# Predicate Privacy in Encryption Systems

Emily Shen
MIT
eshen@csail.mit.edu

Elaine Shi
CMU/PARC
eshi@parc.com

Brent Waters*
UT Austin
bwaters@cs.utexas.edu

December 24, 2008

**Abstract**

*Predicate encryption* is a new encryption paradigm which gives a master secret key owner fine-grained control over access to encrypted data. The master secret key owner can generate secret key tokens corresponding to predicates. An encryption of data $x$ can be evaluated using a secret token corresponding to a predicate $f$; the user learns whether the data satisfies the predicate, i.e., whether $f(x) = 1$.

Prior work on public-key predicate encryption has focused on the notion of data or plaintext privacy, the property that ciphertexts reveal no information about the encrypted data to an attacker other than what is inherently revealed by the tokens the attacker possesses. In this paper, we consider a new notion called *predicate privacy*, the property that tokens reveal no information about the encoded query predicate. Predicate privacy is inherently impossible to achieve in the public-key setting and has therefore received little attention in prior work. In this work, we consider predicate encryption in the symmetric-key setting and present a symmetric-key predicate encryption scheme which supports inner product queries. We prove that our scheme achieves both plaintext privacy and predicate privacy.

## 1 Introduction

In traditional public-key encryption, a user encrypts a message under a public key, and only the owner of the corresponding secret key can decrypt the ciphertext. In some applications, however, the user may wish to have more fine-grained control over what is revealed about the encrypted data. For example, in a medical context an administrative assistant might only be able to learn whether an encrypted record was generated at a certain clinic. *Predicate encryption* is a new encryption paradigm which allows for such fine-grained control over access to encrypted data. In a predicate encryption scheme, the owner of a master secret key can create and issue secret key *tokens* to other users. Tokens are associated with *predicates* which can be evaluated over encrypted data. Specifically, an encryption of a data $x$ can be evaluated using a token $TK_f$ associated with a predicate $f$ to learn whether $f(x) = 1$.

Prior work on public-key predicate encryption [7, 12, 1, 9, 19, 11, 28, 27] has focused on the security property that ciphertexts reveal no information about the underlying plaintext or data other than what is implied by the tokens in one's possession. More specifically, an adversary

*Supported by NSF CNS-0524252, CNS-0716199; the U.S. Department of Homeland Security under Grant Award Number 2006-CS-001-000001.

in possession of tokens $TK_{f_1}, \ldots, TK_{f_\ell}$ for predicates $f_1, \ldots, f_\ell$ learns no information about the underlying plaintext $x$ other than the values of $f_1(x), \ldots, f_\ell(x)$[1]. We refer to the above property as plaintext or data privacy.

In this work, we consider a different dimension of predicate encryption – *predicate privacy*. In addition to protecting the privacy of plaintexts, we would like to protect the description of the predicates encoded by tokens. Informally, predicate privacy says that a token hides all information about the encoded predicate other than what is implied by the ciphertexts in one's possession. Unfortunately, predicate privacy is inherently impossible to achieve in the public-key setting. Since encryption does not require a secret key, an adversary can encrypt any plaintext of his choice and evaluate a token on the resulting ciphertext to learn whether the plaintext satisfies the predicate associated with the token. In this way, an adversary can gain information about the predicate encoded in a token. Therefore, it does not make sense to consider the notion of predicate privacy for predicate encryption in the public-key setting.

However, it is interesting to consider predicate privacy in the symmetric-key setting, in applications where we want to hide information about the predicate being tested from the party evaluating a token. For example, suppose a user Alice uses a remote storage service to back up her files. Alice wishes to protect the privacy of her files by encrypting them using her secret key before sending them to the server. (Only Alice possesses her secret key.) Later on, Alice may wish to retrieve all files satisfying a certain predicate. To do this, Alice can create a token (using her secret key) for this predicate and issue the token to the server. The server can then evaluate the predicate on the encrypted files and return those files which satisfy the predicate. We want to guarantee that the server learns nothing about the predicate it evaluates on Alice's behalf.

### 1.1 Our Results

In this paper, we present formal definitions of security for predicate encryption in the symmetric-key setting, for general classes of predicates. We present a symmetric-key predicate encryption scheme that achieves both plaintext privacy and predicate privacy. Our construction supports the class of predicates corresponding to the evaluation of inner products. We take the set of plaintexts to be $\Sigma = \mathbb{Z}_N^n$ and the class of predicates to be $\mathcal{F} = \{f_{\vec{v}} | \vec{v} \in \mathbb{Z}_N^n\}$ where $f_{\vec{v}}(\vec{x}) = 1$ iff $\langle \vec{v}, \vec{x} \rangle = 0$, where $\langle \vec{v}, \vec{x} \rangle$ denotes the inner product $\sum_{i=1}^n v_i \cdot x_i \mod N$ of vectors $\vec{v}$ and $\vec{x}$. Our construction is based on the KSW construction [21], which uses bilinear groups whose order is the product of three primes. Our construction uses groups whose order is the product of four primes. Our complexity assumptions have all been introduced in prior work but for the case of groups whose order is the product of fewer than four primes.

**Why Inner Product Queries?** An important goal in predicate encryption is to support complex, expressive queries. Prior work has focused on achieving more expressive schemes, with the most expressive scheme to date being that of Katz, Sahai and Waters [21]. The KSW scheme supports inner product queries, which are strictly more expressive than conjunctive queries and, as shown in [21], imply conjunctions, disjunctions, CNF/DNF formulas, polynomial evaluation, and exact thresholds. Therefore, our goal in this work is to construct a symmetric-key predicate encryption scheme that supports inner product queries.

[1]In some works the authors also distinguish an extra "payload message" $M$ such that in the case that one of $f_1(x), \ldots, f_\ell(x)$ evaluates to 1, the adversary learns the payload message $M$. In our work we solely consider the predicate encryption system property where the evaluation reveals $f(x)$.

### 1.2 Related Work

**Public-Key Predicate Encryption.** The earliest examples of public-key predicate encryption are *anonymous identity-based encryption* (A-IBE) schemes with *keyword search* (which corresponds to an equality predicate) [7, 12, 1, 9]. Since then, more expressive schemes such as those supporting conjunctive queries [19, 11, 28] and multi-dimensional range queries [27] have been proposed. The most expressive scheme known to date is due to Katz, Sahai and Waters [21] and supports inner-product queries. As explained above, the KSW scheme is strictly more expressive than previously proposed predicate encryption schemes.

**Searchable Encrypted Databases.** A related line of research is secure searching on outsourced encrypted databases. The problem was considered by Goldreich and Ostrovsky [22, 18] when cast as a problem of oblivious RAM, and they provided general solutions. Song, Wagner, and Perrig [29] later gave more efficient solutions for equality searches, but made a tradeoff of letting a storage server learn the access pattern of a user. Curtmola et al. [17] considered stronger security definitions in a similar setting. While we do not directly address searchable encrypted databases in this work, our predicate encryption solution allows for more complex queries to be made in this particular application.

**Identity-Based Encryption and Attribute-Based Encryption.** Identity-based encryption (IBE) [26, 8, 16] can be viewed as a special, more limited, case of predicate encryption for the class of equality tests. In attribute-based encryption (ABE) [25, 20, 3, 24, 15, 23], a user can receive a capability representing an access control policy over the attributes of an encrypted record.

In both IBE and ABE schemes, the identity or attributes are not hidden in the ciphertext. In fact, access to the encrypted data itself is inherently "all-or-nothing." The important distinction between these systems and the ones we consider is that they only hide a "payload message" $M$. In particular, the ciphertext is associated with a payload message $M$ and some extra structure $x$ (e.g., the "identity" or set of attributes associated with the ciphertext). The security guarantee of these systems is that $M$ remains hidden as along as the attacker does not have a secret key associated with a predicate function $f$ such that $f(x) = 1$; however, there is no guarantee about hiding the structure of $x$, which in general might be leaked to the attacker. One advantage, however, is that this relaxation might allow for more expressive access predicates.

## 2 Definitions

In this section, we formally define symmetric-key predicate encryption and its security. For simplicity, we consider the *predicate-only* variant, in which evaluating a token on a ciphertext outputs a bit indicating whether the encrypted plaintext satisfies the predicate corresponding to the token. We note that a predicate-only scheme can easily be extended to obtain a full-fledged predicate encryption scheme, in which evaluating a token on a ciphertext outputs the encrypted plaintext if the plaintext satisfies the predicate corresponding to the token, using techniques from prior work such as [11, 27, 21].

We give definitions for the general case of an arbitrary set of plaintexts $\Sigma$ and an arbitrary set of predicates $\mathcal{F}$. Our construction in Section 4 will be for the specific case of $\Sigma = \mathbb{Z}_N^n$ and

$\mathcal{F} = \{f_{\vec{v}} | \vec{v} \in \mathbb{Z}_N^n\}$ with $f_{\vec{x}}(\vec{v}) = 1$ iff $\langle \vec{x}, \vec{v} \rangle = 0 \mod N$, where $\langle \vec{x}, \vec{v} \rangle$ denotes the inner product $\sum_{i=1}^n x_i \cdot v_i \mod N$ of vectors $\vec{x}$ and $\vec{v}$. We follow the notation of [21].

## 2.1 Symmetric-Key Predicate-Only Encryption

Let $\Sigma$ denote a finite set of plaintexts, and let $\mathcal{F}$ denote a finite set of predicates $f : \Sigma \to \{0, 1\}$. We say that $x \in \Sigma$ satisfies a predicate $f$ if $f(x) = 1$.

**Definition 2.1** (Symmetric-Key Predicate-Only Encryption Scheme)**.** A *symmetric-key predicate-only encryption scheme* for the class of predicates $\mathcal{F}$ over the set of attributes $\Sigma$ consists of the following probabilistic polynomial time (PPT) algorithms.

***Setup***$(1^\lambda)$**:** Takes as input a security parameter $1^\lambda$ and outputs a secret key $SK$.

***Encrypt***$(SK, x)$**:** Takes as input a secret key $SK$ and a plaintext $x \in \Sigma$ and outputs a ciphertext $CT$.

***GenToken***$(SK, f)$**:** Takes as input a secret key $SK$ and a description of a predicate $f \in \mathcal{F}$ and outputs a token $TK_f$.

***Query***$(TK_f, CT)$**:** Takes as input a token $TK_f$ for a predicate $f$ and a ciphertext $CT$. It outputs either 0 or 1, indicating the value of the predicate $f$ evaluated on the underlying plaintext.

**Correctness.** For correctness, we require the following condition. For all $\lambda$, all $x \in \Sigma$, and all $f \in \mathcal{F}$, letting $SK \leftarrow \mathit{Setup}(1^\lambda)$, $TK_f \leftarrow \mathit{GenToken}(SK, f)$, and $CT \leftarrow \mathit{Encrypt}(SK, x)$,

- If $f(x) = 1$, then $\mathit{Query}(TK_f, CT) = 1$.
- If $f(x) = 0$, then $\Pr[\mathit{Query}(TK_f, CT) = 0] > 1 - \epsilon(\lambda)$ where $\epsilon$ is a negligible function.

## 2.2 Security Definitions

We now give formal definitions of security for a symmetric-key predicate-only encryption scheme. We first define *full security*, which, roughly speaking, says that given a set of tokens for predicates $f_1, \ldots, f_k$ and a set of encryptions of plaintexts $x_1, \ldots, x_\ell$, an adversary $\mathcal{A}$ gains no information about any of the predicates $f_1, \ldots, f_k$ or the plaintexts $x_1, \ldots, x_\ell$ (other than the value of each of the predicates evaluated on each of the plaintexts).

However, the full security notion turns out to be difficult to work with in our proofs of security. Therefore, we introduce a second security notion called *single challenge security*, which resembles the security notions used in previous work such as [11, 21]. As we show later, full security implies single challenge security, and, for the specific case of inner product predicates, single challenge security implies full security in the sense that, given a single challenge secure scheme for inner product predicates over $\Sigma = \mathbb{Z}_N^{2n}$, we can construct a fully secure scheme for inner product predicates over $\Sigma = \mathbb{Z}_N^n$. Therefore, for our construction it suffices to consider the single challenge security definition. To prove the security of our construction, we will use the *selective* relaxation of single challenge security. The notion of selective security was first introduced by [13] and has been used widely in the literature [13, 14, 5, 11, 12, 27].

### 2.2.1 Full Security

We define full security of a symmetric-key predicate-only encryption scheme using the following game between an adversary $\mathcal{A}$ and a challenger.

**Setup:** The challenger runs $Setup(1^\lambda)$ and keeps $SK$ to itself. The challenger picks a random bit $\mathsf{b}$.

**Queries:** $\mathcal{A}$ adaptively issues queries, where each query is of one of two types:

- Ciphertext query. On the $j$th ciphertext query, $\mathcal{A}$ outputs a bit $t = 0$ (indicating a ciphertext query) and two plaintexts $x_{j,0}, x_{j,1} \in \Sigma$. The challenger responds with $Encrypt(SK, x_{j,\mathsf{b}})$.
- Token query. On the $i$th token query, $\mathcal{A}$ outputs a bit $t = 1$ (indicating a token query) and descriptions of two predicates $f_{i,0}, f_{i,1} \in \mathcal{F}$. The challenger responds with $GenToken(SK, f_{i,\mathsf{b}})$.

$\mathcal{A}$'s queries are subject to the restriction that, for all ciphertext queries $(x_{j,0}, x_{j,1})$ and all predicate queries $(f_{i,0}, f_{i,1})$, $f_{i,0}(x_{j,0}) = f_{i,1}(x_{j,1})$.

**Guess:** $\mathcal{A}$ outputs a guess $\mathsf{b}'$ of $\mathsf{b}$.

The advantage of $\mathcal{A}$ is defined as $\mathrm{Adv}_{\mathcal{A}} = \left|\Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}\right|$.

**Definition 2.2** (Full Security)**.** A symmetric-key predicate-only encryption scheme is *fully secure* if, for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the above game is negligible in $\lambda$.

### 2.2.2 Single Challenge Security

In order to prove the security of our construction, we will need to introduce a second security definition, which we refer to as *single challenge security*. Whereas in the full security game, each of the adversary's queries is considered part of the challenge, in the single challenge security game, the challenge consists of only one pair of plaintexts or predicates. The single challenge security game resembles security games used previously in the IBE and predicate encryption literature. The game proceeds as follows.

**Setup:** The challenger runs $Setup(1^\lambda)$ and keeps $SK$ to itself.

**Query Phase 1:** $\mathcal{A}$ adaptively issues queries, where each query is of one of two types:

- Ciphertext query. On the $j$th ciphertext query, $\mathcal{A}$ outputs a bit $t = 0$ (indicating a ciphertext query) and a plaintext $x_j$. The challenger responds with $Encrypt(SK, x_j)$.
- Token query. On the $j$th token query, $\mathcal{A}$ outputs a bit $t = 1$ (indicating a token query) and a description of a predicate $f_j$. The challenger responds with $GenToken(SK, f_j)$.

**Challenge:** $\mathcal{A}$ outputs a request for one of the following:

- Ciphertext challenge. $\mathcal{A}$ outputs a bit $t = 0$ (indicating a ciphertext challenge) and two plaintexts $x_0^*$ and $x_1^*$ such that, for all previous token queries $f_j$, $f_j(x_0^*) = f_j(x_1^*)$. The challenger picks a random bit $\mathsf{b}$ and responds with $Encrypt(SK, x_{\mathsf{b}}^*)$.

