Delegating Capabilities in Predicate Encryption Systems

Elaine Shi*
rshi@cmu.edu

Brent Waters†
bwaters@csl.sri.com

Abstract

In predicate encryption systems, given a capability, one can evaluate one or more predicates on the encrypted data, while all other information about the plaintext remains hidden. We consider the first such systems to permit *delegation* of capabilities. In a system that supports delegation, a user Alice who has a capability can delegate to Bob a more restrictive capability, which allows him to learn less about encrypted data than she did.

We formally define delegation in predicate encryption systems, and propose a new security definition for delegation. In addition, we present 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, 10, 9, 22, 1, 7, 18] (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 with different Internet Service Providers (ISPs) contributing network audit logs to an untrusted repository. The audit logs will later be used to study network intrusions and worms. Because of privacy concerns, 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 the decryption of suspicious log entries, e.g., flows satisfying the following characteristic: $(\text{PORT} \in [p_1, p_2]) \wedge (\text{TIME} \in \text{LAST MONTH})$. Meanwhile, the privacy of all other log entries is 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 those she currently holds. For example, Carnegie Mellon University can have the capability to decrypt all log entries satisfying characteristics of the SQL Slammer worm. The university may ask a specific group of researchers to study the SQL

*Army Research Office under the CyberTA Grant No. W911NF-06-1-0316.

†Supported by NSF CNS-0749931, CNS-0524252, CNS-0716199; the U.S. Army Research Office under the CyberTA Grant No. W911NF-06-1-0316; and the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001.

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 that are more restrictive than the capability 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 delegatable predicate encryption scheme supporting conjunctive queries. 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 finer-grained and more expressive decryption capabilities. Shamir [21] 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 [14], a number of new crypto-systems have provided increasing functionality and expressiveness of decryption capabilities. In Attribute-Based Encryption systems [20, 16, 2, 19, 13] a user can receive a private capability that represents complex access control policies over the attributes of an encrypted record. These systems permit much more expressive access control over encrypted data. However, access to the data itself is still inherently all-or-nothing. The decryptor either will be able to decrypt the data and learn everything or will not and thus learn nothing.

We are interested in what we call *predicate encryption* systems, where the capability performs a function (or predicate) over the encrypted data itself and the evaluator learns only the output of this function. The first examples of this were called *keyword search* (or *anonymous IBE*) [5, 10, 9, 22, 1, 7, 18] systems. 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 [17] in the form of Hierarchical Identity-Based Encryption (HIBE) [17, 4, 15]. In an HIBE scheme both private keys and ciphertexts are associated with ordered lists of identities. A user with a given hierarchical identity can decrypt any ciphertext where his identity is a prefix of the ciphertext's identity; moreover, a user can delegate by creating any other private key 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 the introduction of HIBE, the principle of delegation has been applied to other access control systems such as attribute-based encryption systems [16].

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 [23, 1, 5, 10, 9, 22, 18]. Apart from the aforementioned network audit log example, delegation in predicate encryption can also be useful in other scenarios. For example, 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 her incoming email messages, but only over this period. To do this, Alice can create a more restrictive capability that

decrypts all such messages sent during the next two weeks. In another example, Alice's email gateway has the capability to decrypt certain labels of the email and make forwarding decisions accordingly. Emails labeled as "urgent" by her boss should be sent to her pager; emails from her family should be forwarded to her home computer, and so on. 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 email as "urgent" and "nonurgent" and give preference in forwarding the urgent email.

Delegation in predicate encryption poses a unique set of challenges and is typically harder to realize than delegation in Identity-Based Encryption (IBE) or Attribute-Based Encryption because in an IBE system, a user can 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 hide 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. As a result, realizing delegation is much more challenging. For instance, in anonymous HIBE systems one must be careful that the delegation components themselves cannot be used to answer queries.

Definitional Issues One major difficulty in building delegation into encryption systems is that previous definitions for security of HIBE are incomplete. In the existing definitions of HIBE security, the attacker plays a game where he receives *all* 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 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 [15] 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 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. To do this we create a general definition that accounts for how predicate capabilities are 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 [9] to allow for delegation. 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 [9], the simulator typically creates keys 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. 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 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 *structure of 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 delegatable 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

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. In general, a user who has a delegated capability can in turn create an even more restricted capability. 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., $\mathcal{F} := \{f \mid f : \{0,1\}^\ell \to \{0,1\}\}$.

We define a token as a capability that allows one to evaluate from the ciphertext a set of functions on X. Tokens will be associated with a set $\mathcal{G} = \{g_1, g_2, \ldots, g_m\} \subseteq \mathcal{F}$ that can compute a subset of all available functions. We remark that a token might be represented much more succintly than $|\mathcal{G}|$. For instance, if one had the capability to learn each individual bit of X one could have a small token, but still compute all 2^{2^ℓ} predicate functions on the input.

A delegatable 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.

[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.

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. If CT is an encryption of the plaintext X, then 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 $\mathcal{G}' \subseteq \mathcal{G}$. 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 [9]; we focus our efforts on permitting delegation in such systems.

More recently, Katz, Sahai, and Waters proposed a novel predicate encryption system supporting inner product queries [18] and realized a more expressive system. An interesting open direction is to figure out what types of delegation one might realize in their system.

2.2 Security

We now define the security for delegation in predicate encryption systems. 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. 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 of functions $\mathcal{G} \subseteq \mathcal{F}$. The challenger creates a token for $\mathcal{G}$ without giving it to 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 $\mathcal{G}' \subseteq \mathcal{G}$. 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 sees a token only 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 2.1. *We say that a delegatable 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.*

2.2.1 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 been used earlier in the literature [11, 12, 3, 9, 10, 22].

We say that a delegatable 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. In prior work on delegation in identity-based encryption systems (e.g., Hierarchical Identity-Based Encryption (HIBE) [4], Anonymous Hierarchical Identity-Based Encryption (AHIBE) [10]), the security game was under-specified. In these definitions, the adversary did not get to specify from whom each queried token is delegated.

One way to deal with this is to create systems where all tokens are generated from the same probability distribution. For instance, the AHIBE [10] work uses this approach. While this allows us to prove the security of these systems, it can be an overkill. Under our security definition, the delegated token need not be picked from the same probability distribution as the nondelegated 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 [9]. As shown by Figure 1, 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 $[a_1, a_2]$, $[a_3, a_4]$, $[a_5, a_6]$ and four points x, y, z, u. 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 $a_1 \le X \le a_2$, $a_3 \le X \le a_4$, and $a_5 \le X \le a_6$. 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 $\mathit{Encrypt}(\mathsf{PK}, z)$.

Alice performs a delegation and computes a child token for the function $g(X) = f_{a_1,a_2}(X) \vee f_{a_3,a_4}(X)$, and Bob receives this delegated token from Alice. Bob can decide whether $(a_1 \le X \le a_2) \vee (a_3 \le X \le a_4)$; 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 $\mathit{Encrypt}(\mathsf{PK}, y)$.

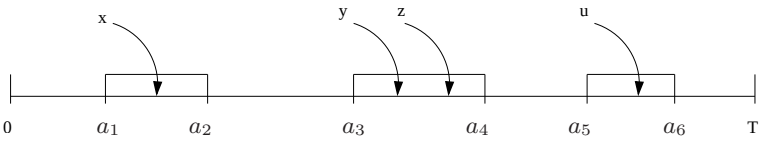

Figure 1: A simple example of predicate encryption similar to the one described in BW06 [9].

3 Delegatable Hidden Vector Encryption (dHVE)

We propose a primitive called delegatable hidden vector encryption (dHVE), where we add delegation to the HVE construction proposed in BW06 [9]. This is an interesting special case to the general definition given in Section 2.1, and represents an initial step toward our bigger vision of enabling expressive queries and delegations in predicate encryption systems.

3.1 Delegatable HVE overview (dHVE)

In our dHVE system, a plaintext 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 "delegatable", or choose not to include a field in a query. For example, the query (IP = ?) $\wedge$ (PORT = 80) $\wedge$ (TIME = 02/10/08) fixes the values of the PORT and TIME fields, and makes the IP field delegatable. The LENGTH field is not included in the query. A party in possession of this token can fill in any appropriate value for the delegatable field IP; however, she cannot change the values of a fixed field such as PORT 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 delegatable 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, (IP = ?)$\wedge$(PORT = 80)$\wedge$(TIME = 02/10/08), the IP field is delegatable, LENGTH is "don't care", and the remaining fields are fixed.

Plaintext Space. In dHVE, our plaintext is composed of a message $\mathsf{M} \in \{0,1\}^*$ and ℓ fields, denoted by $X = (x_1, x_2, \ldots, x_\ell) \in \Sigma^\ell$. Capabilities will be evaluated over X, and the M component is an extra message that will be divulged in case the predicate evaluates to true.

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 delegatable 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 delegatable 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. 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 delegatable 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 delegatable fields $\mathcal{W}(\sigma) - W'$, we fill them with "don't care". If σ has no delegatable 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 delegatable 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 delegatable field unchanged (with the ? symbol). In this case, Bob will be able to perform further delegation on that field.

Definition 3.1. *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 delegatable 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 delegatable 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. 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, A delegates to B the 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 allows 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 and 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, a 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 $f_{\sigma'}(X)$; if $f_{\sigma'}(X) = 1$, it also outputs the message M.

Remark 4. 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. In addition, 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 [9].

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 Appendix D.

- **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 requires $O(D^2)$ private key size, our new construction has $O(D)$ private key size, where D is the maximum depth of the hierarchy. See Appendix E 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 Appendix C.

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 requires only 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 $P(\lambda)$:

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

Define algorithm $\mathcal{A}$'s advantage in solving the composite Bilinear Diffie-Hellman problem as

$$\mathsf{cBDH}\,\mathsf{Adv}_{\mathsf{GG},\mathcal{A}}(\lambda) := \big|\Pr[\mathcal{A}(\bar{Z}, T) = 1] - \Pr[\mathcal{A}(\bar{Z}, R) = 1]\big|$$

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 $\mathcal{A}$, $\mathsf{cBDH}\,\mathsf{Adv}_{\mathsf{GG},\mathcal{A}}(\lambda)$ 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 [9]. For a given group generator GG define the following distribution $P(\lambda)$:

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

Define algorithm $\mathcal{A}$'s advantage in solving the generalized composite 3-party Diffie-Hellman problem for GG as $\mathsf{C3DH}\,\mathsf{Adv}_{\mathsf{GG},\mathcal{A}}(\lambda) := \big|\Pr[\mathcal{A}(\bar{Z}, T) = 1] - \Pr[\mathcal{A}(\bar{Z}, R) = 1]\big|$, 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 $\mathsf{C3DH}\,\mathsf{Adv}_{\mathsf{GG},\mathcal{A}}(\lambda)$ 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 our dHVE scheme by extending the HVE construction by Boneh and Waters [9] (also referred to as the BW06 scheme). One of the challenges that we must overcome is how to add delegation in anonymous IBE systems.

Our primary challenges arise from providing delegation in the anonymous setting. Delegation is easier in nonanonymous 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 restriction on rerandomizing components is the primary hurdle we must overcome.

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 technique 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 DK and a delegation component denoted 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 delegatable 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

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

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 [9]. 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 DK, and (2) a delegation component denoted DL.

