Function-Hiding Inner Product Encryption

Allison Bishop[1] [⋆], Abhishek Jain[2] [⋆⋆], and Lucas Kowalczyk[3] [⋆⋆⋆]

[1] Columbia University, allison@cs.columbia.edu
[2] Johns Hopkins University, abhishek@cs.jhu.edu
[3] Columbia University, luke@cs.columbia.edu

Abstract. We extend the reach of functional encryption schemes that are provably secure under simple assumptions against unbounded collusion to include function-hiding inner product schemes. Our scheme is a private key functional encryption scheme, where ciphertexts correspond to vectors $\vec{x}$, secret keys correspond to vectors $\vec{y}$, and a decryptor learns $\langle \vec{x}, \vec{y} \rangle$. Our scheme employs asymmetric bilinear maps and relies only on the SXDH assumption to satisfy a natural indistinguishability-based security notion where arbitrarily many key and ciphertext vectors can be simultaneously changed as long as the key-ciphertext dot product relationships are all preserved.

1 Introduction

Functional encryption (FE) [25, 8, 23] is an exciting paradigm for non-interactively computing on encrypted data. In a functional encryption scheme for a family $\mathcal{F}$, it is possible to derive "special-purpose" decryption keys K_f for any function $f \in \mathcal{F}$ from a master secret key. Given such a decryption key K_f and an encryption of some input x, a user should be able to learn $f(x)$ and nothing else about x.

A driving force behind FE has been to understand what class of functions $\mathcal{F}$ can be supported and what notions of security can be achieved. In terms of functionality, research in FE started with the early works on attribute-based encryption [25, 16], progressively evolving to support more expressive classes of functions, leading to the state of art works that are now able to support computation of general polynomial-size circuits [24, 15, 14, 11]. In terms of security, most of the prior work in this area focuses on the privacy of (encrypted) *messages* (see, e.g., [8, 23, 10] for various security definitions considered in the literature for message privacy).

In many application scenarios, however, it is important to also consider privacy of the *function* being computed. Consider the following motivating example: suppose a hospital subscribes to a cloud service provider to store medical records of its patients. To protect the privacy of the data, these records are stored in an encrypted form. At a later point in time, the hospital can request the cloud to

[⋆] Supported in part by NSF CNS 1413971 and NSF CCF 1423306.
[⋆⋆] Supported in part by a DARPA/ARL Safeware Grant W911NF-15-C-0213 and NSF CNS-1414023.
[⋆⋆⋆] Supported by an NSF Graduate Research Fellowship DGE-11-44155

perform some analysis on the encrypted records by releasing a decryption key K_f for a function f of its choice. If the FE scheme in use does not guarantee any hiding of the function (which is the case for many existing FE schemes), then the key K_f might reveal f completely to the cloud, which is undesirable when f itself contains sensitive information.

This has motivated the study of function privacy in FE, starting with the work of Shen et al. [26], and more recently by [6, 7, 2, 9]. Intuitively speaking, function privacy requires that given a decryption key K_f for a function f, one should not be able to learn any unnecessary information about f. Using the analogy to secure computation, function private FE can be seen as the non-interactive analogue of private function evaluation (which guarantees the privacy of both the input x and the function f being computed on x) just like standard FE can be seen as the non-interactive analogue of secure function evaluation (which only guarantees privacy of the input x). One may also observe that the notion of function privacy is similar in spirit to program obfuscation [5, 11]. Indeed, in the public-key setting, function private FE, in fact, implies program obfuscation.[4] In the secret-key setting, however, no such implication is known.

In this work, we continue the study of function privacy in FE. In particular, we focus on the inner product functionality $\mathcal{IP}$: a function $\mathrm{IP}_{\vec{y}} \in \mathcal{IP}$ in this function family is parametrized by a vector $\vec{y}$ in the finite field $\mathbb{Z}_p$. On an input $\vec{x} \in \mathbb{Z}_p$, $\mathrm{IP}_{\vec{y}}(\vec{x}) = \langle \vec{x}, \vec{y} \rangle$, where $\langle \vec{x}, \vec{y} \rangle$ denotes the inner product $\sum_{i=1}^{n} x_i y_i \in \mathbb{Z}_p$. Inner product is a particularly useful function for statistical analysis. In particular, in the context of FE, (as shown by [17]) it enables computation of conjunctions, disjunctions, CNF/DNF formulas, polynomial evaluation and exact thresholds.

Prior work on FE for inner product can be cast into the following two categories:

– *Generic constructions*: By now, we have a large sequence of works [24, 15, 14, 10, 11, 27, 12, 4] on FE that support computation of general circuits. Very recently, Brakerski and Segev [9] give a general transformation from any FE scheme for general circuits into one that achieves function privacy. Then, by combining [9] with the aforementioned works, one can obtain a function-private FE scheme for inner product as a special case.
  We note, however, that these generic FE constructions use heavy-duty tools for secure computation (such as fully-homomorphic encryption [13] and program obfuscation [5, 11]) and are therefore extremely inefficient. Furthermore, in order to achieve collusion-resistance – one of the central goals in FE since its inception – the above solution would rely on indistinguishability obfuscation [5, 11], which is a strong assumption.
– *Direct constructions*: To the best of our knowledge, the only "direct" construction of FE for inner product that avoids the aforementioned expensive tools is due to the recent work of Abdalla et al. [1]. Their work, however, does not consider function privacy.

[4] Here, the security definition for function privacy determines the security notion of program obfuscation that we obtain. See, e.g., [6] for further discussion on this connection.

We clarify that our formulation of inner product FE is different from that considered in the works of [17, 26, 18, 3, 21, 22, 7]. Very briefly, these works study inner product in the context of predicate encryption, where a message m is encrypted along with a tag $\vec{x}$ and decryption with a key K_y yields m iff $\langle \vec{x}, \vec{y} \rangle = 0$. In contrast, as discussed above, we are interested in learning the actual inner product value (in $\mathbb{Z}_p$).

In summary, the state of the art leaves open the problem of constructing a collusion-resistant, function-private FE scheme for inner product from standard assumptions. We stress that unless we put some restrictions on the distribution of the messages (as in the work of [6, 7]), this question only makes sense in the secret-key setting.

Our Results. In this work, we resolve this open problem. Specifically, we construct a function-private secret-key FE scheme for the inner product functionality that supports any arbitrary polynomial number of key queries and message queries. Our construction makes use of asymmetric bilinear maps and is significantly more efficient than the generic solutions discussed earlier. The security notion we prove for our construction is a natural indistinguishability-based notion, and we establish it under the Symmetric External Diffie-Hellman Assumption (SXDH). To obtain correctness for our scheme, we assume that inner products will be contained in a polynomially-sized range. This assumption is quite reasonable for statistical applications, where the average or count of some bounded quantity over a polynomially-sized database will naturally be in a polynomial range.

Our Techniques. We begin with the basic idea for inner product encryption developed in [17], which is the observation that one can place two vectors in the exponents on opposite sides of a bilinear group and compute the dot product via the pairing. This already provides some protection for the vectors as discrete log is thought to be hard in these groups, but without further randomization, this is vulnerable to many attacks, such as guessing the vector or learning whether two coordinates of a vector are the same. For this reason, the construction in [17] multiplies each of the exponent vectors by a random scalar value and uses additional subgroups in a composite order group to supply more randomization. The use of composite order groups in this way is by no means inherent, as subsequent works [18, 21, 22] (for example) demonstrate how to supply a sufficient amount of randomization in prime order bilinear groups using dual pairing vector spaces. However, in all of these schemes, the random scalars prevent a decryptor from learning the actual value of the inner product. Of course this is intentional and required, as these are predicate encryption schemes where the prescribed functionality only depends on whether the inner product is zero or nonzero. To adapt these methods to allow a decryptor to learn the inner product, we must augment the construction with additional group elements that produce the same product of scalars in the exponent. Then the decryptor can produce the value by finding a ratio between the exponents of two group elements. This will be efficiently computable when the value of the inner product is in a known polynomially-sized range. Crucially, we must prove that these additional group elements do not reveal any unintended information.

We note that the construction in [17] is not known to be function-hiding. And since we are further allowing the inner product itself to be learned, function-hiding for our scheme means something different than function-hiding for schemes such as [17]. In particular, function hiding for a public key scheme in our setting would be impossible: one could simply create ciphertexts for a basis of vectors and test decryption of one's key against all of them to fully reconstruct the vector embedded in the key. It is thus fundamental that the public key scheme in [1] is not function-hiding. Indeed, their secret keys include vectors given in the clear and have no hiding properties.

To prove function-hiding for our construction, we thus leverage our private key setting to obtain a perfect symmetry between secret keys and ciphertexts, both in our construction and in our security reduction. Since no public parameters for encryption need to be published, the same techniques that we use to hide the underlying vectors in the ciphertexts can be flipped to argue that function-hiding holds for the secret keys.

The core of our security argument is an information-theoretic step (in the setting of dual pairing vector spaces as introduced by Okamoto and Takashima [19, 20]). Essentially, our master secret key consists of two dual orthonormal bases that will be employed in the exponents to encode the vectors for ciphertexts and secret keys respectively. Secret keys and ciphertexts thus correspond to linear combinations of these basis vectors in the exponent. Since the bases themselves are never made public, if all of the secret keys (for example) are orthogonal to a particular vector, then there is a hidden "dimension" in the bases that can be used to argue that the ciphertext vector can be switched to another vector that has the same inner products with the provided keys. In fact, if we did not want any function privacy and instead only wanted to hide whether a single ciphertext corresponded to a vector $\vec{x}_0$ or $\vec{x}_1$ while giving out secret keys for vectors $\vec{y}$ orthogonal to $\vec{x}_0 - \vec{x}_1$, then we would do this information-theoretically. When we instead have many ciphertexts and we also demand function privacy for the keys, we use a hybrid argument, employing various applications of the SXDH assumption to move things around in the exponent bases and isolate a single ciphertext or key in a particular portion of the bases to apply our information-theoretic argument. The symmetry between keys and ciphertexts in our construction allows us to perform the same hybrid argument to obtain function privacy as in the case of multiple-ciphertext security.

2 Preliminaries

2.1 Functional Encryption Specifications and Security Definitions

In the rest of this paper, we will consider a specialization of the general definition of functional encryption to the particular functionality of computing dot products of n-length vectors over a finite field $\mathbb{Z}_p$. A private key functional encryption scheme for this class of functions will have the following PPT algorithms:

Setup$(1^\lambda, n) \to \mathrm{PP}, \mathrm{MSK}$ The setup algorithm will take in the security parameter λ and the vector length parameter n (a positive integer that is polynomial in λ). It will produce a master secret key MSK and public parameters PP. (Note that this is not a public key scheme, so the PP are not sufficient to encrypt - they are just parameters that do not need to be kept secret.)

