Hidden Vector Encryption Fully Secure Against Unrestricted Queries
No Question Left Un answered

Angelo De Caro Vincenzo Iovino Giuseppe Persiano

Dipartimento di Informatica ed Applicazioni, Università di Salerno, 84084 Fisciano (SA), Italy.
{decaro, iovino, giuper}@dia.unisa.it.

March 4, 2011

Abstract

Predicate encryption is an important cryptographic primitive (see [3, 6, 10, 11]) that enables fine-grained control on the decryption keys. Let $P$ be a binary predicate. Roughly speaking, in a predicate encryption scheme for predicate $P$ the owner of the master secret key $\text{Msk}$ can derive secret key $\text{Sk}_{\vec{y}}$, for any vector $\vec{y}$. In encrypting a message $M$, the sender can specify an attribute vector $\vec{x}$ and the resulting ciphertext $\tilde{X}$ can be decrypted only by using keys $\text{Sk}_{\vec{y}}$ such that $P(\vec{x}, \vec{y}) = 1$.

Our main contribution is the first construction of a predicate encryption scheme that can be proved fully secure against unrestricted queries by probabilistic polynomial-time adversaries under non-interactive constant sized (that is, independent of the length $\ell$ of the attribute vectors) hardness assumptions on bilinear groups of composite order.

Specifically, we consider hidden vector encryption (HVE in short), a notable case of predicate encryption introduced by Boneh and Waters [6]. In a HVE scheme, the ciphertext attributes are vectors $\vec{x} = (x_1, \ldots, x_\ell)$ of length $\ell$ over alphabet $\Sigma$, keys are associated with vectors $\vec{y} = (y_1, \ldots, y_\ell)$ of length $\ell$ over alphabet $\Sigma \cup \{\star\}$ and we consider the $\text{Match}(\vec{x}, \vec{y})$ predicate which is true if and only if, for all $i$, $y_i \neq \star$ implies $x_i = y_i$. Previous constructions restricted the proof of security to adversaries that could ask only non-matching queries; that is, for challenge attribute vectors $\vec{x}_0$ and $\vec{x}_1$, the adversary could ask only for keys of vectors $\vec{y}$ such that $\text{Match}(\vec{x}_0, \vec{y}) = \text{Match}(\vec{x}_1, \vec{y}) = 0$.

Our proof employs the dual system methodology of Waters [19], that gave one of the first fully secure construction in this area, blended with a careful design of intermediate security games that keep into account the relationship between challenge ciphertexts and key queries.

Keywords: predicate encryption, full security, pairing-based cryptography.
Contents

1 Introduction and related work ........................................... 3

2 Hidden Vector Encryption .............................................. 4
   2.1 Hidden Vector Encryption ........................................ 4
   2.2 Security definitions for HVE .................................... 5

3 Complexity Assumptions .............................................. 5

4 Constructing HVE ..................................................... 6

5 Security of our HVE scheme .......................................... 7
   5.1 The first step of the proof ..................................... 8
   5.2 The second step of the proof .................................. 9
      5.2.1 Some simple observations about GBadQ and GBadQ2 .... 10
      5.2.2 Description of simulator S ................................ 10
      5.2.3 Properties of simulator S ................................... 12
      5.2.4 The advantage of $A$ in GPK ............................. 15
   5.3 Wrapping up ..................................................... 17

6 Conclusions and future directions ................................... 17

7 Acknowledgements ................................................... 17

A An alternative description of our HVE ............................. 20

B Correctness of our HVE scheme ..................................... 20

C Generic security of our Complexity Assumptions ................. 22

D Full-Fledged HVE .................................................... 23
   D.1 Security definition .............................................. 24
   D.2 Full-fledged scheme for HVE ................................. 24

E Reductions .......................................................... 25
   E.1 Boolean Satisfaction Encryption ............................. 25
   E.2 Security definitions for Boolean Satisfaction Encryption .... 26
   E.3 Reducing k-CNF to HVE .................................... 26
   E.4 Reducing Disjunctions to HVE .............................. 27
   E.5 Reducing k-DNF to k-CNF ................................. 27
1 Introduction and related work

Predicate encryption is an important cryptographic primitive (see [3, 6, 10, 11]) that enables fine-grained control on the decryption keys. Let \( P \) be a binary predicate. Roughly speaking, in a predicate encryption scheme for predicate \( P \) the owner of the master secret key \( \text{Msk} \) can derive secret key \( \text{Sk}_y \) for any vector \( \vec{y} \). In encrypting a message \( M \), the sender can specify an attribute vector \( \vec{x} \) and the resulting ciphertext \( \tilde{X} \) can be decrypted only by using keys \( \text{Sk}_y \) such that \( P(\vec{x}, \vec{y}) = 1 \). Thus a predicate encryption scheme enables the owner of the master secret key to delegate the decryption of different types of ciphertexts (as specified by the attribute vector) to different entities.

Our main contribution is the first construction of a predicate encryption scheme that can be proved fully secure against unrestricted queries from probabilistic polynomial-time adversaries under non-interactive constant sized (that is independent of \( \ell \)) hardness assumptions on bilinear groups of composite order.

More specifically, we consider hidden vector encryption (HVE in short), a notable case of predicate encryption introduced by [6]. In a HVE scheme, the ciphertext attributes are vectors \( \vec{x} = \langle x_1, \ldots, x_\ell \rangle \) of length \( \ell \) over alphabet \( \Sigma \), keys are associated with vectors \( \vec{y} = \langle y_1, \ldots, y_\ell \rangle \) of length \( \ell \) over alphabet \( \Sigma \cup \{ \star \} \) and we consider the \text{Match}(\vec{x}, \vec{y}) \) predicate which is true if and only if, for all \( i, y_i \neq \star \) implies \( x_i = y_i \).