- Token challenge. $\mathcal{A}$ outputs a bit $t = 1$ (indicating a token challenge) and descriptions of two predicates $f_0^*$ and $f_1^*$ such that, for all previous ciphertext queries $x_j$, $f_0^*(x_j) = f_1^*(x_j)$. The challenger picks a random bit $\mathsf{b}$ and responds with $GenToken(SK, f_{\mathsf{b}}^*)$.

**Query Phase 2:** $\mathcal{A}$ adaptively issues additional queries as in Query Phase 1, subject to the same restriction with respect to the challenge as above.

**Guess:** $\mathcal{A}$ outputs a guess $\mathsf{b}'$ of $\mathsf{b}$.

The advantage of $\mathcal{A}$ is defined as $\mathrm{Adv}_{\mathcal{A}} = \left|\Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}\right|$.

**Definition 2.3** (Single Challenge Security)**.** A symmetric-key predicate-only encryption scheme is *single challenge secure* if, for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the above game is negligible in $\lambda$.

**Selective Single Challenge Security.** We will need to use the *selective* variant of single challenge security, defined below. The notion of selective security was first introduced by [13] and has been used previously by [13, 14, 5, 11, 12, 27].

**Definition 2.4** (Selective Single Challenge Security)**.** In the selective single challenge security game, the adversary $\mathcal{A}$ outputs the challenge strings at the start of the game during an **Init** phase (instead of during a **Challenge** phase). The rest of the game proceeds in the same way as in the single challenge security game. We say that a symmetric-key predicate-only encryption scheme is *selective single challenge secure* if, for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the selective single challenge game is negligible in $\lambda$.

For our proofs of security, it will be useful to define separate notions of plaintext privacy and predicate privacy, which correspond to a ciphertext challenge and a token challenge, respectively, in the selective single challenge security game.

**Definition 2.5** (Plaintext Privacy)**.** A symmetric-key predicate-only encryption scheme has *selective single challenge plaintext privacy* (*plaintext privacy*, for short) if, for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the selective single challenge game for a ciphertext challenge is negligible in $\lambda$.

**Definition 2.6** (Predicate Privacy)**.** A symmetric-key predicate-only encryption scheme has *selective single challenge predicate privacy* (*predicate privacy*, for short) if, for all PPT adversaries $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the selective single challenge game for a token challenge is negligible in $\lambda$.

We note that plaintext privacy and predicate privacy, together, are equivalent to selective single challenge security.

### 2.2.3 Relationship Between Single Challenge Security and Full Security

It is useful to consider the relationship between the security definitions introduced above. The full security notion implies single challenge security. For the specific case of inner product query predicates, a single challenge secure scheme for vectors of length $2n$ can be used to construct a fully secure scheme for vectors of length $n$. Therefore, we consider single challenge security to be a sufficiently strong notion of security for our construction.

These relationships are stated formally in the following theorems.

**Theorem 2.7.** *If a symmetric-key predicate-only encryption scheme is fully secure, then it is single challenge secure.*

*Proof.* Suppose an adversary $\mathcal{A}$ wins the single challenge security game with advantage $\epsilon$. We can define an adversary $\mathcal{B}$ that wins the full security game with advantage $\epsilon$ as follows. When $\mathcal{A}$ makes a ciphertext query $\vec{x}$, $\mathcal{B}$ in turn makes the ciphertext query $(\vec{x}, \vec{x})$ to $\mathcal{B}$'s challenger and responds to $\mathcal{A}$ with the ciphertext it receives. Similarly, when $\mathcal{A}$ makes a token query $\vec{v}$, $\mathcal{B}$ in turn makes the token query $(\vec{v}, \vec{v})$ to $\mathcal{B}$'s challenger and responds to $\mathcal{A}$ with the token it receives. When $\mathcal{A}$ issues its challenge request, $\mathcal{B}$ outputs the challenge request as a query to its challenger and responds to $\mathcal{A}$ with the answer it receives. $\mathcal{B}$ outputs the same guess $\mathsf{b}'$ as $\mathcal{A}$ does. It is clear that all of $\mathcal{B}$'s responses to $\mathcal{A}$ are properly constructed, and $\mathcal{B}$ wins the full security game with the same advantage $\epsilon$ with which $\mathcal{A}$ wins the single challenge security game. $\square$

**Theorem 2.8.** *Let* $\text{Scheme}_{2n}$ *denote a single challenge secure symmetric-key predicate-only encryption scheme for inner product queries, where plaintext and predicate vectors have length* $2n$*. Then* $\text{Scheme}_{2n}$ *can be used to construct a fully secure symmetric-key predicate-only encryption scheme* $\text{Scheme}_n$ *for inner product queries, where plaintext and predicate vectors have length* $n$*.*

The proof of this theorem is deferred to Appendix B.

# 3 Background and Complexity Assumptions

Our symmetric-key predicate encryption scheme uses bilinear groups of composite order, first introduced by [10]. While the public-key predicate encryption scheme of [21] uses bilinear groups whose order is the product of three distinct primes, we use bilinear groups whose order is the product of four distinct primes.

We briefly review some facts about bilinear groups and then state the assumptions we use to prove security of our construction.

## 3.1 Bilinear Groups of Composite Order

Let $\mathcal{G}$ denote a group generator algorithm that takes as input a security parameter $1^\lambda$ and outputs a tuple $(p, q, r, s, \mathbb{G}, \mathbb{G}_T, e)$ where $p, q, r, s$ are distinct primes, $\mathbb{G}$ and $\mathbb{G}_T$ are two cyclic groups of order $N = pqrs$, and $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ satisfies the following properties:

- (Bilinear) $\forall u, v \in \mathbb{G}$, $\forall a, b \in \mathbb{Z}$, $e(u^a, v^b) = e(u, v)^{ab}$.
- (Non-degenerate) $\exists g \in \mathbb{G}$ such that $e(g, g)$ has order $N$ in $\mathbb{G}_T$.

We assume that group operations in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the bilinear map $e$ can be computed in time polynomial in $\lambda$.

We use the notation $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_s$ to denote the subgroups of $\mathbb{G}$ having order $p, q, r, s$, respectively.

We will use the following facts about bilinear groups of composite order. Although these facts are stated in terms of $\mathbb{G}_p$ and $\mathbb{G}_q$, similar facts hold in general for distinct subgroups of a composite order bilinear group.

- Let $a_p \in \mathbb{G}_p, b_q \in \mathbb{G}_q$ denote two elements from distinct subgroups. Then $e(a_p, b_q) = 1$.

- Let $\mathbb{G}_{pq} = \mathbb{G}_p \times \mathbb{G}_q$, $a, b \in \mathbb{G}_{pq}$. $a$ and $b$ can be rewritten (uniquely) as $a = a_p a_q$, $b = b_p b_q$, where $a_p, b_q \in \mathbb{G}_p$, and $a_q, b_q \in \mathbb{G}_q$. Then $e(a, b) = e(a_p, b_p)e(a_q, b_q)$.

## 3.2 Our Assumptions

The security of our symmetric-key predicate-only encryption scheme relies on three assumptions. All of these assumptions have been introduced previously but in groups whose order is the product of at most three distinct primes. Specifically, Assumption 1 involves 3 subgroups, C3DH involves 2 subgroups, and DL involves 1 subgroup. We assume that these assumptions hold when the relevant subgroups are contained in a larger group whose order is the product of four distinct primes. Note that the naming of subgroups is not significant in our assumptions; that is, the assumptions are the same if the subgroups are renamed.

**Assumption 1** We use Assumption 1 of KSW [21], which was used for bilinear groups whose order is the product of three distinct primes. We restate the assumption in the context of a bilinear group whose order is the product of four distinct primes.

Let $\mathcal{G}$ be a group generator algorithm as above. Run $\mathcal{G}(1^\lambda)$ to obtain $(p, q, r, s, \mathbb{G}, \mathbb{G}_T, e)$. Let $N = pqrs$ and let $g_p, g_q, g_r, g_s$ be random generators of $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_s$, respectively. Choose random $Q_1, Q_2, Q_3 \in \mathbb{G}_q$, random $R_1, R_2, R_3 \in \mathbb{G}_r$, random $a, b, c \in \mathbb{Z}_p$, and a random bit $\mathsf{b}$. If $\mathsf{b} = 0$, let $T = g_p^{b^2c} R_3$; if $\mathsf{b} = 1$, let $T = g_p^{b^2c} Q_3 R_3$. Give the adversary $\mathcal{A}$ the description of the bilinear group, $(N, \mathbb{G}, \mathbb{G}_T, e)$, along with the following values:

$$\Big(g_p,\ \ g_r,\ \ g_s,\ \ g_q R_1,\ \ g_p^b,\ \ g_p^{b^2},\ \ g_p^a g_q,\ \ g_p^{ab} Q_1,\ \ g_p^c,\ \ g_p^{bc} Q_2 R_2,\ \ T\Big)$$

The adversary $\mathcal{A}$ outputs a guess $\mathsf{b}'$ of $\mathsf{b}$. The advantage of $\mathcal{A}$ is defined as $\mathrm{Adv}_{\mathcal{A}} = \left|\Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}\right|$.

**Definition 3.1.** We say that $\mathcal{G}$ satisfies Assumption 1 if, for all PPT algorithms $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the above game is negligible in the security parameter $\lambda$.

We note that Assumption 1 implies the hardness of finding a non-trivial factor of $N$.

**Generalized 3-Party Diffie-Hellman Assumption (C3DH).** We use the composite 3-party Diffie-Hellman assumption first introduced by [11]. We restate the assumption in the context of a bilinear group whose order is the product of four distinct primes.

Let $\mathcal{G}$ be a group generator algorithm as above. Run $\mathcal{G}(1^\lambda)$ to obtain $(p, q, r, s, \mathbb{G}, \mathbb{G}_T, e)$. Let $N = pqrs$ and let $g_p, g_q, g_r, g_s$ be random generators of $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_s$, respectively. Choose random $R_1, R_2, R_3 \in \mathbb{G}_r$, random $a, b, c \in \mathbb{Z}_N$, and a random bit $\mathsf{b}$. If $\mathsf{b} = 0$, let $T = g_p^c \cdot R_3$; if $\mathsf{b} = 1$, let $T$ be a random element in $\mathbb{G}_{pr} = \mathbb{G}_p \times \mathbb{G}_r$. Give the adversary $\mathcal{A}$ the description of the bilinear group, $(N, \mathbb{G}, \mathbb{G}_T, e)$, along with the following values:

$$\Big(g_p,\ \ g_q,\ \ g_r,\ \ g_s,\ \ g_p^a,\ \ g_p^b,\ \ g_p^{ab} \cdot R_1,\ \ g_p^{abc} \cdot R_2,\ \ T\Big)$$

The adversary $\mathcal{A}$ outputs a guess $\mathsf{b}'$ of $\mathsf{b}$. The advantage of $\mathcal{A}$ is defined as $\mathrm{Adv}_{\mathcal{A}} = \left|\Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}\right|$.

**Definition 3.2.** We say that $\mathcal{G}$ satisfies the C3DH assumption if for all PPT algorithms $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the above game is negligible in the security parameter $\lambda$.

We note that the C3DH assumption implies the hardness of finding a non-trivial factor $N$.

**Decisional Linear assumption (DLinear).** We use the Decisional Linear assumption introduced by [6]. We restate the assumption in the context of a bilinear group whose order is the product of four distinct primes.

Let $\mathcal{G}$ be a group generator algorithm as above. Run $\mathcal{G}(1^\lambda)$ to obtain $(p, q, r, s, \mathbb{G}, \mathbb{G}_T, e)$. Let $N = pqrs$ and let $g_p, g_q, g_r, g_s$ be random generators of $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_s$, respectively. Choose random $z_1, z_2, z_3, z_4 \in \mathbb{Z}_p$ and a random bit $\mathsf{b}$. If $\mathsf{b} = 0$, let $Z = g_p^{z_3+z_4}$; if $\mathsf{b} = 1$, let $Z$ be a random element in $\mathbb{G}_p$. Give the adversary $\mathcal{A}$ the description of the bilinear group, $(N, \mathbb{G}, \mathbb{G}_T, e)$, along with the following values:

$$(g_p,\quad g_q,\quad g_r,\quad g_s,\quad g_p^{z_1},\quad g_p^{z_2},\quad g_p^{z_1 z_3},\quad g_p^{z_2 z_4},\quad Z)$$

The adversary $\mathcal{A}$ outputs a guess $\mathsf{b}'$ of $\mathsf{b}$. The advantage of $\mathcal{A}$ is defined as $\mathrm{Adv}_\mathcal{A} = \left|\Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}\right|$.

**Definition 3.3.** We say that $\mathcal{G}$ satisfies the DLinear assumption if for all PPT algorithms $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the above game is negligible in the security parameter $n$.

# 4 Construction

Our goal is to construct a symmetric-key predicate encryption scheme supporting inner product queries that has both plaintext privacy and predicate privacy. The KSW construction [21] is a public-key predicate encryption scheme supporting inner product queries that has plaintext privacy. A natural first attempt might be to convert the KSW scheme into a symmetric-key scheme simply by withholding the public key. Such a scheme would immediately inherit plaintext privacy from the KSW construction. However, it is difficult to prove the predicate privacy of such a scheme. Our primary challenge is to achieve predicate privacy.

To achieve predicate privacy, we use the observation that, for inner product queries, ciphertexts and tokens play symmetric roles in the scheme and the security definitions. In particular, a token and a ciphertext each encode a vector in $\mathbb{Z}_N^n$, and the inner product $\langle \vec{x}, \vec{v} \rangle$ is commutative. Furthermore, for inner products, ciphertexts and tokens have symmetric roles in the security definitions. One way to interpret this observation is to view a ciphertext as an encryption of a plaintext vector and a token as an encryption of a predicate vector.

Based on this observation, our general approach is to start from a construction that resembles the KSW construction, so that we can prove plaintext privacy in a relatively straightforward manner. We then show through a series of modifications to our construction that it is indistinguishable from one in which ciphertexts and tokens are formed symmetrically. Using this symmetry, we can leverage the plaintext privacy proven for our main construction to achieve predicate privacy as well. Taken all together, the "native" formation of our system gives us plaintext privacy by a KSW type of approach, and the indistinguishability of our construction from one in which ciphertexts and tokens are symmetrically formed implies that our construction also has predicate privacy.

## 4.1 A Symmetric-Key Predicate Encryption Scheme

Our main construction is a symmetric-key predicate-only encryption scheme supporting inner product queries. We take the class of plaintexts to be $\Sigma = \mathbb{Z}_N^n$ and the class of predicates to be $\mathcal{F} = \{f_{\vec{v}} | \vec{v} \in \mathbb{Z}_N^n\}$ with $f_{\vec{x}}(\vec{v}) = 1$ iff $\langle \vec{x}, \vec{v} \rangle = 0 \mod N$.

We now describe our construction in detail.