Encrypt$(\mathrm{MSK}, \mathrm{PP}, \vec{x}) \to \mathrm{CT}$ The encryption algorithm will take in the master secret key MSK, the public parameters PP, and a vector $\vec{x} \in \mathbb{Z}_p^n$. It produces a ciphertext CT.

KeyGen$(\mathrm{MSK}, \mathrm{PP}, \vec{y}) \to \mathrm{SK}$ The key generation algorithm will take in the master secret key MSK, the public parameters PP, and a vector $\vec{y} \in \mathbb{Z}_p^n$. It produces a secret key SK.

Decrypt$(\mathrm{PP}, \mathrm{CT}, \mathrm{SK}) \to m \in \mathbb{Z}_p \ or \perp$ The decryption algorithm will take in the public parameters PP, a ciphertext CT, and a secret key SK. It will output either a value $m \in \mathbb{Z}_p$ or $\perp$.

For correctness, we will require the following. We suppose that $\mathrm{PP}, \mathrm{MSK}$ are the result of calling $\mathrm{Setup}(1^\lambda, n)$, and CT, SK are then the result of calling $\mathrm{Encrypt}(\mathrm{MSK}, \mathrm{PP}, \vec{x})$ and $\mathrm{KeyGen}(\mathrm{MSK}, \mathrm{PP}, \vec{y})$ respectively. We then require that the output of $\mathrm{Decrypt}(\mathrm{PP}, \mathrm{CT}, \mathrm{SK})$ must be either $m = \langle \vec{x}, \vec{y} \rangle$ or $\perp$. We will only require that it is $\langle \vec{x}, \vec{y} \rangle$ and not $\perp$ when $\langle \vec{x}, \vec{y} \rangle$ is from a fixed polynomial range of values inside $\mathbb{Z}_p$, as this will allow a decryption algorithm to compute it as a discrete log in a group where discrete log is generally hard.

Security We will consider an indistinguishability-based security notion defined by a game between a challenger and an attacker. At the beginning of the game, the challenger calls $\mathrm{Setup}(1^\lambda, n, B)$ to produce $\mathrm{MSK}, \mathrm{PP}$. It gives PP to the attacker. The challenger also selects a random bit b.

Throughout the game, the attacker can (adaptively) make two types of a queries. To make a *key query*, it submits two vectors $\vec{y}^0, \vec{y}^1 \in \mathbb{Z}_p^n$ to the challenger, who then runs $\mathrm{KeyGen}(\mathrm{MSK}, \mathrm{PP}, \vec{y}^b)$ and returns the resulting SK to the attacker. To make a *ciphertext query*, the attacker submits two vectors $\vec{x}^0, \vec{x}^1 \in \mathbb{Z}_p^n$ to the challenger, who then runs $\mathrm{Encrypt}(\mathrm{MSK}, \mathrm{PP}, \vec{x}^b)$ and returns the resulting ciphertext to the attacker. The attacker can make any polynomial number of key and ciphertext queries throughout the game. At the end of the game, the attacker must submit a guess b' for the bit b. We require that for all key queries $\vec{y}^0, \vec{y}^1$ and all ciphertext queries $\vec{x}^0, \vec{x}^1$, it must hold that

$$\langle \vec{y}^0, \vec{x}^0 \rangle = \langle \vec{y}^0, \vec{x}^1 \rangle = \langle \vec{y}^1, \vec{x}^0 \rangle = \langle \vec{y}^1, \vec{x}^1 \rangle$$

The attacker's advantage is defined to be the probability that $b' = b$ minus $\frac{1}{2}$.

Definition 1. *We say a private key functional encryption scheme for dot products over $\mathbb{Z}_p^n$ satisfies function-hiding indistinguishability-based security if any PPT attacker's advantage in the above game is negligible as a function of the security parameter λ.*

Remark 1. We note that the attacker can trivially win the security game if we allowed a key query $\vec{y}^0, \vec{y}^1$ and ciphertext query $\vec{x}^0, \vec{x}^1$ such that $\langle \vec{y}^0, \vec{x}^0 \rangle \neq \langle \vec{y}^1, \vec{x}^1 \rangle$. Our stronger requirement that $\langle \vec{y}^0, \vec{x}^1 \rangle$ and $\langle \vec{y}^1, \vec{x}^0 \rangle$ is used for our hybrid security proof, but it might be possible to remove it by developing different proof techniques.

2.2 Asymmetric Bilinear Groups

We will construct our scheme in aymmetric bilinear groups. We let $\mathcal{G}$ denote a group generator - an algorithm which takes a security parameter λ as input and outputs a description of prime order groups G_1, G_2, G_T with a bilinear map $e : G_1 \times G_2 \to G_T$. We define $\mathcal{G}$'s output as (p, G_1, G_2, G_T, e), where p is a prime, G_1, G_2 and G_T are cyclic groups of order p, and $e : G_1 \times G_2 \to G_T$ is a map with the following properties:

1. (Bilinear) $\forall g_1 \in G_1, g_2 \in G_2,\ a, b \in \mathbb{Z}_p,\ e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$
2. (Non-degenerate) $\exists g_1 \in G_1, g_2 \in G_2$ such that $e(g_1, g_2)$ has order p in G_T.

We refer to G_1 and G_2 as the source groups and G_T as the target group. We assume that the group operations in G_1, G_2, and G_T and the map e are computable in polynomial time with respect to λ, and the group descriptions of G_1, G_2, and G_T include a generator of each group.

The SXDH Assumption The security of our construction relies on the hardness of the SXDH assumption. Given prime order groups $(p, G_1, G_2, G_T, e) \leftarrow \mathcal{G}(\lambda)$, we define the SXDH problem as distinguishing between the following two distributions:

$$D_1 = (g_1, g_1^a, g_1^b, g_1^{ab}, g_2)$$

and

$$D_2 = (g_1, g_1^a, g_1^b, g_1^{ab+r}, g_2)$$

where g_1, g_2 are generators of G_1, G_2, and $a, b, r \leftarrow \mathbb{Z}_p$.

The SXDH Assumption states that no polynomial-time algorithm can achieve non-negligible advantage in deciding between D_1 and D_2. It also states that the same is true for the analogous distributions formed from switching the roles of G_1, G_2 (that is, $D_1 = (g_2, g_2^a, g_2^b, g_2^{ab}, g_1)$ and $D_2 = (g_2, g_2^a, g_2^b, g_2^{ab+r}, g_1)$)

2.3 Dual Pairing Vector Spaces

In addition to referring to individual elements of G_1 and G_2, we will also consider "vectors" of group elements. For $\vec{v} = (v_1, ..., v_m) \in \mathbb{Z}_p^m$ and $g_1 \in G_1$, we write $g_1^{\vec{v}}$ to denote the m-tuple of elements of G_1:

$$g_1^{\vec{v}} := (g_1^{v_1}, ..., g_1^{v_m})$$

We can also perform scalar multiplication and exponentiation in the exponent. For any $a \in \mathbb{Z}_p$ and $\vec{v}, \vec{w} \in \mathbb{Z}_p^m$, we have:

$$g_1^{a\vec{v}} := (g_1^{av_1}, ..., g_1^{av_m})$$
$$g_1^{\vec{v}+\vec{w}} = (g_1^{v_1+w_1}, ..., g_1^{v_m+w_m})$$

We abuse notation slightly and also let e denote the product of the component wise pairings:

$$e(g_1^{\vec{v}}, g_2^{\vec{w}}) := \prod_{i=1}^{m} e(g_1^{v_i}, g_2^{w_i}) = e(g_1, g_2)^{\langle \vec{v}, \vec{w} \rangle}$$

Here, the dot product is taken modulo p.

Dual Pairing Vector Spaces We will employ the concept of dual pairing vector spaces from [19, 20]. We will choose two random sets of vectors: $\mathbb{B} := \{\vec{b}_1, \ldots, \vec{b}_m\}$ and $\mathbb{B}^* = \{\vec{b}_1^*, \ldots, \vec{b}_m^*\}$ of $\mathbb{Z}_p^m$ subject to the constraint that they are "dual orthonormal" in the following sense:

$$\langle \vec{b}_i, \vec{b}_i^* \rangle = 1 \pmod p \text{ for all } i$$
$$\langle \vec{b}_i, \vec{b}_j^* \rangle = 0 \pmod p \text{ for all } j \neq i.$$

We note that choosing sets $(\mathbb{B}, \mathbb{B}^*)$ at random from sets satisfying these dual orthonormality constraints can be realized by choosing a set of m vectors $\mathbb{B}$ uniformly at random from $\mathbb{Z}_p^m$ (these vectors will be linearly independent with high probability), then determining each vector of $\mathbb{B}^*$ from its orthonormality constraints. We will denote choosing random dual orthonormal sets this way as: $(\mathbb{B}, \mathbb{B}^*) \leftarrow Dual(\mathbb{Z}_p^m)$.

3 Construction

We now present our construction in asymmetric bilinear groups. We will choose dual orthonormal bases $\mathbb{B}$ and $\mathbb{B}^*$ that will be used in the exponent to encode ciphertext and key vectors respectively. Vectors will be encoded twice to create space for a hybrid security proof and will be additionally masked by random scalars (these basic features are also present in [17]). We will use additional dual bases $\mathbb{D}, \mathbb{D}^*$ to separately encode these same scalars in the exponent so that their effect can be removed from the final decryption result. We view it as a core feature of our construction that the structure of keys and ciphertexts in our scheme is perfectly symmetric, just on different sides of dual orthonormal bases. This enables us to prove function hiding for the keys with exactly the same techniques we use to prove indistinguishability security for the ciphertexts.

Setup$(1^\lambda, n), \to \mathrm{MSK}, \mathrm{PP}$ The setup algorithm takes in the security parameter λ and a positive integer n specifying the desired length of vectors for the keys and ciphertexts. It chooses an asymmetric bilinear group consisting of G_1, G_2, G_T, all with prime order p. It fixes generators g_1, g_2 of G_1 and G_2 respectively. It then samples dual orthonormal bases $\mathbb{B}, \mathbb{B}^* \leftarrow Dual(\mathbb{Z}_p^{2n})$ and dual orthonormal bases $\mathbb{D}, \mathbb{D}^* \leftarrow Dual(\mathbb{Z}_p^2)$. It defines the master secret key as $\mathrm{MSK} := \mathbb{B}, \mathbb{B}^*, \mathbb{D}, \mathbb{D}^*$. The groups G_1, G_2, G_T, the generators g_1, g_2, and p are set to be public parameters.