- The decryption key component DK is composed in a way similar to that of the original HVE construction [9]. 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:

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

Remark 7. Later, if we want to delegate on the k^{th} field by fixing it to $\mathcal{I} \in \Sigma$, 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 delegatable 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 delegatable 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 all other 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 delegatable 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:

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

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

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

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:

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

Step 3: 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':

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

Step 4: 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:

$$\forall i \in \mathcal{W}(\sigma'): \quad \mathsf{DL}_i' = \left(\begin{array}{l} L_{i,h}' = \widehat{L}_{i,h} \Gamma_i Y_{i,h}', \quad L_{i,u}' = \widehat{L}_{i,u} Y_{i,u}', \\ L_{i,0}' = \widehat{L}_{i,0} \Gamma_{i,0} Y_{i,0}', \quad L_{i,\phi}' = \widehat{L}_{i,\phi} \Gamma_{i,\phi} Y_{i,\phi}', \\ L_{i,i}' = \widehat{L}_{i,i} Y_{i,i}', \quad L_{i,k}' = \Gamma_{i,k} Y_{i,k}', \quad \forall j \in \mathcal{S}(\sigma): \quad L_{i,j}' = \widehat{L}_{i,j} \Gamma_{i,j} Y_{i,j}' \end{array} \right)$$

Type 2: In Type 2 delegation, 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 Appendix B, we show that Type 1 delegated tokens "appear" (in a computational sense) as if they 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 $\sigma \in \Sigma_{?,\perp}^{\ell}$ 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 delegatable 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 $\mathsf{CT} = \left(\widetilde{C}, \quad C, \quad C_0, \quad C_\phi, \quad \forall j \in \ell : C_j\right)$.

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 Appendix A.

5.2 Security of our construction

Theorem 5.1. *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 explain the main techniques used in the proof; however, we defer the detailed proof to Appendix B. In our main construction, delegated tokens have certain correlations with their parent tokens. As a result, the distribution of delegated tokens differs 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 $\mathbb{G}_p$.

6 Conclusions and Future Directions

We study delegation in predicate encryption systems. We propose new security definitions for delegation in predicate encryption and construct a delegatable predicate encryption scheme supporting conjunctive queries. Our main contributions follow:

- We propose a new definition for the security of delegation in predicate encryption and identity-based encryption systems. This new definition allows an adversary to specify to the challenger the path of derivation for each token being queried. Using the new security definition, delegated tokens need not be identically distributed as tokens directly generated by the authority. This turns out to be a crucial observation in both our construction and our proof.

- We realized a delegatable predicate encryption scheme for Hidden Vector Encryption. Our main challenge was to provide a way for the delegator to "rerandomize" keys. This proved to be challenging since the natural techniques from nonanonymous schemes did not apply. We created a new technique in our construction, where we multiply each element in the token with a random group element from the third subgroup $\mathbb{G}_r$. In this way, a delegated token derived using the *Delegate* algorithm would appear computationally indistinguishable from a token picked freshly at random.

- As a side product of our construction, we show that similar techniques can be used to obtain a more efficient AHIBE construction with linear private key size. (In comparison, the BW construction [10] has quadratic private key size.) Our construction also implies an HVE construction with the ciphertext size cut down by half asymptotically, in comparison with the BW06 construction [9].

In the future it would be interesting to explore the limits of delegation in predicate encryption systems. One attractive target is the recent system of Katz, Sahai, and Waters [18] that allows for inner-product predicates and currently represents the most expressive predicate encryption system. A first step for this would be even to decide where to define the desired delegation semantics. For instance, it is unclear whether one should attempt to do this at the core inner-product layer or at some other level.

Acknowledgment

We thank John Bethencourt and Jason Franklin for insightful suggestions and comments. In addition, we are grateful to the anonymous reviewers for their helpful evaluation.

References

[1] Michel Abdalla, Mihir Bellare, Dario Catalano, Eike Kiltz, Tadayoshi Kohno, Tanja Lange, John Malone-Lee, Gregory Neven, Pascal Paillier, and Haixia Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. In *CRYPTO*, 2005.

[2] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, 2007.

[3] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In *EUROCRYPT*, 2004.

[4] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, pages 440–456, 2005.

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

[6] Dan Boneh and Matt Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *Proceedings of Crypto 2001*, volume 2139 of *LNCS*, pages 213–29. Springer-Verlag, 2001.

[7] Dan Boneh, Craig Gentry, and Michael Hamburg. Space-efficient identity based encryption without pairings. In *Proceedings of FOCS*, 2007.

[8] Dan Boneh, Amit Sahai, and Brent Waters. Fully collusion resistant traitor tracing with short ciphertexts and private keys. In *EUROCRYPT*, pages 573–592, 2006.

[9] Dan Boneh and Brent Waters. A fully collusion resistant broadcast trace and revoke system with public traceability. In *ACM Conference on Computer and Communication Security (CCS)*, 2006.

[10] Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *CRYPTO*, 2006.

[11] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.

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

[13] Melissa Chase. Multi-authority attribute based encryption. In *TCC*, pages 515–534, 2007.

[14] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *Proceedings of the 8th IMA International Conference on Cryptography and Coding*, pages 360–363, London, UK, 2001. Springer-Verlag.

[15] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, 2002.

[16] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security (CCS)*, 2006.

[17] Jeremy Horwitz and Ben Lynn. Towards hierarchical identity-based encryption. In *Proceedings of Eurocrypt*, 2002.

[18] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Eurocrypt '08, to appear*, 2008.

[19] Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. Secure attribute-based systems. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, 2006.

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

[21] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of Crypto*, 1984.

[22] Elaine Shi, John Bethencourt, T-H. Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimension range query over encrypted data. In *IEEE Symposium on Security and Privacy*, May 2007.

[23] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, 2000.

A Correctness

We explain why the *Query* algorithm is correct. Let (M, X) denote the plaintext encrypted, and let σ' denote the conjunctive query being evaluated in the *Query* algorithm.

- If the plaintext X satisfies the query, i.e., if $f_{\sigma'}(X) = 1$, a simple calculation shows that the *Query* algorithm outputs the message M. The calculation relies on the fact that if $a \in \mathbb{G}_q$ and $b \in \mathbb{G}_r$, then $\mathsf{e}(a, b) = 1$. Observe that in our construction, each term in the ciphertext (except $\widetilde{C}$) contains a random hiding factor from the subgroup $\mathbb{G}_q$, and each term in the token contains a random hiding factor from the subgroup $\mathbb{G}_r$. When one performs a pairing operation on the ciphertext and the token, the subgroups $\mathbb{G}_q$ and $\mathbb{G}_r$ "disappear", and the result of the pairing is an element of $\mathbb{G}_{T,p}$.
- If the plaintext X does not satisfy the query, i.e., if $f_{\sigma'}(X) = 0$, due to an argument similar to the BW06 [9] paper, the probability $\Pr[Query((\mathsf{PK}, \mathsf{TK}_\sigma, \mathsf{CT}, \sigma') \neq 0]$ is negligible. See Lemma 5.2 of BW06 for details.

B Proof

We prove the security of our construction. We prove selective security, where the adversary commits to two strings X_0^* and X_1^* at the beginning of the security game.

The challenge in proving security is that under our new security game, the simulation needs to reflect how tokens are delegated. In other words, delegated tokens are correlated with their parent tokens in some way, and the simulation should reflect this fact.

Our overall strategy is for the simulator to generate tokens by calling the original *GenToken* algorithm whenever possible, even when the token is delegated. More specifically, for all Type 1 delegation queries, the simulator generates a freshly randomized token by calling the *GenToken* algorithm, rather than the *Delegate* algorithm. As we mentioned, this simulation does not reflect the real security game, since the Type 1 delegated tokens are no longer correlated with their parent tokens. However, we overcome this by showing that the simulation is computationally indistinguishable from the real security game. Intuitively, the indistinguishability property comes from the random group element from the third subgroup $\mathbb{G}_r$ that we use to rerandomize the tokens. Our technique is novel in the sense that in proving semantic security over the ciphertext, we actually rely on "semantic security" over the tokens.

B.1 Sequence of games

To prove security, we define a sequence of games, $\mathsf{Game}_0, \mathsf{Game}_1, \ldots, \mathsf{Game}_5$.

Game_0. Let Game_0 denote the real selective security game as defined in Section 2.2.

Game_1. We first modify Game_0 slightly into a new game Game_1. Game_1 is almost identical to Game_0, except in the way the tokens are generated. In Game_1, whenever the adversary issues a "create delegated token" query, depending on which type of delegation query it is, the challenger performs the following:

- **Type 1:** The challenger calls the *GenToken* algorithm to generate a fresh token, and gives it to the adversary.
- **Type 2:** The challenger generates the token in the normal way by calling the *Delegate* algorithm.

Remark 9. The difference between Game_0 and Game_1 lies in the fact that in the real game Game_0, child tokens are always correlated with their parent tokens. In game Game_1, a Type 1 delegated token is no longer correlated with its parent token; however, Type 2 delegated tokens are still correlated with their parent tokens.

Intuitively, if we use the $\mathbb{G}_r$ subgroup to randomize the tokens, no polynomially bounded adversary is able to tell Game_0 apart from Game_1. In other words, the advantage of the adversary in winning Game_0 is almost the same as her advantage in winning Game_1. Therefore, it suffices to prove security using Game_1 instead of Game_0. This simplifies the proof, since in Game_1, Type 1 delegated tokens are formed in the same way as nondelegated tokens.

Lemma B.1. *Assuming that the generalized 3-party Diffie-Hellman assumption holds in $\mathbb{G}$, then no polynomially bounded adversary can successfully distinguish Game_0 and Game_1 with more than negligible advantage.*

Game_2. Next, we modify Game_1 slightly into a new game Game_2. Game_2 differs from Game_1 also in the way tokens are formed. To explain how Game_2 differs from Game_1, first observe that any token σ queried must satisfy one of the following two cases:

- *Matching tokens.* The decryption key part of TK_σ matches both of the two selected points X_0^* and X_1^*. In this case, for all $i \in \mathcal{W}(\sigma)$, $X_{0,i}^* = X_{1,i}^*$, since otherwise TK_σ would separate the two selected points. In this case, we say that the token matches both selected points.
- *Nonmatching tokens.* The decryption key part of TK_σ matches neither of the two selected points X_0^* and X_1^*.

In Game_2, in any Type 1 delegation query, if the token requested matches both of the selected points X_0^* and X_1^*, the challenger picks the two exponents for w and $\overline{w}$ in DK not independently at random, but in a correlated way: At the beginning of the security game, the challenger picks a random $\pi \in \mathbb{Z}_p$, and keeps it secret from the adversary. Now if a token σ requested in a Type 1 delegation query matches both of the selected points, the challenger picks $\overline{\gamma} = \pi\gamma$ when it computes DK. Similarly, for all $i \in \mathcal{W}(\sigma)$, when the challenger computes DL_i, it picks $\overline{\gamma}_i = \pi\gamma_i$, instead of picking the two exponents independently at random.

Lemma B.2. *Assume that the C3DH assumption holds in $\mathbb{G}$, Then for any polynomial time adversary, the difference of advantage in winning Game_1 and Game_2 is negligible.*

Remark 10. In Game_1, all tokens (except Type 2 tokens) are picked independently at random. In Game_2, this is no longer true, in the sense that for certain queries, the exponents of w and $\overline{w}$ are correlated with each other. Because of the third subgroup $\mathbb{G}_r$ that we use to rerandomize the tokens, we will show that this correlation is computationally hidden from the adversary. The motivation for introducing Game_2 is that later the simulator will need to exploit this correlation in γ and $\overline{\gamma}$ in order to successfully perform a simulation.

Game_3. We now further modify Game_2 into Game_3. Game_3 is almost identical to Game_2 except in the challenge ciphertext. In Game_3, if $\mathsf{M}_0 \neq \mathsf{M}_1$, the first term $\widetilde{C}$ in the challenge ciphertext is replaced by a random element from $\mathbb{G}_T$, and the rest of the ciphertext is generated as usual. If $\mathsf{M}_0 = \mathsf{M}_1$, the challenge ciphertext is generated correctly.