***Setup***$(1^\lambda)$**:** The setup algorithm runs $\mathcal{G}(1^\lambda)$ to obtain $(p, q, r, s, \mathbb{G}, \mathbb{G}_T, e)$ with $\mathbb{G} = \mathbb{G}_p \times \mathbb{G}_q \times \mathbb{G}_r \times \mathbb{G}_s$. Next it picks generators $g_p, g_q, g_r, g_s$ of $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_s$, respectively. It chooses $h_{1,i}, h_{2,i}, u_{1,i}, u_{2,i} \in \mathbb{G}_p$ uniformly at random for $i = 1$ to $n$. The secret key is

$$SK = \left(g_p,\ g_q,\ g_r,\ g_s,\ \{h_{1,i}, h_{2,i}, u_{1,i}, u_{2,i}\}_{i=1}^n\right).$$

***Encrypt***$(SK, \vec{x})$**:** Let $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{Z}_N^n$. The encryption algorithm chooses random $y, z, \alpha, \beta \in \mathbb{Z}_N$, random $S, S_0 \in \mathbb{G}_s$, and random $R_{1,i}, R_{2,i} \in \mathbb{G}_r$ for $i = 1$ to $n$. It outputs the ciphertext

$$CT = \left( \begin{array}{c} C = S \cdot g_p^y, \quad C_0 = S_0 \cdot g_p^z, \\ \left\{ C_{1,i} = h_{1,i}^y \cdot u_{1,i}^z \cdot g_q^{\alpha x_i} \cdot R_{1,i}, \quad C_{2,i} = h_{2,i}^y \cdot u_{2,i}^z \cdot g_q^{\beta x_i} \cdot R_{2,i} \right\}_{i=1}^n \end{array} \right).$$

***GenToken***$(SK, \vec{v})$**:** Let $\vec{v} = (v_1, \ldots, v_n) \in \mathbb{Z}_N^n$. The token generation algorithm chooses random $f_1, f_2 \in \mathbb{Z}_N$, random $r_{1,i}, r_{2,i} \in \mathbb{Z}_N$ for $i = 1$ to $n$, random $R, R_0 \in \mathbb{G}_r$, and random $S_{1,i}, S_{2,i} \in \mathbb{G}_s$ for $i = 1$ to $n$. It outputs the token

$$TK_{\vec{v}} = \left( \begin{array}{c} K = R \cdot \prod_{i=1}^n h_{1,i}^{-r_{1,i}} \cdot h_{2,i}^{-r_{2,i}}, \quad K_0 = R_0 \cdot \prod_{i=1}^n u_{1,i}^{-r_{1,i}} \cdot u_{2,i}^{-r_{2,i}}, \\ \left\{ K_{1,i} = g_p^{r_{1,i}} \cdot g_q^{f_1 v_i} \cdot S_{1,i}, \quad K_{2,i} = g_p^{r_{2,i}} \cdot g_q^{f_2 v_i} \cdot S_{2,i} \right\}_{i=1}^n \end{array} \right).$$

***Query***$(TK_{\vec{v}}, CT)$ : Let $CT = (C, C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^n)$ and $TK_{\vec{v}} = (K, K_0, \{K_{1,i}, K_{2,i}\}_{i=1}^n)$ as above. The query algorithm outputs 1 iff

$$e(C, K) \cdot e(C_0, K_0) \cdot \prod_{i=1}^n e(C_{1,i}, K_{1,i}) \cdot e(C_{2,i}, K_{2,i}) \stackrel{?}{=} 1.$$

**Correctness.** Let $CT$ and $TK_{\vec{v}}$ be as above. Then

$$\begin{aligned}
& e(C, K) \cdot e(C_0, K_0) \cdot \prod_{i=1}^n e(C_{1,i}, K_{1,i}) \cdot e(C_{2,i}, K_{2,i}) \\
= \quad & e(S \cdot g_p^y,\ R \cdot \prod_{i=1}^n h_{1,i}^{-r_{1,i}} \cdot h_{2,i}^{-r_{2,i}}) \cdot e(S_0 \cdot g_p^z,\ R_0 \cdot \prod_{i=1}^n u_{1,i}^{-r_{1,i}} \cdot u_{2,i}^{-r_{2,i}}) \\
& \cdot \prod_{i=1}^n e(h_{1,i}^y \cdot u_{1,i}^z \cdot g_q^{\alpha x_i} \cdot R_{1,i},\ g_p^{r_{1,i}} \cdot g_q^{f_1 v_i} \cdot S_{1,i}) \\
& \cdot e(h_{2,i}^y \cdot u_{2,i}^z \cdot g_q^{\beta x_i} \cdot R_{2,i},\ g_p^{r_{2,i}} \cdot g_q^{f_2 v_i} \cdot S_{2,i}) \\
= \quad & e(g_p^y,\ \prod_{i=1}^n h_{1,i}^{-r_{1,i}} \cdot h_{2,i}^{-r_{2,i}}) \cdot e(g_p^z,\ \prod_{i=1}^n u_{1,i}^{-r_{1,i}} \cdot u_{2,i}^{-r_{2,i}}) \\
& \cdot \prod_{i=1}^n e(h_{1,i}^y \cdot u_{1,i}^z \cdot g_q^{\alpha x_i},\ g_p^{r_{1,i}} \cdot g_q^{f_1 v_i}) \cdot e(h_{2,i}^y \cdot u_{2,i}^z \cdot g_q^{\beta x_i},\ g_p^{r_{2,i}} \cdot g_q^{f_2 v_i}) \\
= \quad & \prod_{i=1}^n e(g_q, g_q)^{(\alpha f_1 + \beta f_2) x_i v_i} = e(g_q, g_q)^{(\alpha f_1 + \beta f_2 \mod q)\langle \vec{x}, \vec{v} \rangle}
\end{aligned}$$

If $\langle \vec{x}, \vec{v} \rangle = 0 \mod N$, then the above expression evaluates to 1. If $\langle \vec{x}, \vec{v} \rangle \neq 0 \mod N$, then there are two cases. If $\langle \vec{x}, \vec{v} \rangle = 0 \mod q$, then the above expression evaluates to 1; however, this case would reveal a non-trivial factor of $N$ and, therefore, this case occurs with negligible probability. If $\langle \vec{x}, \vec{v} \rangle \neq 0 \mod q$, then with all but negligible probability $\alpha f_1 + \beta f_2 \neq 0 \mod q$ and the above expression does not evaluate to 1.

### 4.2 Discussion

To understand our construction, it is useful to examine the role of each of the subgroups $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_s$.

The $\mathbb{G}_q$ subgroup is used to encode the plaintext vector $\vec{x}$ in the $C_{1,i}$ and $C_{2,i}$ terms of the ciphertext and the predicate vector $\vec{v}$ in the $K_{1,i}$ and $K_{2,i}$ terms of the token. When a token for $\vec{v}$ is applied to an encryption of $\vec{x}$, the computation of the inner product $\langle \vec{x}, \vec{v} \rangle$ is evaluated in the exponent of the $\mathbb{G}_q$ subgroup.

The $\mathbb{G}_p$ subgroup is used to prevent an adversary from manipulating components of either a ciphertext or a token and then evaluating a query on the improperly formed inputs. The $\mathbb{G}_p$ subgroup encodes an equation which will evaluate to 0 in the exponent if the inputs to the query algorithm are properly formed.

The $\mathbb{G}_r$ subgroup is used for to hide factors from other subgroups and ensure plaintext privacy. In an analogous manner, the $\mathbb{G}_s$ subgroup is used to ensure predicate privacy. Also, the additional subgroup $\mathbb{G}_s$ allows us to construct our scheme in a slightly different manner from KSW. For example, the $\mathbb{G}_s$ subgroup allows us to eliminate the factor $Q$ from the $\mathbb{G}_q$ subgroup in the $K$ term of the token.

As discussed earlier, in our proofs of security we will need to show that our main construction is computationally indistinguishable from a scheme in which ciphertexts and tokens are formed symmetrically. In the KSW construction, all terms in the ciphertext have the same exponent $y$ in the $\mathbb{G}_p$ subgroup. In our construction, we introduce an additional degree of randomness using the exponent $z$. Terms in the ciphertext now contain two degrees of randomness, $y$ and $z$, in the $\mathbb{G}_p$ subgroup. This change is necessary to ensure symmetry of the ciphertext and the token in the $\mathbb{G}_p$ subgroup.

To see why this is the case, recall that Decisional Diffie-Hellman is easy in bilinear groups. That is, for a random vector $g_p^{\alpha_1}, g_p^{\alpha_2}, \ldots, g_p^{\alpha_k}$, it is easy to decide whether the exponents $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ are picked independently at random or picked from a prescribed one-dimensional subspace. On the other hand, an informal interpretation of the Decisional Linear assumption tells us that it is computationally hard to decide whether the exponents $(\alpha_1, \alpha_2, \ldots, \alpha_k)$ are picked independently at random or picked randomly from a prescribed 2-dimensional subspace. The reason for introducing the extra randomness $z$ in the ciphertext is to ensure that the exponents in the $\mathbb{G}_p$ subgroup are picked from a 2-dimensional subspace instead of a 1-dimensional subspace.

Similarly to [11, 12, 21], our construction consists of two parallel sub-systems. Note that $C_{1,i}$ and $C_{2,i}$ (similarly, $K_{1,i}$ and $K_{2,i}$) play identical roles. Our proof of security will rely on having these two parallel sub-systems.

For comparison, we provide a review of the KSW construction in Appendix C.

### 4.3 Proof Overview

Our main security statement is the following theorem.

**Theorem 4.1.** *Under the generalized Assumption 1 of the KSW construction, the generalized C3DH assumption, and the Decisional Linear assumption, the symmetric-key predicate-only encryption scheme presented in Section 4.1 is selectively single challenge secure.*

Our proof technique consists of two steps. First, we prove that our construction achieves plaintext privacy. Second, we prove that, for our construction, plaintext privacy implies predicate privacy. Taken together, these results imply the security of our scheme.

Our construction defined above, which we call SchemeReal, does not immediately yield a proof of these two properties. In order to argue these properties, we define two different schemes that are computationally indistinguishable from our original construction. That is, no adversary can tell whether tokens and ciphertexts are generated from our actual system or from one of the two defined for the purposes of the proof.

We first define a system that we call SchemeQ, which very closely follows the KSW construction. We reduce the plaintext privacy of SchemeQ to the plaintext privacy of the KSW construction.

Next we define a system that we call SchemeSym, in which ciphertexts and tokens are formed symmetrically. For this system it is straightforward to argue that plaintext privacy implies predicate privacy.

We observe that since our main construction and the two variants defined are all computationally indistinguishable (from an adversary's view), it is actually possible to define any of them as the "real" construction that we will actually use. We chose the variant described above due to (relative) notational simplicity and slight efficiency advantages. We prove Theorem 4.1 in Appendix A.

# 5 Conclusions

We examined the idea of protecting the privacy of predicates in predicate encryption systems. While this turns out to be an inherently unachievable in a public-key system, we showed that there exist solutions in the symmetric-key setting. We first provided security definitions for predicate encryption schemes in the symmetric-key setting and then presented a construction supporting inner product queries, which are the most expressive queries supported by currently known schemes.

While semantic security of predicates is inherently impossible in the public-key setting, in the future we might wish to consider relaxations of public-key encryption. For example, is it possible to find interesting systems where predicates are drawn from a high entropy distribution, in a fashion similar to recent work on deterministic encryption [4, 2]? Another open direction is to consider "partial public-key encryption," in which a public key might allow a user to generate only a subset of valid ciphertexts. (The rest may be generated from a secret key or other public keys kept hidden from an attacker.) Thus, certain predicates might be indistinguishable given the partial public keys published.

# Acknowledgments

We gratefully thank Philippe Golle for helpful discussions. Elaine Shi thanks Adrian Perrig for his support while part of this research was conducted.

## 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 *Advances in Cryptology - Proceedings of CRYPTO '05*, pages 205–222. Springer-Verlag, August 2005.

[2] Mihir Bellare, Marc Fischlin, Adam O'Neill, and Thomas Ristenpart. Deterministic encryption: Definitional equivalences and constructions without random oracles. In *CRYPTO*, pages 360–378, 2008.

[3] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 321–334, Washington, DC, USA, 2007. IEEE Computer Society.

[4] Alexandra Boldyreva, Serge Fehr, and Adam O'Neill. On notions of security for deterministic encryption, and efficient constructions without random oracles. In *CRYPTO*, pages 335–359, 2008.

[5] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity based encryption without random oracles. In *Advances in Cryptology—EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Berlin: Springer-Verlag, 2004. Available at `http://www.cs.stanford.edu/~xb/eurocrypt04b/`.

[6] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *CRYPTO*, pages 41–55, 2004.

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

[8] 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.

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

[10] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In Joe Kilian, editor, *Proceedings of Theory of Cryptography Conference 2005*, volume 3378 of *LNCS*, pages 325–342. Springer, 2005.

[11] 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.

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

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

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

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

[16] 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.

[17] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, 2006.

[18] O. Goldreich and R. Ostrovsky. Software protection and simulation by oblivious rams. *JACM*, 1996.

[19] Philippe Golle, Jessica Staddon, and Brent Waters. Secure conjunctive keyword search over encrypted data. In *Proc. of the 2004 Applied Cryptography and Network Security Conference*, 2004.

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

[21] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Eurocrypt*, 2008.

[22] Rafail Ostrovsky. *Software protection and simulation on oblivious RAMs*. PhD thesis, M.I.T, 1992. Preliminary version in STOC 1990.

[23] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, 2007.

[24] 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*, pages 99–112, New York, NY, USA, 2006.

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

[26] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of Crypto '84*, volume 196 of *LNCS*, pages 47–53. Springer-Verlag, 1984.

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

[28] Elaine Shi and Brent Waters. Delegating capabilities in predicate encryption systems. In *Proceedings of ICALP*, 2008. Full version can be found online at `http://sparrow.ece.cmu.edu/~elaine/docs/delegation.pdf`.

[29] Dawn Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *Proceedings of the 2000 IEEE symposium on Security and Privacy (S&P 2000)*, 2000.

# A Proof of Security

In this section, we prove Theorem 4.1, which states the selective single challenge security (cf. Definition 2.4) of our main construction. To do this, it suffices to prove plaintext privacy and predicate privacy separately.

In our proof, we make a series of modifications to our main construction and show that each resulting construction is *computationally indistinguishable* from our main construction. To prove that our main scheme has plaintext privacy (or predicate privacy), it suffices to prove that a scheme that is computationally indistinguishable from our main scheme has plaintext privacy (or predicate privacy). The following definition formalizes the notion of computational indistinguishability of symmetric-key predicate encryption schemes.

**Definition A.1** (Indistinguishability of Symmetric-Key Predicate Encryption Schemes)**.** We say that two symmetric-key predicate encryption schemes $\textsc{Scheme}_0$ and $\textsc{Scheme}_1$ are *computationally indistinguishable* if no polynomial-time adversary $\mathcal{A}$ has more than negligible advantage in winning the following distinguishing game:

**Setup.** The challenger chooses a random bit $\mathsf{b}$. The challenger runs $\textsc{Scheme}_{\mathsf{b}}.\mathit{Setup}(1^\lambda)$ and keeps $SK$ to itself.

**Queries.** The adversary $\mathcal{A}$ adaptively issues queries where each query is one of the following:

- Ciphertext query. $\mathcal{A}$ outputs a bit $t = 0$ (indicating a ciphertext query) and a plaintext $x$. The challenger responds with $\textsc{Scheme}_{\mathsf{b}}.\mathit{Encrypt}(SK, x)$.
- Token query. $\mathcal{A}$ outputs a bit $t = 1$ (indicating a token query) and a description of a predicate $f$. The challenger responds with $\textsc{Scheme}_{\mathsf{b}}.\mathit{GenToken}(SK, f)$.

**Guess.** Finally, $\mathcal{A}$ outputs a guess $\mathsf{b}'$ of $\mathsf{b}$.

The advantage of $\mathcal{A}$ is defined as $\mathrm{Adv}_{\mathcal{A}} = \left|\Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}\right|$.

The notion of computational indistinguishability of two schemes will be useful throughout our proof, as, to prove plaintext privacy (or predicate privacy) of a scheme $\textsc{Scheme}_0$, it suffices to prove plaintext privacy (or predicate privacy) of a scheme $\textsc{Scheme}_1$ which is computationally indistinguishable from $\textsc{Scheme}_0$. This is formally stated in the following proposition.