Encrypt$(\mathrm{MSK}, \mathrm{PP}, \vec{x}) \to \mathrm{CT}$ The encryption algorithm takes in the master secret key $\mathbb{B}, \mathbb{B}^*, \mathbb{D}, \mathbb{D}^*$, the public parameters, and a vector $\vec{x} \in \mathbb{Z}_p^n$. It chooses two independent and uniformly random elements $\alpha, \tilde{\alpha} \in \mathbb{Z}_p$. It then computes:

$$C_1 := g_1^{\alpha(x_1\vec{b}_1^* + \cdots + x_n\vec{b}_n^*) + \tilde{\alpha}(x_1\vec{b}_{n+1}^* + \cdots + x_n\vec{b}_{2n}^*)}$$

$$C_2 := g_1^{\alpha\vec{d}_1^* + \tilde{\alpha}\vec{d}_2^*}.$$

The ciphertext $\mathrm{CT} = \{C_1, C_2\}$.

KeyGen$(\mathrm{MSK}, \mathrm{PP}, \vec{y}) \to \mathrm{SK}$ The secret key generation algorithm takes in the master secret key $\mathbb{B}, \mathbb{B}^*, \mathbb{D}, \mathbb{D}^*$, the public parameters, and a vector $\vec{y} \in \mathbb{Z}_p^n$. It chooses two independent and uniformly random elements $\beta, \tilde{\beta} \in \mathbb{Z}_p$. It then computes:

$$K_1 := g_2^{\beta(y_1\vec{b}_1 + \cdots + y_n\vec{b}_n) + \tilde{\beta}(y_1\vec{b}_{n+1} + \cdots + y_n\vec{b}_{2n})}$$

$$K_2 := g_2^{\beta\vec{d}_1 + \tilde{\beta}\vec{d}_2}.$$

The secret key $\mathrm{SK} = \{K_1, K_2\}$.

Decrypt$(\mathrm{PP}, \mathrm{CT}, \mathrm{SK}) \to m \in \mathbb{Z}_p$ *or* $\perp$ The decryption algorithm takes in the public parameters, the ciphertext C_1, C_2, and the secret key K_1, K_2. It computes:

$$D_1 := e(C_1, K_1)$$

$$D_2 := e(C_2, K_2).$$

It then computes an m such that $D_2^m = D_1$ as elements of G_T. It outputs m. We note that we can guarantee that the decryption algorithm runs in polynomial time when we restrict to checking a fixed, polynomially size range of possible values for m and output $\perp$ when none of them satisfy the criterion $D_2^m = D_1$.

Correctness We observe that for a ciphertext formed by calling $\mathrm{Encrypt}(\mathrm{MSK}, \mathrm{PP}, \vec{x})$ and a key formed by calling $KeyGen(\mathrm{MSK}, \mathrm{PP}, \vec{y})$, we have

$$D_1 = e(C_1, K_1) = e(g_1, g_2)^{\alpha\beta\langle\vec{x},\vec{y}\rangle + \tilde{\alpha}\tilde{\beta}\langle\vec{x},\vec{y}\rangle} = e(g_1, g_2)^{(\alpha\beta + \tilde{\alpha}\tilde{\beta})\langle\vec{x},\vec{y}\rangle}$$

$$\text{and } D_2 = e(C_2, K_2) = e(g_1, g_2)^{\alpha\beta + \tilde{\alpha}\tilde{\beta}}.$$

This follows immediately from the definitions of C_1, C_2, K_1, K_2 and the fact that $\mathbb{B}, \mathbb{B}^*$ and $\mathbb{D}, \mathbb{D}^*$ are dual orthonormal bases pairs. Thus, if $\langle\vec{x},\vec{y}\rangle$ is in the polynomial range of possible values for m that the decryption algorithm checks, it will output $m := \langle\vec{x},\vec{y}\rangle$ as desired.

4 Security Proof

Our security proof is structured as a hybrid argument over a series of games which differ in how the ciphertext and keys are constructed. Intuitively, if there were only one ciphertext, we could embed the difference of the two possible ciphertext vectors, namely $\vec{x}^0 - \vec{x}^1$, into the definition of the bases $\mathbb{B}, \mathbb{B}^*$ to argue that this difference is hidden when only key vectors orthogonal to $\vec{x}^0 - \vec{x}^1$ are provided. In other words, there is ambiguity in the choice of $\mathbb{B}, \mathbb{B}^*$ left conditioned on the provided keys, and this can be exploited to switch $\vec{x}^0$ for $\vec{x}^1$. But there is a limited amount of such ambiguity, so to re-purpose it for many ciphertexts, we employ a hybrid argument that isolates each ciphertext in turn in a portion of the basis. Since keys and ciphertexts are constructed and treated symmetrically in our scheme, we can apply the same hybrid argument over keys to prove function hiding, just reversing the roles of $\mathbb{B}$ and $\mathbb{B}^*$.

Notice that a normal ciphertext for a vector $\vec{x}$ contains two parallel copies of $\vec{x}$ in the exponent of C_1: one attached to $\vec{b}_i^*$'s and one attached to $\vec{b}_{n+i}^*$'s for $i = 1, ..., n$. We will refer to this as a type-$(\vec{x}, \vec{x})$ ciphertext. We will use this notation to define a type-$(\vec{0}, \vec{x})$ ciphertext - one which is normally formed but has no $\vec{b}_i^*$ components for $i = 1, ..., n$ and no $\vec{d}_1^*$ component in C_2. We will also use the same terminology to refer to keys (i.e: type-$(\vec{y}, \vec{y})$ / type-$(\vec{0}, \vec{y})$ / type-$(\vec{y}, \vec{0})$ keys).

Letting Q_1 denote the total number of ciphertext queries the attacker makes, we define 7 games for each $j = 0, ..., Q_1$:

Game$^1_{j,Z}$ In Game$^1_{j,Z}$ all ciphertexts before the jth ciphertext are of type-$(\vec{0}, \vec{x}_i^1)$, the jth ciphertext is of type-$(\vec{0}, \vec{x}_i^0)$, all ciphertexts after the jth ciphertext are also type-$(\vec{0}, \vec{x}_i^0)$ ciphertexts, and all keys are of type-$(\vec{y}_i^0, \vec{y}_i^0)$.

Game$^2_{j,Z}$ Game$^2_{j,Z}$ is the same as Game$^1_{j,Z}$ except that the jth ciphertext is now of type-$(\vec{x}_j^0, \vec{x}_j^0)$.

Game$^3_{j,Z}$ Game$^3_{j,Z}$ is the same as Game$^2_{j,Z}$ except that the jth ciphertext is now of type-$(\vec{x}_j^0, \vec{x}_j^1)$.

Game$^4_{j,Z}$ Game$^4_{j,Z}$ is the same as Game$^3_{j,Z}$ except that all ciphertexts before the jth ciphertext are now of type-$(\vec{x}_i^1, \vec{x}_i^1)$ and all ciphertexts after the jth ciphertext are now type-$(\vec{x}_i^0, \vec{x}_i^0)$ ciphertexts.

Game$^5_{j,Z}$ Game$^5_{j,Z}$ is the same as Game$^4_{j,Z}$ except that all ciphertexts before the jth ciphertext are now of type-$(\vec{x}_i^1, \vec{0})$ and all ciphertexts after the jth ciphertext are now type-$(\vec{x}_i^0, \vec{0})$ ciphertexts.

Game$^6_{j,Z}$ Game$^6_{j,Z}$ is the same as Game$^5_{j,Z}$ except that the jth ciphertext is now of type-$(\vec{x}_j^1, \vec{x}_j^1)$.

Game$^7_{j,Z}$ Game$^7_{j,Z}$ is the same as Game$^6_{j,Z}$ except that all ciphertexts before the jth ciphertext are now of type-$(\vec{x}^1_i, \vec{x}^1_i)$ and all ciphertexts after the jth ciphertext are now type-$(\vec{x}^0_i, \vec{x}^0_i)$ ciphertexts.

Letting Q_2 denote the total number of key requests the attacker makes, we define 7 additional games for each $j = 0, ..., Q_2$:

Game$^1_{O,j}$ In Game$^1_{O,j}$ all keys before the jth key are of type-$(\vec{0}, \vec{y}^1_i)$, the jth key is of type-$(\vec{0}, \vec{y}^0_i)$, all keys after the jth key are also type-$(\vec{0}, \vec{y}^0_i)$ keys, and all ciphertexts are of type-$(\vec{x}^1_i, \vec{x}^1_i)$.

Game$^2_{O,j}$ Game$^2_{O,j}$ is the same as Game$^1_{O,j}$ except that the jth key is now of type-$(\vec{y}^0_j, \vec{y}^0_j)$.

Game$^3_{O,j}$ Game$^3_{O,j}$ is the same as Game$^2_{O,j}$ except that the jth key is now of type-$(\vec{y}^0_j, \vec{y}^1_j)$.

Game$^4_{O,j}$ Game$^4_{O,j}$ is the same as Game$^3_{O,j}$ except that all keys before the jth key are now of type-$(\vec{y}^1_i, \vec{y}^1_i)$ and all keys after the jth key are now type-$(\vec{y}^0_i, \vec{y}^0_i)$ keys.

Game$^5_{O,j}$ Game$^5_{O,j}$ is the same as Game$^4_{O,j}$ except that all keys before the jth key are now of type-$(\vec{y}^1_i, \vec{0})$ and all keys after the jth key are now type-$(\vec{y}^0_i, \vec{0})$ keys.

Game$^6_{O,j}$ Game$^6_{O,j}$ is the same as Game$^5_{O,j}$ except that the jth key is now of type-$(\vec{y}^1_j, \vec{y}^1_j)$.

Game$^7_{O,j}$ Game$^7_{O,j}$ is the same as Game$^6_{O,j}$ except that all keys before the jth key are now of type-$(\vec{y}^1_i, \vec{y}^1_i)$ and all keys after the jth key are now type-$(\vec{y}^0_i, \vec{y}^0_i)$ keys.

Note that Game$^7_{0,Z}$ is the real security game played with $b = 0$ and Game$^7_{O,Q_2}$ is the real security game played with $b = 1$. Note also that Game$^7_{Q_1,Z}$ and Game$^7_{O,0}$ are identical.