Lemma B.3. *Assume that the BDH and C3DH assumptions hold in $\mathbb{G}$. Then no polynomial time adversary can successfully distinguish Game_2 and Game_3 with more than negligible probability.*

Game_4. Next, we modify Game_3 into a new game Game_4. Game_3 and Game_4 are identical except in the challenge ciphertext. In Game_4, the simulator creates the challenge ciphertext according to the following distribution:

$$C_0 = W^\rho g_p^{-\pi\rho'} Z_0, \quad C_\phi = \overline{W}^\rho g_p^{\rho'} Z_\phi$$

where ρ' is picked at random from $\mathbb{Z}_p$.

Lemma B.4. *Assume that the C3DH assumption holds in $\mathbb{G}$, Then no polynomial time adversary can successfully distinguish games Game_3 and Game_4 with more than negligible probability.*

Game_5. Let $\overline{E}$ denote the set of indices i such that $X^*_{0,i} \neq X^*_{1,i}$, where X^*_0 and X^*_1 are the two committed points in the selective security game. We now define a new game Game_5. Game_5 differs from Game_4 in that for all $i \in \overline{E}$, the ciphertext component C_i is replaced by a random element from $\mathbb{G}_{pq}$.

Lemma B.5. *Assume that the C3DH assumption holds in $\mathbb{G}$, Then no polynomial time adversary can successfully distinguish Game_4 and Game_5 with more than negligible probability.*

Notice that in Game_5, the ciphertext gives no information about the point X^*_b or the message M_b encrypted. Therefore, the adversary can win Game_5 with probability at most 1/2.

We prove the above lemmas. First, we observe that from Game_0 to Game_2, the simulation changes in the way the tokens are generated. We show that these changes remain computationally hidden from any poly-time adversary.

B.2 Indistinguishability of Game_0 and Game_1

We prove Lemma B.1 and show that games Game_0 and Game_1 are computationally indistinguishable. To do this, we perform a hybrid argument on the number of Type 1 "Create delegated token" queries issued by the adversary, henceforth referred to as *T1-delegation query* for short.

Definition B.6. *Let $\mathsf{Game}_{0,0} := \mathsf{Game}_0$ denote the real game. Let q denote the number of T1-delegation queries issued by the adversary. Define a sequence of hybrid games $\mathsf{Game}_{0,i}$ for all $1 \leq i \leq \mathsf{q}$. $\mathsf{Game}_{0,i}$ differs from Game_0 in the fact that when the adversary issues the first i T1-delegation queries, instead of generating the delegated tokens faithfully using the Delegate algorithm, the challenger calls the GenToken algorithm instead to generate these delegated tokens. For all the remaining queries, the challenger computes tokens and responds faithfully as in the real game Game_0. Under the above definition, $\mathsf{Game}_{0,\mathsf{q}}$ is the same as Game_1.*

Claim B.7. *For all $0 \leq d \leq \mathsf{q}-1$, no polynomially bounded adversary can distinguish $\mathsf{Game}_{0,d}$ from $\mathsf{Game}_{0,d+1}$ with more than negligible advantage.*

If we can prove the above Claim B.7, then Lemma B.1 follows by the hybrid argument.

We focus on proving Claim B.7. Intuitively, Claim B.7 relies on the following observation. Pick $h_1, h_2, \ldots, h_\ell \xleftarrow{R} \mathbb{G}_p$, an exponent $\tau \xleftarrow{R} \mathbb{Z}_p$, and randomizing factors $Y_1, Y_2, \ldots, Y_\ell,\ Z_1, Z_2, \ldots, Z_\ell \xleftarrow{R} \mathbb{G}_r$. Now the tuple

$$(h_1 Z_1, \ldots, h_\ell Z_\ell,\ h_1^\tau Y_1, \ldots, h_\ell^\tau Y_\ell)$$

is computationally indistinguishable from

$$(h_1 Z_1, \ldots, h_\ell Z_\ell, R_1, \ldots, R_\ell),$$

where $(R_1, \ldots, R_\ell)$ are picked independently at random from $\mathbb{G}_{pr} = \mathbb{G}_p \times \mathbb{G}_r$. It is not hard to see that this is the equivalent of the Decisional Diffie-Hellman (DDH) assumption for bilinear groups of composite order. Since we can compute pairing in such groups, normally DDH is easy in group $\mathbb{G}$. However, if we use subgroup $\mathbb{G}_r$ to hide subgroup $\mathbb{G}_p$, DDH becomes hard in $\mathbb{G}_p$. For this reason, we can rerandomize tokens by raising all elements to the same exponent τ, and the rerandomized token is computationally indistinguishable from a completely rerandomized token.

We formalize the above intuition into the ℓ-composite 3-party Diffie-Hellman assumption (ℓ-C3DH). Lemma B.8 proves that the ℓ-C3DH assumption is implied by the generalized C3DH assumption. Therefore, we are not introducing a new assumption here.

Given a group generator GG, define the following distribution $P(\lambda)$:

$$
\begin{aligned}
&(p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda, 1), \quad n \leftarrow pqr, \\
&g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_p, \quad g_r \xleftarrow{R} \mathbb{G}_r \\
&Y_1, Y_2, \ldots, Y_\ell, \ Z_1, Z_2, \ldots, Z_\ell \xleftarrow{R} \mathbb{G}_r \\
&h_1, h_2, \ldots, h_\ell \xleftarrow{R} \mathbb{G}_p \\
&\tau \xleftarrow{R} \mathbb{Z}_p \\
&X \leftarrow ((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}), \ \ g_p, \ g_q, \ g_r, \ h_1 Z_1, \ h_2 Z_2, \ \ldots, \ h_\ell Z_\ell) \\
&Q \leftarrow (h_1^\tau Y_1, \ h_2^\tau Y_2, \ \ldots, \ h_\ell^\tau Y_\ell) \\
&\text{Output} \ \ (X, Q)
\end{aligned}
$$

For an algorithm $\mathcal{A}$, define $\mathcal{A}$'s advantage in solving the above problem:

$$
\ell\mathsf{C3DH}\ \mathsf{Adv}_{\mathsf{GG},\mathcal{A}}(\lambda) := \Big| \Pr[\mathcal{A}(X, Q) = 1] - \Pr[\mathcal{A}(X, R) = 1] \Big|
$$

where $(X, Q) \leftarrow P(\lambda)$, and

$$
R = (R_1, R_2, \ldots, R_\ell) \xleftarrow{R} \mathbb{G}_{pr}^{\ell}
$$

Lemma B.8 (ℓ-composite 3-party Diffie-Hellman)**.** *Assume that the generalized composite 3-party Diffie-Hellman assumption holds in $\mathbb{G}$. All probabilistic polynomial time adversaries have negligible advantage in solving the ℓ-C3DH problem.*

Proof. By hybrid argument. ■

Proof of Claim B.7: Recall that in game $\mathsf{Game}_{0,d}$, when the challenger receives the first d T1-delegation queries, it creates a completely randomized token. We show that no polynomially bounded adversary has more than negligible advantage in distinguishing $\mathsf{Game}_{0,d}$ from $\mathsf{Game}_{0,d+1}$.

We use the following sequence of games to prove Claim B.7.

$\mathsf{Game}'_{0,d}$ In *Step 1* of the *Delegate* algorithm, for the $d+1^{\text{th}}$ T1-delegation query, instead of generating $\widehat{\mathsf{DL}} = [\widehat{\mathsf{DL}}_i]_{i \in \mathcal{W}(\sigma)}$ faithfully by raising every element in DL to a random exponent μ, the challenger picks $\widehat{\mathsf{DL}}$ to be a fresh random delegation component. We show that a polynomial time adversary cannot distinguish between the two cases.

$\mathsf{Game}''_{0,d}$ In *Step 2*, instead of computing each pDK_i faithfully, the challenger picks them as fresh random decryption keys (except without the g^α term). We show that a polynomial time adversary cannot distinguish between these two cases.

It is not hard to see that if $\widehat{\mathsf{DL}}$ were a completely rerandomized delegation component for σ, while each pDK_i were independently rerandomized decryption keys (except without the g^α part), then the delegated token $\mathsf{TK}_{\sigma'}$ would be a truly rerandomized token, as if it were generated by directly calling the *GenToken* algorithm. In other words, $\mathsf{Game}''_{0,d} = \mathsf{Game}_{0,d+1}$. We show below that $\mathsf{Game}_{0,d}$ is indistinguishable from $\mathsf{Game}'_{0,d}$ and that $\mathsf{Game}'_{0,d}$ is indistinguishable from $\mathsf{Game}''_{0,d}$.

■

$\mathsf{Game}_{0,d}$ *is indistinguishable from* $\mathsf{Game}'_{0,d}$. We prove the above *Step* 1, i.e., $\mathsf{Game}'_{0,d}$ is computationally indistinguishable from $\mathsf{Game}_{0,d}$. Suppose a polynomial time adversary $\mathcal{A}$ can successfully distinguish between the above two games. Let q_0 denote the maximum number of "create token" and "create Type 1 delegated token" queries made by the adversary. We build a simulator $\mathcal{B}$ that leverages $\mathcal{A}$ to break the following $((\ell+1)(\ell+2)\mathsf{q}_0)$-C3DH assumption. We use the notation $\forall i,j,k$ to denote $\forall i \in [\mathsf{q}_0], 0 \le j \le \ell, k \in [\ell+2]$.

$$
\begin{aligned}
&(p,q,r,\mathbb{G},\mathbb{G}_T,\mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda,1), \quad n \leftarrow pqr,\\
&g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_p, \quad g_r \xleftarrow{R} \mathbb{G}_r\\
&\forall i,j,k: \quad Y_{i,j,k},\ Z_{i,j,k} \xleftarrow{R} \mathbb{G}_r, \quad v_{i,j,k} \xleftarrow{R} \mathbb{G}_p\\
&\tau \xleftarrow{R} \mathbb{Z}_p\\
&X \leftarrow ((n,\mathbb{G},\mathbb{G}_T,\mathsf{e}), \quad g_p,\ g_q,\ g_r, \quad \forall i,j,k: \quad v_{i,j,k}Z_{i,j,k})\\
&Q \leftarrow \left(\forall i,j,k: \quad v_{i,j,k}^{\tau}Y_{i,j,k}\right)
\end{aligned}
$$

Then the challenger randomly decides to give $(X, Q'=Q)$ or $(X, Q'=R)$, where R is a random vector drawn from $(\mathbb{G}_{pr})^{(\ell+1)(\ell+2)\mathsf{q}_0}$.

The simulator will leverage the adversary $\mathcal{A}$ to distinguish between the above two cases.

Init and Setup. At the beginning of the security game, the adversary commits two points X_0^* and X_1^*.

The simulator picks $v \xleftarrow{R} \mathbb{G}_p$, and for $1 \le i \le \ell$, the simulator sets $u_i = v^{x_i}, h_i = v^{y_i}$, where x_i and y_i are random exponents from $\mathbb{Z}_n$. The simulator also picks $w = v^z$ and $\overline{w} = v^{\overline{z}}$. The remaining public parameters and secret key components are picked normally according to the *Setup* algorithm.

Query 1 and 2. Recall that the adversary makes a number of queries of the following types: 1) create token, 2) create delegated token, 3) reveal token. In this simulation, the simulator computes and saves a token internally whenever a "create token" or "create delegated token" query is made. The simulator simply reveals the saved token whenever the adversary makes a "reveal token" query.