We model our security notion by means of a game against a PPT adversary \( A \) that sees the public key (thus we depart from the selective model of security), is allowed to ask for keys of vectors \( \vec{y} \) of his choice and gives two challenge vectors \( \vec{x}_0 \) and \( \vec{x}_1 \). \( A \) then receives a challenge ciphertext (an encryption of a randomly chosen challenge vectors) and has to guess which of the two challenge vectors has been encrypted. The adversary \( A \) is allowed to ask queries even after seeing the challenge ciphertext. Unlike previous work, we only require the adversary \( A \) to ask for keys of vectors \( \vec{y} \) that do not discriminate the two challenge vectors; that is, for which \( \text{Match}(\vec{x}_0, \vec{y}) = \text{Match}(\vec{x}_1, \vec{y}) \). It can be readily seen that this condition is necessary. Previous constructions restricted the proof of security to adversaries that could ask only non-matching queries; that is, ask for keys of vectors \( \vec{y} \) such that \( \text{Match}(\vec{x}_0, \vec{y}) = \text{Match}(\vec{x}_1, \vec{y}) = 0 \). Thus our construction is the first to be proved fully secure against unrestricted PPT adversaries \( A \).

Besides being one of the first predicates for which constructions have been given, HVE can be used as building block for several other predicates. Specifically in [6], it is shown that HVE implies predicate encryption schemes for conjunctions, comparison, range queries and subset queries. For completeness, in Appendix E we describe also constructions of secure predicate encryption for boolean predicates that can be expressed as \( k \)-CNF and \( k \)-DNF (for any constant \( k \)).

We also stress that the two computational assumptions on which we base our proof of security are very natural. Specifically, our two assumptions posit the difficulty of a subgroup decision problem and of a problem that can be seen as the generalization of Decision Diffie-Hellman to groups of composite order.

Related Work. The first implementation of HVE is due to [6] that proved the security of their construction under assumptions on bilinear groups of composite order in the selective model. In this security model (introduced by [2] in the context of IBE), the adversary must commit to its challenge vectors before seeing the public key of the HVE scheme. In a recent series of papers Waters [19] and Lewko and Waters [13] introduced the concept of a dual system encryption scheme that was used to construct efficient and fully secure Identity Based Encryption (IBE) and Hierarchical IBE from simple assumptions. Previous fully secure construction of these primitives either used a partitioning strategy (see [2], [18]) or used complexity assumptions of non-constant size (see [8], [12]). Partitioning strategy and the approaches of [8] and [9] do not seem to be helpful in proving full
security of more complex primitives like HVE.

For HVE, fully secure constructions of HVE can be derived, via the reduction given in [11], from the fully secure constructions for inner-product encryption given by [12, 14]. The resulting constructions are proved secure against adversaries that are allowed to ask only non-matching key queries; that is, key queries for vectors \( \vec{y} \) such that \( \text{Match}(\vec{x}_0, \vec{y}) = \text{Match}(\vec{x}_1, \vec{y}) = 0 \). We call queries for vectors \( \vec{y} \) such that \( \text{Match}(\vec{x}_0, \vec{y}) = \text{Match}(\vec{x}_1, \vec{y}) = 1 \) matching. We believe the constructions of [14, 12] can be modified so that they are secure against adversaries that either ask only matching queries or ask only non-matching queries but do not mix the two types of queries. Our construction instead poses no restriction on the queries that adversaries can ask. The limitation of having security against non-matching adversaries was already pointed out in [11] and [17] described a scheme secure against non-matching adversaries that was not secure against unrestricted adversaries. Our security notion is game-based. Recently, Boneh et al. [5] showed that for predicate encryption the game-based and simulation-based notions of security do not coincide and showed that simulation-based security is impossible to achieve even for IBE and Anonymous IBE.

**Proof technique.** Our result is based on the dual system encryption methodology introduced by Waters [19] and gives extra evidence of the power of this proof technique. However, to overcome the difficulty of having to deal also with matching queries, we have to carefully look at the space of matching queries and at how they relate to the challenge vectors. This enables us to craft a new security game in which the challenge ciphertext is constructed in a way that guarantees that keys obtained by the adversary give the expected result when tested against the challenge ciphertext and, at the same time, the challenge ciphertext is independent from the challenge vector used to construct it. Then we show, by means of a sequence of intermediate security games, that the real security game is computationally indistinguishable from this new game. It is not immediate to obtain a similar relation between matching queries and challenge vectors for other predicates (e.g., inner product) that would make our approach viable.

## 2 Hidden Vector Encryption

In this section we give formal definitions for Hidden Vector Encryption (HVE) and its security properties. For sake of simplicity, we present predicate-only definitions and constructions for HVE instead of full-fledged ones. In Appendix D we will briefly discuss how to extend our scheme to the full-fledged version. For the same reason, we give our definitions and constructions for binary alphabets. Extensions to general alphabets is straightforward.

Following standard terminology, we call a function \( \nu(\lambda) \) negligible if for all constants \( c > 0 \) and sufficiently large \( \lambda \), \( \nu(\lambda) < 1/\lambda^c \) and denote by \([n]\) the set of integers \( \{1, \ldots, n\} \). Moreover the writing “\( a \leftarrow A \)' for a finite set \( A \), denotes that \( a \) is randomly and uniformly selected from \( A \).

### 2.1 Hidden Vector Encryption

Let \( \vec{x} \) be vector of length \( \ell \) over the alphabet \( \{0, 1\} \) and \( \vec{y} \) vector of the same length over the alphabet \( \{0, 1, \star\} \). Define the predicate \( \text{Match}(\vec{x}, \vec{y}) = \text{TRUE} \) if and only if for any \( i \in [\ell] \), it holds that \( x_i = y_i \) or \( y_i = \star \). That is, the two vectors must match in the positions \( j \) where \( y_j \neq \star \).