**Proposition A.2.** *Let* $\textsc{Scheme}_0$ *and* $\textsc{Scheme}_1$ *denote two computationally indistinguishable symmetric-key predicate encryption schemes. Then* $\textsc{Scheme}_0$ *has plaintext privacy (or predicate privacy) if and only if* $\textsc{Scheme}_1$ *has plaintext privacy (or predicate privacy).*

*Proof.* Suppose for the purpose of contradiction, without loss of generality, that $\textsc{Scheme}_0$ has plaintext privacy and $\textsc{Scheme}_1$ does not have plaintext privacy. Then there exists a polynomial-time adversary $\mathcal{A}$ that can win the plaintext privacy game of $\textsc{Scheme}_1$ with non-negligible probability $\epsilon$. We can leverage this adversary $\mathcal{A}$ to build a simulator $\mathcal{B}$ that distinguishes between $\textsc{Scheme}_0$ and $\textsc{Scheme}_1$. The simulator $\mathcal{B}$ plays the role of the challenger in the plaintext privacy game with $\mathcal{A}$, forwarding $\mathcal{A}$'s queries to its own challenger and forwarding its challenger's responses to $\mathcal{A}$. If $\mathcal{A}$ wins the plaintext privacy game, $\mathcal{B}$ outputs a guess of $\mathsf{b}' = 1$; otherwise, it outputs $\mathsf{b}' = 0$. The simulator $\mathcal{B}$ has advantage at least $\epsilon/2$ in distinguishing between $\textsc{Scheme}_0$ and $\textsc{Scheme}_1$, contradicting the assumption that $\textsc{Scheme}_0$ and $\textsc{Scheme}_1$ are computationally indistinguishable. $\square$

### A.1 Assumptions

In addition to the three assumptions stated in Section 3.2, we will also use the following assumptions, which are implied by those previously stated. Note that these are not new assumptions, as they can be reduced to those assumptions already introduced.

**Assumption W** The Assumption W game is identical to the Assumption 1 game, except that, whereas the Assumption 1 game gives the adversary $\mathcal{A}$ the tuple

$$V = \left(g_p,\ \ g_r,\ \ g_s,\ \ g_q R_1,\ \ g_p^b,\ \ g_p^{b^2},\ \ g_p^a g_q,\ \ g_p^{ab} Q_1,\ \ g_p^c,\ \ g_p^{bc} Q_2 R_2,\ \ T = g_p^{b^2 c} g_q^{\gamma} R_3\right),$$

the Assumption W game gives the adversary $\mathcal{A}$ a subset $V' \subset V$:

$$V' = \left\{g_p,\ \ g_r,\ \ g_s,\ \ g_q R_1,\ \ g_p^{b^2},\ \ g_p^a g_q,\ \ g_p^c,\ \ T = g_p^{b^2 c} g_q^{\gamma} R_3\right\},$$

**Definition A.3.** We say that $\mathcal{G}$ satisfies the Assumption W if, for all PPT algorithms $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the above game is negligible in the security parameter $\lambda$.

Clearly, Assumption W is weaker than Assumption 1.

**$\ell$-C3DH Assumption** Let $\mathcal{G}$ be a group generator algorithm. Run $\mathcal{G}(1^\lambda)$ to obtain $(p, q, r, s, \mathbb{G}, \mathbb{G}_T, e)$. Let $N = pqrs$ and let $g_p, g_q, g_r, g_s$ be random generators of $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_s$, respectively. Choose random $w_1, w_2, \ldots,\ w_\ell \xleftarrow{\text{R}} \mathbb{G}_q$, random $\omega \xleftarrow{\text{R}} \mathbb{Z}_N$, random $\widehat{S}_1, \ldots, \widehat{S}_\ell,\ \widetilde{S}_1, \ldots, \widetilde{S}_\ell,\ \overline{S}_1, \ldots, \overline{S}_\ell \xleftarrow{\text{R}} \mathbb{G}_s$, and random $Q_1, Q_2, \ldots, Q_\ell \xleftarrow{\text{R}} \mathbb{G}_q$. Choose a random bit $\mathsf{b}$. Give the adversary $\mathcal{A}$ the description of the bilinear group, $(N, \mathbb{G}, \mathbb{G}_T, e)$. If $\mathsf{b} = 0$, give $\mathcal{A}$ the tuple

$$(w_1 \widehat{S}_1, \ldots, w_\ell \widehat{S}_\ell,\ w_1^\omega \widetilde{S}_1, \ldots, w_\ell^\omega \widetilde{S}_\ell).$$

If $\mathsf{b} = 1$, give $\mathcal{A}$ the tuple

$$(w_1 \widehat{S}_1, \ldots, w_\ell \widehat{S}_\ell,\ Q_1 \overline{S}_1, \ldots, Q_\ell \overline{S}_\ell).$$

The adversary $\mathcal{A}$ outputs a guess $\mathsf{b}'$ of $\mathsf{b}$. The advantage of $\mathcal{A}$ is defined as $\mathrm{Adv}_{\mathcal{A}} = \left|\Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}\right|$.

**Definition A.4.** We say that $\mathcal{G}$ satisfies the $\ell$-C3DH assumption if, for all PPT algorithms $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the above game is negligible in the security parameter $\lambda$.

The $\ell$-C3DH assumption can be shown to follow from the generalized C3DH assumption using a simple hybrid argument. Shi and Waters [28] also used the $\ell$-C3DH assumption as an intermediate assumption in their proofs.

**$\ell$-DLinear Assumption** Let $\mathcal{G}$ be a group generator algorithm. Run $\mathcal{G}(1^\lambda)$ to obtain $(p, q, r, s, \mathbb{G}, \mathbb{G}_T, e)$. Let $N = pqrs$ and let $g_p, g_q, g_r, g_s$ be random generators of $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_s$, respectively. Let $\ell$ be an integer greater than 2. Choose two random vectors $\vec{y} = (y_1, y_2, \ldots, y_\ell) \xleftarrow{\text{R}} \mathbb{F}_p^\ell$ and $\vec{z} = (z_1, z_2, \ldots, z_\ell) \xleftarrow{\text{R}} \mathbb{F}_p^\ell$. Choose a random bit $\mathsf{b}$. Choose a vector $\vec{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_\ell)$ in one of two ways, depending on the value of $\mathsf{b}$. If $\mathsf{b} = 0$, choose $\gamma_1, \gamma_2, \ldots, \gamma_\ell$ independently at random from $\mathbb{Z}_p$. In other words, the vector $\vec{\gamma}$ is picked at random from the vector space $\mathbb{F}_p^\ell$. If $\mathsf{b} = 1$, choose the vector $\vec{\gamma} = (\gamma_1, \gamma_2, \ldots, \gamma_\ell)$ from the 2-dimensional subspace $\mathsf{span}(\vec{y}, \vec{z})$ of $\mathbb{F}_p^\ell$ generated by $\vec{y}, \vec{z}$[2]. Specifically, choose random $w, t \xleftarrow{\text{R}} \mathbb{Z}_p$ and let $\vec{\gamma} = w\vec{y} + t\vec{z}$. Define the following notation:

$$g_p^{\vec{x}} := \left(g_p^{x_1},\ g_p^{x_2},\ \ldots,\ g_p^{x_\ell}\right) \qquad \text{where } \vec{x} \in \mathbb{F}_p^\ell.$$

Give the adversary the description of the group, $(N = pqrs,\ \mathbb{G},\ \mathbb{G}_T,\ e)$, generators of each subgroup, $g_p, g_q, g_r, g_s$, and the following tuple:

$$\left(g_p^{\vec{y}},\ \ g_p^{\vec{z}},\ \ g_p^{\vec{\gamma}}\right).$$

The adversary outputs a guess $\mathsf{b}'$ of the bit $\mathsf{b}$. The advantage of $\mathcal{A}$ is defined as $\mathrm{Adv}_\mathcal{A} = \left|\Pr[\mathsf{b}' = \mathsf{b}] - \frac{1}{2}\right|$.

**Definition A.5.** We say that $\mathcal{G}$ satisfies the $\ell$-DLinear assumption if, for all PPT algorithms $\mathcal{A}$, the advantage of $\mathcal{A}$ in winning the above game is negligible in the security parameter $\lambda$.

We can show that the $\ell$-DLinear problem is at least as hard as the Decision Linear problem using a hybrid argument, as follows.

We define the following sequence of games, where $*$ represents a random element from $\mathbb{G}_p$.

| Game | What the challenger gives to the adversary |
|---|---|
| $\mathrm{Game}_\ell$ | $(g_p^{\vec{y}},\ g_p^{\vec{z}},\ g_p^{wy_1+tz_1},\ g_p^{wy_2+tz_2},\ g_p^{wy_3+tz_3},\ \ldots,\ g_p^{wy_{\ell-2}+tz_{\ell-2}},\ g_p^{wy_{\ell-1}+tz_{\ell-1}},\ g_p^{wy_\ell+tz_\ell})$ |
| $\mathrm{Game}_{\ell-1}$ | $(g_p^{\vec{y}},\ g_p^{\vec{z}},\ g_p^{wy_1+tz_1},\ g_p^{wy_2+tz_2},\ g_p^{wy_3+tz_3},\ \ldots,\ g_p^{wy_{\ell-2}+tz_{\ell-2}},\ g_p^{wy_{\ell-1}+tz_{\ell-1}},\ *)$ |
| $\mathrm{Game}_{\ell-2}$ | $(g_p^{\vec{y}},\ g_p^{\vec{z}},\ g_p^{wy_1+tz_1},\ g_p^{wy_2+tz_2},\ g_p^{wy_3+tz_3},\ \ldots,\ g_p^{wy_{\ell-2}+tz_{\ell-2}},\ *,\ *)$ |
| ... | ... |
| $\mathrm{Game}_2$ | $(g_p^{\vec{y}},\ g_p^{\vec{z}},\ g_p^{wy_1+tz_1},\ g_p^{wy_2+tz_2},\ *,\ \ldots,\ *,\ *,\ *)$ |

It is clear that $\mathrm{Game}_\ell$ is equivalent to the $\ell$-DLinear experiment when $\mathsf{b} = 1$; and $\mathrm{Game}_2$ is equivalent to the $\ell$-DLinear experiment when $\mathsf{b} = 0$. Using a hybrid argument, it suffices to prove that no polynomial-time adversary can distinguish between two adjacent games.

We now show that if there exists a PPT adversary $\mathcal{A}$ that can distinguish between two adjacent games $\mathrm{Game}_d$ and $\mathrm{Game}_{d-1}$ with $\epsilon$ advantage, then we can build a simulator $\mathcal{B}$ that uses $\mathcal{A}$ and wins the DLinear experiment also with $\epsilon$ advantage.

Suppose $\mathcal{B}$ is given the DLinear instance $(g_p,\ g_p^a,\ g_p^b,\ g_p^{a\rho},\ g_p^{b\tau},\ Y)$ and tries to decide whether $Y = g_p^{\rho+\tau}$ or $Y \xleftarrow{\text{R}} \mathbb{G}_p$. The simulator chooses random elements $k_2,\ k_3,\ \ldots,\ k_{d-1},\ y_d,\ y_{d+1},\ \ldots,\ y_\ell$, and $w_2,\ w_3,\ \ldots,\ w_{d-1},\ z_d,\ z_{d+1},\ \ldots,\ z_\ell$ from $\mathbb{Z}_N$, and implicitly sets:

$$\vec{y} = (a,\ k_2a,\ k_3a,\ \ldots,\ k_{d-1}a,\ \ y_d,\ y_{d+1},\ \ldots,\ y_\ell)$$

[2]In the unlikely event that $\vec{y}$ and $\vec{z}$ are linearly dependent, $\dim(\mathsf{span}(\vec{y}, \vec{z})) < 2$. However, this happens with negligible probability.

$$\vec{z} = (b,\ w_2 b,\ w_3 b,\ \ldots,\ w_{d-1}b,\ \ z_d,\ z_{d+1},\ \ldots,\ z_\ell)$$

It also implicitly sets

$$w = \rho y_d^{-1}, \quad t = \tau z_d^{-1}$$

where multiplicative inverses are taken modular $N$. (For our purposes, this is equivalent to taking multiplicative inverses modular $p$.) Note that the simulator does not know the values of $a, b, \rho, \tau$. It merely sets the above parameters implicitly, without actually computing them.

The simulator $\mathcal{B}$ computes the following tuple and gives it to $\mathcal{A}$:

$$g_p^{\vec{y}},\ g_p^{\vec{z}},\ \ (g_p^{a\rho})^{y_d^{-1}}(g_p^{b\tau})^{z_d{}^{-1}},\ \ \left\{(g_p^{a\rho})^{k_i y_d^{-1}}(g_p^{b\tau})^{w_i z_d^{-1}}\right\}_{i=2}^{d-1},\ \ Y,\ *,\ *,\ \ldots,\ *$$

Clearly, if $Y = g_p^{\rho+\tau}$, then the above experiment is identically distributed as in Game$_d$. Otherwise, if $Y$ is a random element in $\mathbb{G}_p$, then the above experiment is identically distributed as in Game$_{d-1}$. Hence, if $\mathcal{A}$ can distinguish between Game$_{d-1}$ and Game$_d$ with $\epsilon$ advantage, then $\mathcal{B}$ can win the DLinear experiment with $\epsilon$ advantage as well.

## A.2 Proof Overview

The proof of Theorem 4.1 consists of two main parts.

1. **Plaintext privacy.** First, we show in Section A.3 that our main construction (henceforth referred to as SchemeReal) achieves plaintext privacy. This part of the proof consists of two steps.
   (a) First, we show that SchemeReal is computationally indistinguishable from a variant scheme which we call SchemeQ.
   (b) Second, we show that SchemeQ achieves plaintext privacy. More specifically, SchemeQ bears enough resemblance to the KSW construction that we can reuse the KSW proof of plaintext privacy in a blackbox fashion.
2. **Predicate privacy.** Next, we show in Section A.4 that SchemeReal is computationally indistinguishable from an alternative scheme which we call SchemeSym, in which the tokens and ciphertexts are formed symmetrically. As SchemeSym and SchemeReal are computationally indistinguishable, it suffices to prove plaintext and predicate privacy for SchemeSym. The plaintext privacy of SchemeSym follows from the plaintext privacy of SchemeReal. Since tokens and ciphertexts are symmetrically formed in SchemeSym, the plaintext privacy of SchemeSym implies predicate privacy as well.

## A.3 Plaintext Privacy of SchemeReal

In this section, we prove the following lemma.

**Lemma A.6** (Plaintext Privacy of SchemeReal)**.** *Under the generalized C3DH assumption and Assumption 1,* SchemeReal *has plaintext privacy.*

We know that the KSW construction has plaintext privacy (in the public-key setting). To prove the plaintext privacy of our construction, SchemeReal, we first observe the differences between our construction and KSW.

1. Our construction introduces the $u_{1,i}, u_{2,i}$ terms. As a result, we need an extra component in both the ciphertext and the token: the $C$ and $K$ terms in KSW become $C, C_0$ and $K, K_0$ in our construction.

2. Our construction removes the $\mathbb{G}_q$ elements from the $K$ and $K_0$ components of the token. (To compare, notice the $Q_6 \xleftarrow{\text{R}} \mathbb{G}_q$ element in the $K$ term in KSW.)

The intuition behind the proof of plaintext privacy of SchemeReal is to show that these modifications preserve the plaintext privacy of the KSW construction.

The proof of Lemma A.6 consists of two parts:

1. We construct a hybrid scheme called SchemeQ, in which we add back the random hiding factors from $\mathbb{G}_q$ to the $K$ and $K_0$ terms in the token. We show that SchemeReal and SchemeQ are computationally indistinguishable.