Throughout the simulation, whenever the adversary asks the simulator to create a Type 2 delegated token, the simulator generates it faithfully by deriving it from its parent token. This correctly reflects the relation between the child token and the parent token.

From now on, we focus on how the simulator generates Type 1 delegated tokens and nondelegated tokens.

- Before the adversary issues the $(d+1)^{\text{th}}$ T1-delegation query, the simulator computes tokens using the following strategy. Whenever the adversary asks the simulator to create a Type 1 token or nondelegated token, the simulator incorporates elements from the $(\ell+1)(\ell+2)\mathsf{q}_0$-C3DH instance into these tokens, in a way such that all the exponents are distributed uniformly at random. In particular, let i $(1 \le i \le \mathsf{q}_0)$ denote the index of the current query. We note that i is a counter for all "create delegated token" or "create Type 1 delegated token" queries, and d is a counter for all "create Type 1 delegated token" queries. The simulator lets

$$K_0 = v_{i,0,\ell+1}Z_{i,0,\ell+1}, \quad K_\phi = v_{i,0,\ell+2}Z_{i,0,\ell+2}, \quad \forall k \in \mathcal{S}(\sigma): \quad K_k = v_{i,0,k}Z_{i,0,k}$$

For all $j \in \mathcal{W}(\sigma)$, the simulator lets

$$L_{j,0} = v_{i,j,\ell+1} Z_{i,j,\ell+1}, \quad L_{j,\phi} = v_{i,j,\ell+2} Z_{i,j,\ell+2}, \quad \forall k \in \mathcal{S}(\sigma) \cup \{j\} : \quad L_{j,k} = v_{i,j,k} Z_{i,j,k}$$

As the simulator knows the dlog of $w, \overline{w}, u_1, \ldots, u_\ell, h_1, \ldots, h_\ell$ base v, the remaining components of the token can be generated efficiently:

$$K = g^\alpha K_0^z K_\phi^{\overline{z}} \left(\prod_{j \in \mathcal{S}(\sigma)} K_j^{x_j \sigma_j + y_j} \right) Y, \qquad \text{where } Y \xleftarrow{R} \mathbb{G}_r \tag{5}$$

$$\forall j \in \mathcal{W}(\sigma) : \quad \begin{array}{l} L_{j,h} = L_{j,j}^{y_j} L_{j,0}^z L_{j,\phi}^{\overline{z}} \left(\prod_{k \in \mathcal{S}(\sigma)} L_{j,k}^{x_k \sigma_k + y_k} \right) Y_{j,h} \\ L_{j,u} = L_{j,j}^{x_j} Y_{j,u} \end{array} \quad \text{where } Y_{j,h}, Y_{j,u} \xleftarrow{R} \mathbb{G}_r \tag{6}$$

- The adversary makes the $(d+1)^{\text{th}}$ T1-delegation query. In particular, the adversary specifies a parent token, and asks to fix a delegatable field to some value $\mathcal{I} \in \Sigma$. Assume the parent token was created in the i^{th} query, $1 \le i \le \mathsf{q}_0$. When performing *Step 1* of the *Delegate* algorithm, for all $j \in \mathcal{W}(\sigma)$, the simulator lets

  $$\widehat{L}_{j,0} = Q'_{i,j,\ell+1}, \quad \widehat{L}_{j,\phi} = Q'_{i,j,\ell+2}, \quad \forall k \in \mathcal{S}(\sigma) \cup \{j\} : \quad \widehat{L}_{j,k} = Q'_{i,j,k}$$

  Here we use the notation $Q'_{i,j,k}$ to index into the vector Q' from the $((\ell+1)(\ell+2)\mathsf{q}_0)$-C3DH problem. As the simulator knows the dlog of $w, \overline{w}, u_1, \ldots, u_\ell, h_1, \ldots, h_\ell$ base v, the remaining components of the token can be generated efficiently due to Equations (5) and (6).
- For all the remaining queries, the simulator responds faithfully as in the real game.

Clearly, if $Q' = Q$ in the $((\ell+1)(\ell+2)\mathsf{q}_0)$-C3DH instance, then the above simulation is identically distributed as $\mathsf{Game}_{0,d}$. Otherwise, the above simulation is identically distributed as $\mathsf{Game}'_{0,d}$.

Challenge. The simulator generates the challenge ciphertext as normal.

Guess. If the adversary has ϵ difference in its advantage in $\mathsf{Game}_{0,d}$ and $\mathsf{Game}'_{0,d}$, it is not hard to see that the simulator has a comparable advantage in solving the C3DH instance.

$\mathsf{Game}'_{0,d}$ is indistinguishable from $\mathsf{Game}''_{0,d}$. Similarly, we can show that Step 2 above is also true, i.e., no polynomial time adversary can distinguish between $\mathsf{Game}'_{0,d}$ and $\mathsf{Game}''_{0,d}$ with nonnegligible probability. To prove this, we further define a sequence of hybrid games. Suppose that in $\mathsf{Game}'_{0,d,c}$ where $0 \le c \le \mathcal{W}(\sigma')$, the first c pDK_i's are replaced by independent random decryption keys (without the g^α part). We show that a polynomial time adversary cannot distinguish between $\mathsf{Game}'_{0,d,c}$ and $\mathsf{Game}'_{0,d,c+1}$. Then, by the hybrid argument, $\mathsf{Game}'_{0,d}$ and $\mathsf{Game}''_{0,d}$ (which is identically distributed as $\mathsf{Game}_{0,d+1}$) are computationally indistinguishable.

The simulator tries to solve the following ℓ-C3DH instance:

$$\begin{array}{l} (p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda, 1), \quad n \leftarrow pqr, \\ g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_p, \quad g_r \xleftarrow{R} \mathbb{G}_r \\ Y_1, Y_2, \ldots, Y_\ell, \ Z_1, Z_2, \ldots, Z_\ell \xleftarrow{R} \mathbb{G}_r \\ v_1, v_2, \ldots, v_\ell \xleftarrow{R} \mathbb{G}_p \\ \tau \xleftarrow{R} \mathbb{Z}_p \\ X \leftarrow ((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}), \ g_p, \ g_q, \ g_r, \ v_1 Z_1, \ v_2 Z_2, \ \ldots, \ v_\ell Z_\ell) \\ Q \leftarrow (v_1^\tau Y_1, \ v_2^\tau Y_2, \ \ldots, \ v_\ell^\tau Y_\ell) \end{array}$$

The simulator tries to distinguish between $(X, Q' = Q)$ and $(X, Q' = R)$, where R is a random vector from $\mathbb{G}_{pr}$. The simulator leverages an adversary $\mathcal{A}$ who can distinguish between $\mathsf{Game}'_{0,d,c}$ and $\mathsf{Game}'_{0,d,c+1}$.

Init and Setup. At the beginning of the game, the simulator sets up public parameters and a secret key by choosing $v \xleftarrow{R} \mathbb{G}_p$. For $1 \leq i \leq \ell$, the simulator sets $u_i = v^{x_i}, h_i = v^{y_i}$, where x_i and y_i are random exponents from $\mathbb{Z}_n$. The simulator also picks $w = v^z$ and $\overline{w} = v^{\overline{z}}$. The remaining public parameters and secret key components are picked normally according to the *Setup* algorithm.

Query 1 and 2. The adversary issues a number of queries to the simulator. Like before, the simulator internally computes and saves a token whenever it receives a "create token" or "create delegated token" query. The simulator simply reveals to the adversary the previously computed token in a "reveal token" query.

The simulator treats Type 2 tokens as a special case. Whenever the adversary asks the simulator to create a Type 2 token, the simulator computes it faithfully by deriving the token from the specified parent. This correctly reflects the relation between the child token and its parent. Henceforth, we focus on how the simulator computes Type 1 delegated tokens and nondelegated tokens.

- Before the adversary makes the $(d+1)^{\text{th}}$ T1-delegation query, the simulator always computes each Type 1 delegated token and nondelegated token freshly at random.
- At the $(d+1)^{\text{th}}$ T1-delegation query, the adversary specifies a parent token, and requests to fix the k^{th} coordinate to some value $\mathcal{I} \in \Sigma$. To answer this query, the simulator first generates $\widehat{\mathsf{DL}}_i$ for all $i \in \mathcal{W}(\sigma)$, and pDK. For $i \in \mathcal{W}(\sigma) \backslash \{k\}$, the simulator picks at random $\widehat{L}_{i,0}, \widehat{L}_{i,\phi}$ and $\widehat{L}_{i,j}$ for all $j \in \mathcal{S}(\sigma) \cup \{i\}$. The simulator lets

$$T_0 = \widehat{L}_{k,0} = v_{\ell+1} Z_{\ell+1}, \quad T_\phi = \widehat{L}_{k,\phi} = v_{\ell+2} Z_{\ell+2}, \quad \forall j \in \mathcal{S}(\sigma') : \quad T_j = \widehat{L}_{k,j} = v_j Z_j$$

  As the simulator knows the dlog of $w, \overline{w}, u_1, \ldots, u_\ell, h_1, \ldots, h_\ell$ base v, the remaining components of $\widehat{\mathsf{DL}}_i$'s and pDK can be generated efficiently in a way similar to Equations (5) and (6). The only difference is that pDK does not contain the g^α term, while a decryption key DK does.

  The simulator picks the first c pDK_i's as fresh random (partial) decryption keys.

  Let i be the $c+1^{\text{th}}$ index in $\mathcal{W}(\sigma')$. For pDK_i, the simulator sets

$$\Gamma_{i,0} = Q'_{\ell+1}, \quad \Gamma_{i,\phi} = Q'_{\ell+2} \qquad \forall j \in \mathcal{S}(\sigma') : \quad \Gamma_{i,j} = Q'_j$$

  We use the notation Q'_j to index into the j^{th} element of the vector Q' from the ℓ-C3DH problem. Again, since the simulator knows the dlog of $w, \overline{w}, u_1, \ldots, u_\ell, h_1, \ldots, h_\ell$ base v, the remaining terms in pDK_i can be generated efficiently.

  For all the remaining pDK_i's, the simulator generates them normally as in the original *Delegate* algorithm.
- For all the remaining queries, the simulator generates them faithfully.

Challenge. The simulator generates the challenge ciphertext as normal.

Guess. Notice that if $Q' = Q$ in the ℓ-C3DH problem, then the above simulation is identically distributed as $\mathsf{Game}_{0,d,c}$; otherwise, the above simulation is identically distributed as $\mathsf{Game}_{0,d,c+1}$. Therefore, if a polynomial time adversary could successfully distinguish between $\mathsf{Game}_{0,d,c}$ and $\mathsf{Game}_{0,d,c+1}$, then the simulator would be able to solve the ℓ-C3DH problem with nonnegligible probability.

B.3 Indistinguishability of Game_1 and Game_2

We prove Lemma B.2.

Let q denote the maximum number of T1-delegation queries for a matching token made by the adversary. We show that if a poly-time adversary has nonnegligible difference in its advantage in Game_1 and Game_2, we can build a simulator that leverages this adversary to break the modified $\mathsf{q}(\ell+1)$-C3DH assumption. In the following, we use $\forall i, j$ to mean $\forall i \in [\mathsf{q}], 0 \leq j \leq \ell$.

$$
\begin{aligned}
&(p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda, 1), \quad n \leftarrow pqr, \\
&g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_p, \quad g_r \xleftarrow{R} \mathbb{G}_r \\
&\forall i, j: \quad Y_{i,j,1}, Y_{i,j,2} \xleftarrow{R} \mathbb{G}_r \\
&v_1, v_2 \xleftarrow{R} \mathbb{G}_p \\
&\forall i, j: \quad \tau_{i,j} \xleftarrow{R} \mathbb{Z}_p \\
&Q \leftarrow \left(v_1^{\tau_{i,j}} Y_{i,j,1},\ v_2^{\tau_{i,j}} Y_{i,j,2}\right)
\end{aligned}
$$

The modified $\mathsf{q}(\ell+1)$-C3DH assumption says that given randomly $(Q' = Q)$ or $(Q' = R)$ where R is a random vector of length $\mathsf{q}(\ell+1)$ from $\mathbb{G}_{pr}$, a poly-time adversary cannot distinguish whether $Q' = Q$ or $Q' = R$. The modified $\mathsf{q}(\ell+1)$-C3DH assumption follows from the generalized C3DH assumption by the hybrid argument. Hence, we are not introducing a new assumption here.