We will use a hybrid argument to transition between the two to show that no polynomial attacker can achieve non-negligible advantage in the security game (distinguishing between $b = 0$ and $b = 1$.). Our hybrid works in two parts, first transitioning all ciphertexts from type-$(\vec{x}^0_i, \vec{x}^0_i)$ to type-$(\vec{x}^1_i, \vec{x}^1_i)$ (using the Game$^i_{j,Z}$'s), then transitioning all keys from type-$(\vec{y}^0_i, \vec{y}^0_i)$ to type-$(\vec{y}^1_i, \vec{y}^1_i)$ (using the Game$^i_{O,j}$'s).

First we will transition from Game$^7_{0,Z}$ (the real security game played with $b = 0$) to Game$^1_{1,Z}$. We then transition from Game$^1_{1,Z}$ to Game$^2_{1,Z}$, to Game$^3_{1,Z}$, ..., to Game$^7_{1,Z}$, to Game$^1_{2,Z}$ etc. until reaching Game$^7_{Q_1,Z}$, where all ciphertexts are of type $(\vec{x}_i^1, \vec{x}_i^1)$ (but all keys are still of type $(\vec{y}_i^0, \vec{y}_i^0)$). Recall that $\text{Game}^7_{Q_1,Z}$ is identical to $\text{Game}^7_{O,0}$. We will then transition from $\text{Game}^7_{Q_1,Z} = \text{Game}^7_{O,0}$ to $\text{Game}^1_{O,1}$, to $\text{Game}^2_{O,1}$, ..., to $\text{Game}^7_{O,1}$, to $\text{Game}^1_{O,2}$, etc. until reaching Game^7_{O,Q_2}, where all keys are of type $(\vec{y}_i^1, \vec{y}_i^1)$ (and all ciphertexts are of type $(\vec{x}_i^1, \vec{x}_i^1)$). This is identical to the real security game played with $b = 1$.

We begin the first transition in a hybrid over the Q_1 ciphertexts. Recall that the real security game played with $b = 0$ is identical to $\text{Game}^7_{0,Z}$, so in particular, the following lemma allows us to make the first transition from $\text{Game}^7_{0,Z}$ to $\text{Game}^1_{1,Z}$.

Lemma 1. *No polynomial-time attacker can achieve a non-negligible difference in advantage between* $\text{Game}^7_{(j-1),Z}$ *and* $\text{Game}^1_{j,Z}$ *for* $j = 1, ..., Q_1$ *under the SXDH assumption.*

Proof. Given an attacker that achieves non-negligible difference in advantage between $\text{Game}^7_{(j-1),Z}$ and $\text{Game}^1_{j,Z}$ for some $j \in [1, Q_1]$, we could achieve non-negligible advantage in deciding the SXDH problem as follows:

Given SXDH instance $g_1, g_1^a, g_1^b, T = g_1^{ab+r}, g_2$ where either $r \leftarrow \mathbb{Z}_p$ or $r = 0$, use g_1, g_2 as the generators of the same name used to form ciphertexts and keys respectively. Generate bases $(\mathbb{F}, \mathbb{F}^*) \leftarrow Dual(\mathbb{Z}_p^{2n})$, $(\mathbb{H}, \mathbb{H}^*) \leftarrow Dual(\mathbb{Z}_p^2)$ and implicitly define new bases $(\mathbb{B}, \mathbb{B}^*)$, $(\mathbb{D}, \mathbb{D}^*)$ as the following:

$$\begin{aligned}
\vec{b}_i &= \vec{f}_i - a\vec{f}_{n+i} \text{ for } i = 1, ..., n \\
\vec{b}_{n+i} &= \vec{f}_{n+i} \text{ for } i = 1, ..., n \\
\vec{b}_i^* &= \vec{f}_i^* \text{ for } i = 1, ..., n \\
\vec{b}_{n+i}^* &= \vec{f}_{n+i}^* + a\vec{f}_i^* \text{ for } i = 1, ..., n
\end{aligned}$$

$$\begin{aligned}
\vec{d}_1 &= \vec{h}_1 - a\vec{h}_2 \\
\vec{d}_2 &= \vec{h}_2 \\
\vec{d}_1^* &= \vec{h}_1^* \\
\vec{d}_2^* &= \vec{h}_2^* + a\vec{h}_1^*
\end{aligned}$$

Note that these bases are distributed exactly the same as those output by $Dual(\mathbb{Z}_p^{2n})$ and $Dual(\mathbb{Z}_p^2)$ respectively (they are created by applying an invertible linear transformation to the output of $Dual(\mathbb{Z}_p^{2n})$ and $Dual(\mathbb{Z}_p^2)$).

To construct any key (for, say, vector $\vec{y}^0$), generate random $\beta, \tilde{\beta}' \leftarrow \mathbb{Z}_p$, implicitly define $\tilde{\beta} = \beta a + \tilde{\beta}'$, and compute:

$$\begin{aligned}
K_1 &= g_2^{\beta(y_1^0\vec{f}_1+\cdots+y_n^0\vec{f}_n)+\tilde{\beta}'(y_1^0\vec{f}_{n+1}+\cdots+y_n^0\vec{f}_{2n})} \\
&= g_2^{\beta(y_1^0\vec{f}_1+\cdots+y_n^0\vec{f}_n)+(\tilde{\beta}-\beta a)(y_1^0\vec{f}_{n+1}+\cdots+y_n^0\vec{f}_{2n})} \\
&= g_2^{\beta(y_1^0(\vec{f}_1-a\vec{f}_{n+1})+\cdots+y_n^0(\vec{f}_n-a\vec{f}_{2n}))+\tilde{\beta}(y_1^0\vec{f}_{n+1}+\cdots+y_n^0\vec{f}_{2n})} \\
&= g_2^{\beta(y_1^0\vec{b}_1+\cdots+y_n^0\vec{b}_n)+\tilde{\beta}(y_1^0\vec{b}_{n+1}+\cdots+y_n^0\vec{b}_{2n})}
\end{aligned}$$

$$\begin{aligned}
K_2 &= g_2^{\beta\vec{h}_1+\tilde{\beta}'\vec{h}_2} \\
&= g_2^{\beta\vec{h}_1+(\tilde{\beta}-\beta a)\vec{h}_2} \\
&= g_2^{\beta(\vec{h}_1-a\vec{h}_2)+\tilde{\beta}\vec{h}_2} \\
&= g_2^{\beta\vec{d}_1+\tilde{\beta}\vec{d}_2}
\end{aligned}$$

a properly distributed type-$(\vec{y}^0, \vec{y}^0)$ key (as expected in both $\text{Game}^7_{(j-1),Z}$ and $\text{Game}^1_{j,Z}$).

For the jth ciphertext and all ciphertexts after, draw $\alpha'_i, \tilde{\alpha}'_i \leftarrow \mathbb{Z}_p$ and compute:

$$\begin{aligned}
C_{1,i} &= (g_1^b)^{\alpha'_i(x^0_{1,i}\vec{f}^*_{n+1}+\cdots+x^0_{n,i}\vec{f}^*_{2n})}(T)^{\alpha'_i(x^0_{1,i}\vec{f}^*_1+\cdots+x^0_{n,i}\vec{f}^*_n)}g_1^{\tilde{\alpha}'_i(x^0_{1,i}\vec{f}^*_{n+1}+\cdots+x^0_{n,i}\vec{f}^*_{2n})} \\
&\quad \cdot (g_1^a)^{\tilde{\alpha}'_i(x^0_{1,i}\vec{f}^*_1+\cdots+x^0_{n,i}\vec{f}^*_n)} \\
&= g_1^{\alpha'_i r(x^0_{1,i}\vec{f}^*_1+\cdots+x^0_{n,i}\vec{f}^*_n)+(\tilde{\alpha}'_i+\alpha'_i b)(x^0_{1,i}(\vec{f}^*_{n+1}+a\vec{f}^*_1)+\cdots+x^0_{n,i}(\vec{f}^*_{2n}+a\vec{f}^*_n))} \\
&= g_1^{\alpha'_i r(x^0_{1,i}\vec{b}^*_1+\cdots+x^0_{n,i}\vec{b}^*_n)+(\tilde{\alpha}'_i+\alpha'_i b)(x^0_{1,i}\vec{b}^*_{n+1}+\cdots+x^0_{n,i}\vec{b}^*_{2n})}
\end{aligned}$$

$$\begin{aligned}
C_{2,i} &= (g_1^b)^{\alpha'_i\vec{h}^*_2}(T)^{\alpha'_i\vec{h}^*_1}g_1^{\tilde{\alpha}'_i\vec{h}^*_2}(g_1^a)^{\tilde{\alpha}'_i\vec{h}^*_1} \\
&= g_1^{\alpha'_i r\vec{h}^*_1+(\tilde{\alpha}'_i+\alpha'_i b)(\vec{h}^*_2+a\vec{h}^*_1)} \\
&= g_1^{\alpha'_i r\vec{d}^*_1+(\tilde{\alpha}'_i+\alpha'_i b)\vec{d}^*_2}
\end{aligned}$$

For ciphertexts before the jth ciphertext, draw $\alpha'_i, \tilde{\alpha}'_i \leftarrow \mathbb{Z}_p$ and compute:

$$\begin{aligned}
C_{1,i} &= (g_1^b)^{\alpha'_i(x^1_{1,i}\vec{f}^*_{n+1}+\cdots+x^1_{n,i}\vec{f}^*_{2n})}(T)^{\alpha'_i(x^1_{1,i}\vec{f}^*_1+\cdots+x^1_{n,i}\vec{f}^*_n)}g_1^{\tilde{\alpha}'_i(x^1_{1,i}\vec{f}^*_{n+1}+\cdots+x^1_{n,i}\vec{f}^*_{2n})} \\
&\quad \cdot (g_1^a)^{\tilde{\alpha}'_i(x^1_{1,i}\vec{f}^*_1+\cdots+x^1_{n,i}\vec{f}^*_n)} \\
&= g_1^{\alpha'_i r(x^1_{1,i}\vec{f}^*_1+\cdots+x^1_{n,i}\vec{f}^*_n)+(\tilde{\alpha}'_i+\alpha'_i b)(x^1_{1,i}(\vec{f}^*_{n+1}+a\vec{f}^*_1)+\cdots+x^1_{n,i}(\vec{f}^*_{2n}+a\vec{f}^*_n))} \\
&= g_1^{\alpha'_i r(x^1_{1,i}\vec{b}^*_1+\cdots+x^1_{n,i}\vec{b}^*_n)+(\tilde{\alpha}'_i+\alpha'_i b)(x^1_{1,i}\vec{b}^*_{n+1}+\cdots+x^1_{n,i}\vec{b}^*_{2n})}
\end{aligned}$$

$$\begin{aligned}
C_{2,i} &= (g_1^b)^{\alpha'_i\vec{h}^*_2}(T)^{\alpha'_i\vec{h}^*_1}g_1^{\tilde{\alpha}'_i\vec{h}^*_2}(g_1^a)^{\tilde{\alpha}'_i\vec{h}^*_1} \\
&= g_1^{\alpha'_i r\vec{h}^*_1+(\tilde{\alpha}'_i+\alpha'_i b)(\vec{h}^*_2+a\vec{h}^*_1)} \\
&= g_1^{\alpha'_i r\vec{d}^*_1+(\tilde{\alpha}'_i+\alpha'_i b)\vec{d}^*_2}
\end{aligned}$$

(The only difference from the prior ciphertext construction is that $\vec{x}^1$ is used instead of $\vec{x}^0$).

