \mathbb{G}_{T,p}$. We say that GG satisfies the composite bilinear Diffie-Hellman assumption (cBDH) if for any polynomial time algorithm $\mathcal{A}$, $\mathsf{cBDH\ Adv}_{\mathsf{GG},\mathcal{A}}(\lambda)$ is a negligible function of λ.

The generalized composite 3-party Diffie-Hellman assumption. We also rely on the composite 3-party Diffie-Hellman assumption first introduced by Boneh and Waters [8]. For a given group generator GG define the following distribution $P(\lambda)$:

$$
\begin{array}{l}
(p, q, r_1, \ldots, r_k, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda, k), \quad n \leftarrow pq \prod_{i=1}^{k} r_i, \\
g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_q, \quad h_1 \xleftarrow{R} \mathbb{G}_{r_1}, \quad \ldots, \quad h_k \xleftarrow{R} \mathbb{G}_{r_k} \\
R_1, R_2, R_3 \xleftarrow{R} \mathbb{G}_q, \quad a, b, c \xleftarrow{R} \mathbb{Z}_n \\
\bar{Z} \leftarrow \left((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}),\ g_q,\ g_p,\ h_1,\ h_2,\ \ldots,\ h_k,\ g_p^a,\ g_p^b,\ g_p^{ab} \cdot R_1,\ g_p^{abc} \cdot R_2\right) \\
T \leftarrow g_p^c \cdot R_3 \\
\text{Output } (\bar{Z},\ T)
\end{array}
$$

Define algorithm $\mathcal{A}$'s advantage in solving the generalized composite 3-party Diffie-Hellman problem for GG as $\mathsf{C3DH\ Adv}_{\mathsf{GG},\mathcal{A}}(\lambda) := \left|\Pr[\mathcal{A}(\bar{Z}, T) = 1] - \Pr[\mathcal{A}(\bar{Z}, R) = 1]\right|$, where $(\bar{Z}, T) \xleftarrow{R} P(\lambda)$ and $R \xleftarrow{R} \mathbb{G}$. We say that GG satisfies the composite 3-party Diffie-Hellman assumption (C3DH) if for any polynomial time algorithm $\mathcal{A}$, its advantage $\mathsf{C3DH\ Adv}_{\mathsf{GG},\mathcal{A}}(\lambda)$ is a negligible function of 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.

Remark 5. Consider bilinear groups of order $n = pqr$, where p, q and r are three distinct primes. In the above generalized composite 3-party Diffie-Hellman assumption, whether to call a prime p, q or r is merely a nominal issue. So equivalently, we may assume that it is hard to test for Diffie-Hellman tuples in the subgroup $\mathbb{G}_p$, if each element is multiplied by a random element from $\mathbb{G}_r$ instead of $\mathbb{G}_q$.

5 dHVE Construction

We construct a dHVE scheme by extending the HVE construction by Boneh and Waters [8] (also referred to as the BW06 scheme). One of the challenges that we have to overcome is how to add delegation in anonymous IBE systems. We note that delegation is easier in non-anonymous IBE systems, such as in HIBE [4]. In the HIBE construction [4], the public key contains an element corresponding to each attribute, and the delegation algorithm can use these elements in the public key to rerandomize the tokens. In anonymous systems, however, as the encryption now has to hide the attributes as well, we have extra constraints on what information we can release in the public key. This makes delegation harder in anonymous settings.

5.1 Construction

In our construction, the public key and the ciphertext are constructed in a way similar to the BW06 scheme. However, we use a new trick to reduce the number of group elements in the ciphertext asymptotically by one half. Our token consists of two parts, a decryption key part denoted as DK and a delegation component denoted as DL. The decryption key part DK is similar to that in the BW06 scheme. The delegation component DL is more difficult to construct, since we need to make sure that the delegation component itself does not leak unintended information about the plaintext encrypted.

We will use $\Sigma = \mathbb{Z}_m$ for some integer m. Recall that $\Sigma_{?,\perp} := \Sigma \cup \{?, \perp\}$, where $?$ denotes a delegateable field, and $\perp$ denotes a "don't care" field.