2. We prove the plaintext privacy of SchemeQ. The proof is a reduction showing that if there exists a PPT adversary $\mathcal{A}$ that breaks the plaintext privacy of SchemeQ, then we can build a simulator $\mathcal{B}$ that leverages the adversary $\mathcal{A}$ to break the plaintext privacy of the KSW construction.

We show that SchemeReal is indistinguishable from a scheme SchemeQ using a series of hybrid schemes, defined as follows. For the reader's convenience, we underline the parts where each scheme differs from the previous scheme.

Scheme1: The *Encrypt* and *Query* algorithms are the same as in SchemeReal. The *Setup* algorithm is the same as in SchemeReal except that it chooses a random $\omega \in \mathbb{Z}_N$ and includes it in the secret key. In the *GenToken* algorithm, instead of choosing $f_1, f_2 \in \mathbb{Z}_N$ independently at random, we choose $f \in \mathbb{Z}_N$ at random, and lets $f_1 = f$, and $f_2 = \omega f$. The token is computed as

$$TK_{\vec{v}} = \begin{pmatrix} K = R \cdot \prod_{i=1}^{n} h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}}, \quad K_0 = R_0 \cdot \prod_{i=1}^{n} u_{1,i}^{-r_{1,i}} u_{2,i}^{-r_{2,i}}, \\ \left\{ K_{1,i} = g_p^{r_{1,i}} \underline{g_q^{f v_i}} S_{1,i}, \quad K_{2,i} = g_p^{r_{2,i}} \underline{g_q^{\omega f v_i}} S_{2,i} \right\}_{i=1}^{n} \end{pmatrix}.$$

Scheme2: The *Setup*, *Encrypt*, and *Query* algorithms are the same as in Scheme1. The *GenToken* algorithm chooses a random $Q \in \mathbb{G}_q$, and multiplies $Q$ to $K$. Note that a fresh $Q$ is chosen each time *GenToken* is called. The token is computed as

$$TK_{\vec{v}} = \begin{pmatrix} K = \underline{Q} R \cdot \prod_{i=1}^{n} h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}}, \quad K_0 = R_0 \cdot \prod_{i=1}^{n} u_{1,i}^{-r_{1,i}} u_{2,i}^{-r_{2,i}}, \\ \left\{ K_{1,i} = g_p^{r_{1,i}} g_q^{f v_i} S_{1,i}, \quad K_{2,i} = g_p^{r_{2,i}} g_q^{\omega f v_i} S_{2,i} \right\}_{i=1}^{n} \end{pmatrix}.$$

Scheme3: The *Setup*, *Encrypt*, and *Query* algorithms are the same as in Scheme2. The *GenToken* algorithm chooses a random $Q_0 \in \mathbb{G}_q$ and multiplies $Q_0$ to $K_0$. The rest of the algorithm is the same as in Scheme2. The token is computed as

$$TK_{\vec{v}} = \begin{pmatrix} K = QR \cdot \prod_{i=1}^{n} h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}}, \quad K_0 = \underline{Q_0} R_0 \cdot \prod_{i=1}^{n} u_{1,i}^{-r_{1,i}} u_{2,i}^{-r_{2,i}}, \\ \left\{ K_{1,i} = g_p^{r_{1,i}} g_q^{f v_i} S_{1,i}, \quad K_{2,i} = g_p^{r_{2,i}} g_q^{\omega f v_i} S_{2,i} \right\}_{i=1}^{n} \end{pmatrix}.$$

SchemeQ: The *Setup*, *Encrypt*, and *Query* algorithms are the same as in Scheme3. The *GenToken* algorithm chooses the exponents $f_1, f_2 \in \mathbb{Z}_N$ independently at random as in SchemeReal. The rest of the algorithm is the same as in Scheme3. The token is computed as

$$TK = \left( \begin{array}{c} K = QR \cdot \prod_{i=1}^{n} h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}}, \quad K_0 = Q_0 R_0 \cdot \prod_{i=1}^{n} u_{1,i}^{-r_{1,i}} u_{2,i}^{-r_{2,i}}, \\ \left\{ K_{1,i} = g_p^{r_{1,i}} \underline{g_q^{f_1 v_i}} S_{1,i}, \quad K_{2,i} = g_p^{r_{2,i}} \underline{g_q^{f_2 v_i}} S_{2,i} \right\}_{i=1}^{n} \end{array} \right)$$

### A.3.1 Computational Indistinguishability of SchemeReal and Scheme1

In this section, we prove the following lemma:

**Lemma A.7** (Computational indistinguishability of SchemeReal and Scheme1)**.** *Under the $\ell$-C3DH assumption,* Scheme1 *is computationally indistinguishable from* SchemeReal*.*

*Proof.* Assuming $\ell$-C3DH, we can prove that SchemeReal and Scheme1 are indistinguishable through a simple reduction argument. Suppose there exists an adversary $\mathcal{A}$, making $\ell$ token queries, that can distinguish between SchemeReal and Scheme1, we can use $\mathcal{A}$ to build a simulator $\mathcal{B}$ that can win the above $\ell$-C3DH game. The simulator $\mathcal{B}$ is randomly given one of the following two tuples. In the case $\mathsf{b} = 0$, $\mathcal{B}$ is given

$$(w_1 \widehat{S}_1, \ldots, w_\ell \widehat{S}_\ell,\ W_1 = w_1^{\omega} \widetilde{S}_1, \ldots, W_\ell = w_\ell^{\omega} \widetilde{S}_\ell).$$

In the case $\mathsf{b} = 1$, $\mathcal{B}$ is given

$$(w_1 \widehat{S}_1, \ldots, w_\ell \widehat{S}_\ell,\ W_1 = Q_1 \overline{S}_1, \ldots, W_\ell = Q_\ell \overline{S}_\ell).$$

The goal of the simulator is to determine the bit $\mathsf{b}$.

In the Setup phase of the game, the simulator generates the secret key (without the $\omega$ component) and keeps the secret key to itself. (The simulator can generate the secret key given the generators of the different subgroups.)

In answer to the $j$th token query, the simulator uses the terms $w_j \widehat{S}_j$ and $W_j$ from the $\ell$-C3DH instance to create the following token:

$$TK = \left( \begin{array}{c} K = R \cdot \prod_{i=1}^{n} h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}}, \quad K_0 = R_0 \cdot \prod_{i=1}^{n} u_{1,i}^{-r_{1,i}} u_{2,i}^{-r_{2,i}}, \\ \left\{ K_{1,i} = g_p^{r_{1,i}} \underline{(w_j \widehat{S}_j)^{v_i}} S_{1,i}, \quad K_{2,i} = g_p^{r_{2,i}} \underline{W_j^{v_i}} S_{2,i} \right\}_{i=1}^{n} \end{array} \right)$$

Clearly, if $\mathsf{b} = 0$, the tokens are formed as in Scheme1; if $\mathsf{b} = 1$, the tokens are formed as in SchemeReal.

If $\mathcal{A}$ outputs a guess of Scheme1, the simulator outputs a guess $\mathsf{b}' = 0$; if $\mathcal{A}$ outputs a guess of SchemeReal, the simulator outputs a guess of $\mathsf{b}' = 1$. In this way, if $\mathcal{A}$ has $\epsilon$ advantage in distinguishing SchemeReal and Scheme1, the simulator will have $\epsilon$ in winning the $\ell$-C3DH game. □

**Remark 1.** *When proving the computational indistinguishability of* SchemeReal *and* Scheme1*, we rely on the $\mathbb{G}_s$ subgroup. Therefore, our proof of computationally indistinguishability of* SchemeReal *and* SchemeQ *also relies on the $\mathbb{G}_s$ subgroup. Thus, although we are able to computationally remove the $\mathbb{G}_q$ subgroup from the $K$ and $K_0$ terms in the token in our construction, this does* not *imply that we can do the same thing for the KSW construction, as the KSW construction does not have the $\mathbb{G}_s$ subgroup. That is, our proof does* not *imply that one can safely remove the $\mathbb{G}_q$ subgroup from the $K$ term in the token of the KSW construction.*

### A.3.2 Computational Indistinguishability of Scheme1 and Scheme2

In this section, we prove the following lemma.

**Lemma A.8** (Computational Indistinguishability of SCHEME1 and SCHEME2)**.** *Under Assumption W,* SCHEME2 *is computationally indistinguishable from* SCHEME1*.*

*Proof.* We build a simulator $\mathcal{B}$ that tries to break Assumption W. The simulator uses an adversary $\mathcal{A}$ that tries to distinguish SCHEME1 from SCHEME2. If the adversary $\mathcal{A}$ has advantage $\epsilon$ in distinguishing SCHEME1 from SCHEME2, then the simulator $\mathcal{B}$ has advantage $\epsilon$ in breaking Assumption W.

The simulator $\mathcal{B}$ is given an instance of Assumption W, and it plays the following distinguishing game with the adversary $\mathcal{A}$. The adversary makes queries for ciphertexts and tokens, and in answer to these queries, the simulator computes ciphertexts and tokens following a certain strategy. The resulting ciphertexts and tokens are distributed either according to SCHEME1 or according to SCHEME2. In particular, if the simulator is given $T = g_p^{b^2c}R_3$ from the Assumption W instance, then the encryption scheme used would be identically distributed as SCHEME1; otherwise, if $T = g_p^{b^2c}Q_3R_3$, the encryption scheme used would be identically distributed as SCHEME2.

- **Setup.** The simulator is given an instance of Assumption W, and it uses this to create the following secret key:

  $$SK = \left(\omega,\ \ g_p,\ g_r,\ g_s,\ \ \left\{h_{1,i} = (g_p^{b^2})^{\omega y_i},\ h_{2,i} = g_p^{z_i}(g_p^{b^2})^{-y_i},\ \ u_{1,i} = g_p^{c_i},\ u_{2,i} = g_p^{d_i}\right\}_{i=1}^n\right)$$

  where $\omega, \{z_i, y_i, c_i, d_i\}_{i=1}^n$ are random exponents from $\mathbb{Z}_N$. In the above $SK$, the following elements are inherited from the Assumption W instance: $g_p$, $g_r$, $g_s$ and $g_p^{b^2}$.

  Notice that the simulator does not know $g_q$, which ought to be part of the secret key. We show that the simulator is still able to answer ciphertext queries and token queries from the adversary appropriately, in spite of not knowing $g_q$.

- **Ciphertext query.** In spite of not knowing $g_q$, the simulator is able to compute ciphertexts, as it knows a generator $g_r$ of $\mathbb{G}_r$ and $g_qR_1$ from the Assumption W instance.

- **Token query.** To answer a token query, the simulator picks random values $r, f$ from $\mathbb{Z}_N$, random hiding factors $R, R_0$ from the subgroup $\mathbb{G}_r$, and random $\{S_{1,i}, S_{2,i}\}_{i=1}^n$ from $\mathbb{G}_s$.

  The simulator uses the following strategy to choose the values of $\{r_{1,i}, r_{2,i}\}_{i=1}^n$. First, the simulator picks random exponents $\{\tau_i, \overline{r}_{1,i}, \overline{r}_{2,i}\}_{i=1}^n$ from $\mathbb{Z}_N$. The simulator then implicitly sets the values of $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ to be the following (without actually computing them):

  $$\forall i \in [n]: \quad \begin{aligned} r_{1,i} &= afv_i + \tau_i c + \overline{r}_{1,i} \\ r_{2,i} &= af\omega v_i + \overline{r}_{2,i} \end{aligned} \tag{1}$$

  Using the above implicit values for $\{r_{1,i}, r_{2,i}\}_{i=1}^n$, the simulator computes a token as below:

  $$\forall i \in [n]: \quad \begin{aligned} K_{1,i} &= S_{1,i} \cdot (g_p^a g_q)^{fv_i} (g_p^c)^{\tau_i} g_p^{\overline{r}_{1,i}} \\ K_{2,i} &= S_{2,i} \cdot (g_p^a g_q)^{\omega f v_i} g_p^{\overline{r}_{2,i}} \end{aligned} \tag{2}$$

In addition,

$$K = \quad R \cdot \prod_{i=1}^{n} T^{-\omega y_i \tau_i} \cdot h_{1,i}^{-\overline{r}_{1,i}} \cdot (g_p^{-a} R_1)^{f \omega z_i v_i} \cdot h_{2,i}^{-\overline{r}_{2,i}} \tag{3}$$

$$K_0 = \quad R_0 \cdot \prod_{i=1}^{n} \left( (g_p^{-a} R_1)^{-f v_i} (g_p^c)^{\tau_i} g_p^{\overline{r}_{1,i}} \right)^{-c_i} \left( (g_q^{-a} R_1)^{\omega f v_i} g_p^{\overline{r}_{2,i}} \right)^{-d_i} \tag{4}$$

Notice that the above equations make use of the term $g_p^{-a} R_1$. This can be obtained from the terms $g_q R_1$ and $g_p^a g_q$ inherited from the Assumption W instance:

$$g_p^{-a} R_1 = \frac{g_q R_1}{g_p^a g_q}$$

It can be seen that $\{K_{1,i}, K_{2,i}\}_{i=1}^n$ as defined in Equation (2), and $K_0$ as defined in Equation (4), are correctly formed as in Scheme1 (or Scheme2). Recall that $K_0, \{K_{1,i}, K_{2,i}\}_{i=1}^n$ have the same form in both Scheme1 and Scheme2. We now show that if $\gamma$ from the Assumption W instance is equal to 0, then $K$ as defined in Equation (3) is distributed as in Scheme1. Otherwise, if $\gamma \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_N$, $K$ as defined in Equation (3) is distributed as in Scheme2.

To see that $K$ has the correct distribution, let $K_{0,p}, K_{0,q}, K_{0,r}$ denote the projections of $K$ into the subgroups $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r$ respectively. Clearly, $K_{0,r}$ has the correct distribution.

We now verify that $K_{0,q}$ and $K_{0,p}$ have the correct distribution.

One can see that

$$K_{0,q} = g_q^{-\gamma\kappa} \qquad \text{where } \kappa = \omega \sum_{i=1}^{n} y_i \tau_i$$

Clearly, if $\gamma = 0$ in the Assumption W instance, then $K_{0,q}$ is distributed as in Scheme1, i.e., $K$ does not contain an element from the subgroup $\mathbb{G}_q$. If $\gamma \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_N$, observe that $\kappa$ is distributed uniformly at random in $\mathbb{Z}_N$, and is independent of $\{r_{1,i}, r_{2,i}\}_{i=1}^n$. Therefore, the distribution of $K_{0,q}$ is statistically close to uniform at random. (In fact, it suffices to pick $\tau_1 \stackrel{\text{R}}{\leftarrow} \mathbb{Z}_N$, and fix $\tau_i = 0$ for $i \in [2, n]$.)

It remains to verify that $K_{0,p}$ has the correct distribution. The correct distribution of $K_{0,p}$ is:

$$\begin{aligned} \textstyle\prod_{i=1}^n h_{1,i}^{-r_{1,i}} h_{2,i}^{-r_{2,i}} &= \textstyle\prod_{i=1}^n (g_p^{-b^2 \omega y_i})^{a f v_i + \tau_i c + \overline{r}_{1,i}} \cdot (g_p^{-z_i} g_p^{b^2 y_i})^{a f \omega v_i + \overline{r}_{2,i}} \\ &= \textstyle\prod_{i=1}^n g_p^{-b^2 c \omega y_i \tau_i} h_{1,i}^{-\overline{r}_{1,i}} g_p^{-a f \omega z_i v_i} h_{2,i}^{-\overline{r}_{2,i}} \end{aligned} \tag{5}$$

One can see that the $K$ defined in Equation (3) has the same $\mathbb{G}_p$ component as the above Equation (5). A crucial observation here is that all terms involving $g_p^{ab^2}$ (which is unknown to the simulator) cancel out. This is the reason why the simulator can generate the token efficiently.