When $r \leftarrow \mathbb{Z}_p$, all ciphertexts before the jth ciphertext are properly distributed type-$(\vec{x}^1_i, \vec{x}^1_i)$ ciphertexts and the remaining ciphertexts are properly distributed type-$(\vec{x}^0_i, \vec{x}^0_i)$ ciphertexts where $\alpha_i = \alpha'_i r$ and $\tilde{\alpha}_i = \tilde{\alpha}'_i + \alpha'_i b$. This is as would be expected in $\text{Game}^7_{(j-1),Z}$.
When $r = 0$, all ciphertexts before the jth ciphertext are properly distributed type-$(\vec{0}, \vec{x}^1_i)$ ciphertexts and the remaining ciphertexts are properly distributed type-$(\vec{0}, \vec{x}^0_i)$ ciphertexts where $\tilde{\alpha}_i = \tilde{\alpha}'_i + \alpha'_i b$. This is as would be expected in $\text{Game}^1_{j,Z}$.

Our simulation is therefore identical to either $\text{Game}^7_{(j-1),Z}$ or $\text{Game}^1_{j,Z}$ depending on the SXDH challenge $T = g_1^{ab+r}$ having $r \leftarrow \mathbb{Z}_p$ or $r = 0$ respectively. Therefore, by playing the security game in this manner with the supposed attacker and using the attacker's output as an answer to the SXDH challenge, we will enjoy the same non-negligible advantage as the supposed attacker in deciding the SXDH problem.

By the SXDH assumption, this is not possible, so no such adversary can exist.

Lemma 2. *No polynomial-time attacker can achieve a non-negligible difference in advantage between Game*$^1_{j,Z}$ *and Game*$^2_{j,Z}$ *for* $j = 1, ..., Q_1$ *under the SXDH assumption.*

Proof. Given an attacker that achieves non-negligible difference in advantage between $\text{Game}^1_{j,Z}$ and $\text{Game}^2_{j,Z}$ for some $j \in [1, Q_1]$, we could achieve non-negligible advantage in deciding the SXDH problem as follows:

Given SXDH instance $g_1, g_1^a, g_1^b, T = g_1^{ab+r}, g_2$ where either $r \leftarrow \mathbb{Z}_p$ or $r = 0$, use g_1, g_2 as the generators of the same name used to form ciphertexts and keys respectively.. Generate bases $(\mathbb{F}, \mathbb{F}^*) \leftarrow Dual(\mathbb{Z}_p^{2n})$, $(\mathbb{H}, \mathbb{H}^*) \leftarrow Dual(\mathbb{Z}_p^2)$ and implicitly define new bases $(\mathbb{B}, \mathbb{B}^*)$, $(\mathbb{D}, \mathbb{D}^*)$ as the following:

$$\begin{aligned}
\vec{b}_i &= \vec{f}_i - a\vec{f}_{n+i} \text{ for } i = 1, ..., n \\
\vec{b}_{n+i} &= \vec{f}_{n+i} \text{ for } i = 1, ..., n \\
\vec{b}_i^* &= \vec{f}_i^* \text{ for } i = 1, ..., n \\
\vec{b}_{n+i}^* &= \vec{f}_{n+i}^* + a\vec{f}_i^* \text{ for } i = 1, ..., n
\end{aligned}$$

$$\begin{aligned}
\vec{d}_1 &= \vec{h}_1 - a\vec{h}_2 \\
\vec{d}_2 &= \vec{h}_2 \\
\vec{d}_1^* &= \vec{h}_1^* \\
\vec{d}_2^* &= \vec{h}_2^* + a\vec{h}_1^*
\end{aligned}$$

Note that these bases are distributed exactly the same as those output by $Dual(\mathbb{Z}_p^{2n})$ and $Dual(\mathbb{Z}_p^2)$ respectively (they are created by applying an invertible linear transformation to the output of $Dual(\mathbb{Z}_p^{2n})$ and $Dual(\mathbb{Z}_p^2)$).

To construct any key (for, say, vector $\vec{y}^0$), generate random $\beta, \tilde{\beta}' \leftarrow \mathbb{Z}_p$, implicitly define $\tilde{\beta} = \beta a + \tilde{\beta}'$, and compute:

$$\begin{aligned}
K_1 &= g_2^{\beta(y_1^0\vec{f}_1+\cdots+y_n^0\vec{f}_n)+\tilde{\beta}'(y_1^0\vec{f}_{n+1}+\cdots+y_n^0\vec{f}_{2n})} \\
&= g_2^{\beta(y_1^0\vec{f}_1+\cdots+y_n^0\vec{f}_n)+(\tilde{\beta}-\beta a)(y_1^0\vec{f}_{n+1}+\cdots+y_n^0\vec{f}_{2n})} \\
&= g_2^{\beta(y_1^0(\vec{f}_1-a\vec{f}_{n+1})+\cdots+y_n^0(\vec{f}_n-a\vec{f}_{2n}))+\tilde{\beta}(y_1^0\vec{f}_{n+1}+\cdots+y_n^0\vec{f}_{2n})} \\
&= g_2^{\beta(y_1^0\vec{b}_1+\cdots+y_n^0\vec{b}_n)+\tilde{\beta}(y_1^0\vec{b}_{n+1}+\cdots+y_n^0\vec{b}_{2n})}
\end{aligned}$$

$$\begin{aligned}
K_2 &= g_2^{\beta\vec{h}_1+\tilde{\beta}'\vec{h}_2} \\
&= g_2^{\beta\vec{h}_1+(\tilde{\beta}-\beta a)\vec{h}_2} \\
&= g_2^{\beta(\vec{h}_1-a\vec{h}_2)+\tilde{\beta}\vec{h}_2} \\
&= g_2^{\beta\vec{d}_1+\tilde{\beta}\vec{d}_2}
\end{aligned}$$

a properly distributed type-$(\vec{y}^0, \vec{y}^0)$ key (as expected in both $\text{Game}_{j,Z}^1$ and $\text{Game}_{j,Z}^2$).

For ciphertexts before the jth ciphertext draw random $\tilde{\alpha}_i \leftarrow \mathbb{Z}_p$ and compute:

$$\begin{aligned}
C_{1,i} &= g_1^{\tilde{\alpha}_i(x^1_{1,i}\vec{f}^*_{n+1}+\cdots+x^1_{n,i}\vec{f}^*_{2n})}(g_1^a)^{\tilde{\alpha}_i(x^1_{1,i}\vec{f}^*_1+\cdots+x^1_{n,i}\vec{f}^*_n)}\\
&= g_1^{\tilde{\alpha}_i(x^1_{1,i}(\vec{f}^*_{n+1}+a\vec{f}^*_1)+\cdots+x^1_{n,i}(\vec{f}^*_{2n}+a\vec{f}^*_n))}\\
&= g_1^{\tilde{\alpha}_i(x^1_{1,i}\vec{b}^*_{n+1}+\cdots+x^1_{n,i}\vec{b}^*_{2n})}
\end{aligned}$$

$$\begin{aligned}
C_{2,i} &= g_1^{\tilde{\alpha}_i\vec{h}^*_2}(g_1^a)^{\tilde{\alpha}_i\vec{h}^*_1}\\
&= g_1^{\tilde{\alpha}_i(\vec{h}^*_2+a\vec{h}^*_1)}\\
&= g_1^{\tilde{\alpha}_i\vec{d}^*_2}
\end{aligned}$$

a properly distributed type-$(\vec{0},\vec{x}^1_i)$ ciphertext (as expected in both $\text{Game}^1_{j,Z}$ and $\text{Game}^2_{j,Z}$).

For ciphertexts after the jth ciphertext draw random $\tilde{\alpha}_i \leftarrow \mathbb{Z}_p$ and compute:

$$\begin{aligned}
C_{1,i} &= g_1^{\tilde{\alpha}_i(x^0_{1,i}\vec{f}^*_{n+1}+\cdots+x^0_{n,i}\vec{f}^*_{2n})}(g_1^a)^{\tilde{\alpha}_i(x^0_{1,i}\vec{f}^*_1+\cdots+x^0_{n,i}\vec{f}^*_n)}\\
&= g_1^{\tilde{\alpha}_i(x^0_{1,i}(\vec{f}^*_{n+1}+a\vec{f}^*_1)+\cdots+x^0_{n,i}(\vec{f}^*_{2n}+a\vec{f}^*_n))}\\
&= g_1^{\tilde{\alpha}_i(x^0_{1,i}\vec{b}^*_{n+1}+\cdots+x^0_{n,i}\vec{b}^*_{2n})}
\end{aligned}$$

$$\begin{aligned}
C_{2,i} &= g_1^{\tilde{\alpha}_i\vec{h}^*_2}(g_1^a)^{\tilde{\alpha}_i\vec{h}^*_1}\\
&= g_1^{\tilde{\alpha}_i(\vec{h}^*_2+a\vec{h}^*_1)}\\
&= g_1^{\tilde{\alpha}_i\vec{d}^*_2}
\end{aligned}$$

a properly distributed type-$(\vec{0},\vec{x}^0_i)$ ciphertext (as expected in both $\text{Game}^1_{j,Z}$ and $\text{Game}^2_{j,Z}$). Note that this construction is the same as that of the first $j-1$ ciphertexts except for the $\vec{x}^0_i$ components used instead of $\vec{x}^1_i$.