Setup(1^λ)**.** The setup algorithm first chooses random large primes $p, q, r > m$ and creates a bilinear group $\mathbb{G}$ of composite order $n = pqr$, as specified in Section 4. Next, it picks random elements

$$(u_1, h_1), \ldots, (u_\ell, h_\ell) \in \mathbb{G}_p^2 \,, \quad g, v, w, \overline{w} \in \mathbb{G}_p \,, \quad g_q \in \mathbb{G}_q, \quad g_r \in \mathbb{G}_r$$

and an exponent $\alpha \in \mathbb{Z}_p$. It keeps all these as the secret key MSK.
It then chooses $2\ell + 3$ random blinding factors in $\mathbb{G}_q$:

$$(R_{u,1}, R_{h,1}), \ldots, (R_{u,\ell}, R_{h,\ell}) \in \mathbb{G}_q \text{ and } R_v, R_w, \overline{R}_w \in \mathbb{G}_q.$$

For the public key, PK, it publishes the description of the group $\mathbb{G}$ and the values

$$g_q,\ \ g_r,\quad V = vR_v,\quad W = wR_w,\quad \overline{W} = \overline{w}R_w,\quad A = \mathsf{e}(g,v)^\alpha,\quad \begin{pmatrix} U_1 = u_1R_{u,1},\ \ H_1 = h_1R_{h,1} \\ \cdots \\ U_\ell = u_\ell R_{u,\ell},\ \ H_\ell = h_\ell R_{h,\ell} \end{pmatrix}$$

The message space $\mathcal{M}$ is set to be a subset of $\mathbb{G}_T$ of size less than $n^{1/4}$.

Encrypt$(\mathsf{PK},\ X \in \Sigma^\ell,\ \mathsf{M} \in \mathcal{M} \subseteq \mathbb{G}_T)$**.** Assume that $\Sigma \subseteq \mathbb{Z}_m$. Let $X = (x_1, \ldots, x_\ell) \in \mathbb{Z}_m^\ell$. The encryption algorithm first chooses a random $\rho \in \mathbb{Z}_n$ and random $Z,\ Z_0,\ Z_\phi,\ Z_1, Z_2, \ldots, Z_\ell \in \mathbb{G}_q$. (The algorithm picks random elements in $\mathbb{G}_q$ by raising g_q to random exponents from $\mathbb{Z}_n$.) Then, the encryption algorithm outputs the ciphertext:

$$\mathsf{CT} = \left(\widetilde{C} = \mathsf{M}A^\rho,\ \ C = V^\rho Z,\ \ C_0 = W^\rho Z_0,\ \ C_\phi = \overline{W}^\rho Z_\phi,\ \ \begin{pmatrix} C_1 = (U_1^{x_1}H_1)^\rho Z_1, \\ C_2 = (U_2^{x_2}H_2)^\rho Z_2, \\ \cdots\cdots \\ C_\ell = (U_\ell^{x_\ell}H_\ell)^\rho Z_\ell \end{pmatrix}\ \right)$$

Remark 6. We note that the ciphertext size is cut down by roughly a half when compared to the BW06 construction [8]. Therefore, our construction immediately implies an HVE scheme with asymptotically half the ciphertext size as the original BW06 construction.

GenToken$(\mathsf{PK}, \mathsf{MSK},\ \ \sigma \in \Sigma_{?,\perp}^\ell)$**.** The token generation algorithm will take as input the master secret key MSK and an ℓ-tuple $\sigma = (\sigma_1, \ldots, \sigma_\ell) \in \Sigma_{?,\perp}^\ell$. The token for σ consists of two parts: (1) a decryption key component denoted as DK, and (2) a delegation component denoted DL.