- **Guess.** The simulator $\mathcal{B}$ outputs the same guess $\mathsf{b}'$ output by the adversary $\mathcal{A}$.

Clearly, if the adversary $\mathcal{A}$ has advantage $\epsilon$ in distinguishing Scheme1 and Scheme2, then the simulator $\mathcal{B}$ also has advantage $\epsilon$ in breaking Assumption W. This completes the proof of Lemma A.8. $\square$

### A.3.3 Computational Indistinguishability of Scheme2, Scheme3, SchemeQ

In this section, we prove the indistinguishability of SCHEME2, SCHEME3, and SCHEMEQ.

**Lemma A.9** (Computational indistinguishability of SCHEME2 and SCHEME3)**.** *Under Assumption 1 of KSW [21],* SCHEME3 *is computationally indistinguishable from* SCHEME2*.*

*Proof.* The proof of this claim is very similar to that of Claim A.8. The only difference is that in this proof, the simulator needs to re-randomize the term $K$ with a random element from the subgroup $\mathbb{G}_{qr} = \mathbb{G}_q \times \mathbb{G}_r$. The simulator can do this because it knows the terms $g_q R_1$ and $g_r$. (In comparison, in the proof of Lemma A.8, the simulator re-randomizes the term $K_0$ with an element from $\mathbb{G}_r$). □

Finally, we show that SCHEME3 and SCHEMEQ are computationally indistinguishable, completing the sequence of hybrid schemes, proving that SCHEMEREAL and SCHEMEQ are computationally indistinguishable.

**Lemma A.10.** *Under the generalized C3DH assumption,* SCHEME3 *and* SCHEMEQ *are computationally indistinguishable.*

*Proof.* Similar to that of Claim A.7. □

### A.3.4 Plaintext Privacy of SchemeQ

We have shown that SCHEMEREAL is computationally indistinguishable from SCHEMEQ. We now show that SCHEMEQ has plaintext privacy. This implies that SCHEMEREAL has plaintext privacy as well.

The original KSW construction runs in a bilinear group of order $N = pqr$. This part of the proof relies on the observation that if we run the KSW construction in the subgroup $\mathbb{G}_{pqr} = \mathbb{G}_p \times \mathbb{G}_q \times \mathbb{G}_r$ residing in a larger bilinear group of order $N = pqrs$, the KSW construction still has plaintext privacy. Fundamentally, this relies on the fact that Assumption 1 still holds when the bilinear group $\mathbb{G}_{pqr}$ in question resides in the context of a larger group $\mathbb{G} = \mathbb{G}_{pqrs}$.

**Lemma A.11.** *If Assumption 1 holds in the bilinear group $\mathbb{G}$, then* SCHEMEQ *has plaintext privacy.*

*Proof.* The proof is based on the plaintext privacy of the KSW construction. We show that if there exists a polynomial-time adversary $\mathcal{A}$ that can break the plaintext privacy of SCHEMEQ, we can build a polynomial-time simulator $\mathcal{B}$ that leverages $\mathcal{A}$ to break the plaintext privacy of the KSW construction. Recall that the KSW construction uses a bilinear group of order $N = pqr$. We assume that this group resides in a larger group of size $N = pqrs$, and that Assumption 1 still holds in the context of this larger group.

The simulator $\mathcal{B}$ acts as the challenger of the SCHEMEQ adversary $\mathcal{A}$ and tries to break the plaintext privacy of KSW, interacting with its own challenger $\mathcal{C}$. The simulator $\mathcal{B}$ uses the following strategy to interact with $\mathcal{A}$: whenever $\mathcal{A}$ submits a ciphertext or token query, the simulator $\mathcal{B}$ simply forwards it along to the challenger $\mathcal{C}$. In return, $\mathcal{B}$ obtains a KSW ciphertext or token. Now $\mathcal{B}$ augments the KSW ciphertext or token before handing the answer over to the adversary $\mathcal{A}$. For example, part of the augmentation performed by $\mathcal{B}$ is to fill in the terms $C_0$ and $K_0$.

- **Init.** The SCHEMEQ adversary $\mathcal{A}$ commits to a ciphertext challenge $(\vec{x}_0, \vec{x}_1)$ to the simulator $\mathcal{B}$. $\mathcal{B}$ forwards the same challenge $(\vec{x}_0, \vec{x}_1)$ to $\mathcal{C}$.

- **Setup.** $\mathcal{C}$ runs the *Setup* algorithm of KSW, and gives the following public key to the simulator $\mathcal{B}$.
$$PK = (g_p,\ g_r,\ g_s,\quad Q = g_q \cdot R,\quad \{H_{1,i}, H_{2,i}\}_{i=1}^n)$$
In addition, $\mathcal{B}$ generates the following secrets:
$$\{u_{1,i} = g_p^{y_i},\ u_{1,i} = g_p^{z_i}\}_{i=1}^n$$
where $\{y_i, z_i\}_{i=1}^n$ are random in $\mathbb{Z}_N$.

- **Ciphertext queries.** Whenever the adversary $\mathcal{A}$ submits a ciphertext query for the vector $\vec{x} \in (\mathbb{Z}_N)^n$, $\mathcal{B}$ computes the following ciphertext and returns it to the adversary. Pick random exponents $y, z, \alpha, \beta$ from $\mathbb{Z}_N$, random hiding factors $S, S_0$ from $\mathbb{G}_s$, and random $\{R_{1,i}, R_{2,i}\}_{i=1}^n$ from $\mathbb{G}_r$.
$$CT = \left( \begin{array}{c} C = S \cdot g_p^y, \quad C_0 = S_0 \cdot g_p^z, \\ \left\{ C_{1,i} = H_{1,i}^y u_{1,i}^z Q^{\alpha x_i} R_{1,i}, \quad C_{2,i} = H_{2,i}^y u_{2,i}^z Q^{\beta x_i} R_{2,i} \right\}_{i=1}^n \end{array} \right)$$

- **Token queries.** Whenever the adversary $\mathcal{A}$ makes a token query for a vector $\vec{v} \in (\mathbb{Z}_N)^n$, the simulator asks $\mathcal{C}$ to generate a KSW token for the same vector $\vec{v}$. Suppose the KSW token for $\vec{v}$ is formed as below:
$$\mathsf{KSW}.TK = (k_0,\ \{k_{1,i}, k_{2,i}\}_{i=1}^n)$$
The simulator now transforms the KSW token into a SchemeQ token as below. The simulator picks a random exponent $r \xleftarrow{R} \mathbb{Z}_N$; a random hiding factor $R_0$ from the subgroup $\mathbb{G}_r$; and random $\{S_{1,i}, S_{2,i}\}_{i=1}^n$ from $\mathbb{G}_s$.
$$TK = \left( \begin{array}{c} K = k_0, \quad K_0 = R_0 Q^r \prod_{i=1}^n k_{1,i}^{-y_i} k_{2,i}^{-z_i}, \\ \{K_{1,i} = k_{1,i} S_{1,i}, \quad K_{2,i} = k_{2,i} S_{2,i}\}_{i=1}^n \end{array} \right)$$

- **Challenge.** When $\mathcal{A}$ requests a challenge, $\mathcal{B}$ requests a challenge from $\mathcal{C}$. Suppose $\mathcal{B}$ obtains the following challenge ciphertext from $\mathcal{C}$:
$$\mathsf{KSW}.CT = (c_0,\quad \{c_{1,i}, c_{2,i}\}_{i=1}^n)$$
The simulator transforms the above KSW ciphertext to a SchemeQ ciphertext. It picks $t \xleftarrow{R} \mathbb{Z}_N$, $S, S_0 \xleftarrow{R} \mathbb{G}_s$, and computes:
$$CT = \left( \begin{array}{c} C = S \cdot c_0, \quad C_0 = S_0 \cdot g_p^t, \\ \left\{ C_{1,i} = c_{1,i} u_{1,i}^t, \quad C_{2,i} = c_{2,i} u_{2,i}^t \right\}_{i=1}^n \end{array} \right)$$

- **More ciphertext and token queries.** Same as above.

- **Guess.** The simulator $\mathcal{B}$ outputs the same guess as the adversary $\mathcal{A}$.

It is straightforward to verify that in the above simulation, the ciphertexts and tokens computed by $\mathcal{B}$ has the correct distribution. Clearly, if $\mathcal{A}$ has $\epsilon$ advantage in breaking SchemeQ, then the simulator $\mathcal{B}$ has $\epsilon$ advantage in breaking KSW. This completes the proof of Lemma A.11. □

### A.4 Indistinguishability of SchemeReal and SchemeSym

We now show that SchemeReal is computationally indistinguishable from a scheme called SchemeSym, where the tokens and the ciphertexts are symmetrically formed. The proof is carried out in the following two steps:

1. We first define SchemeSym, and show that SchemeReal is computationally indistinguishable from SchemeSym.

2. Next, we show that in SchemeSym, the tokens and ciphertexts are symmetrically formed.

#### A.4.1 SchemeSym

We modify SchemeReal to obtain an alternative scheme called SchemeSym. Specifically, we modify the way the *GenToken* algorithm picks the exponents $\{r_{1,i}, r_{2,i}\}_{i=1}^n$. In SchemeSym, the exponents $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ are no longer picked completely at random from $\mathbb{Z}_p$. Instead, these exponents are now picked at random from a two-dimensional subspace of the vector space $\mathbb{F}_p^{2n}$. The scheme SchemeSym is defined as follows.

*Setup*$(1^\lambda)$: The setup algorithm first chooses a secret key as in SchemeReal. Additionally, it chooses the following random exponents from $\mathbb{Z}_p$, and keeps them secret.

$$\{y_{1,i},\ z_{1,i},\ y_{2,i},\ z_{2,i}\}_{i=1}^n$$

*Encrypt*$(SK, \vec{x})$: Same as in SchemeReal.

*GenToken*$(SK, \vec{v})$: Instead of picking $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ independently at random from $\mathbb{Z}_p$, the *GenToken* picks random $\rho, \tau \xleftarrow{\text{R}} \mathbb{Z}_p$, and sets the values of $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ as below:

$$\begin{aligned} \forall i \in [n]: \qquad & r_{1,i} = \rho y_{1,i} + \tau z_{1,i} \\ & r_{2,i} = \rho y_{2,i} + \tau z_{2,i} \end{aligned}$$

The rest of the *GenToken* proceeds as in SchemeReal.

*Query*$(TK_{\vec{v}}, CT)$: Same as in SchemeReal.

One way to understand the above construction SchemeSym is as follows. Let $\vec{y} = \{y_{1,i}, y_{2,i}\}_{i=1}^n$, let $\vec{z} = \{z_{1,i}, z_{2,i}\}_{i=1}^n$, let $\vec{r} = \{r_{1,i}, r_{2,i}\}_{i=1}^n$. It is not hard to see that $\vec{r}$ is chosen at random from a 2-dimensional subspace generated by $\vec{y}$ and $\vec{z}$. Essentially, SchemeSym always chooses a 2-dimensional subspace during the setup phase. Later, when constructing tokens, SchemeSym always picks the exponents $\vec{r}$ at random from this prescribed 2-dimensional subspace. Due to the Decisional Linear assumption, picking the exponents from a 2-dimensional subspace is computationally indistinguishable from picking the exponents completely at random from the entire vector space $\mathbb{F}_p^{2n}$. We state this intuition in the following lemma.

So far, it may not be entirely clear why the ciphertexts and tokens are symmetrically formed in SchemeSym. We explain why this is the case in Appendix A.4.2.

**Lemma A.12.** *Assuming that the $\ell$-DLinear assumption holds in $\mathbb{G}_p$,* SchemeSym *is computationally indistinguishable from* SchemeReal.

*Proof.* Let $\ell = 2n$. We show that distinguishing between SchemeReal and SchemeSym is at least as hard as the $\ell$-DLinear problem as stated in Observation A.5. Our proof relies on a hybrid argument on the number of token queries made by the adversary. Let $k$ denote the number of token queries made by the adversary. We define a sequence of games, $\mathrm{Game}_0, \mathrm{Game}_1, \ldots, \mathrm{Game}_k$. In $\mathrm{Game}_d$ ($0 \le d \le k$), for the first $d$ tokens queried, the challenger picks the exponents $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ from a pre-determined 2-dimensional subspace. For the remaining token queries $d+1, \ldots, k$, the challenger picks completely random exponents $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ from $\mathbb{F}_p^{2n}$.

More specifically, $\mathrm{Game}_d$ ($0 \le d \le k$) is formally defined as below.

- **Setup.** The challenger picks two random vectors

$$\begin{aligned} \vec{y} = \{y_{1,i},\ y_{2,i}\}_{i=1}^n &\xleftarrow{\mathrm{R}} \mathbb{F}_p^{2n} \\ \vec{z} = \{z_{1,i},\ z_{2,i}\}_{i=1}^n &\xleftarrow{\mathrm{R}} \mathbb{F}_p^{2n} \end{aligned}$$

  and keeps them secret. These two vectors determine a 2-dimensional subspace $\mathsf{span}(\vec{y}, \vec{z})$. Later, when the challenger answers the first $d$ token queries made by the adversary, it will pick the exponents $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ at random from this subspace. The challenger now calls the *Setup* algorithm to generate a secret key as in SchemeReal.

- **Ciphertext queries.** The challenger answers all ciphertext queries by directly calling the *Encrypt* algorithm.

- **Token queries.** For the first $d$ token queries, the challenger picks exponents $\{r_{1,i}, r_{2,i}\}_{i=1}^n$ as below. Pick two random numbers $\rho, \tau \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, and sets the values of $\vec{r} := \{r_{1,i}, r_{2,i}\}_{i=1}^n$ to be the following:

$$\forall i \in [n]: \qquad \begin{aligned} r_{1,i} &= \rho y_{1,i} + \tau z_{1,i} \\ r_{2,i} &= \rho y_{2,i} + \tau z_{2,i} \end{aligned}$$

  Expressed in the vector form,

$$\vec{r} = \rho\vec{y} + \tau\vec{z}$$

  In other words, $\vec{r}$ is picked at random from the 2-dimensional subspace $\mathsf{span}(\vec{y}, \vec{z})$.

  For the remaining token queries $d+1, \ldots, k$, the challenger generates tokens normally by calling the *GenToken* algorithm.

It is not hard to see that $\mathrm{Game}_0$ is identically distributed as SchemeReal and $\mathrm{Game}_k$ is identically distributed as SchemeSym. Using a hybrid argument, it suffices to show that no PPT adversary is able to distinguish between two adjacent games $\mathrm{Game}_{d-1}$ and $\mathrm{Game}_d$ ($1 \le d \le k$) with more than negligible advantage.