A Hidden Vector Encryption scheme is a tuple of four efficient probabilistic algorithms (Setup, Encrypt, KeyGen, Test) with the following semantics.

- **Setup(1^{\lambda}, 1^{\ell})**: takes as input a security parameter \( \lambda \) and a length parameter \( \ell \) (given in unary), and outputs the public parameters \( Pk \) and the master secret key \( Msk \).
KeyGen(Msk, ⃗y): takes as input the master secret key Msk and a vector ⃗y ∈ \{0, 1, ⋆\}^ℓ, and outputs a secret key Sk_{⃗y}.

Encrypt(Pk, ⃗x): takes as input the public parameters Pk and a vector ⃗x ∈ \{0, 1\}^ℓ and outputs a ciphertext Ct.

Test(Pk, Ct, Sk_{⃗y}): takes as input the public parameters Pk, a ciphertext Ct encrypting ⃗x and a secret key Sk_{⃗y} and outputs Match(⃗x, ⃗y).

For correctness we require that, for pairs (Pk, Msk) output by Setup(1^λ, 1^ℓ), it holds that for all vectors ⃗x ∈ \{0, 1\}^ℓ and ⃗y ∈ \{0, 1, ⋆\}^ℓ, we have that Test(Pk, Encrypt(Pk, ⃗x), KeyGen(Msk, ⃗y)) = Match(⃗x, ⃗y) except with negligible in λ probability.

2.2 Security definitions for HVE

In this section we formalize our security requirement by means of a security game GReal between a probabilistic polynomial time adversary A and a challenger C. GReal consists of a Setup phase and of a Query Answering phase. In the Query Answering phase, the adversary can issue a polynomial number of Key Queries and one Challenge Construction query and at the end of this phase A outputs a guess. We stress that key queries can be issued by A even after he has received the challenge from C. In GReal the adversary is restricted to queries for vectors ⃗y such that Match(⃗y, x_0) = Match(⃗y, x_1).

More precisely, we define game GReal in the following way.

Setup. C runs the Setup algorithm on input the security parameter λ and the length parameter ℓ (given in unary) to generate public parameters Pk and master secret key Msk. C starts the interaction with A on input Pk.

Key Query Answering. Upon receiving a query for vector ⃗y, C returns KeyGen(Msk, ⃗y).

Challenge Construction. Upon receiving the pair (⃗x_0, ⃗x_1), C picks random η ∈ \{0, 1\} and returns Encrypt(Pk, ⃗x_η).

Winning Condition. Let η' be A’s output. We say that A wins the game if η = η' and for all ⃗y for which A has issued a Key Query, it holds Match(⃗x_0, ⃗y) = Match(⃗x_1, ⃗y).

We define the advantage Adv^A_{\text{HVE}}(λ) of A in GReal to be the probability of winning minus 1/2.

Definition 1. An Hidden Vector Encryption scheme is secure if for all probabilistic polynomial time adversaries A, we have that Adv^A_{\text{HVE}}(λ) is a negligible function of λ.

It is trivial to observe that no scheme can be secure in the sense of Definition 1 against an adversary that possesses a secret key for a vector ⃗y such that Match(⃗y, x_0) ≠ Match(⃗y, x_1).

3 Complexity Assumptions

Composite order bilinear groups were first used in Cryptography by [4] (see also [3]). We suppose the existence of an efficient group generator algorithm G which takes as input the security parameter λ and outputs a description I = (N, G, G_T, e) of a bilinear setting, where G and G_T are cyclic groups of order N, and e : G^2 → G_T is a map with the following properties:

1. (Bilinearity) ∀ g, h ∈ G and a, b ∈ Z_N it holds that e(g^a, h^b) = e(g, h)^{ab}.

2. (Non-degeneracy) ∃ g ∈ G such that e(g, g) has order N in G_T.
We assume that the group descriptions of \( G \) and \( \mathbb{G}_T \) include generators of the respective cyclic groups. We require that the group operations in \( G \) and \( \mathbb{G}_T \) as well as the bilinear map \( e \) are computable in deterministic polynomial time in \( \lambda \). In our construction we will make hardness assumptions for bilinear settings whose order \( N \) is product of four distinct primes each of length \( \Theta(\lambda) \). For an integer \( m \) dividing \( N \), we let \( G_m \) denote the subgroup of \( G \) of order \( m \). From the fact that the group is cyclic, it is easy to verify that if \( g \) and \( h \) are group elements of co-prime orders then \( e(g, h) = 1 \). This is called the orthogonality property and is a crucial tool in our constructions.

We are now ready to give our complexity assumptions.

**Assumption 1.** The first assumption is a subgroup-decision type assumption for bilinear settings. Specifically, Assumption 1 posits the difficulty of deciding whether an element belongs to one of two specified subgroups, even when generators of some of the subgroups of the bilinear group are given. More formally, we have the following definition.

First pick a random bilinear setting \( \mathcal{I} = (N = p_1 p_2 p_3 p_4, G, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda) \) and then pick \( A_3 \leftarrow G_{p_2}, A_{13} \leftarrow G_{p_1 p_2}, A_{12} \leftarrow G_{p_1 p_3}, A_4 \leftarrow G_{p_4}, T_1 \leftarrow G_{p_1 p_3}, T_2 \leftarrow G_{p_2 p_3}, \) and set \( D = (\mathcal{I}, A_3, A_{13}, A_{12}). \) We define the advantage of an algorithm \( A \) in breaking Assumption 1 to be

\[
\text{Adv}_A^1(\lambda) = |\text{Prob}[A(D, T_1) = 1] - \text{Prob}[A(D, T_2) = 1]| 
\]

**Assumption 1.** We say that Assumption 1 holds for generator \( \mathcal{G} \) if for all probabilistic polynomial-time algorithms \( A \), \( \text{Adv}_A^1(\lambda) \) is a negligible function of \( \lambda \).