- The decryption key component DK is composed in a similar way to the original HVE construction [8]. Recall that $\mathcal{S}(\sigma)$ denotes the indices of the fixed fields, i.e., indices j such that $\sigma_j \in \Sigma$. Randomly select $\gamma, \overline{\gamma} \in \mathbb{Z}_p$ and $t_j \in \mathbb{Z}_p$ for all $j \in \mathcal{S}(\sigma)$. Pick random $Y, Y_0, Y_\phi \in \mathbb{G}_r$ and $Y_j \in \mathbb{G}_r$ for all $j \in \mathcal{S}(\sigma)$. Observe that picking random elements from the subgroup $\mathbb{G}_r$ can be done by raising g_r to random exponents in $\mathbb{Z}_n$. Next, output the following decryption key component:

$$\mathsf{DK} = \left(K = g^\alpha w^\gamma \overline{w}^{\overline{\gamma}} \textstyle\prod_{j\in\mathcal{S}(\sigma)} (u_j^{\sigma_j} h_j)^{t_j} Y,\quad K_0 = v^\gamma Y_0,\quad K_\phi = v^{\overline{\gamma}} Y_\phi,\quad \forall j \in \mathcal{S}(\sigma):\ \ K_j = v^{t_j} Y_j\right)$$

- The delegation component DL is constructed as below. Recall that $\mathcal{W}(\sigma)$ denotes the set of all indices i where $\sigma_i = ?$. Randomly select $Y_{i,u}, Y_{i,h} \in \mathbb{G}_r$. For each $i \in \mathcal{W}(\sigma)$, for each $j \in \mathcal{S}(\sigma) \cup \{i\}$, randomly select $s_{i,j} \in \mathbb{Z}_p$, $Y_{i,j} \in \mathbb{G}_r$. For each $i \in \mathcal{W}(\sigma)$, randomly select $\gamma_i, \overline{\gamma}_i \in \mathbb{Z}_p$, $Y_{i,h}, Y_{i,u}, Y_{i,0}, Y_{i,\phi} \in \mathbb{G}_r$. Next, output the following delegation component DL_i for coordinate i.

$$\forall i \in \mathcal{W}(\sigma):\quad \mathsf{DL}_i = \begin{pmatrix} L_{i,h} = h_i^{s_{i,i}} w^{\gamma_i} \overline{w}^{\overline{\gamma}_i} \prod_{j\in\mathcal{S}(\sigma)} (u_j^{\sigma_j} h_j)^{s_{i,j}} Y_{i,h},\quad L_{i,u} = u_i^{s_{i,i}} Y_{i,u} \\ L_{i,0} = v^{\gamma_i} Y_{i,0},\quad L_{i,\phi} = v^{\overline{\gamma}_i} Y_{i,\phi},\quad \forall j \in \mathcal{S}(\sigma) \cup \{i\}:\ \ L_{i,j} = v^{s_{i,j}} Y_{i,j} \end{pmatrix}$$

Remark 7. Later, suppose we want to delegate on the k^{th} field by fixing it to $\mathcal{I} \in \Sigma$. To do so, we will multiply $L_{k,u}^{\mathcal{I}}$ to $L_{k,h}$, resulting in something similar to the decryption key DK (except without the g^α term). Observe that

the $L_{i,h}$ terms encode all the fixed fields (i.e., $\mathcal{S}(\sigma)$). This effectively restricts the use of the delegation components, such that they can only be added on top of the fixed fields, partly ensuring that the delegation components do not leak unintended information.

Delegate$(\mathsf{PK}, \sigma, \sigma')$**.** Given a token for $\sigma \in \Sigma_{?,\perp}^{\ell}$, the *Delegate* algorithm computes a token for $\sigma' \prec \sigma$. Without loss of generality, we assume that σ' fixes only one delegateable field of σ to a symbol in Σ or to $\perp$. Clearly, if we have an algorithm to perform delegation on one field, then we can perform delegation on multiple fields. This can be achieved by fixing the multiple delegateable fields one by one.

We now describe how to compute $\mathsf{TK}_{\sigma'}$ from TK_{σ}. Suppose σ' fixes the k^{th} coordinate of σ. We consider the following two types of delegation: 1) the k^{th} coordinate is fixed to some value in the alphabet Σ, and 2) the k^{th} coordinate is set to $\perp$, i.e., it becomes a "don't care" field.

Type 1: σ' fixes the k^{th} coordinate of σ to $\mathcal{I} \in \Sigma$, and the remaining coordinates of σ remain unchanged. In this case, $\mathcal{S}(\sigma') = \mathcal{S}(\sigma) \cup \{k\}$, and $\mathcal{W}(\sigma') = \mathcal{W}(\sigma) \backslash \{k\}$. (Recall that $\mathcal{S}(\sigma)$ denotes the set of indices j where $\sigma_j \in \Sigma$, and $\mathcal{W}(\sigma)$ denotes the set of delegateable fields of σ.)