We now show that if there exists a PPT adversary $\mathcal{A}$ that can distinguish between $\mathrm{Game}_{d-1}$ and $\mathrm{Game}_d$ ($1 \le d \le k$) with $\epsilon$ advantage, we can build a polynomial-time simulator $\mathcal{B}$ that uses $\mathcal{A}$ to break the $\ell$-DLinear assumption also with $\epsilon$ advantage. Suppose $\mathcal{B}$ is given the $\ell$-DLinear instance $(g_p^{\vec{y}}, g_p^{\vec{z}}, g_p^{\vec{\gamma}})$, where $\vec{y} = \{y_{1,i}, y_{2,i}\}_{i=1}^n$, $\vec{z} = \{z_{1,i}, z_{2,i}\}_{i=1}^n$, and $\vec{\gamma} = \{\gamma_{1,i}, \gamma_{2,i}\}_{i=1}^n$. Now the simulator tries to decide whether $\vec{\gamma} \in \mathsf{span}(\vec{y}, \vec{z})$, or whether $\vec{\gamma}$ is a random vector in $\mathbb{F}_p^{2n}$. To do this, the simulator will set the exponents $\vec{r} = \{r_{1,i}, r_{2,i}\}_{i=1}^n$ in the first $d-1$ tokens to be random vectors in $\mathsf{span}(\vec{y}, \vec{z})$. The simulator sets the exponents $\vec{r}$ in the $d^{\text{th}}$ token to be the vector $\vec{\gamma}$. For the remaining token queries, the simulator chooses random exponents $\vec{r} \xleftarrow{\mathrm{R}} \mathbb{F}_p^{2n}$. In this way, if

$\vec{\gamma} \xleftarrow{R} \mathsf{span}(\vec{y}, \vec{z})$, the simulation is equivalent to Game$_d$; otherwise, if $\vec{\gamma} \xleftarrow{R} \mathbb{F}_p^{2n}$, the simulation is equivalent to Game$_{d-1}$.

- **Setup.** The simulator picks the following secret key:

$$SK = (g_p,\ g_q,\ g_r,\ g_s,\ \ \{h_{1,i} = g_p^{\omega_{1,i}},\ h_{2,i} = g_p^{\omega_{2,i}},\ u_{1,i} = g_p^{\kappa_{1,i}},\ u_{2,i} = g_p^{\kappa_{2,i}}\})$$

  where $\omega_{1,i}, \omega_{2,i}, \kappa_{1,i}, \kappa_{2,i}$ are random exponents in $\mathbb{Z}_p$.

- **Ciphertext queries.** The simulator answers all ciphertext queries by directly calling the *Encrypt* algorithm.

- **Token queries.** For all token queries, the simulator picks random exponents $f_1, f_2$ from $\mathbb{Z}_N$; random hiding factors $R, R_0$ from the subgroup $\mathbb{G}_r$; and random $\{S_{1,i}, S_{2,i}\}_{i=1}^n$ from $\mathbb{G}_s$. It chooses the values of $\vec{r} = \{r_{1,i}, r_{2,i}\}_{i=1}^n$ as follows:

  – For the first $d-1$ token queries, the challenger picks random $\rho, \tau$ from $\mathbb{Z}_p$, ($\rho, \tau$ are picked as fresh random numbers for each of the first $d-1$ token queries.) and implicitly lets $\vec{r} = \{r_{1,i}, r_{2,i}\}_{i=1}^n$ to be the following (without actually computing it):

$$\vec{r} = \rho\vec{y} + \tau\vec{z}$$

  In the above expression, $\vec{y}$ and $\vec{z}$ are inherited from the $\ell$-DLinear instance. Note that the simulator does not know the values of $\vec{y}$ and $\vec{z}$, it implicitly sets the vector $\vec{r}$ without computing its value. Now the simulator computes the following token:

$$TK = \begin{pmatrix} K = R \cdot \prod_{i=1}^n (g_p^{y_{1,i}})^{-\rho\omega_{1,i}} (g_p^{z_{1,i}})^{-\tau\omega_{1,i}} (g_p^{y_{2,i}})^{-\rho\omega_{2,i}} (g_p^{z_{2,i}})^{-\tau\omega_{2,i}}, \\ K_0 = R_0 \cdot \prod_{i=1}^n (g_p^{y_{1,i}})^{-\rho\kappa_{1,i}} (g_p^{z_{1,i}})^{-\tau\kappa_{1,i}} (g_p^{y_{2,i}})^{-\rho\kappa_{2,i}} (g_p^{z_{2,i}})^{-\tau\kappa_{2,i}}, \\ \Big\{K_{1,i} = (g_p^{y_{1,i}})^{\rho} (g_p^{z_{1,i}})^{\tau} g_q^{f_1 v_i} S_{1,i}, \quad K_{2,i} = (g_p^{y_{2,i}})^{\rho} (g_p^{z_{2,i}})^{\tau} g_q^{f_2 v_i} S_{2,i}\Big\}_{i=1}^n \end{pmatrix}$$

  – For the $d^{\text{th}}$ token query, the simulator will implicitly set the exponents $\vec{r} = \vec{\gamma}$, where $\vec{\gamma}$ is adopted from the $\ell$-DLinear instance. The simulator computes the following token:

$$TK = \begin{pmatrix} K = R \cdot \prod_{i=1}^n (g_p^{\gamma_{1,i}})^{-\omega_{1,i}} (g_p^{\gamma_{2,i}})^{-\omega_{2,i}}, \quad K_0 = R_0 \cdot \prod_{i=1}^n (g_p^{\gamma_{1,i}})^{-\kappa_{1,i}} (g_p^{\gamma_{2,i}})^{-\kappa_{2,i}}, \\ \Big\{K_{1,i} = g_p^{\gamma_{1,i}} g_q^{f_1 v_i} S_{1,i}, \quad K_{2,i} = g_p^{\gamma_{2,i}} g_q^{f_2 v_i} S_{2,i}\Big\}_{i=1}^n \end{pmatrix}$$

  – For the remaining token queries $d+1, \ldots, k$, the simulator generates tokens by directly calling the *GenToken* algorithm. In this case, $\vec{r}$ is chosen as a random vector in $\mathbb{F}_p^{2n}$.

- **Guess.** If the adversary guesses that it is playing Game$_d$, the simulator guesses that $\vec{\gamma} \xleftarrow{R} \mathsf{span}(\vec{y}, \vec{z})$. Otherwise, if the adversary guesses that it is playing Game$_{d-1}$, the simulator guesses that $\vec{\gamma} \xleftarrow{R} \mathbb{F}_p^{2n}$.

Clearly, if the adversary has $\epsilon$ advantage in distinguishing Game$_d$ and Game$_{d-1}$ ($1 \le d \le k$), the simulator also has $\epsilon$ advantage in the $\ell$-DLinear experiment. □

### A.4.2 Symmetry of Token and Ciphertext in SchemeSym

So far, it may not be clear why the tokens and ciphertexts are symmetrically formed in SchemeSym. To show that this indeed is the case, we describe a scheme called SchemeSymII and show that the statistical distance between the distribution of tokens and ciphertexts in SchemeSym and the distribution of tokens and ciphertexts in SchemeII is negligible. From the description of SchemeII, it is clear that tokens and ciphertexts are symmetrically formed.

Before we formally define SchemeSymII, we first give some intuition. Notice that in SchemeSym, both the ciphertext and token have $2n+2$ terms. Clearly, in SchemeSym, tokens and ciphertexts are symmetric in the $\mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_s$ subgroups. In particular, the $\mathbb{G}_q$ subgroup has the same form in both the ciphertext and the token, and the $\mathbb{G}_r$ and $\mathbb{G}_s$ subgroups "mirror" each other.

However, it may not entirely obvious that the $\mathbb{G}_p$ subgroup is symmetric as well; this is what we are about to show. Let us now focus on the elements in the $\mathbb{G}_p$ subgroup in the ciphertext and token. We represent elements in the $\mathbb{G}_p$ subgroup in the canonical form $g_p^x$, where $g_p$ is a generator of $\mathbb{G}_p$, and $x \in \mathbb{Z}_p$. In both the ciphertext and the token, the exponents in the $\mathbb{G}_p$ subgroup (base $g_p$) form a vector in $\mathbb{F}_p^{2n+2}$. We now show that the statistical distance between the distribution of these exponents and the following distribution is negligible:

- Pick two random 2-dimensional subspaces $Z_1, Z_2 \subset \mathbb{F}_p^{2n+2}$ that are orthogonal to each other, i.e., $Z_1 \perp Z_2$. The fact that $Z_1 \perp Z_2$ ensures that the $\mathbb{G}_p$ subgroup cancels out in the *Query* algorithm.

- For every ciphertext generated, pick a random vector $\vec{\mu} \xleftarrow{R} Z_1$ to be the exponents in the $\mathbb{G}_p$ subgroup (base $g_p$).

- For every token generated, pick a random vector in $\vec{\nu} \xleftarrow{R} Z_2$ to be the exponents in the $\mathbb{G}_p$ subgroup (base $g_p$).

**Definition A.13** (SchemeSymII)**.** We define the following encryption scheme called SchemeSymII. From the description of SchemeSymII, it is clear that tokens and ciphertexts are symmetrically formed.

*Setup*$(1^\lambda)$: The setup algorithm runs $\mathcal{G}(1^\lambda)$ to obtain $(p, q, r, s, \mathbb{G}, \mathbb{G}_T, e)$ and chooses generators $g_p,\ g_q,\ g_r,\ g_s$ from subgroups $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r, \mathbb{G}_s$ respectively. The setup algorithm also chooses two orthogonal subspaces from $\mathbb{F}_p^{2n+2}$. To do so, the setup algorithm picks the following random exponents from $\mathbb{Z}_p$:

$$\begin{gathered}\vec{\mu}_1 = \left(c,\ c_0,\ \{c_{1,i},\ c_{2,i}\}_{i=1}^n\right), \quad \vec{\mu}_2 = \left(d,\ d_0,\ \{d_{1,i},\ d_{2,i}\}_{i=1}^n\right), \\ \vec{\nu}_1 = \left(y,\ y_0,\ \{y_{1,i},\ y_{2,i}\}_{i=1}^n\right), \quad \vec{\nu}_2 = \left(z,\ z_0,\ \{z_{1,i},\ z_{2,i}\}_{i=1}^n\right), \\ \text{s.t.} \qquad \forall (i,j) \in [2] \times [2], \quad \langle \vec{\mu}_i, \vec{\nu}_j \rangle = 0\end{gathered}$$

In the above, the notation $\langle \vec{\mu}, \vec{\nu} \rangle$ denotes inner product. For example,

$$\langle \vec{\mu}_1, \vec{\nu}_1 \rangle := cy + c_0 y_0 + \sum_{i=1}^{n} (c_{1,i} y_{1,i} + c_{2,i} y_{2,i})$$

All of the above parameters are kept as the secret key. Intuitively, by picking $\vec{\mu}_1, \vec{\mu}_2$ and $\vec{\nu}_1, \vec{\nu}_2$, we are picking two random 2-dimensional subspaces in $\mathbb{F}_p^{2n+2}$ that are orthogonal to each other:

$$\mathsf{span}(\vec{\mu}_1, \vec{\mu}_2) \perp \mathsf{span}(\vec{\nu}_1, \vec{\nu}_2)$$

In the unlikely event that $\vec{\mu}_1$ and $\vec{\mu}_2$ (or $\vec{\nu}_1$ and $\vec{\nu}_2$) are linearly dependent, the dimension of $\mathsf{span}(\vec{\mu}_1, \vec{\mu}_2)$ (or $\mathsf{span}(\vec{\nu}_1, \vec{\nu}_2)$) may be smaller than 2. However, this happens only with negligible probability.

*Encrypt*$(SK, \vec{x})$: Let $\vec{x} = (x_1, x_2, \ldots, x_n) \in (\mathbb{Z}_N)^n$. The encryption algorithm first picks random exponents $w, t, \alpha, \beta$ from $\mathbb{Z}_p$. Then, it chooses random hiding factors $S, S_0$ from the subgroup $\mathbb{G}_s$; and random $\{R_{1,i}, R_{2,i}\}_{i=1}^n$ from $\mathbb{G}_r$. The encryption algorithm computes the following ciphertext:

$$CT = \left( \begin{array}{l} C = S \cdot g_p^{wc+td}, \quad C_0 = S_0 \cdot g_p^{wc_0+td_0}, \\ \left\{ C_{1,i} = g_p^{wc_{1,i}+td_{1,i}} g_q^{\alpha x_i} R_{1,i}, \quad C_{2,i} = g_p^{wc_{2,i}+td_{2,i}} g_q^{\beta x_i} R_{2,i} \right\}_{i=1}^n \end{array} \right)$$

**Remark 2.** *In the above ciphertext, the exponents in the $\mathbb{G}_p$ subgroup form the following vector:*

$$\vec{\mu} = \left( wc + td,\ wc_0 + td_0, \quad \{wc_{1,i} + td_{1,i},\ wc_{2,i} + td_{2,i}\}_{i=1}^n \right) = w\vec{\mu}_1 + t\vec{\mu}_2$$

*One can see that $\vec{\mu}$ is chosen as a random vector in the 2-dimensional subspace defined by $\mathsf{span}(\vec{\mu}_1, \vec{\mu}_2)$.*

*GenToken*$(SK, \vec{v})$: Let $\vec{v} = (v_1, v_2, \ldots, v_n) \in (\mathbb{Z}_N)^n$. The *GenToken* algorithm behaves symmetrically to the *Encrypt* algorithm. It first picks random exponents $\rho, \tau, f_1, f_2$ from $\mathbb{Z}_p$. Then, it chooses random hiding factors $R, R_0$ from the subgroup $\mathbb{G}_r$; and random $\{S_{1,i}, S_{2,i}\}_{i=1}^n$ from $\mathbb{G}_s$. The token is formed as below:

$$TK = \left( \begin{array}{l} K = R \cdot g_p^{\rho y+\tau z}, \quad K_0 = R_0 \cdot g_p^{\rho y_0+\tau z_0} \\ \left\{ K_{1,i} = g_p^{\rho y_{1,i}+\tau z_{1,i}} g_q^{f_1 v_i} S_{1,i}, \quad K_{2,i} = g_p^{\rho y_{2,i}+\tau z_{2,i}} g_q^{f_2 v_i} S_{2,i} \right\}_{i=1}^n \end{array} \right)$$

**Remark 3.** *In the above token, the exponents in the $\mathbb{G}_p$ subgroup form the following vector:*

$$\vec{\nu} = \left( \rho y + \tau z,\ \rho y_0 + \tau z_0, \quad \{\rho y_{1,i} + \tau z_{1,i},\ \rho y_{2,i} + \tau z_{2,i}\}_{i=1}^n \right) = \rho\vec{\nu}_1 + \tau\vec{\nu}_2$$

*One can see that $\vec{\nu}$ is chosen as a random vector in the 2-dimensional subspace defined by $\mathsf{span}(\vec{\nu}_1, \vec{\nu}_2)$.*

*Query*$(TK_{\vec{v}}, CT)$: Same as the *Query* algorithm of SchemeReal. Note that as the two subspaces $\mathsf{span}(\vec{\mu}_1, \vec{\mu}_2)$ and $\mathsf{span}(\vec{\nu}_1, \vec{\nu}_2)$ are orthogonal to each other, $\langle \vec{\mu}, \vec{\nu} \rangle = 0$. Hence, in the *Query* algorithm, elements in the $\mathbb{G}_p$ subgroup cancel out, resulting in $1 \in \mathbb{G}_p$.

**Lemma A.14.** *The statistical distance between the distribution of tokens and ciphertexts in* SchemeSymII *and in* SchemeSym *is negligible.*

*Proof.* Let us now focus on SCHEMESYM. We first show that in the ciphertext, exponents in the $\mathbb{G}_p$ subgroup are chosen as a random vector in a pre-determined 2-dimensional subspace (also chosen at random) in $\mathbb{F}_p^{2n+2}$.

For $1 \leq i \leq n$, let $\omega_{1,i}, \omega_{2,i}$ denote the discrete log of $h_{1,i}, h_{2,i}$ (base $g_p$); let $\kappa_{1,i}, \kappa_{2,i}$ denote the discrete log of $u_{1,i}, u_{2,i}$ (base $g_p$). $\{\omega_{1,i}, \omega_{2,i}\}_{i=1}^n$ and $\{\kappa_{1,i}, \kappa_{2,i}\}_{i=1}^n$ are chosen independently at random from $\mathbb{Z}_p$ in the *Setup* algorithm.