Suppose the simulator is randomly given $(Q' = Q)$ or $(Q' = R)$ where R is a random vector of length $\mathsf{q}(\ell+1)$ from $\mathbb{G}_{pr}$. Now the simulator tries to distinguish between the two cases.

The simulator first generates public and secret keys. The simulator picks $v \in \mathbb{G}_p$ at random. For $1 \leq i \leq \ell$, the simulator sets $u_i = v^{x_i}, h_i = v^{y_i}$, where x_i and y_i are random exponents from $\mathbb{Z}_n$. The simulator also picks $w = v^z$ and $\overline{w} = v^{\overline{z}}$, where z and $\overline{z}$ are also random exponents from $\mathbb{Z}_n$. The simulator proceeds and generates the rest of public and secret keys as normal.

We explain how the simulator answers the adversary's queries. When the adversary makes the i^{th} T1-delegation query for a matching token, the simulator computes a token by letting $K_0 = Q'_{i,0,1}$ and $K_\phi = Q'_{i,0,2}$. This fixes the exponents γ and $\overline{\gamma}$, although the simulator does not know what γ and $\overline{\gamma}$ really are. The simulator picks the remaining parameters needed as normal, and computes the decryption key part DK. Notice that even though the simulator does not know γ or $\overline{\gamma}$, DK can be efficiently computed, since the simulator knows the dlog of $w, \overline{w}$ base v.

Similarly, for delegation component DL_j where $j \in \mathcal{W}(\sigma)$, the simulator lets $L_{i,0} = Q'_{i,j,1}$, $L_{i,\phi} = Q'_{i,j,2}$, picks the remaining parameters needed as normal, and computes DL_j. By the same reasoning, even though the simulator does not know γ_j or $\overline{\gamma}_j$, DL_j can be efficiently computed since the simulator knows the dlog of $w, \overline{w}$ base v.

We observe that if $Q' = Q$, then the above simulation would be identically distributed as Game_2. Otherwise, if $Q' = R$, the above simulation would be identically distributed as Game_1. Therefore, if a poly-time adversary has nonnegligible difference in its advantage in distinguishing Game_1 and Game_2, the simulator would be able to break the modified $\mathsf{q}(\ell+1)$-C3DH assumption.

Until now, we have shown that the simulator can change the way tokens are computed such that these changes remain computationally hidden from the adversary. Now we show that if the simulator changes certain parts of the ciphertext to random, a poly-time adversary cannot distinguish with more than negligible advantage.

In all the simulations described below, the simulator will compute tokens only when a "reveal token" query is made. When the adversary makes a "create token" or "create delegated token" query, the simulator simply records that query without computing the actual token created. In particular, in some of these simulations, the simulator is not able to compute all tokens. However, the simulator

is always able to compute a token in a "reveal token" query. Recall that a token σ represents a set of conjunctive queries over the point X encrypted. Any token σ requested in a "reveal token" query must satisfy the condition that for any function $f \in \mathcal{C}_\sigma$ (f is a conjunctive query on $X \in \mathbb{Z}_m^\ell$), $f(X_0^*) = f(X_1^*)$. Henceforth, we use the terminology σ *does not separate the two selected points* X_0^* *and* X_1^* to describe the above condition. In all the simulations below, the simulator is always able to compute a token σ, as long as σ does not separate the two selected points.

In the simulations described below that change certain parts of the ciphertext, an adversary can ask the simulator to reveal a token of the following types: 1) nondelegated, 2) Type 1 delegated, 3) Type 2 delegated. Clearly, nondelegated tokens are distributed independently from other tokens. Due to Lemma B.1, Type 1 tokens appear to be uncorrelated with their parent tokens. Therefore, the simulator always computes nondelegated and Type 1 tokens freshly at random. By contrast, Type 2 tokens are correlated with their ancestor tokens, and thus require special treatment. The simulator must construct Type 2 tokens such that they reflect the correct relationship with their ancestors. Before explaining how the simulations are performed, we describe a general strategy the simulator uses to generate Type 2 delegated tokens, since they require special treatment different from that for nondelegated tokens and Type 1 delegated tokens.

B.4 Generating Type 2 delegated tokens

The simulator uses a "book-keeping" technique. We use the notation $\mathsf{TK}_{\sigma'} \prec_2 \mathsf{TK}_\sigma$ to mean that $\mathsf{TK}_{\sigma'}$ is derived from TK_σ through a Type 2 delegation operation. Whenever the adversary asks the simulator to reveal a Type 2 delegated token, instead of computing a fresh token, the simulator examines the history of queries, and finds the sequence of Type 2 delegation queries that created this token,

$$\mathsf{TK}_{\sigma_k} \prec_2 \mathsf{TK}_{\sigma_{k-1}} \prec_2 \ldots \prec_2 \mathsf{TK}_{\sigma_1}$$

where $\mathsf{TK}_{\sigma_k} := \mathsf{TK}_\sigma$ is the currently requested token, and TK_{σ_1} is a nondelegated token or a Type 1 delegated token. We note that the simulator might not be able to compute all these tokens. However, the simulator can compute a token if the token does not separate the two selected points X_0^* and X_1^*.

If a token $\mathsf{TK}_{\sigma_i} (1 \leq i \leq k)$ in the above sequence has been computed by the simulator in the past, the simulator simply derives TK_σ from TK_{σ_i} using the *Delegate* algorithm, and returns it to the adversary. In particular, σ fixes some delegatable coordinates of σ_i to $\perp$, and the simulator simply removes the corresponding delegation components from TK_{σ_i} to form TK_σ. Otherwise, if no token in the above sequence has been computed by the simulator in the history, the simulator finds the earliest ancestor $\mathsf{TK}_{\sigma_i} (1 \leq i \leq k)$ in the above sequence, such that TK_{σ_i} does not separate the two selected points X_0^* and X_1^*. The simulator generates TK_{σ_i} freshly at random, and then it follows the *Delegate* algorithm to generate TK_σ from TK_{σ_i} (by removing the fields set to $\perp$ from the delegation components).

We now describe a sequence of simulations that replace ciphertext components by random group elements. In these simulations, we focus on how the simulator can compute nondelegated and Type 1 tokens. Type 2 tokens are always treated as a special case using the algorithm described earlier in this section.

B.5 Indistinguishability of Game_2 and Game_3

In Game_3, if $\mathsf{M}_0 \neq \mathsf{M}_1$, the challenger replaces the ciphertext component $\widetilde{C}$ by a random group element from $\mathbb{G}_T$.

The proof that Game_2 and Game_3 are indistinguishable to a poly-time adversary is similar to that in the original BW06 paper [9].

We prove this in two steps:

- Game_2': If $\mathsf{M}_0 \neq \mathsf{M}_1$, the challenger replaces the ciphertext component $\widetilde{C}$ by a random group element from $\mathbb{G}_{T,p}$. No poly-time adversary can distinguish Game_2' from Game_2 with more than negligible probability.

- Because of the subgroup decision assumption (implied by the C3DH assumption), if the simulator replaces the ciphertext component $\widetilde{C}$ by a random group element from $\mathbb{G}_T$ instead of $\mathbb{G}_{T,p}$, the adversary cannot distinguish this case from Game_2'.

We first prove that Game_2 is computationally indistinguishable from Game_2'. Suppose the simulator tries to solve the following BDH instance:

$$(p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda), \quad n \leftarrow pqr, \quad g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_q, \quad g_r \xleftarrow{R} \mathbb{G}_r$$
$$a, b, c \xleftarrow{R} \mathbb{Z}_n$$
$$\bar{Z} \leftarrow \big((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}), \quad g_p, \ g_q, \ g_r, \quad g_p^a, \quad g_p^b, \quad g_p^c\big)$$
$$Q \leftarrow \mathsf{e}(g_p, g_p)^{abc}$$

The simulator is randomly given $(\bar{Z},\ Q' = Q)$ or $(\bar{Z},\ Q' = R)$ where R is a random element in $\mathbb{G}_T$, and it tries to distinguish between these two cases.

If there exists a poly-time adversary $\mathcal{A}$ that has non-negligible difference in its advantage in Game_2 and Game_3, we can build the following simulation to solve the BDH instance.

Init. The adversary commits to two selected points X_0^* and X_1^*. The challenger picks a random coin β internally.

Setup. The simulator chooses random $(R_{u,1}, R_{h,1}), (R_{u,2}, R_{h,2}), \ldots, (R_{u,\ell}, R_{h,\ell}) \in \mathbb{G}_q$, $R_v, R_w, \overline{R}_w \in \mathbb{G}_q$, and random $z_1, y_1, \ldots, t_\ell, y_\ell \in \mathbb{Z}_n$, and $x, \overline{x} \in \mathbb{Z}_n$. The simulator publishes the group description $g_q, g_r, V = g_p R_v$. It lets $A = \mathsf{e}(g_p^a, g_p^b)$ and creates

$$U_i = (g_p^b)^{z_i} R_{u,i}, \quad H_i = (g_p^b)^{-z_i X^*_{\beta,i}} g_p^{y_i} R_{h,i}$$

Finally, the simulator creates:

$$W = g^x, \quad \overline{W} = g^{\overline{x}}$$

We observe that the parameters are distributed identically to the real scheme.

Query 1. The simulator does not compute any token when the adversary makes "create token" or "create delegated token" queries. It computes tokens only when "reveal token" queries are made.

Recall that in Section B.4, we pointed out that Type 2 tokens require special treatment. In addition, we gave an algorithm for the simulator to generate Type 2 tokens such that they reflect the correct relationship with their parent tokens. Now it suffices to show that the simulator can always compute a fresh random token, so long as the token does not separate the two selected points X_0^* and X_1^*.

Whenever the adversary makes a "reveal token" query for a matching token, the simulator simply aborts and takes a random guess. The reason is that by our definition, when the adversary asks the simulator to reveal a matching token, the challenge messages M_0 and M_1 must be equal. However, in this case, Game_2 and Game_3 are identical, so there can be no difference in the adversary's advantage in between these two games.