Step 1: Let $(\mathsf{DK}, \mathsf{DL})$ denote the parent token. Pick a random exponent $\mu \in \mathbb{Z}_n$ and rerandomize the delegation component DL by raising every element in DL to μ. Denote the rerandomized delegation component as:

$$\forall i \in \mathcal{W}(\sigma): \quad \widehat{\mathsf{DL}}_i = \left(\begin{array}{l} \widehat{L}_{i,h} = L_{i,h}^{\mu}, \;\; \widehat{L}_{i,u} = L_{i,u}^{\mu}, \\ \widehat{L}_{i,0} = L_{i,0}^{\mu}, \;\; \widehat{L}_{i,\phi} = L_{i,\phi}^{\mu}, \;\; \forall j \in \mathcal{S}(\sigma) \cup \{i\}: \;\; \widehat{L}_{i,j} = L_{i,j}^{\mu} \end{array} \right)$$

In addition, compute a partial decryption key component with the k^{th} coordinate fixed to $\mathcal{I}$:

$$\mathsf{pDK} = \left(T = \widehat{L}_{k,u}^{\mathcal{I}} \widehat{L}_{k,h}, \quad T_0 = \widehat{L}_{k,0}, \quad T_\phi = \widehat{L}_{k,\phi}, \quad \forall j \in \mathcal{S}(\sigma'): \;\; T_j = \widehat{L}_{k,j} \right)$$

The partial decryption key pDK is formed similarly to the decryption key DK, except that pDK does not contain the term g^{α}.

Step 2: Compute $|\mathcal{W}(\sigma')|$ rerandomized versions of the above. For all $i \in \mathcal{W}(\sigma')$, randomly select $\tau_i \in Z_n$, and compute:

$$\mathsf{pDK}_i = \left(\Gamma_i = T^{\tau_i}, \quad \Gamma_{i,0} = T_0^{\tau_i}, \quad \Gamma_{i,\phi} = T_\phi^{\tau_i}, \quad \forall j \in \mathcal{S}(\sigma'): \;\; \Gamma_{i,j} = T_j^{\tau_i} \right)$$

Step 3: We are now ready to compute the decryption key component DK' of the child token. DK' is computed from two things: 1) DK, the decryption key component of the parent token and 2) pDK, the partial decryption key computed in Step 1. In particular, pDK is the partial decryption key with the k^{th} field fixed; however, as pDK does not contain the g^{α} term, we need to multiply appropriate components of pDK to those in DK.

To compute DK', first, randomly select $Y', Y_0', Y_\phi' \in \mathbb{G}_r$. For all $j \in \mathcal{S}(\sigma')$, randomly select $Y_j' \in \mathbb{G}_r$. Now output the following DK':

$$\mathsf{DK}' = \left(K' = KTY', \quad K_0' = K_0 T_0 Y_0', \quad K_\phi' = K_\phi T_\phi Y_\phi', \quad K_k' = T_k Y_k', \quad \forall j \in \mathcal{S}(\sigma): \;\; K_j' = K_j T_j Y_j' \right)$$

Step 4: We now explain how to compute the delegation component DL' of the child token. DL' is composed of a portion DL'_i for each $i \in \mathcal{W}(\sigma')$. Moreover, each DL'_i is computed from two things: 1) $\widehat{\mathsf{DL}}_i$ as computed in Step 1 and 2) pDK_i as computed in Step 2.