**Assumption 2.** Our second assumption can be seen as the Decision Diffie-Hellman Assumption. Specifically, Assumption 2 posits the difficulty of deciding if a triple of elements constitute a Diffie-Hellman triplet with respect to one of the factors of the order of the group, even when given, for each prime divisor \( p \) of the group order, a generator of the subgroup of order \( p \). Notice that for bilinear groups of prime order the Diffie-Hellman assumption does not hold. More formally, we have the following definition.

First pick a random bilinear setting \( \mathcal{I} = (N = p_1 p_2 p_3 p_4, G, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda) \) and then pick \( A_1 \leftarrow G_{p_1}, A_2 \leftarrow G_{p_2}, A_3 \leftarrow G_{p_1}, A_4 \leftarrow G_{p_4}, B_1, C_4, D_4 \leftarrow G_{p_4}, \alpha, \beta \leftarrow \mathbb{Z}_{p_1}, T_2 \leftarrow G_{p_1 p_4}, \) and set \( T_1 = A_1^{\alpha \beta} \cdot D_4 \) and \( D = (\mathcal{I}, A_1, A_2, A_3, A_4, A_1^\alpha \cdot B_4, A_1^\beta \cdot C_4). \) We define the advantage of an algorithm \( A \) in breaking Assumption 2 to be

\[
\text{Adv}_A^2(\lambda) = |\text{Prob}[A(D, T_1) = 1] - \text{Prob}[A(D, T_2) = 1]| 
\]

**Assumption 2.** We say that Assumption 2 holds for generator \( \mathcal{G} \) if for all probabilistic polynomial-time algorithms \( A \), \( \text{Adv}_A^2(\lambda) \) is a negligible function of \( \lambda \).

In Appendix C we prove that Assumption 1 and 2 hold in the generic group model.

### 4 Constructing HVE

In this section we describe our construction for an HVE scheme. To make our descriptions and proofs simpler, we add to all vectors \( \vec{x} \) and \( \vec{y} \) two dummy components and set both of them equal to 0. We can thus assume that all vectors have at least two non-star positions.

**Setup(1^\lambda, 1^\lambda):** The setup algorithm chooses a description of a bilinear group \( \mathcal{I} = (N = p_1 p_2 p_3 p_4, G, \mathbb{G}_T, e) \) with known factorization by running a generator algorithm \( \mathcal{G} \) on input \( 1^\lambda \). The setup
algorithm chooses random \( g_1 \in \mathbb{G}_{p_1}, \ g_2 \in \mathbb{G}_{p_2}, \ g_3 \in \mathbb{G}_{p_3}, \ g_4 \in \mathbb{G}_{p_4} \), and, for \( i \in [\ell] \) and \( b \in \{0,1\} \), random \( t_{i,b} \in \mathbb{Z}_N \) and random \( R_{i,b} \in \mathbb{G}_{p_3} \) and sets \( T_{i,b} = g_1^{t_{i,b}} \cdot R_{i,b} \).

The public parameters are \( \mathbb{P}k = [N, g_3, (T_{i,b})_{i \in [\ell], b \in \{0,1\}}] \) and the master secret key is \( \mathbb{M}sk = [g_{12}, g_4, (t_{i,b})_{i \in [\ell], b \in \{0,1\}}] \), where \( g_{12} = g_1 \cdot g_2 \).

**KeyGen(\( \mathbb{M}sk, \vec{y} \)):** Let \( S_{\vec{y}} \) be the set of indices \( i \) such that \( y_i \neq \star \). The key generation algorithm chooses random \( a_i \in \mathbb{Z}_N \) for \( i \in S_{\vec{y}} \) under the constraint that \( \sum_{i \in S_{\vec{y}}} a_i = 0 \). For \( i \in S_{\vec{y}} \), the algorithm chooses random \( W_i \in \mathbb{G}_{p_4} \) and sets

\[
Y_i = g_{12}^{a_i/t_{i,y_i}} \cdot W_i.
\]

The algorithm returns the tuple \( (Y_i)_{i \in S_{\vec{y}}} \). Here we use the fact that \( S_{\vec{y}} \) has size at least 2.

**Encrypt(\( \mathbb{P}k, \vec{x} \)):** The encryption algorithm chooses random \( s \in \mathbb{Z}_N \). For \( i \in [\ell] \), the algorithm chooses random \( Z_i \in \mathbb{G}_{p_3} \) and sets

\[
X_i = T_{i,x_i}^s \cdot Z_i,
\]

and returns the tuple \( (X_i)_{i \in [\ell]} \).

**Test(\( \mathbb{C}t, \mathbb{S}k_{\vec{y}} \)):** The test algorithm computes \( T = \prod_{i \in S_{\vec{y}}} e(X_i, Y_i) \). It returns TRUE if \( T = 1 \), FALSE otherwise.

**Correctness**  In Appendix \[\text{[\text{A}]\text{]}\] we prove that the above scheme is correct.

**Remark 2.** Let \( \mathbb{P}k = [N, g_3, (T_{i,b})_{i \in [\ell], b \in \{0,1\}}] \) and \( \mathbb{M}sk = [g_1 : g_2, g_4, (t_{i,b})_{i \in [\ell], b \in \{0,1\}}] \) be a pair of public parameter and master secret key output by the Setup algorithm and consider \( \mathbb{P}k' = [N, g_3, (T_{i,b}')_{i \in [\ell], b \in \{0,1\}}] \) and \( \mathbb{M}sk' = [g_1' : g_2, g_4, (t_{i,b}')_{i \in [\ell], b \in \{0,1\}}] \) with \( T_{i,b}' = g_1^{t_{i,b}} \cdot R_{i,b} \) for some \( g_1' \in \mathbb{G}_{p_1} \) and \( R_{i,b}' \in \mathbb{G}_{p_3} \). We make the following easy observations.