In the *Encrypt* algorithm of SCHEMESYM, we pick two random exponents $w, t \xleftarrow{\text{R}} \mathbb{Z}_p$, and in the ciphertext, the exponents in the $\mathbb{G}_p$ subgroup (base $g_p$) have the following form:

$$\vec{\mu} := \big(w,\ t,\ \ \{w\omega_{1,i} + t\kappa_{1,i},\ w\omega_{2,i} + t\kappa_{2,i}\}_{i=1}^n\big) \tag{6}$$

Define the following two vectors:

$$\begin{aligned} \vec{\mu}_1 &:= (1, 0,\ \{\omega_{1,i}, \omega_{2,i}\}_{i=1}^n) \in \mathbb{F}_p^{2n+2} \\ \vec{\mu}_2 &:= (0, 1,\ \{\kappa_{1,i}, \kappa_{2,i}\}_{i=1}^n) \in \mathbb{F}_p^{2n+2} \end{aligned} \tag{7}$$

Equation (6) can be expressed in the following form:

$$\vec{\mu} = w\vec{\mu}_1 + t\vec{\mu}_2$$

Therefore, an equivalent way to think of SCHEMESYM is as follows. In the *Setup* algorithm, we pick two vectors $\vec{\mu}_1$ and $\vec{\mu}_2$ as in Equation (7), and $\mathsf{span}(\vec{\mu}_1, \vec{\mu}_2)$ defines a random 2-dimensional subspace in $\mathbb{F}_p^{2n+2}$ (except with negligible probability). Later, when computing ciphertexts, we always pick the exponents in the $\mathbb{G}_p$ subgroup as a random vector in $\mathsf{span}(\vec{\mu}_1, \vec{\mu}_2)$.

We now examine the tokens in SCHEMESYM. It remains to show that in the tokens, exponents in the $\mathbb{G}_p$ subgroup are chosen as random vectors from a random 2-dimensional subspace orthogonal to $\mathsf{span}(\vec{\mu}_1, \vec{\mu}_2)$. We can see that in the tokens of SCHEMESYM, the exponents of the $\mathbb{G}_p$ subgroup are picked from a subspace orthogonal to $\mathsf{span}(\vec{\mu}_1, \vec{\mu}_2)$, since in the *Query* algorithm, the $\mathbb{G}_p$ subgroup always cancels out, resulting in $1 \in \mathbb{G}_p$. Now, we just need to show that the exponents in the token form a 2-dimensional subspace (as opposed to 1 dimension or other number of dimensions.) To understand why this is the case, we now present alternative way to understand the formation of tokens in SCHEMESYM. In the *Setup* phase, pick the vectors $\vec{y} = (y, y_0,\ \{y_{1,i}, y_{2,i}\}_{i=1}^n)$ and $\vec{z} = (z, z_0,\ \{z_{1,i}, z_{2,i}\}_{i=1}^n)$ as below:

1. Pick $2n$ out of the $2n+2$ coordinates at random, that is, pick $\{y_{1,i}, y_{2,i}\}_{i=1}^n$ at random from $\mathbb{Z}_p$.

2. Given the constraints that $\langle \vec{y}, \vec{\mu}_1 \rangle = 0$, and $\langle \vec{y}, \vec{\mu}_2 \rangle = 0$, the first two coordinates $y, y_0$ can be solved through a system of linear equations. We have two linear equations with two indeterminants. The coefficients of the linear equations are linearly independent except with negligible probability. This means that except with negligible probability, $y, y_0$ can be uniquely solved.

3. Pick $\vec{z}$ in exactly the same way as we did for $\vec{y}$.

By picking the vectors $\vec{y} = (y, y_0,\ \{y_{1,i}, y_{2,i}\}_{i=1}^n)$ and $\vec{z} = (z, z_0,\ \{z_{1,i}, z_{2,i}\}_{i=1}^n)$ in the manner specified above, we are equivalently picking a random subspace that is orthogonal to the subspace $\mathsf{span}(\vec{\mu}_1, \vec{\mu}_2)$.

Later, when computing tokens, the *GenToken* algorithm picks the exponents in the $\mathbb{G}_p$ subgroup as a random vector from $\mathsf{span}(\vec{y}, \vec{z})$.

This concludes the proof of Lemma A.14. □

# B Proof of Theorem 2.8

Here, we prove that a single challenge secure symmetric-key predicate-only encryption scheme supporting inner product queries for vectors of length $2n$ can be used to construct a fully secure symmetric-key predicate-only encryption scheme supporting inner product queries for vectors length $n$. Our proof is inspired by the hybrid argument used by [21].

*Proof.* Let $\textsc{Scheme}_{2n}$ be a single challenge secure symmetric-key predicate-only encryption scheme supporting inner product queries over $\mathbb{Z}_N^{2n}$. We construct a fully secure symmetric-key predicate-only encryption scheme $\textsc{Scheme}_n$ supporting inner product queries over $\mathbb{Z}_N^n$.

For any two vectors $\vec{x} = (x_1, \dots x_n), \vec{y} = (y_1, \dots, y_n) \in \mathbb{Z}_N^n$, define $\vec{x}\|\vec{y} = (x_1, \dots, x_n, y_1, \dots, y_n)$ to be the vector obtained by concatenating $\vec{x}$ and $\vec{y}$.

Informally, $\textsc{Scheme}_n$ works as follows. To encrypt a vector $\vec{x} \in \mathbb{Z}_N^n$, encrypt the vector $\vec{x}\|\vec{x} \in \mathbb{Z}_N^{2n}$ using $\textsc{Scheme}_{2n}$. Similarly, to construct a token for the vector $\vec{v} \in \mathbb{Z}_N^n$, use $\textsc{Scheme}_{2n}$ to construct a token for the vector $\vec{v}\|\vec{v} \in \mathbb{Z}_N^{2n}$. The algorithms of $\text{Scheme}_n$ are defined as follows.

$\textsc{Scheme}_n.\mathit{Setup}(1^\lambda)$: Run $\textsc{Scheme}_{2n}.\mathit{Setup}(1^\lambda)$. The secret key $SK$ is the same as that generated by $\textsc{Scheme}_{2n}$.

$\textsc{Scheme}_n.\mathit{Encrypt}(SK, \vec{x})$: Output $\textsc{Scheme}_{2n}.\mathit{Encrypt}(SK, \vec{x}\|\vec{x})$.

$\textsc{Scheme}_n.\mathit{GenToken}(SK, \vec{v})$: Output $\textsc{Scheme}_{2n}.\mathit{GenToken}(SK, \vec{v}\|\vec{v})$.

$\textsc{Scheme}_n.\mathit{Query}(TK_{\vec{v}}, CT)$: Output $\textsc{Scheme}_{2n}.\mathit{Query}(TK_{\vec{v}}, CT)$.

The correctness of $\textsc{Scheme}_n$ results from the fact that for vectors $\vec{x}, \vec{v} \in \mathbb{Z}_N^n$,

$$\langle \vec{x}, \vec{v} \rangle = 0 \quad \textit{iff} \quad \langle \vec{x}\|\vec{x},\ \vec{v}\|\vec{v} \rangle = 0.$$

We now show that $\textsc{Scheme}_n$ is fully secure. Recall the full security game defined in Section 2.2.1. First, the challenger picks a random bit $\mathsf{b}$. Next, the adversary $\mathcal{A}$ adaptively issues queries to the challenger. If a query is a ciphertext query $(\vec{x}_{j,0}, \vec{x}_{j,1})$, the challenger responds with an encryption of $\vec{x}_{j,\mathsf{b}}$. If a query is a token query $(\vec{v}_{i,0}, \vec{v}_{i,1})$, the challenger responds with a token for $\vec{v}_{i,\mathsf{b}}$. $\mathcal{A}$'s queries are subject to the restriction that, for all ciphertext queries $(x_{j,0}, x_{j,1})$ and all predicate queries $(f_{i,0}, f_{i,1})$, $f_{i,0}(x_{j,0}) = f_{i,1}(x_{j,1})$. At the end of the game, $\mathcal{A}$ outputs a guess $\mathsf{b}'$ of $\mathsf{b}$ and wins if $\mathsf{b}' = \mathsf{b}$.

Suppose that the adversary $\mathcal{A}$ makes $c$ ciphertext queries, $(\vec{x}_{1,0},\ \vec{x}_{1,1})$, $\dots$, $(\vec{x}_{c,0},\ \vec{x}_{c,1})$, and $t$ token queries, $(\vec{v}_{1,0}, \vec{v}_{1,1}), \dots, (\vec{v}_{t,0}, \vec{v}_{t,1})$.

Our task is to show that $\mathcal{A}$ cannot distinguish between two experiments: one where the challenger constructs ciphertexts for $\vec{x}_{1,0}, \dots, \vec{x}_{c,0}$ and tokens for $\vec{v}_{1,0}, \dots, \vec{v}_{t,0}$ (call this Game 0), and one where the challenger constructs ciphertexts for $\vec{x}_{1,1}, \dots, \vec{x}_{c,1}$ and tokens for $\vec{v}_{1,1}, \dots, \vec{v}_{t,1}$ (call this Game 1). To do this, we construct a series of hybrid games as follows.

**Game 0** : The challenger calls $\textsc{Scheme}_{2n}$ and computes ciphertexts for $\vec{x}_{1,0}\|\vec{x}_{1,0}$, $\vec{x}_{2,0}\|\vec{x}_{2,0}$, $\dots$, $\vec{x}_{c,0}\|\vec{x}_{c,0}$ and tokens for $\vec{v}_{1,0}\|\vec{v}_{1,0}$, $\vec{v}_{2,0}\|\vec{v}_{2,0}$, $\dots$, $\vec{v}_{t,0}\|\vec{v}_{t,0}$.

**Game A** : The challenger calls $\textsc{Scheme}_{2n}$ and computes ciphertexts for $\vec{x}_{1,0}\|\vec{0}$, $\vec{x}_{2,0}\|\vec{0}$, $\dots$, $\vec{x}_{c,0}\|\vec{0}$ and tokens for $\vec{v}_{1,0}\|\vec{v}_{1,0}$, $\vec{v}_{2,0}\|\vec{v}_{2,0}$, $\dots$, $\vec{v}_{t,0}\|\vec{v}_{t,0}$.

**Game B** : The challenger calls $\textsc{Scheme}_{2n}$ and computes ciphertexts for $\vec{x}_{1,0}\|\vec{0}$, $\vec{x}_{2,0}\|\vec{0}$, …, $\vec{x}_{c,0}\|\vec{0}$ and tokens for $\vec{v}_{1,0}\|\vec{v}_{1,1}$, $\vec{v}_{2,0}\|\vec{v}_{2,1}$, …, $\vec{v}_{t,0}\|\vec{v}_{t,1}$.

**Game M** : The challenger picks a random $\alpha \xleftarrow{R} \mathbb{Z}_N$, calls $\textsc{Scheme}_{2n}$ and computes ciphertexts for $\vec{x}_{1,0}\|\alpha\vec{x}_{1,1}$, $\vec{x}_{2,0}\|\alpha\vec{x}_{2,1}$, …, $\vec{x}_{c,0}\|\alpha\vec{x}_{c,1}$ and tokens for $\vec{v}_{1,0}\|\vec{v}_{1,1}$, $\vec{v}_{2,0}\|\vec{v}_{2,1}$, …, $\vec{v}_{t,0}\|\vec{v}_{t,1}$.

Notice that in the above sequence of hybrid games, the outcomes of the predicates corresponding to the generated tokens on the plaintexts in $\mathbb{Z}_N^{2n}$ encrypted by the challenger remain the same between all pairs of adjacent games, except with negligible probability.

**Claim 1.** *If $\textsc{Scheme}_{2n}$ is single challenge secure, then no PPT adversary $\mathcal{A}$ has more than negligible advantage in distinguishing between any pair of adjacent games in the above sequence of games.*

*Proof.* By a hybrid argument. □

Similarly, we can construct a sequence of hybrid games connecting Game $M$ and Game 1. Using a hybrid argument, we conclude that no PPT adversary has more than negligible advantage in distinguishing between Game 0 and Game 1.

# C KSW Predicate Encryption Scheme

To aid in the understanding of our construction and the proof of security, we review the KSW public key predicate-only encryption scheme for inner product queries [21].

Let $\mathcal{G}'$ denote a group generator algorithm for a bilinear group whose order is the product of three distinct primes.

*Setup*$(1^\lambda)$: The setup algorithm runs $\mathcal{G}'(1^\lambda)$ to obtain $(p, q, r, \mathbb{G}, \mathbb{G}_T, e)$ with $\mathbb{G} = \mathbb{G}_p \times \mathbb{G}_q \times \mathbb{G}_r$. Next it picks generators $g_p, g_q, g_r$ from subgroups $\mathbb{G}_p, \mathbb{G}_q, \mathbb{G}_r$, respectively. It then chooses, uniformly at random, $h_{1,i}, h_{2,i} \in \mathbb{G}_p$, $R_{1,i}, R_{2,i} \in \mathbb{G}_r$ for $i = 1$ to $n$, and $R_0 \in \mathbb{G}_r$.

The public key consists of:

$$PK = (g_p,\ g_r,\ Q = g_q \cdot R_0,\ \{H_{1,i} = h_{1,i} \cdot R_{1,i},\ H_{2,i} = h_{2,i} \cdot R_{2,i}\}_{i=1}^n)$$

The secret key is set to:

$$SK = \left(p, q, r, g_q, \{h_{1,i}, h_{2,i}\}_{i=1}^n\right).$$

*Encrypt*$(PK, \vec{x})$: Let $\vec{x} = (x_1, \ldots, x_n) \in \mathbb{Z}_N^n$. The encryption algorithm first picks random exponents $y, \alpha, \beta$ from $\mathbb{Z}_N$, and it chooses random $R_{3,i}, R_{4,i} \in \mathbb{G}_r$ for $i = 1$ to $n$. It outputs the ciphertext

$$CT = \left(C = g_p^y,\ \left\{C_{1,i} = H_{1,i}^y \cdot Q^{\alpha x_i} \cdot R_{3,i},\quad C_{2,i} = H_{2,i}^y \cdot Q^{\beta x_i} \cdot R_{4,i}\right\}_{i=1}^n\right).$$

*GenToken*$(SK, \vec{v})$: Let $\vec{v} = (v_1, \ldots, v_n) \in \mathbb{Z}_N^n$. The token generation algorithm chooses random $f_1, f_2, \{r_{1,i}, r_{2,i}\}_{i=1}^n$ from $\mathbb{Z}_N$, random $R_5 \in \mathbb{G}_r$, and random $Q_6 \in \mathbb{G}_q$. It outputs the token

$$TK_{\vec{v}} = \left(\begin{array}{c} K = R_5 \cdot Q_6 \cdot \prod_{i=1}^n h_{1,i}^{-r_{1,i}} \cdot h_{2,i}^{-r_{2,i}}, \\ \left\{K_{1,i} = g_p^{r_{1,i}} \cdot g_q^{f_1 v_i},\ K_{2,i} = g_p^{r_{2,i}} \cdot g_q^{f_2 v_i}\right\}_{i=1}^n \end{array}\right).$$

*Query*$(TK_{\vec{v}}, CT)$: Let $CT = (C, \{C_{1,i}, C_{2,i}\}_{i=1}^n)$ and $TK_{\vec{v}} = (K, \{K_{1,i}, K_{2,i}\}_{i=1}^n)$ as above. The query algorithm outputs 1 iff

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