Follow the steps below to compute DL'. For each $i \in \mathcal{W}(\sigma')$, randomly select $Y'_{i,h}, Y'_{i,u}, Y'_{i,0}, Y'_{i,\phi}$ from $\mathbb{G}_r$. For each $i \in \mathcal{W}(\sigma')$, for each $j \in \mathcal{S}(\sigma) \cup \{i, k\}$, pick at random $Y'_{i,j}$ from $\mathbb{G}_r$. Compute the delegation component DL' of the child token as below:

$$\forall i \in \mathcal{W}(\sigma') : \quad \mathsf{DL}'_i = \begin{pmatrix} L'_{i,h} = \widehat{L}_{i,h} \Gamma_i Y'_{i,h}, & L'_{i,u} = \widehat{L}_{i,u} Y'_{i,u} \\ L'_{i,0} = \widehat{L}_{i,0} \Gamma_{i,0} Y'_{i,0}, & L'_{i,\phi} = \widehat{L}_{i,\phi} \Gamma_{i,\phi} Y'_{i,\phi}, \\ L'_{i,i} = \widehat{L}_{i,i} Y'_{i,i}, & L'_{i,k} = \Gamma_{i,k} Y'_{i,k}, \quad \forall j \in \mathcal{S}(\sigma) : \quad L'_{i,j} = \widehat{L}_{i,j} \Gamma_{i,j} Y'_{i,j} \end{pmatrix}$$

Type 2: We now go on to explain how to perform a Type 2 delegation. Suppose σ' fixes the k^{th} coordinate of σ to $\perp$. In this case, $\mathcal{S}(\sigma') = \mathcal{S}(\sigma)$, and $\mathcal{W}(\sigma') = \mathcal{W}(\sigma) \backslash \{k\}$. The child token is formed by removing the part DL_k from the parent token:

$$\mathsf{TK}_{\sigma'} = (\mathsf{DK}, \quad \mathsf{DL} \backslash \{\mathsf{DL}_k\})$$

Remark 8. It is not hard to verify that delegated tokens have the correct form, except that their exponents are no longer distributed independently at random, but are correlated with the parent tokens. In the proof in the Appendix, we show that Type 1 delegated tokens "appear" (in a computational sense) as if there were generated directly by calling the *GenToken* algorithm, that is, with exponents completely at random. This constitutes an important idea in our security proof.

Query$(\mathsf{PK}, \mathsf{TK}_\sigma, \mathsf{CT}, \sigma')$**.** A token for $\sigma \in \Sigma^{\ell}_{?,\perp}$ allows one to evaluate a set of functions $\mathcal{C}_\sigma$ defined by Equation (1) from the ciphertext. Let $\sigma' \prec \sigma$ and assume σ' has no delegateable fields. Then σ' represents a single function $f_{\sigma'}$ (a conjunctive equality test), and the *Query* algorithm allows us to evaluate $f_{\sigma'}$ over the ciphertext.

To evaluate $f_{\sigma'}$ from the ciphertext using TK_σ, first call the *Delegate* algorithm to compute a decryption key for σ'. Write this decryption key in the form $\mathsf{DK} = (K, \quad K_0, \quad K_\phi, \quad \forall j \in \mathcal{S}(\sigma') : \quad K_j)$. Furthermore, parse the ciphertext as $\mathsf{CT} = \left(\widetilde{C}, \quad C, \quad C_0, \quad C_\phi, \quad \forall j \in \ell : C_j\right)$.

Now use the same algorithm as the original HVE construction to perform the query. First, compute

$$\mathsf{M} \leftarrow \widetilde{C} \cdot \mathsf{e}(C, K)^{-1} \cdot \mathsf{e}(C_0, K_0) \mathsf{e}(C_\phi, K_\phi) \prod_{j \in \mathcal{S}(\sigma')} \mathsf{e}(C_j, K_j) \tag{4}$$

If $\mathsf{M} \notin \mathcal{M}$, output 0, indicating that $f_{\sigma'}$ is not satisfied. Otherwise, output 1, indicating that $f_{\sigma'}$ is satisfied and also output M. We explain why the *Query* algorithm is correct in the Appendix.