1. For every \( \vec{y} \in \{0,1,\star\}^{\ell} \), the distributions \( \text{KeyGen}(\mathbb{M}sk, \vec{y}) \) and \( \text{KeyGen}(\mathbb{M}sk', \vec{y}) \) are identical.
2. Similarly, for every \( \vec{x} \in \{0,1\}^{\ell} \), the distributions \( \text{Encrypt}(\mathbb{P}k, \vec{x}) \) and \( \text{Encrypt}(\mathbb{P}k', \vec{x}) \) are identical.

5 Security of our HVE scheme

We start by giving an informal description of the idea behind our proof of security and show how we overcome the main technical difficulty of having to deal with matching keys.

The first step of our proof strategy consists in projecting the public key (and thus the ciphertexts the adversary constructs by himself) to a different subgroup from the one of the challenge ciphertext. Specifically, we defined a new security game \( \mathbb{G}PK \) in which the public key is constructed so that all relevant information (that is, the \( t_{i,b}'s \)) is encoded in the \( \mathbb{G}_{p_2} \) part of the public key instead of the \( \mathbb{G}_{p_1} \) as in the real game. The challenge ciphertext and the answers to the key queries are instead constructed as in the real security game \( \mathbb{G}Real \). Thus, ciphertexts constructed by the adversary are completely independent from the challenge ciphertext (as they encode information in two different subgroups). The view of an adversary in \( \mathbb{G}PK \) is still indistinguishable from the view of the real security game \( \mathbb{G}Real \). We observe that since keys are constructed as in the real security game, they carry information about \( \vec{y} \) both in the \( \mathbb{G}_{p_1} \) and \( \mathbb{G}_{p_2} \) parts. Thus when the adversary tests a ciphertext he has constructed by using the public key against a key obtained by means of a query, he obtains the expected result because of the information encoded in the \( \mathbb{G}_{p_2} \) part of the key and of the ciphertext. The same holds for the challenge ciphertext but now thanks to the \( \mathbb{G}_{p_1} \) part of the
key. The only difference is in the public key but, under Assumption 1 (a natural subgroup decision hardness assumption), we can prove that the view remains indistinguishable.

The second step proves that the keys obtained from queries do not help the adversary. Since the challenge ciphertext carries information about the randomly selected challenge vector $\mathbf{x}_\eta$ in its $G_1$ part, in this informal discussion when we refer to key we mean its $G_1$ part. The $G_2$ parts of the keys are always correctly computed.

In our construction, testing a ciphertext against a non-matching key gives a random value (from the target group) whereas testing it against a matching key returns a specified value (the identity of the target group). One possible avenue for proving security against an adversary that asks only non-matching queries is to show that replying key queries by returning random elements from the appropriate subgroups instead of well-formed keys, gives a new game which is indistinguishable from the real security game and in which it is easy to prove that the adversary can win with probability essentially 1/2. This approach fails for matching queries as a random reply to a matching query is unlikely to return the correct answer when tested against the challenge ciphertext. Instead we modify the construction of the challenge ciphertext in the following way: the challenge ciphertext is well-formed in all the positions where the two challenge vectors are equal and random in all the other positions. We observe that testing such a challenge ciphertext against matching and non-matching keys always gives the correct answer and that no adversary (even an all powerful one) can guess which of the two challenge vectors has been used to construct the challenge ciphertext.

We thus prove security of our construction by proving that the above game (that is called $\text{GBadCh}(\ell + 1)$) is indistinguishable from $GPK$. We achieve this by means of $\ell + 1$ intermediate games: $\text{GBadCh}(1) = GPK, \ldots, \text{GBadCh}(\ell + 1)$ where in each game we switch to random one more component of the challenge ciphertext which corresponds to position in which the challenge vectors differ. Now let us consider two consecutive games $\text{GBadCh}(f)$ and $\text{GBadCh}(f + 1)$. It is easy to see that if the challenge vectors coincide in the $f$-th component the two games are exactly the same. Let us now discuss the case in which the two games differ in the $f$-th component. To do so, we consider intermediate games in which we modify the answer to the queries but, for the reasons discussed above, we cannot naively reply randomly to the queries. Rather, for adversaries that ask $q$ key queries, we consider $q + 1$ intermediate games $\text{GBadQ}(f, 0) = \text{GBadCh}(f), \ldots, \text{GBadQ}(f, q)$. In the $k$-th intermediate game we alter the distribution of the reply to the first $k$ key queries based on the the following observation that relates matching queries to the challenge vectors: if the challenge vectors differ in the $f$-th component, a query is matching only if it has a $\star$ in position $f$. Thus, if the key has a $\star$ in position $f$, then its $f$-th component is empty and this is easy to simulate. If instead a query has a non-$\star$ in the position $f$, then it must be a non matching query and thus it safe to randomize the reply. Using Assumption 2, we prove that this modification in the answers to the key queries still guarantees indistinguishability.

At this point, we would like to prove that $\text{GBadQ}(f, q)$ is indistinguishable from $\text{GBadQ}(f + 1, 0)$ (which is equal to $\text{GBadCh}(f + 1)$). Unfortunately, this is not the case and we need to resort to extra security games $\text{GBadQ2}(f, q), \ldots, \text{GBadQ2}(f, 0)$ to complete the proof.