For the jth ciphertext, compute:

$$\begin{aligned}
C_{1,j} &= (g_1^b)^{x^0_{1,j}\vec{f}^*_{n+1}+\cdots+x^0_{n,j}\vec{f}^*_{2n}}(T)^{x^0_{1,j}\vec{f}^*_1+\cdots+x^0_{n,j}\vec{f}^*_n}\\
&= g_1^{r(x^0_{1,j}\vec{f}^*_1+\cdots+x^0_{n,j}\vec{f}^*_n)+b(x^0_{1,j}(\vec{f}^*_{n+1}+a\vec{f}^*_1)+\cdots+x^0_{n,j}(\vec{f}^*_{2n}+a\vec{f}^*_n))}\\
&= g_1^{r(x^0_{1,j}\vec{b}^*_1+\cdots+x^0_{n,j}\vec{b}^*_n)+b(x^0_{1,j}\vec{b}^*_{n+1}+\cdots+x^0_{n,j}\vec{b}^*_{2n})}
\end{aligned}$$

$$\begin{aligned}
C_{2,j} &= (g_1^b)^{\vec{h}^*_2}(T)^{\vec{h}^*_1}\\
&= g_1^{r\vec{h}^*_1+b(\vec{h}^*_2+a\vec{h}^*_1)}\\
&= g_1^{r\vec{d}^*_1+b\vec{d}^*_2}
\end{aligned}$$

When $r = 0$, this is a properly distributed type-$(\vec{0}, \vec{x}_j^0)$ ciphertext where $\tilde{\alpha}_j = b$ (as would be expected in $\text{Game}^1_{j,Z}$).
When $r \leftarrow \mathbb{Z}_p$, this is a properly distributed type-$(\vec{x}_j^0, \vec{x}_j^0)$ ciphertext where $\alpha_j = r$ and $\tilde{\alpha}_j = b$ (as would be expected in $\text{Game}^2_{j,Z}$).

Our simulation is therefore identical to either $\text{Game}^1_{j,Z}$ or $\text{Game}^2_{j,Z}$ depending on the SXDH challenge $T = g_1^{ab+r}$ having $r = 0$ or $r \leftarrow \mathbb{Z}_p$ respectively. Therefore, by playing the security game in this manner with the supposed attacker and using the attacker's output as an answer to the SXDH challenge, we will enjoy the same non-negligible advantage as the supposed attacker in deciding the SXDH problem.

By the SXDH assumption, this is not possible, so no such adversary can exist.

Lemma 3. *No attacker can achieve a non-negligible difference in advantage between $\text{Game}^2_{j,Z}$ and $\text{Game}^3_{j,Z}$ for $j = 1, ..., Q_1$.*

Proof. This lemma is unconditionally true because both games are information-theoretically identical.

In the simulation of the security $\text{Game}^2_{j,Z}$, one draws bases $(\mathbb{B}, \mathbb{B}^*) \leftarrow Dual(\mathbb{Z}_p^{2n})$. However, imagine knowing the jth ciphertext vectors $\vec{x}_j^0, \vec{x}_j^1$ ahead of time and drawing the bases in the following way: first, draw $\mathbb{B}, \mathbb{B}^* \leftarrow Dual(\mathbb{Z}_p^{2n})$ then apply following invertible linear transformation to get $(\mathbb{F}, \mathbb{F}^*)$, which is used (along with a normally drawn $(\mathbb{D}, \mathbb{D}^*) \leftarrow Dual(\mathbb{Z}_p^2)$) as the basis:

$$\vec{f}_i = \vec{b}_i \text{ for } i = 1, ..., n$$

$$\vec{f}_{n+i} = \vec{b}_{n+i} + \frac{\tilde{\alpha}_j(x_{i,j}^0 - x_{i,j}^1)}{\alpha_j x_{1,j}^0}\vec{b}_1 \text{ for } i = 1, ..., n$$

$$\vec{f}_1^* = \vec{b}_1^* - \sum_{i=1}^{n} \frac{\tilde{\alpha}_j(x_{i,j}^0 - x_{i,j}^1)}{\alpha_j x_{1,j}^0}\vec{b}_{n+i}^*$$

$$\vec{f}_i^* = \vec{b}_i^* \text{ for } i = 2, ..., 2n$$

where $\alpha_j, \tilde{\alpha}_j$ are randomly drawn values used in the creation of the jth ciphertext. Recall that the distribution of the $(\mathbb{F}, \mathbb{F}^*)$ produced this way is identical to that produced by $Dual(\mathbb{Z}_p^{2n})$).

Consider simulating $\text{Game}^2_{j,Z}$ with this basis. Any key (for, say, vector $\vec{y}^0$) looks like:

$$K_1 = g_2^{\beta(y_1^0\vec{f}_1+\cdots+y_n^0\vec{f}_n)+\tilde{\beta}(y_1^0\vec{f}_{n+1}+\cdots+y_n^0\vec{f}_{2n})}$$

$$K_2 = g_2^{\beta\vec{d}_1+\tilde{\beta}\vec{d}_2}.$$

where:

$$\begin{aligned}
K_1 &= g_2^{\beta(y_1^0\vec{f}_1+\cdots+y_n^0\vec{f}_n)+\tilde{\beta}(y_1^0\vec{f}_{n+1}+\cdots+y_n^0\vec{f}_{2n})} \\
&= g_2^{\beta(y_1^0\vec{b}_1+\cdots+y_n^0\vec{b}_n)+\tilde{\beta}(y_1^0(\vec{b}_{n+1}+\frac{\tilde{\alpha}_j(x_{1,j}^0-x_{1,j}^1)}{\alpha_j x_{1,j}^0}\vec{b}_1)+\cdots+y_n^0(\vec{b}_{2n}+\frac{\tilde{\alpha}_j(x_{n,j}^0-x_{n,j}^1)}{\alpha_j x_{1,j}^0}\vec{b}_1))} \\
&= g_2^{\beta(y_1^0\vec{b}_1+\cdots+y_n^0\vec{b}_n)+\tilde{\beta}(y_1^0\vec{b}_{n+1}+\cdots+y_n^0\vec{b}_{2n})+\frac{\tilde{\beta}\tilde{\alpha}_j\langle\vec{y}^0,\vec{x}_j^0-\vec{x}_j^1\rangle}{\alpha_j x_{1,j}^0}\vec{b}_1} \\
&= g_2^{\beta(y_1^0\vec{b}_1+\cdots+y_n^0\vec{b}_n)+\tilde{\beta}(y_1^0\vec{b}_{n+1}+\cdots+y_n^0\vec{b}_{2n})}
\end{aligned}$$

The last step above comes from the fact that $\langle\vec{y}^0,\vec{x}_j^0-\vec{x}_j^1\rangle=0$.
($\vec{x}_j^0,\vec{x}_j^1$ are vectors requested in the game where we require $\langle\vec{y}^0,\vec{x}_j^0\rangle=\langle\vec{y}^0,\vec{x}_j^1\rangle \implies \langle\vec{y}^0,\vec{x}_j^0-\vec{x}_j^1\rangle=0$). This is a type-$(\vec{y}^0,\vec{y}^0)$ key in the $(\mathbb{B},\mathbb{B}^*)$ basis (as expected in both $\text{Game}_{j,Z}^2$ and $\text{Game}_{j,Z}^3$).

All ciphertexts created before the jth ciphertext look like properly distributed type-$(\vec{0},\vec{x}_i^1)$ ciphertexts in the $(\mathbb{B},\mathbb{B}^*)$ basis:

$$\begin{aligned}
C_{1,i} &= g_1^{\tilde{\alpha}_i(x_{1,i}^1\vec{f}_{n+1}^*+\cdots+x_{n,i}^1\vec{f}_{2n}^*)} \\
&= g_1^{\tilde{\alpha}_i(x_{1,i}^1\vec{b}_{n+1}^*+\cdots+x_{n,i}^1\vec{b}_{2n}^*)}
\end{aligned}$$

$$C_{2,i} = g_1^{\tilde{\alpha}_i\vec{d}_2^*}$$

Similarly, all ciphertexts created after the jth ciphertext look like properly distributed type-$(\vec{0},\vec{x}_i^0)$ ciphertexts in the $(\mathbb{B},\mathbb{B}^*)$ basis:

$$\begin{aligned}
C_{1,i} &= g_1^{\tilde{\alpha}_i(x_{1,i}^0\vec{f}_{n+1}^*+\cdots+x_{n,i}^0\vec{f}_{2n}^*)} \\
&= g_1^{\tilde{\alpha}_i(x_{1,i}^0\vec{b}_{n+1}^*+\cdots+x_{n,i}^0\vec{b}_{2n}^*)}
\end{aligned}$$

$$C_{2,i} = g_1^{\tilde{\alpha}_i\vec{d}_2^*}$$

However, the jth ciphertext constructed (as a type-$(\vec{x}_j^0,\vec{x}_j^0)$ ciphertext) looks like a type-$(\vec{x}_j^0,\vec{x}_j^1)$ ciphertext in the $(\mathbb{B},\mathbb{B}^*)$ basis:

$$C_{1,j} = g_1^{\alpha_j(x_{1,j}^0\vec{f}_1^*+\cdots+x_{n,j}^0\vec{f}_n^*)+\tilde{\alpha}_j(x_{1,j}^0\vec{f}_{n+1}^*+\cdots+x_{n,j}^0\vec{f}_{2n}^*)}$$

$$= g_1^{\alpha_j(x_{1,j}^0(\vec{b}_1^*-\sum_{i=1}^{n}\frac{\tilde{\alpha}_j(x_{i,j}^0-x_{i,j}^1)}{\alpha_j x_{1,j}^0}\vec{b}_{n+i}^*)+\cdots+x_{n,j}^0\vec{b}_n^*)+\tilde{\alpha}_j(x_{1,j}^0\vec{b}_{n+1}^*+\cdots+x_{n,j}^0\vec{b}_{2n}^*)}$$

$$= g_1^{\alpha_j(x_{1,j}^0\vec{b}_1^*+\cdots+x_{n,j}^0\vec{b}_n^*)+\tilde{\alpha}_j(x_{1,j}^1\vec{b}_{n+1}^*+\cdots+x_{n,j}^1\vec{b}_{2n}^*)}$$

$$C_{2,j} = g_1^{\alpha_j\vec{d}_1^*+\tilde{\alpha}_j\vec{d}_2^*}$$

This construction of the jth ciphertext is the only difference between $\text{Game}^2_{j,Z}$ and $\text{Game}^3_{j,Z}$. So, drawing $(\mathbb{B}, \mathbb{B}^*)$ directly from $Dual(\mathbb{Z}_p^{2n})$ and using it to play $\text{Game}^2_{j,Z}$ results in $\text{Game}^2_{j,Z}$ in the $(\mathbb{B}, \mathbb{B}^*)$ basis. However, transforming this basis to $(\mathbb{F}, \mathbb{F}^*)$ and using it instead results in playing $\text{Game}^3_{j,Z}$ with the $(\mathbb{B}, \mathbb{B}^*)$ basis. However, since $(\mathbb{B}, \mathbb{B}^*)$ and $(\mathbb{F}, \mathbb{F}^*)$ have the same distribution (the one produced by $Dual(\mathbb{Z}_p^{2n})$), this means that the two Games are actually information-theoretically identical. Therefore, no attacker can achieve non-negligible difference advantage in distinguishing between $\text{Game}^2_{j,Z}$ and $\text{Game}^3_{j,Z}$.