Whenever the adversary asks the simulator to reveal a nonmatching token, the simulator needs to compute a token of the correct form. First, notice that the delegation components DL can be efficiently computed, since they do not contain any unknown parameters. However, computing the decryption key component DK is slightly more tricky. Recall that because of the way the public key is

formed, $g^\alpha = g_p^{ab}$. Therefore, the decryption key component DK contains the term g_p^{ab}. Unfortunately, the simulator does not know g_p^{ab}, so it has to find some way to cancel out that term and still form a correctly distributed token. The intuition is that since the token is nonmatching, there exists a dimension i where $X^*_{0,i} \neq \sigma_i$ and $X^*_{1,i} \neq \sigma_i$. We observe that the term $u_i^{\sigma_i} h_i = (g_p^b)^{\Delta_i z_i} g_p^{y_i}$ contains $(g_p^b)^{\Delta_i z_i}$, where $\Delta_i = \sigma_i - X^*_{\beta,i} \neq 0$. Therefore, the simulator can pick $\widehat{t_i}$ at random from $\mathbb{Z}_n$, and let

$$t_i = \widehat{t_i} - a/(\Delta_i z_i)$$

without actually computing it. And this t_i is used to generate the decryption key component DK. If the simulator picks t_i in the way specified above, it is able to compute DK, since all terms containing the unknown parameter g_p^{ab} cancel out. In particular, in the decryption key DK, K is a product of several terms. Rewrite K:

$$\begin{aligned} K &= g_p^{ab} w^\gamma \overline{w}^{\overline{\gamma}} \prod_{j \in \mathcal{S}(\sigma)} (u_j^{\sigma_j} h_j)^{t_j} Y \\ &= \left(g_p^{ab} (u_i^{\sigma_i} h_i)^{t_i} \right) \cdot \left(w^\gamma \overline{w}^{\overline{\gamma}} \prod_{j \in \mathcal{S}(\sigma), j \neq i} (u_j^{\sigma_j} h_j)^{t_j} Y \right) \end{aligned}$$

The product term

$$g_p^{ab} (u_i^{\sigma_i} h_i)^{t_i} = (u_i^{\sigma_i} h_i)^{\widehat{t_i}} \cdot (g_p^a)^{-y_i/(\Delta_i z_i)}$$

can be efficiently computed, since all terms involving g_p^{ab} cancel out. It is not hard to see that the remaining terms in K can be efficiently generated, since the simulator knows all parameters needed. As the simulator knows g_p^a, the term $K_i = v^{t_i}$ can be efficiently computed.

Challenge. The adversary gives the simulator two messages, M_0 and M_1. If $\mathsf{M}_0 = \mathsf{M}_1$, the simulator aborts and takes a random guess for the reason stated above.

Otherwise, the simulator chooses random $Z, Z_0, Z_\phi, Z_1, Z_2, \ldots, Z_\ell \in \mathbb{G}_q$, and outputs the following challenge ciphertext:

$$\widetilde{C} = \mathsf{M}_\beta Q', \quad C = (g_p^c) Z, \quad C_0 = (g_p^c)^x Z_0, \quad C_\phi = (g_p^c)^{\overline{x}} Z_\phi, \quad \forall i \in [\ell] : \; C_i = (g_p^c)^{y_i} Z_i$$

Query 2. Same as phase Query 1.

Guess. The adversary outputs a guess β'. If $\beta = \beta'$, the simulator guesses that $Q' = Q$. Otherwise, the simulator guesses that $Q' = R$. We observe that if $Q' = Q$, the ciphertext component $\widetilde{C}$ is a faithful encryption of M_β; otherwise, $\widetilde{C}$ is distributed at random in $\mathbb{G}_{T,p}$. Therefore, if the adversary has ϵ advantage in guessing β, the simulator also has ϵ advantage in solving the BDH instance.

To show that Game_2' is computationally indistinguishable from Game_3, we rely on the Bilinear Subgroup Decision (BSD) assumption introduced by Boneh, Sahai and Waters [8]. Bilinear Subgroup Decision assumption is implied by the generalized composite 3-party Diffie-Hellman assumption.

The simulator gets the following BSD instance:

$$\begin{aligned} &(p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda), \quad n \leftarrow pqr, \quad g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_q, \quad g_r \xleftarrow{R} \mathbb{G}_r \\ &\bar{Z} \leftarrow ((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}), \;\; g_p, \; g_q, \; g_r) \\ &Q \leftarrow \mathbb{G}_{T,p} \end{aligned}$$

The simulator is also randomly given $Q' = Q$ or $Q' = R$ where $R \xleftarrow{R} \mathbb{G}_T$. The BSD assumption posits that no poly-time algorithm can distinguish between the above two cases with more than negligible advantage.

The simulation proceeds as follows.

Init. The attacker gives the simulator two identities X_0^*, X_1^*. The challenger then flips the coin β internally.

Setup. The simulator sets up the parameters as would the real setup algorithm. All the simulator needs to do this is g_p, g_q, g_r from the assumption.

Query 1. The simulator answers queries as the real authority would. One small difference is that the simulator chooses exponents from $\mathbb{Z}_n$ instead of $\mathbb{Z}_p$. However, this does not change anything since the both the simulator and a real authority will raise the elements from $\mathbb{G}_p$ to the exponents.

Challenge. The adversary first gives the simulator messages $\mathsf{M}_0, \mathsf{M}_1$. If $\mathsf{M}_0 = \mathsf{M}_1$ then the simulator simply encrypts the message to the point X_β^*. Otherwise, the simulator creates the challenge ciphertext of message M_β to X_β^* as normal with the exception that C' is multiplied by Q'.

If $Q' = Q$, then the simulator is playing Game_2'; otherwise it is playing Game_3.

Query 2. Same as Query Phase 1.

Guess. The adversary outputs a guess β'. If $\beta = \beta'$, the simulator guesses that $Q' = Q$; otherwise it guesses that $Q' = R$. By our assumption the probability that the adversary guesses β correctly in Game_2' has a nonnegligible ϵ difference from that of it guessing it correctly in Game_3. However, it is in Game_3 if and only if the challenger gave the simulator $Q' = R$ instead of $Q' = Q$. Therefore, the simulator has advantage ϵ in the Bilinear Subgroup Decision game, implying that the simulator has an advantage of ϵ in the Composite 3-Party Diffie-Hellman game.

B.6 Indistinguishability of Game_3 and Game_4

If a polynomial time adversary $\mathcal{A}$ has nonnegligible difference ϵ between its advantage in Game_3 and Game_4, we can build a simulator $\mathcal{B}$ that breaks the C3DH assumption with probability ϵ.

The challenger first creates a 3-Party challenge:

$$
\begin{aligned}
&(p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda), \quad n \leftarrow pq, \quad g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_q, \quad g_r \xleftarrow{R} \mathbb{G}_r \\
&R_1, R_2, R_3 \xleftarrow{R} \mathbb{G}_q \\
&a, b, c \xleftarrow{R} \mathbb{Z}_n \\
&\bar{Z} \leftarrow \big((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}), \quad g_p,\ g_q, g_r, \quad g_p^a, \quad g_p^b, \quad \Gamma = g_p^{ab} \cdot R_1, \quad Y = g_p^{abc} \cdot R_2\big) \\
&Q \leftarrow g_p^c \cdot R_3
\end{aligned}
$$

It then randomly decides whether to give $(\bar{Z},\ Q' = Q)$ or $(\bar{Z},\ Q' = R)$ where R is a random element in $\mathbb{G}_{pq}$.

We create the following simulation:

Init. The adversary commits to two points X_0^* and X_1^*. The simulator flips a random coin β internally.

Setup. The simulator picks $V = g_p R_v$, where R_v is picked at random from $\mathbb{G}_q$. The simulator also picks from $\mathbb{Z}_n$ random exponents $y, x, \overline{x}, \mu_i, z_i$ for each $i \in [\ell]$, and lets

$$W = g_p^y \cdot \Gamma^x, \quad \overline{W} = \Gamma^{\overline{x}}$$

The simulator creates:

$$\forall i \in [\ell]: \quad U_i = (g_p^b)^{\mu_i} R_{u,i}, \quad H_i = (g_p^b)^{-\mu_i X_{\beta,i}^*} g_p^{z_i} R_{h,i}$$

where $R_{u,i}$ and $R_{h,i}$'s are random group elements from $\mathbb{G}_q$. The simulator also chooses a random $\alpha \in \mathbb{Z}_n$, and computes $A = \mathsf{e}(g_p, V)^\alpha$.

Query 1. Recall that each query σ defines a set of conjunctive queries $\mathcal{C}_\sigma$ on the encrypted point X. Whenever the adversary asks the simulator to reveal a token for σ, σ must satisfy the condition that for any function $f \in \mathcal{C}_\sigma$ (f is a conjunctive query on $X \in \mathbb{Z}_m^\ell$), $f(X_0^*) = f(X_1^*)$. Henceforth, we use the terminology σ does not separate the two selected points X_0^* and X_1^* to denote the above condition.

We now describe how the simulator responds to the adversary's "reveal token" queries. The token can be nondelegated, Type 1 delegated, or Type 2 delegated. Type 1 delegated tokens and nondelegated tokens should be generated freshly at random, while Type 2 tokens should reflect the correct relation with their parent tokens. In Section B.4, we gave an algorithm for generating Type 2 tokens. Hence, it suffices to show how the simulator can compute fresh random tokens.

- If the token matches both selected points, the simulator first picks a random τ from $\mathbb{Z}_n$, and lets $\gamma = -\overline{x}\tau$, and $\overline{\gamma} = x\tau$. Similarly, the simulator picks a random $\tau_i \in \mathbb{Z}_n$ for each $i \in \mathcal{W}(\sigma)$, and lets $\gamma_i = -\overline{x}\tau_i$, and $\overline{\gamma}_i = x\tau_i$. Except for the above, the simulator follows the *GenToken* algorithm and computes the token. Notice that the token can be computed efficiently, since the only unknown term involving g_p^{ab} cancels out because of the way the simulator chose $\gamma, \overline{\gamma}$, and the way the simulator chose γ_i and $\overline{\gamma}_i$'s. In particular, consider the term K in the decryption key component DK. Group the terms in K:

$$K = \left(w^\gamma \overline{w}^{\overline{\gamma}}\right)\left(g^\alpha \prod_{j \in \mathcal{S}(\sigma)} (u_j^{\sigma_j} h_j)^{t_j} Y\right)$$

  In the above, the product term $w^\gamma \overline{w}^{\overline{\gamma}}$ can be efficiently computed since all terms involving g_p^{ab} cancel out:

$$w^\gamma \overline{w}^{\overline{\gamma}} = g_p^{-\overline{x}\tau y}$$

  Similarly, for all $i \in \mathcal{W}(\sigma)$, the following term in the delegation component DL_i can be efficiently computed:

$$w^{\gamma_i} \overline{w}^{\overline{\gamma}_i} = g_p^{-\overline{x}\tau_i y}$$

  Clearly, all remaining terms in DK or DL can be efficiently computed, since the simulator knows all necessary parameters.

- If the token matches neither selected point, there exists coordinate $c \in \mathcal{S}(\sigma)$, such that $\Delta_c = \sigma_c - X_{\beta,c}^* \neq 0$. In this case, the simulator uses the following strategy to compute the decryption key component DK. The simulator first picks random $\gamma, \overline{\gamma} \in \mathbb{Z}_n$. It also picks random $\widehat{t}_c \in \mathbb{Z}_n$, and lets

$$t_c = \widehat{t}_c - \frac{a(\gamma x + \overline{\gamma x})}{\mu_c \Delta_c}$$

  without actually computing t_c. Except for the above, the simulator follows the *GenToken* algorithm to compute the token requested. Notice that the token can be computed efficiently, since all terms involving the unknown parameter g_p^{ab} cancel out. In particular, in the decryption key components DK, group the terms in K:

$$K = \left(w^\gamma \overline{w}^{\overline{\gamma}} (u_c^{\sigma_c} h_c)^{t_c}\right)\left(g^\alpha \prod_{j \in \mathcal{S}(\sigma), j \neq c} (u_j^{\sigma_j} h_j)^{t_j} Y\right)$$

  The product term $w^\gamma \overline{w}^{\overline{\gamma}} (u_c^{\sigma_c} h_c)^{t_c}$ can be efficiently computed, since all terms involving the unknown parameter g_p^{ab} cancel out:

$$w^\gamma \overline{w}^{\overline{\gamma}} (u_c^{\sigma_c} h_c)^{t_c} = g_p^{y\gamma} (u_c^{\sigma_c} h_c)^{\widehat{t}_c} (g_p^a)^{-z_c \Theta_c}$$

where $\Theta_c = \frac{(\gamma x + \overline{\gamma x})}{\mu_c \Delta_c}$. In addition, we observe that the term $K_c = v^{t_c} Y_c$ can be computed efficiently since the simulator knows g_p^a. Clearly, all other terms in DK can be computed efficiently.