5.2 Security of Our Construction

Theorem 1. *Assuming that the bilinear Diffie-Hellman assumption and the generalized composite 3-party Diffie-Hellman assumptions hold in $\mathbb{G}$, then the above dHVE construction is selectively secure.*

We now explain the main techniques used in the proof; however, we defer the detailed proof to the Appendix. In our main construction, delegated tokens have certain correlations with their parent tokens. As a result, the distribution of delegated tokens differ from tokens generated freshly at random by calling the *GenToken* algorithm. A major technique used in the proof is *"token indistinguishability"*: although delegated tokens have correlations with their parent tokens, they are in fact computationally indistinguishable from tokens freshly generated through the *GenToken* algorithm. (Strictly speaking, Type 1 delegated tokens are computationally indistinguishable from freshly generated tokens.) This greatly simplifies our simulation, since now the simulator can pretend that all Type 1 tokens queried by the adversary are freshly generated, without having to worry about their correlation with parent tokens. Intuitively, the above notion of token indistinguishability relies on the C3DH assumption: if we use a random hiding factor from $\mathbb{G}_r$ to randomize each term in the token, then DDH becomes hard for the subgroup $\mathbb{G}_p$.

Acknowledgement

We would like to thank John Bethencourt and Jason Franklin for helpful suggestions and comments. We also would like to thank the anonymous reviewers for their helpful reviews.

References

1. Abdalla, M., Bellare, M., Catalano, D., Kiltz, E., Kohno, T., Lange, T., Malone-Lee, J., Neven, G., Paillier, P., Shi, H.: Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621. Springer, Heidelberg (2005)
2. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: Proceedings of the 2007 IEEE Symposium on Security and Privacy (2007)
3. 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. Springer, Heidelberg (2004)
4. Boneh, D., Boyen, X., Goh, E.-J.: Hierarchical identity based encryption with constant size ciphertext. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 440–456. Springer, Heidelberg (2005)
5. Boneh, D., Di Crescenzo, G., Ostrovsky, R., Persiano, G.: Public key encryption with keyword search. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 506–522. Springer, Heidelberg (2004)
6. 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)

7. Boneh, D., Gentry, C., Hamburg, M.: Space-efficient identity based encryption without pairings. In: Proceedings of FOCS (2007)
8. Boneh, D., Waters, B.: A fully collusion resistant broadcast trace and revoke system with public traceability. In: ACM Conference on Computer and Communication Security (CCS) (2006)
9. Boyen, X., Waters, B.: Anonymous hierarchical identity-based encryption (without random oracles). In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117. Springer, Heidelberg (2006)
10. Canetti, R., Halevi, S., Katz, J.: A forward-secure public-key encryption scheme. In: EUROCRYPT, pp. 255–271 (2003)
11. Canetti, R., Halevi, S., Katz, J.: Chosen-ciphertext security from identity-based encryption. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 207–222. Springer, Heidelberg (2004)
12. Chase, M.: Multi-authority attribute based encryption. In: TCC, pp. 515–534 (2007)
13. Cocks, C.: An identity based encryption scheme based on quadratic residues. In: Proceedings of the 8th IMA International Conference on Cryptography and Coding, London, UK, pp. 360–363. Springer, Heidelberg (2001)
14. Gentry, C., Silverberg, A.: Hierarchical id-based cryptography. In: Zheng, Y. (ed.) ASIACRYPT 2002. LNCS, vol. 2501. Springer, Heidelberg (2002)
15. 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 communications security (CCS) (2006)
16. Horwitz, J., Lynn, B.: Towards hierarchical identity-based encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332. Springer, Heidelberg (2002)
17. Katz, J., Sahai, A., Waters, B.: Predicate encryption supporting disjunctions, polynomial equations, and inner products. In: Eurocrypt (to appear, 2008)
18. Pirretti, M., Traynor, P., McDaniel, P., Waters, B.: Secure attribute-based systems. In: CCS 2006: Proceedings of the 13th ACM conference on Computer and communications security (2006)
19. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R.J.F. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005)
20. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Proceedings of Crypto (1984)
21. Shi, E., Bethencourt, J., Chan, T.-H.H., Song, D., Perrig, A.: Multi-dimension range query over encrypted data. In: IEEE Symposium on Security and Privacy (May 2007)
22. Shi, E., Waters, B.: Delegating capabilities in predicate encryption systems. In: Aceto, L., Damgaard, I., Goldberg, L.A., Halldorsson, M.M., Ingolfsdottir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5125. Springer, Heidelberg (2008), `http://sparrow.ece.cmu.edu/~elaine/docs/delegation.pdf`
23. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE Symposium on Security and Privacy (2000)

Appendix

Due to limit of space, please refer to the online full version of this paper for the appendix [22].