5.1 The first step of the proof

We start by defining $GPK(\lambda, \ell)$ that differs from $G\text{Real}(\lambda, \ell)$ as in the Setup phase, $C$ prepares two sets of public parameters, $Pk$ and $Pk'$, and one master secret key $Msk$. $Pk$ is given as input to $A$, $Msk$ is used to answer $A$'s key queries and $Pk'$ is used to construct the challenge ciphertext. Specifically,

Setup. $C$ chooses a description of a bilinear group $\mathcal{I} = (N = p_1 p_2 p_3 p_4, G, G_T, e)$ with known
factorization by running a generator algorithm $G$ on input $1^\lambda$. $C$ chooses random $g_1 \in G_{p_1}$, $g_2 \in G_{p_2}$, $g_3 \in G_{p_3}$, $g_4 \in G_{p_4}$ and sets $g_{12} = g_1 \cdot g_2$. For each $i \in \ell$ and $b \in \{0, 1\}$, $C$ chooses random $t_{i,b} \in \mathbb{Z}_N$ and $R_{i,b} \in G_{p_3}$ and sets $T_{i,b} = g_{12}^{t_{i,b}} \cdot R_{i,b}$. Then $C$ sets $\text{Pk} = [N, g_3, (T_{i,b})_{i \in \ell, b \in \{0, 1\}}]$, $\text{Pk}' = [N, g_3, (T'_{i,b})_{i \in \ell, b \in \{0, 1\}}]$, and $\text{Msk} = [g_{12}, g_4, (T_{i,b})_{i \in \ell, b \in \{0, 1\}}]$. Finally, $C$ starts the interaction with $A$ on input $\text{Pk}$.

**Key Query Answering**($\bar{y}$). $C$ returns the output of $\text{KeyGen}(\text{Msk}, \bar{y})$.

**Challenge Query Answering**($\bar{x}_0, \bar{x}_1$). Upon receiving the pair $(\bar{x}_0, \bar{x}_1)$ of challenge vectors, $C$ picks random $\eta \in \{0, 1\}$ and returns the output of $\text{Encrypt}(\text{Pk}', \bar{x}_\eta)$.

**Winning Condition.** Like in $G_{\text{Real}}(\lambda, \ell)$.

The next lemma shows that, the advantages of an adversary in $G_{\text{Real}}(\lambda, \ell)$ and $G_{\text{PK}}(\lambda, \ell)$ are the same, up to a negligible factor.

**Lemma 3.** If Assumption 1 holds, then for any PPT adversary $A$, $|\text{Adv}^{A}[G_{\text{Real}}(\lambda, \ell)] - \text{Adv}^{A}[G_{\text{PK}}(\lambda, \ell)]|$ is negligible.

**Proof.** We show a PPT algorithm $B$ which receives $(T, A_3, A_4, A_{13}, A_{12})$ and $T$ and, depending on the nature of $T$, simulates $G_{\text{Real}}(\lambda, \ell)$ or $G_{\text{PK}}(\lambda, \ell)$ with $A$. This suffices to prove the Lemma.

**Setup.** $B$ starts by constructing public parameters $\text{Pk}$ and $\text{Pk}'$ in the following way. $B$ sets $g_{12} = A_{12}, g_3 = A_3, g_4 = A_4$ and, for each $i \in \ell$ and $b \in \{0, 1\}$, $B$ chooses random $t_{i,b} \in \mathbb{Z}_N$ and sets $T_{i,b} = T'^{t_{i,b}}$ and $T'_{i,b} = A_{13}^{t_{i,b}}$. Then $B$ sets $\text{Pk} = [N, g_3, (T_{i,b})_{i \in \ell, b \in \{0, 1\}}]$, $\text{Msk} = [g_{12}, g_4, (T_{i,b})_{i \in \ell, b \in \{0, 1\}}]$, and $\text{Pk}' = [N, g_3, (T'_{i,b})_{i \in \ell, b \in \{0, 1\}}]$ and starts the interaction with $A$ on input $\text{Pk}$.

**Key Query Answering**($\bar{y}$). $B$ runs algorithm $\text{KeyGen}$ on input $\text{Msk}$ and $\bar{y}$.

**Challenge Query Answering**($\bar{x}_0, \bar{x}_1$). The challenge is created by $B$ by picking random $\eta \in \{0, 1\}$ and running the $\text{Encrypt}$ algorithm on input $\bar{x}_\eta$ and $\text{Pk}'$.

This concludes the description of algorithm $B$.

Now suppose $T \in G_{p_1 p_3}$, and thus it can be written as $T = h_1 \cdot h_3$ for $h_1 \in G_{p_1}$ and $h_3 \in G_{p_3}$. This implies that $\text{Pk}$ received in input by $A$ in the interaction with $B$ has the same distribution as in $G_{\text{Real}}$. Moreover, by writing $A_{13}$ as $A_{13} = \tilde{h}_1 \cdot h_3$ for $\tilde{h}_1 \in G_{p_1}$ and $h_3 \in G_{p_3}$ which is possible since by assumption $A_{13} \in G_{p_{1 p_3}}$, we notice that that $\text{Pk}$ and $\text{Pk}'$ are as in the hypothesis of Remark 2 (with $g_1 = \tilde{h}_1$ and $g_1 = h_1$). Therefore the answers to key queries and the challenge ciphertext given by $B$ to $A$ have the same distribution as the answers and the challenge ciphertext received by $A$ in $G_{\text{Real}}(\lambda, \ell)$. We can thus conclude that, when $T \in G_{p_1 p_3}$, $C$ has simulates $G_{\text{Real}}(\lambda, \ell)$ with $A$.

Let us discuss now the case $T \in G_{p_2 p_3}$. In this case, $\text{Pk}$ provided by $B$ has the same distribution as the public parameters produced by $C$ in $G_{\text{PK}}(\lambda, \ell)$. Therefore, $C$ is simulating $G_{\text{PK}}(\lambda, \ell)$ for $A$.