Lemma 4. *No polynomial-time attacker can achieve a non-negligible difference in advantage between $Game^3_{j,Z}$ and $Game^4_{j,Z}$ for $j = 1, ..., Q_1$ under the SXDH assumption.*

Proof. Given an attacker that achieves non-negligible difference in advantage between $\text{Game}^3_{j,Z}$ and $\text{Game}^4_{j,Z}$ for some $j \in [1, Q_1]$, we could achieve non-negligible advantage in deciding the SXDH problem as follows:

Given SXDH instance $g_1, g_1^a, g_1^b, T = g_1^{ab+r}, g_2$ where either $r \leftarrow \mathbb{Z}_p$ or $r = 0$, use g_1, g_2 as the generators of the same name used to form ciphertexts and keys respectively. Generate bases $(\mathbb{F}, \mathbb{F}^*) \leftarrow Dual(\mathbb{Z}_p^{2n})$, $(\mathbb{H}, \mathbb{H}^*) \leftarrow Dual(\mathbb{Z}_p^2)$ and implicitly define new bases $(\mathbb{B}, \mathbb{B}^*)$, $(\mathbb{D}, \mathbb{D}^*)$ as the following:

$$\begin{aligned}
\vec{b}_i &= \vec{f}_i - a\vec{f}_{n+i} \text{ for } i = 1, ..., n \\
\vec{b}_{n+i} &= \vec{f}_{n+i} \text{ for } i = 1, ..., n \\
\vec{b}_i^* &= \vec{f}_i^* \text{ for } i = 1, ..., n \\
\vec{b}_{n+i}^* &= \vec{f}_{n+i}^* + a\vec{f}_i^* \text{ for } i = 1, ..., n
\end{aligned}$$

$$\begin{aligned}
\vec{d}_1 &= \vec{h}_1 - a\vec{h}_2 \\
\vec{d}_2 &= \vec{h}_2 \\
\vec{d}_1^* &= \vec{h}_1^* \\
\vec{d}_2^* &= \vec{h}_2^* + a\vec{h}_1^*
\end{aligned}$$

Note that these bases are distributed exactly the same as those output by $Dual(\mathbb{Z}_p^{2n})$ and $Dual(\mathbb{Z}_p^2)$ respectively (they are created by applying an invertible linear transformation to the output of $Dual(\mathbb{Z}_p^{2n})$ and $Dual(\mathbb{Z}_p^2)$).

To construct any key (for, say, vector $\vec{y}^0$), generate random $\beta, \tilde{\beta}' \leftarrow \mathbb{Z}_p$, implicitly define $\tilde{\beta} = \beta a + \tilde{\beta}'$, and compute:

$$\begin{aligned}
K_1 &= g_2^{\beta(y_1^0 \vec{f}_1 + \cdots + y_n^0 \vec{f}_n) + \tilde{\beta}'(y_1^0 \vec{f}_{n+1} + \cdots + y_n^0 \vec{f}_{2n})} \\
&= g_2^{\beta(y_1^0 \vec{f}_1 + \cdots + y_n^0 \vec{f}_n) + (\tilde{\beta} - \beta a)(y_1^0 \vec{f}_{n+1} + \cdots + y_n^0 \vec{f}_{2n})} \\
&= g_2^{\beta(y_1^0 (\vec{f}_1 - a\vec{f}_{n+1}) + \cdots + y_n^0 (\vec{f}_n - a\vec{f}_{2n})) + \tilde{\beta}(y_1^0 \vec{f}_{n+1} + \cdots + y_n^0 \vec{f}_{2n})} \\
&= g_2^{\beta(y_1^0 \vec{b}_1 + \cdots + y_n^0 \vec{b}_n) + \tilde{\beta}(y_1^0 \vec{b}_{n+1} + \cdots + y_n^0 \vec{b}_{2n})}
\end{aligned}$$

$$\begin{aligned}
K_2 &= g_2^{\beta \vec{h}_1 + \tilde{\beta}' \vec{h}_2} \\
&= g_2^{\beta \vec{h}_1 + (\tilde{\beta} - \beta a) \vec{h}_2} \\
&= g_2^{\beta(\vec{h}_1 - a\vec{h}_2) + \tilde{\beta} \vec{h}_2} \\
&= g_2^{\beta \vec{d}_1 + \tilde{\beta} \vec{d}_2}
\end{aligned}$$

a properly distributed type-$(\vec{y}^0, \vec{y}^0)$ key (as expected in both $\mathrm{Game}^3_{j,Z}$ and $\mathrm{Game}^4_{j,Z}$).

For the jth ciphertext, draw $\alpha_j, \tilde{\alpha}_j \leftarrow \mathbb{Z}_p$ and compute:

$$\begin{aligned}
C_{1,j} &= g_1^{\alpha_j (x_{1,j}^0 \vec{f}_1^* + \cdots + x_{n,j}^0 \vec{f}_n^*) + \tilde{\alpha}_j (x_{1,j}^1 \vec{f}_{n+1}^* + \cdots + x_{n,j}^1 \vec{f}_{2n}^*)} (g_1^a)^{\tilde{\alpha}_j (x_{1,j}^1 \vec{f}_1^* + \cdots + x_{n,j}^1 \vec{f}_n^*)} \\
&= g_1^{\alpha_j (x_{1,i}^0 \vec{f}_1^* + \cdots + x_{n,i}^0 \vec{f}_n^*) + \tilde{\alpha}_j (x_{1,j}^1 (\vec{f}_{n+1}^* + a\vec{f}_1^*) + \cdots + x_{n,j}^1 (\vec{f}_{2n}^* + a\vec{f}_n^*))} \\
&= g_1^{\alpha_j (x_{1,j}^0 \vec{b}_1^* + \cdots + x_{n,j}^0 \vec{b}_n^*) + \tilde{\alpha}_j (x_{1,j}^1 \vec{b}_{n+1}^* + \cdots + x_{n,j}^1 \vec{b}_{2n}^*)}
\end{aligned}$$

$$\begin{aligned}
C_{2,j} &= g_1^{\alpha_j \vec{h}_1^*} g_1^{\tilde{\alpha}_j \vec{h}_2^*} (g_1^a)^{\tilde{\alpha}_j \vec{h}_1^*} \\
&= g_1^{\alpha_j \vec{h}_1^* + \tilde{\alpha}_j (\vec{h}_2^* + a\vec{h}_1^*)} \\
&= g_1^{\alpha_j \vec{d}_1^* + \tilde{\alpha}_j \vec{d}_2^*}
\end{aligned}$$

a properly distributed type-$(\vec{x}_j^0, \vec{x}_j^1)$ ciphertext (as expected in both $\mathrm{Game}^3_{j,Z}$ and $\mathrm{Game}^4_{j,Z}$).

For all ciphertexts after the jth ciphertext, draw $\alpha'_i, \tilde{\alpha}'_i \leftarrow \mathbb{Z}_p$ and compute:

$$\begin{aligned}
C_{1,i} &= (g_1^b)^{\alpha'_i(x^0_{1,i}\vec{f}^*_{n+1}+\cdots+x^0_{n,i}\vec{f}^*_{2n})}(T)^{\alpha'_i(x^0_{1,i}\vec{f}^*_1+\cdots+x^0_{n,i}\vec{f}^*_n)} g_1^{\tilde{\alpha}'_i(x^0_{1,i}\vec{f}^*_{n+1}+\cdots+x^0_{n,i}\vec{f}^*_{2n})} \\
&\cdot (g_1^a)^{\tilde{\alpha}'_i(x^0_{1,i}\vec{f}^*_1+\cdots+x^0_{n,i}\vec{f}^*_n)} \\
&= g_1^{\alpha'_i r(x^0_{1,i}\vec{f}^*_1+\cdots+x^0_{n,i}\vec{f}^*_n)+(\tilde{\alpha}'_i+\alpha'_i b)(x^0_{1,i}(\vec{f}^*_{n+1}+a\vec{f}^*_1)+\cdots+x^0_{n,i}(\vec{f}^*_{2n}+a\vec{f}^*_n))} \\
&= g_1^{\alpha'_i r(x^0_{1,i}\vec{b}^*_1+\cdots+x^0_{n,i}\vec{b}^*_n)+(\tilde{\alpha}'_i+\alpha'_i b)(x^0_{1,i}\vec{b}^*_{n+1}+\cdots+x^0_{n,i}\vec{b}^*_{2n})}
\end{aligned}$$

$$\begin{aligned}
C_{2,i} &= (g_1^b)^{\alpha'_i\vec{h}^*_2}(T)^{\alpha'_i\vec{h}^*_1} g_1^{\tilde{\alpha}'_i\vec{h}^*_2}(g_1^a)^{\tilde{\alpha}'_i\vec{h}^*_1} \\
&= g_1^{\alpha'_i r\vec{h}^*_1+(\tilde{\alpha}'_i+\alpha'_i b)(\vec{h}^*_2+a\vec{h}^*_1)} \\
&= g_1^{\alpha'_i r\vec{d}^*_1+(\tilde{\alpha}'_i+\alpha'_i b)\vec{d}^*_2}
\end{aligned}$$