To generate the delegation components DL, we can apply the same trick, i.e., by letting

$$s_{i,c} = \widehat{s}_{i,c} - \frac{a(\gamma_i x + \overline{\gamma}_i \overline{x})}{\mu_c \Delta_c}$$

for every $i \in \mathcal{W}(\sigma)$. $\widehat{s}_{i,c}$ is picked at random from $\mathbb{Z}_n$.

Challenge. The adversary submits two messages M_0 and M_1 to the simulator. The simulator creates the following ciphertext:

$$C = Q', \quad C_0 = Q'^{y} Y^x Z_0, \quad C_\phi = Y^{\overline{x}} Z_\phi, \quad \forall i \in [\ell]: \quad C_i = Q'^{z_i} Z_i$$

In addition, if $\mathsf{M}_0 = \mathsf{M}_1$, the simulator lets $\widetilde{C} = \mathsf{e}(g_p, Q')^\alpha$. Otherwise, $\widetilde{C}$ is replaced by a random element from $\mathbb{G}_T$. Observe that if $Q' = Q$, the ciphertext is identically distributed as in Game_3. Otherwise, if Q' is a random element from $\mathbb{G}_{pq}$, the ciphertext is identically distributed as in Game_4.
Query 2. Same as the **Query 1** stage.
Guess. The adversary outputs a guess β' of β. By the C3DH assumption, a poly-time adversary cannot have more than negligible difference in its advantage in Game_3 and Game_4.

B.7 Indistinguishability of Game_4 and Game_5

Let $\overline{E}$ denote the set of indices i where the two committed points are not equal, i.e., $X^*_{0,i} \neq X^*_{1,i}$. Let $\mathsf{Game}_{4,0} := \mathsf{Game}_4$. We define a sequence of games $\mathsf{Game}_{4,1}, \mathsf{Game}_{4,2}, \ldots, \mathsf{Game}_{4,|\overline{E}|}$. Let $\widetilde{E}_i \subseteq \overline{E}$ denote the first i indices in $\overline{E}$. In $\mathsf{Game}_{4,i}$ $(1 \leq i \leq |\overline{E}|)$, the challenger creates ciphertext components $\widetilde{C}, C$, and C_j normally for all $j \notin \widetilde{E}_i$. For all $j \in \widetilde{E}_i$, the challenger replaces C_j with a random group element from $\mathbb{G}_{pq}$. For C_0, C_ϕ, the challenger creates the following ciphertext components like in game Game_4:

$$C_0 = W^\rho g_p^{-\pi\rho'} Z_0, \quad C_\phi = \overline{W}^\rho g_p^{\rho'} Z_\phi$$

where ρ' is a random group element from $\mathbb{Z}_p$. Recall that the simulator picks $\pi \in \mathbb{Z}_p$ at random prior to the game starts, and π is hidden from the adversary. Whenever the adversary makes a query that matches both selected points, the simulator picks the exponents for w and $\overline{w}$ in a correlated way such that $\overline{\gamma} = \pi\gamma$, $\overline{\gamma}_i = \pi\gamma_i$ for all $i \in \mathcal{W}(\sigma)$. It is not hard to see that $\mathsf{Game}_{4,|\overline{E}|} = \mathsf{Game}_5$.

We now prove Lemma B.5, and show that a poly-time adversary cannot have more than negligible difference in its advantage in Game_4 and Game_5. Because of the hybrid argument, it suffices to show that $\mathsf{Game}_{4,d}$ is computationally indistinguishable from $\mathsf{Game}_{4,d+1}$, where $0 \leq d < |\overline{E}|$.

We prove this by supposing that a poly-time adversary $\mathcal{A}$ has more than negligible difference in its advantage against $\mathsf{Game}_{4,d}$ and $\mathsf{Game}_{4,d+1}$. Now we build a simulator $\mathcal{B}$ that leverages $\mathcal{A}$ to solve the C3DH problem.

The challenger first creates a 3-Party challenge:

$$(p, q, r, \mathbb{G}, \mathbb{G}_T, \mathsf{e}) \xleftarrow{R} \mathsf{GG}(\lambda), \quad n \leftarrow pq, \quad g_p \xleftarrow{R} \mathbb{G}_p, \quad g_q \xleftarrow{R} \mathbb{G}_q, \quad g_r \xleftarrow{R} \mathbb{G}_r$$
$$R_1, R_2, R_3 \xleftarrow{R} \mathbb{G}_q$$
$$a, b, c \xleftarrow{R} \mathbb{Z}_n$$
$$\bar{Z} \leftarrow \big((n, \mathbb{G}, \mathbb{G}_T, \mathsf{e}), \quad g_p, \ g_q, g_r, \quad g_p^a, \quad g_p^b, \quad \Gamma = g_p^{ab} \cdot R_1, \quad Y = g_p^{abc} \cdot R_2\big)$$
$$Q \leftarrow g_p^c \cdot R_3$$

It then randomly decides whether to give $(\bar{Z}, \ Q' = Q)$ or $(\bar{Z}, \ Q' = R)$ where R is a random element in $\mathbb{G}_{pq}$.

We create the following simulation:

Init. The adversary commits two points to the simulator, X_0^* and X_1^*. The challenger flips a random coin β internally.

Setup. Let δ denote the $d+1$-th index in $\overline{E}$.

The simulator first chooses random $(R_{u,1}, R_{h,1}), \ldots, (R_{u,\ell}, R_{h,\ell}) \in \mathbb{G}_q^3$ and random $\mu_1, y_1, \ldots, \mu_\ell, y_\ell \in \mathbb{Z}_n$.

The simulator first publishes the group description and $g_q, g_r, V = g_p R_v$, where R_v is a random element from subgroup $\mathbb{G}_r$. It picks a random $\alpha \in \mathbb{Z}_n$ and lets $A = \mathsf{e}(V, g_p)^\alpha$. It creates

$$U_\delta = g_p^{\mu_\delta} R_{u,\delta}, \quad H_\delta = g_p^{-\mu_\delta X^*_{\beta,\delta}} \Gamma^{y_\delta} R_{h,\delta}$$

Next, for all $i \neq \delta$ it creates

$$U_i = g_p^{\mu_i} R_{u,i}, \quad H_i = g_p^{-\mu_i X^*_{\beta,i}} g_p^{y_i} R_{h,i}$$

Finally, the simulator picks random $R_w, R_{\overline{w}}$ from $\mathbb{G}_q$, and random exponents $x, y, \overline{y}$ from $\mathbb{Z}_n$, and computes

$$W = g_p^x (g_p^b)^y R_w, \quad \overline{W} = (g_p^b)^{\overline{y}} R_{\overline{w}}$$

We observe that the parameters are distributed identically to the real scheme.

The simulator also sets $\pi = -y/\overline{y}$. We observe that π is information theoretically hidden from the adversary.

Query 1. Whenever the simulator receives a "reveal token" query from the adversary, it needs to compute a token of the appropriate form and return it to the adversary. The token that the adversary is requesting can be one of the following three cases: 1) nondelegated, 2) Type 1 delegated, 3) Type 2 delegated. Recall that the simulator generates Type 1 and nondelegated tokens freshly at random. Meanwhile, Section B.4 provides an algorithm for generating Type 2 tokens. It suffices now to show how to generate tokens freshly at random.

Consider that the simulator has received a query from the adversary for a nondelegated token or a Type 1 delegated token σ. Recall that σ should not separate the two committed points X_0^* and X_1^*. Hence, exactly one of the following two cases must be true. Let E denote the set of indices i where the two committed points are equal, i.e., $X^*_{0,i} \neq X^*_{1,i}$, and $\overline{E} = [\ell] \backslash E$ denote the set of indices where X_0^* and X_1^* are not equal.

Case 1. $\delta \notin \mathcal{S}(\sigma) \cup \mathcal{W}(\sigma)$.

Case 2. $\delta \in \mathcal{S}(\sigma) \cup \mathcal{W}(\sigma)$. There must exist $i, j \in \mathcal{S}(\sigma)$, such that $\sigma_i \neq X_0^*$ and $\sigma_j \neq X_1^*$. In other words, the query σ does not match either of the committed identities.

Case 1. In Case 1, $\delta \notin \mathcal{S}(\sigma) \cup \mathcal{W}(\sigma)$. The simulator checks if the requested token matches both selected points. If so, the simulator picks correlated exponents for w and $\overline{w}$: $\overline{\gamma} = \pi\gamma$, and $\overline{\gamma}_i = \pi\gamma_i$ for all $i \in \mathcal{W}(\sigma)$. (Recall that the simulator sets $\pi = -y/\overline{y}$.) The simulator proceeds to generate the remaining parts of the token according to the *GenToken* algorithm. Otherwise, if the requested token matches neither of the selected points, the simulator simply follows the *GenToken* algorithm to generate the token. It is not hard to see that the token can be efficiently computed in this case, since the simulator knows u_i, h_i for all $i \neq \delta$, as well as other parameters needed.

Case 2. This is the more complicated case, since the simulator does not know h_δ which contains the term g_p^{ab}. Also, in this case, the token queried does not match either of the selected points. Therefore, the simulator will leverage w and $\overline{w}$ to cancel out the unknown parameters in h_δ.

We first describe how to generate the decryption key component DK. If $\delta \notin \mathcal{S}(\sigma)$, then it is trivial for the simulator to generate DK, since the unknown parameter h_δ does not appear in DK, and the

simulator knows all parameters required. If $\delta \in \mathcal{S}(\sigma)$ the simulator picks $t_\delta, \overline{\gamma}' \in \mathbb{Z}_n$ at random, and lets $\overline{\gamma}$ be the following without actually computing it.

$$\overline{\gamma} = \overline{\gamma}' - a t_\delta y_\delta / \overline{y}$$

Now the simulator follows the *GenToken* algorithm to generate remaining parts of the decryption key DK. DK can be efficiently computed, even though the simulator does not know g_p^{ab}, as all terms involving g_p^{ab} cancel out in DK. In particular, consider the term K in DK. Group the terms in K:

$$K = \left(\overline{w}^{\overline{\gamma}} (u_\delta^{\sigma_\delta} h_\delta)^{t_\delta} \right) \left(g^\alpha \prod_{j \in \mathcal{S}(\sigma), j \neq \delta} (u_j^{\sigma_j} h_j)^{t_j} Y \right)$$

The product term $\overline{w}^{\overline{\gamma}} (u_\delta^{\sigma_\delta} h_\delta)^{t_\delta}$ can be efficiently computed since all terms involving g_p^{ab} cancel out:

$$\overline{w}^{\overline{\gamma}} (u_\delta^{\sigma_\delta} h_\delta)^{t_\delta} = (g_p^b)^{\overline{y}\overline{\gamma}'} g_p^{\mu_\delta \Delta_\delta t_\delta}$$

where $\Delta_\delta = \sigma_\delta - X^*_{\beta,\delta}$. Meanwhile, the term $K_\phi = v^{\overline{\gamma}} Y_\phi$ can be efficiently computed since the simulator knows g_p^a. It is not hard to see that all remaining terms in DK can be efficiently computed.