This concludes the proof of the lemma. \hfill \Box

### 5.2 The second step of the proof

We start the second step of the proof by describing, for $1 \leq f \leq \ell + 1$ and $0 \leq k \leq q$, the $\text{GBadQ}(f, k)$ between the challenger $C$ and an adversary $A$. Not to overburden the notation, we omit $\lambda$ and $\ell$ from the name of the games. $\text{GBadQ}(f, k)$ differs from $G_{\text{PK}}$ both in the way in which key queries are answered and in the way in which the challenge ciphertext is constructed. More precisely, in $\text{GBadQ}(f, k)$ the first $k$ key queries are answered by distinguishing two cases. Queries for $\bar{y}$ such that $\eta_f = \star$ are answered by running $\text{KeyGen}(\text{Msk}, \bar{y})$. Instead queries for $\bar{y}$ such that $\eta_f \neq \star$ are answered by returning keys whose $G_{p_1}$ part is random for all components which
correspond to non-\(\star\) entries. Moreover, in \(\text{GBadQ}(f, k)\), the first \(f - 1\) components of the challenge ciphertext corresponding to positions in which the two challenges differ are random in \(\mathbb{G}_{p_1p_3}\). Let us now formally describe \(\text{GBadQ}(f, k)\).

**Setup.** Like in GPK.

**Key Query Answering** (\(\check{y}\)). \(C\) answers the first \(k\) queries in the following way. If \(y_f \neq \star\), \(C\) chooses, for each \(i \in S_y\) random \(W_i \in \mathbb{G}_{p_1}\), random \(C_i \in \mathbb{G}_{p_1}\) and random \(a_i \in \mathbb{Z}_N\) under the constraint that \(\sum_{i \in S_y} a_i = 0\) and sets \(Y_i = C_i \cdot g_2^{a_i/t_i,y_i} \cdot W_i\). If \(y_f = \star\) then \(C\) returns the output of \(\text{KeyGen}(\check{y}, \text{Msk})\). The remaining \(q - k\) queries are answered by running \(\text{KeyGen}(\check{y}, \text{Msk})\).

**Challenge Query Answering** (\(\langle \vec{x}_0, \vec{x}_1, f, k \rangle\)). \(C\) chooses random \(s \in \mathbb{Z}_N\) and \(\eta \in \{0, 1\}\) and sets \(\vec{x} = \vec{x}_\eta\). For each \(i \in [f - 1]\) such that \(\vec{x}_{0,i} \neq \vec{x}_{1,i}\), \(C\) chooses random \(X_i \in \mathbb{G}_{p_1p_3}\). Then, for each remaining \(i\), \(C\) chooses random \(Z_i \in \mathbb{G}_{p_1}\) and sets \(X_i = T_{i,x_i}^s \cdot Z_i\). \(C\) returns the tuple \((X_i)_{i \in [\ell]}\).

**Winning Condition.** Like in GReal.

In the proofs, we will use the shorthand \(\text{GBadCh}(f)\) for \(\text{GBadQ}(f, 0)\). Moreover, we define \(\text{GBadQ2}(f, k)\), for \(1 \leq f \leq \ell\) and \(0 \leq k \leq q\), as a game in which the setup phase is like in \(\text{GBadQ}(f, k)\), key queries are answered like in \(\text{GBadQ}(f, k)\) and the challenge ciphertext is constructed like in \(\text{GBadQ}(f+1, k)\).

### 5.2.1 Some simple observations about \(\text{GBadQ}\) and \(\text{GBadQ2}\)

**Observation 4.** GPK = \(\text{GBadQ}(1, 0)\).

Straightforward from the definitions of the games.

**Observation 5.** \(\text{GBadQ}(f, q) = \text{GBadQ2}(f, q)\) for \(f = 1, \ldots, \ell\).

From the definitions of the two games, it is clear that all key queries are answered in the same way in both the games and all components \(X_i\) for \(i \neq f\) of the challenge ciphertext are computed in the same way. Let us now look at \(X_f\) and more precisely to its \(\mathbb{G}_{p_1}\) part. In \(\text{GBadQ}(f, q)\), the \(\mathbb{G}_{p_1}\) part of \(X_f\) is computed as \(T_{f,x_f}^s\), which is exactly how it is computed in \(\text{GBadQ2}(f, q)\) when \(x_{0,f} = x_{1,f}\). On the other hand, when \(x_{0,f} \neq x_{1,f}\), the \(\mathbb{G}_{p_1}\) part of \(X_f\) is chosen at random. However, observe that exponents \(t_{f,0} \mod p_1\) and \(t_{f,1} \mod p_1\) have not appeared in the answers to key queries since every query has either a \(\star\) in position \(f\) (in which case position \(f\) of the answer is empty) or a non-\(\star\) value in position \(f\) (in which case the \(\mathbb{G}_{p_1}\) part of the position \(f\) of the answer is random since \(k = q\)). Therefore, we can conclude that the \(\mathbb{G}_{p_1}\) part of the component \(X_f\) of the answer to the challenge query is also random in \(\mathbb{G}_{p_1}\).

**Observation 6.** \(\text{GBadQ2}(f, 0) = \text{GBadQ}(f + 1, 0)\) for \(f = 1, \ldots, \ell - 1\).

Indeed, in both games all key queries are answered correctly, and the challenge query in \(\text{GBadQ2}(f, 0)\) is by definition answered in the same way as in \(\text{GBadQ}(f + 1, 0)\).

**Observation 7.** All adversaries have no advantage in \(\text{GBadCh}(\ell + 1) = \text{GBadQ}(\ell + 1, 0)\).

This follows from the fact that, for positions \(i\) such that \(x_{0,i} \neq x_{1,i}\), the \(\mathbb{G}_{p_1}\) part of \(X_i\) is random. Thus, the challenge ciphertext of \(\text{GBadCh}(\ell + 1)\) is independent from \(\eta\).