For ciphertexts before the jth ciphertext, draw $\alpha'_i, \tilde{\alpha}'_i \leftarrow \mathbb{Z}_p$ and compute:

$$\begin{aligned}
C_{1,i} &= (g_1^b)^{\alpha'_i(x^1_{1,i}\vec{f}^*_{n+1}+\cdots+x^1_{n,i}\vec{f}^*_{2n})}(T)^{\alpha'_i(x^1_{1,i}\vec{f}^*_1+\cdots+x^1_{n,i}\vec{f}^*_n)} g_1^{\tilde{\alpha}'_i(x^1_{1,i}\vec{f}^*_{n+1}+\cdots+x^1_{n,i}\vec{f}^*_{2n})} \\
&\cdot (g_1^a)^{\tilde{\alpha}'_i(x^1_{1,i}\vec{f}^*_1+\cdots+x^1_{n,i}\vec{f}^*_n)} \\
&= g_1^{\alpha'_i r(x^1_{1,i}\vec{f}^*_1+\cdots+x^1_{n,i}\vec{f}^*_n)+(\tilde{\alpha}'_i+\alpha'_i b)(x^1_{1,i}(\vec{f}^*_{n+1}+a\vec{f}^*_1)+\cdots+x^1_{n,i}(\vec{f}^*_{2n}+a\vec{f}^*_n))} \\
&= g_1^{\alpha'_i r(x^1_{1,i}\vec{b}^*_1+\cdots+x^1_{n,i}\vec{b}^*_n)+(\tilde{\alpha}'_i+\alpha'_i b)(x^1_{1,i}\vec{b}^*_{n+1}+\cdots+x^1_{n,i}\vec{b}^*_{2n})}
\end{aligned}$$

$$\begin{aligned}
C_{2,i} &= (g_1^b)^{\alpha'_i\vec{h}^*_2}(T)^{\alpha'_i\vec{h}^*_1} g_1^{\tilde{\alpha}'_i\vec{h}^*_2}(g_1^a)^{\tilde{\alpha}'_i\vec{h}^*_1} \\
&= g_1^{\alpha'_i r\vec{h}^*_1+(\tilde{\alpha}'_i+\alpha'_i b)(\vec{h}^*_2+a\vec{h}^*_1)} \\
&= g_1^{\alpha'_i r\vec{d}^*_1+(\tilde{\alpha}'_i+\alpha'_i b)\vec{d}^*_2}
\end{aligned}$$

(The only difference from the prior ciphertext construction is that $\vec{x}^1$ is used instead of $\vec{x}^0$).

When $r = 0$, all ciphertexts before the jth ciphertext are properly distributed type-$(\vec{0}, \vec{x}^1_i)$ ciphertexts and all ciphertexts after the jth are properly distributed type-$(\vec{0}, \vec{x}^0_i)$ ciphertexts where $\tilde{\alpha}_i = \tilde{\alpha}'_i + \alpha'_i b$. This is as would be expected in $\text{Game}^3_{j,Z}$.
When $r \leftarrow \mathbb{Z}_p$, all ciphertexts before the jth are properly distributed type-$(\vec{x}^1_i, \vec{x}^1_i)$ ciphertexts and all ciphertexts after the jth are properly distributed type-$(\vec{x}^0_i, \vec{x}^0_i)$ ciphertexts where $\alpha_i = \alpha'_i r$ and $\tilde{\alpha}_i = \tilde{\alpha}'_i + \alpha'_i b$. This is as would be expected in $\text{Game}^4_{j,Z}$.

Our simulation is therefore identical to either $\text{Game}^3_{j,Z}$ or $\text{Game}^4_{j,Z}$ depending on the SXDH challenge $T = g_1^{ab+r}$ having $r = 0$ or $r \leftarrow \mathbb{Z}_p$ respectively. Therefore, by playing the security game in this manner with the supposed attacker and using the attacker's output as an answer to the SXDH challenge, we will enjoy the same non-negligible advantage as the supposed attacker in deciding the SXDH problem.

By the SXDH assumption, this is not possible, so no such adversary can exist.

Lemma 5. *No polynomial-time attacker can achieve a non-negligible difference in advantage between Game$^4_{j,Z}$ and Game$^5_{j,Z}$ for $j = 1, ..., Q_1$ under the SXDH assumption.*

Proof. The proof of this lemma is symmetric to that of the previous Lemma 4, just flipping the role of the two parallel bases.

Lemma 4 showed how to create keys of type-$(\vec{y}^0, \vec{y}^0)$ and a type-$(\vec{x}^0_j, \vec{x}^1_j)$ ciphertext while having the ciphertexts before and after the jth be type-$(\vec{0}, \vec{x}^1_i)$ and type-$(\vec{0}, \vec{x}^0_i)$ or type-$(\vec{x}^1_i, \vec{x}^1_i)$ and type-$(\vec{x}^0_i, \vec{x}^0_i)$ respectively based on the challenge elements of the SXDH problem. By symmetrically applying the same embedding to the opposite halves of the parallel bases, we can achieve the same result, but with the ciphertexts before and after the jth being type-$(\vec{x}^1_i, \vec{x}^1_i)$ and type-$(\vec{x}^0_i, \vec{x}^0_i)$ or type-$(\vec{x}^1_i, \vec{0})$ and type-$(\vec{x}^0_i, \vec{0})$ respectively based on the challenge elements of the SXDH problem, which is what we need for this transition.

Lemma 6. *No attacker can achieve a non-negligible difference in advantage between Game$^5_{j,Z}$ and Game$^6_{j,Z}$ for $j = 1, ..., Q_1$.*

Proof. The proof of this lemma is information-theoretic and similarly symmetric to that of Lemma 3, just flipping the role of the two parallel bases (just like the previous lemma did with Lemma 4).

Lemma 7. *No polynomial-time attacker can achieve a non-negligible difference in advantage between Game$^6_{j,Z}$ and Game$^7_{j,Z}$ for $j = 1, ..., Q_1$ under the SXDH assumption.*

Proof. The proof of this lemma is nearly identical to that of Lemma 5 (which transitioned ciphertexts before the jth between type-$(\vec{x}^1_i, \vec{0})$ and type-$(\vec{x}^1_i, \vec{x}^1_i)$ and ciphertexts after the jth between type-$(\vec{x}^0_i, \vec{0})$ and type-$(\vec{x}^0_i, \vec{x}^0_i)$) but instead constructing the jth ciphertext as type-$(\vec{x}^1_i, \vec{x}^1_i)$ instead of type-$(\vec{x}^0_i, \vec{x}^1_i)$.

The previous lemmas let us transition to Game$^7_{Q_1,Z}$, where all ciphertexts are of type-$(\vec{x}^1_i, \vec{x}^1_i)$ and all keys are type-$(\vec{y}^0_i, \vec{y}^0_i)$ keys. Notice that Game$^7_{Q_1,Z}$ is identical to Game$^7_{O,0}$. It is easy to see that we can now use a symmetric set of hybrids to transition all keys to type-$(\vec{y}^1_i, \vec{y}^1_i)$ in a similar manner: starting by transitioning from Game$^7_{Q_1,Z}$ = Game$^7_{O,0}$ to Game$^1_{O,1}$, then proceeding through the Game$^i_{O,j}$ using the same methods as in the Game$^i_{j,Z}$, just switching the roles of the basis vectors (switching all $\vec{b}^*$ with $\vec{b}$, $\vec{d}^*$ with $\vec{d}$, α with β, $\tilde{\alpha}$ with $\tilde{\beta}$, and $\vec{x}_i$ with $\vec{y}_i$, while always producing type $(\vec{x}^1_i, \vec{x}^1_i)$ ciphertexts (instead of always producing type $(\vec{y}^0_i, \vec{y}^0_i)$ keys).

These symmetric arguments let us transition to Game$^7_{O,Q_2}$, where all ciphertexts are of type-$(\vec{x}^1, \vec{x}^1)$ and all keys are type-$(\vec{y}^1_i, \vec{y}^1_i)$ keys. This is identical to the original security game when $b = 1$. So, by a hybrid argument, we have shown that no polynomial-time attacker gan achieve non-negligible difference in advantage in the security game when $b = 0$ ($\mathrm{Game}^7_{0,Z}$) vs when $b = 1$ (Game^7_{O,Q_2}) under the SXDH assumption, so our construction is therefore secure.

References

1. M. Abdalla, F. Bourse, A. De Caro, and D. Pointcheval. Simple functional encryption schemes for inner products. In *PKC*, pages 733–751, 2015.
2. S. Agrawal, S. Agrawal, S. Badrinarayanan, A. Kumarasubramanian, M. Prabhakaran, and A. Sahai. On the practical security of inner product functional encryption. In *PKC*, pages 777–798, 2015.
3. S. Agrawal, D. M. Freeman, and V. Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *ASIACRYPT*, pages 21–40.
4. P. Ananth, Z. Brakerski, G. Segev, and V. Vaikuntanathan. From selective to adaptive security in functional encryption. In *CRYPTO*, 2015.
5. B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.
6. D. Boneh, A. Raghunathan, and G. Segev. Function-private identity-based encryption: Hiding the function in functional encryption. In *CRYPTO*, pages 461–478, 2013.
7. D. Boneh, A. Raghunathan, and G. Segev. Function-private subspace-membership encryption and its applications. In *ASIACRYPT*, pages 255–275, 2013.
8. D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011, Providence, RI, USA, March 28-30, 2011. Proceedings*, pages 253–273, 2011.
9. Z. Brakerski and G. Segev. Function-private functional encryption in the private-key setting. In *TCC*, pages 306–324, 2015.
10. A. De Caro, V. Iovino, A. Jain, A. O'Neill, O. Paneth, and G. Persiano. On the achievability of simulation-based security for functional encryption. In *CRYPTO*, pages 519–535, 2013.
11. S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.
12. Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. Cryptology ePrint Archive, Report 2014/666, 2014. `http://eprint.iacr.org/`.
13. C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.
14. S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. How to run turing machines on encrypted data. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 536–553, 2013.
15. S. Gorbunov, V. Vaikuntanathan, and H. Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, 2012.
16. V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute based encryption for fine-grained access control of encrypted data. In *ACM conference on Computer and Communications Security*, pages 89–98, 2006.

17. J. Katz, A. Sahai, and B. Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.
18. A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.
19. T. Okamoto and K. Takashima. Homomorphic encryption and signatures from vector decomposition. In *Pairing*, pages 57–74, 2008.
20. T. Okamoto and K. Takashima. Hierarchical predicate encryption for inner-products. In *ASIACRYPT*, pages 214–231, 2009.
21. T. Okamoto and K. Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, pages 191–208, 2010.
22. T. Okamoto and K. Takashima. Fully secure unbounded inner-product and attribute-based encryption. In *ASIACRYPT 2012*, pages 349–366, 2012.
23. A. O'Neill. Definitional issues in functional encryption. IACR Cryptology ePrint Archive, 2010.
24. A. Sahai and H. Seyalioglu. Worry-free encryption: functional encryption with public keys. In *CCS*, pages 463–472.
25. A. Sahai and B. Waters. Fuzzy identity based encryption. In *EUROCRYPT*, pages 457–473, 2005.
26. E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *TCC*, pages 457–473, 2009.
27. B. Waters. A punctured programming approach to adaptively secure functional encryption. In *CRYPTO*, 2015.