We show how to generate the delegation components. The simulator can use exactly the same strategy to generate DL. Basically, for all $i \in \mathcal{W}(\sigma)$, the simulator picks $s_{i,\delta}, \overline{\gamma}'_i \in \mathbb{Z}_n$ at random, and lets $\overline{\gamma}_i$ be the following without actually computing it:

$$\overline{\gamma}_i = \overline{\gamma}'_i - a s_{i,\delta} y_\delta / \overline{y}$$

In this way, depending on whether $\delta \in \mathcal{S}(\sigma)$ or $\delta \in \mathcal{W}(\sigma)$ the product $\overline{w}^{\overline{\gamma}_i} (u_\delta^{\sigma_\delta} h_\delta)^{s_{i,\delta}}$ or $\overline{w}^{\overline{\gamma}_i} h_\delta^{s_{i,\delta}}$ can be efficiently computed, since terms involving g_p^{ab} cancel out.

Challenge. The adversary submits two messages M_0 and M_1. Let $\overline{E}$ denote the set of indices i such that $X^*_{0,i} \neq X^*_{1,i}$. Let $\widetilde{E}_d$ denote the first d indices in $\overline{E}$. The simulator picks random $P \in \mathbb{G}_{pq}$, $Z_0, Z_\phi \in \mathbb{G}_q$, and $Z_i \in \mathbb{G}_q$ for all $i \in [\ell]$. The simulator creates the following ciphertext:

$$C = Q', \quad C_0 = Q'^x P^y Z_0, \quad C_\phi = P^{\overline{y}} Z_\phi, \quad C_\delta = Y^{y_\delta} Z_\delta, \quad \forall i \neq \delta \text{ and } i \notin \widetilde{E}_d : \quad C_i = Q'^{y_i} Z_i$$

For all $i \in \widetilde{E}_d$, the simulator picks a random element in $\mathbb{G}_{pq}$ for C_i. In addition, if $\mathsf{M}_0 = \mathsf{M}_1$, the simulator computes $\widetilde{C} = \mathsf{e}(g_p, Q')^\alpha$; otherwise, the simulator replaces $\widetilde{C}$ with a random element from $\mathbb{G}_T$. Notice that if $Q' = Q$, then the above simulation is identically distributed as $\mathsf{Game}_{4,d}$. Otherwise, if $Q' = R$, the simulation is identically distributed as $\mathsf{Game}_{4,d+1}$.

Query 2. Same as phase **Query 1**.

Guess. The adversary outputs a guess β' of β. If the adversary guesses correctly, i.e., $\beta' = \beta$, the simulator guesses that $Q' = Q$ in the C3DH instance. Otherwise, the simulator guesses that $Q' = R$. It is not hard to see that any advantage of the adversary in distinguishing β translates to the simulator's advantage in solving the C3DH problem.

C Background on Composite Order Bilinear Groups

Let GG be an algorithm called a *group generator*. Algorithm GG 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 e is a function $\mathsf{e} : \mathbb{G}^2 \to \mathbb{G}_T$ satisfying the following properties:

- (Bilinear) $\forall u, v \in \mathbb{G}$, $\forall a, b \in \mathbb{Z}$, $\mathsf{e}(u^a, v^b) = \mathsf{e}(u, v)^{ab}$.

- (Nondegenerate) $\exists g \in \mathbb{G}$ such that $\mathsf{e}(g, g)$ has order n in $\mathbb{G}_T$.

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

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 $\mathbb{G}_T$.

D dHVE Full Security

We formally define the security of dHVE through the following security game between a challenger and an adversary.

- **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 challenger answers these queries accordingly.
- **Challenge.** The adversary outputs two pairs $(\mathsf{M}_0, X_0), (\mathsf{M}_1, X_1) \in \{0,1\}^* \times \Sigma^\ell$ subject to the following constraints:

  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.

  1. For all $f \in \mathcal{C}_\sigma$, $f(X_0) = f(X_1)$.
  2. If $\exists f \in \mathcal{C}_\sigma$, $f(X_0) = f(X_1) = 1$, then $\mathsf{M}_0 = \mathsf{M}_1$.

  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 should satisfy the same condition as above.
- **Guess.** The adversary outputs a guess b' of b.

As before, 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 secure if for all polynomial time adversaries, its advantage in the above game is a negligible function of λ.

E Anonymous Hierarchical Identity-Based Encryption with Short Private Keys

In Section 2.2, we propose a new and complete security definition for delegation in these (anonymous) IBE systems. By contrast, previously, researchers have used an under-specified security game, where the adversary does not get to specify how each queried token is derived. We now show one advantage of being able to capture such nuances in our security definition, by giving an Anonymous Hierarchical Identity-Based Encryption (AHIBE) construction with shorter private keys than the original construction by Boyen and Waters [10].

To achieve this, we rely on the same technique that we use for our dHVE construction: we multiply the private keys by random group elements in the third subgroup $\mathbb{G}_r$, so that the private keys are computationally indistinguishable from being picked freshly at random.

For consistency, we build our AHIBE scheme based on composite bilinear groups and the C3DH assumption, rather than the Decisional Linear assumption adopted by the original BW construction. One can easily build the scheme using the Decisional Linear assumption as well.

In comparison, the original BW construction has $O(D^2)$ private key size and our construction has $O(D)$ private key size, where D denotes the depth of the hierarchy. Meanwhile, we preserve all other costs asymptotically, including ciphertext size, encryption cost, and decryption cost.

E.1 Construction

Setup$(1^\lambda, D)$: The setup algorithm takes as input a security parameter a^λ, the maximum depth $D \in \mathbb{N}$, and outputs public parameters PK and the corresponding master secret key MSK. 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 a random $g, v \in \mathbb{G}_p, g_q \in \mathbb{G}_q, g_r \in G_r$, a random exponent $\alpha \in \mathbb{Z}_p$, and random elements

$$\forall n \in [0, D+1], \forall \ell \in [0, D]: \quad u_{n,\ell} \xleftarrow{R} \mathbb{G}_p$$

It keeps all the above as the master secret key MSK. The *Setup* algorithm then chooses the following blinding factors in $\mathbb{G}_q$:

$$R_v, \quad \forall n \in [0, D+1], \forall \ell \in [0, D]: \quad R_{n,\ell} \xleftarrow{R} \mathbb{G}_q$$

The algorithm now publishes the following as the public key PK.

$$g_q,\ g_r,\ V = vR_v, \quad A = \mathsf{e}(g, v)^\alpha, \quad \forall n \in [0, D+1], \forall \ell \in [0, D]: \quad U_{n,\ell} = u_{n,\ell} R_{n,\ell}$$

Extract$(\mathsf{PK}, \mathsf{MSK}, \mathcal{I})$: The *Extract* algorithm takes as input the public key PK, the master secret key MSK, and an ID tuple $\mathcal{I} = (I_0, I_1, \ldots, I_L) \in (\mathbb{Z}_p^\times)^{1+L}$, where $L \in [D]$, and by convention, $I_0 = 1$. The algorithm generates a private key corresponding to the identity $\mathcal{I}$.

- Pick random exponents $r_0, r_1, \ldots, r_{1+D}$ from $\mathbb{Z}_p$. Pick random blinding factors $Y, Y_0, Y_1, \ldots, Y_{1+D}$ from $\mathbb{G}_r$, and random $Y'_{1+L}, Y'_{2+L}, \ldots, Y'_D$ from $\mathbb{G}_r$.
- Compute the decryption key portion of the private key:

$$\mathsf{DK} = \left(K = g^\alpha \prod_{n=0}^{1+D} \prod_{\ell=0}^{L} (u_{n,\ell}^{I_n})^{r_n} \cdot Y, \quad \forall n \in [0,\ 1+D]: \quad K_n = v^{r_n} Y_n \right)$$

- Compute the following delegation components of the decryption key:

$$\mathsf{DL} = \left(\forall \ell \in [1+L, D]: \quad J_\ell = \prod_{n=0}^{1+D} u_{n,\ell}^{r_n} \cdot Y'_\ell \right)$$

Derive$(\mathsf{PK}, \mathsf{Pvk}_{\mathcal{I}|L-1}, \mathcal{I})$ The *Derive* algorithm takes as input the public key PK, and derives a private key for $\mathcal{I} = (I_0, I_1, \ldots, I_L)$ from a parent key for $\mathcal{I}|L-1 := (I_0, I_1, \ldots, I_{L-1})$.

- First, express the parent key using the same notation as before: $\mathsf{Pvk}_{\mathcal{I}|L-1} = (\mathsf{DK}, \mathsf{DL})$, where $\mathsf{DK} = (K, K_0, K_1, \ldots, K_{1+D})$, and $\mathsf{DL} = (J_L, J_{1+L}, \ldots, J_D)$.
- Next, pick a random exponent $\tau \in \mathbb{Z}_n$, and random blinding factors $Y, Y_0, \ldots, Y_{1+D}$, and $Y'_{1+L}, \ldots, Y'_D$ from $\mathbb{G}_r$.
- Compute the decryption key portion of the child key:

$$\mathsf{DK}' = \left(K' = (K \cdot J_L^{I_L})^\tau Y, \quad \forall n \in [0,\ 1+D]: \quad K'_n = K_n^\tau Y_n \right)$$

- Compute the delegation components of the child key:

$$\mathsf{DL}' = \left(\forall \ell \in [1+L, D]: \quad J'_\ell = J_\ell^\tau Y'_\ell \right)$$

Encrypt$(\mathsf{PK}, \mathcal{I}, \mathsf{M})$ The *Encrypt* algorithm takes a public key PK, and encrypts a message M to an identity $\mathcal{I} = (I_0, I_1, \ldots, I_L)$. The algorithm proceeds as follows:

- Pick a random exponent $s \in \mathbb{Z}_n$. Pick random blinding factors $Z, Z_0, Z_1, \ldots, Z_{1+D}$ from $\mathbb{G}_q$.
- Compute the following ciphertext:

$$\mathsf{CT} = \left(\widetilde{C} = \mathsf{M} A^s, \quad C = V^s Z, \qquad \forall n \in [0,\ 1+D] : \quad C_n = (\prod_{\ell=0}^{L} U_{n,\ell}^{I_\ell})^s Z_n \right)$$

Decrypt$(\mathsf{PK}, \mathsf{Pvk}_{\mathcal{I}}, \mathsf{CT})$ The *Decrypt* algorithm takes a public key PK, a private key $\mathsf{Pvk}_{\mathcal{I}}$, and decrypts a ciphertext CT. Using the same notation for the ciphertext and the private key as before, decrypt the message:

$$\widehat{\mathsf{M}} \leftarrow \frac{\widetilde{C} \cdot \prod_{n=0}^{1+D} \mathsf{e}(C_n, K_n)}{\mathsf{e}(C, K)}$$

E.2 Security of construction

Theorem E.1. *The above-defined A-HIBE construction is internally consistent. In addition, it is IND-sID-CPA and ANON-sID-CPA secure under the cBDH and C3DH assumptions in the bilinear group* $\mathbb{G}$.

See the original BW paper [10] for detailed definitions of IND-sID-CPA and ANON-sID-CPA security.

The proof of the consistency is straightforward. Proof of security can be done in the following steps:

- As we multiply all elements of the private key with a random group element from the third subgroup $\mathbb{G}_r$, we can show that private keys generated by the *Derive* algorithm are computationally indistinguishable from being picked freshly at random.
- Show that if private keys were really generated freshly at random rather than by calling the *Derive* algorithm, the scheme would be IND-sID-CPA and ANON-sID-CPA secure. This part of the proof is done in a manner similar to that of the BW construction [10]. The only exception is that we now replace the the Decisional Linear assumption by the C3DH assumption. However, the gist of the proof remains unchanged.

We omit the complete proof in this paper, since it is very similar to the proof of our dHVE construction.