### 5.2.2 Description of simulator \(S\)

In this section, we describe a PPT simulator \(S\) that interacts with an adversary \(A\) and will be used in our proof.

**Input.** Integers \(1 \leq f \leq \ell + 1\) and \(0 \leq k \leq q\), and a randomly chosen instance \((D, T)\) of Assumption 2; recall that \(D = (\mathcal{I}, A_1, A_2, A_3, A_4, A_1^\alpha \cdot B_4, A_3^\beta \cdot C_4)\) and \(T = T_1 = A_1^\beta \cdot D_4\) or \(T = T_2\) random in \(\mathbb{G}_{p_1p_4}\).
Setup. To simulate the Setup phase \( S \) executes the following steps.

1. \( S \) sets \( g_1 = A_1, g_2 = A_2, g_3 = A_3, g_4 = A_4 \) and \( g_{12} = A_1 \cdot A_2 \).
2. For each \( i \in [\ell] \) and \( b \in \{0, 1\}, \)
   \( S \) chooses random \( v_{i,b} \in \mathbb{Z}_N \) and \( R_{i,b} \in \mathbb{G}_{p_3} \), and sets \( T_{i,b} = g_2^{v_{i,b}} \cdot R_{i,b} \).
3. \( S \) sets \( Pk = [N, g_3, (T_{i,b})_{i \in [\ell], b \in \{0, 1\}}] \).
4. \( S \) picks random \( \hat{j} \in [\ell] \) and \( \hat{b} \in \{0, 1\} \) and sets \( \hat{c} = 1 - \hat{b} \).
5. For each \( i \in [\ell] \setminus \{\hat{j}\} \) and \( b \in \{0, 1\}, \)
   \( S \) chooses random \( r_{i,b} \in \mathbb{Z}_N \) and \( R'_{i,b} \in \mathbb{G}_{p_3} \).
   \( S \) sets \( T'_{i,b} = g_1^{r_{i,b}} \cdot R'_{i,b} \).
6. \( S \) chooses random \( r_{j,\hat{c}} \in \mathbb{Z}_N \) and \( R'_{j,\hat{c}} \in \mathbb{G}_{p_3} \).
   \( S \) sets \( T'_{j,\hat{c}} = g_1^{r_{j,\hat{c}}} \cdot R'_{j,\hat{c}} \).

The value of \( r_{i,b} \) and of \( T'_{j,\hat{c}} \) remain unspecified. As we shall see below, in answering key queries, \( S \) will implicitly set \( r_{i,b} = 1/\beta \). We stress that \( \beta \) is the same exponent appearing in \( A_1^\beta \cdot C_4 \) from the instance of Assumption 2 and that \( S \) does not have access to the actual value of \( \beta \).

\( S \) starts the interaction with \( A \) on input \( Pk \).

Key Query Answering(\( \vec{y} \)). To describe how \( S \) answers the first \( k-1 \) queries, we distinguish the following two mutually exclusive cases.

- **Case A.1:** \( y_f \neq * \). In this case, \( S \) outputs a key whose \( \mathbb{G}_{p_1} \) part is random. More precisely, \( S \) executes the following steps. For each \( i \in S_{\vec{y}} \), \( S \) chooses random \( a''_i \) such that \( \sum_{i \in S_{\vec{y}}} a''_i = 0 \), random \( C_i \in \mathbb{G}_{p_1} \), and random \( W_i \in \mathbb{G}_{p_4} \). Then, for each \( i \in S_{\vec{y}} \), \( S \) sets
  \[
  Y_i = C_i \cdot g_2^{a''_{i}/v_{i,y_i}} \cdot W_i.
  \]

- **Case A.2:** \( y_f = * \). In this case, \( S \) tries to output a key that has the same distribution induced by algorithm KeyGen on input \( \vec{y} \). We observe that if \( y_j = \hat{c} \) then \( S \) knows all the \( r_{i,y_i} \)'s and \( v_{i,y_i} \)'s needed. If instead \( y_j = \hat{b} \), then \( S \) is missing \( r_{j,b} \). In this case \( S \) computes \( Y_j \) by using \( A_1^\beta \cdot C_4 \) from the challenge of Assumption 2 received in input.

More precisely, for each \( i \in S_{\vec{y}} \), \( S \) picks random \( W_i \in \mathbb{G}_{p_4} \) and random \( a'_i, a''_i \in \mathbb{Z}_N \) under the constraint that \( \sum_{i \in S_{\vec{y}}} a'_i = \sum_{i \in S_{\vec{y}}} a''_i = 0 \). Then for each \( i \neq j \), \( S \) sets
  \[
  Y_i = g_1^{a'_i/r_{i,y_i}} \cdot g_2^{a''_{i}/v_{i,y_i}} \cdot W_i.
  \]
Moreover, if \( y_j = \hat{c} \), \( S \) sets
  \[
  Y_j = g_1^{a'_j/r_{j,\hat{c}}} \cdot g_2^{a''_j/v_{j,\hat{c}}} \cdot W_j
  \]
otherwise, if \( y_j = \hat{b} \), \( S \) sets
  \[
  Y_j = (A_1^\beta \cdot C_4)^{a'_j} \cdot g_2^{a''_{j}/v_{j,b}} \cdot W_j = g_1^{a'_j} \cdot g_2^{a''_{j}/v_{j,b}} \cdot (C_4^{a'_j} \cdot W_j).
  \]
Notice that this setting implicitly defines \( r_{j,b} = 1/\beta \) which remains unknown to \( S \).

Let us now describe how \( S \) answers the \( k \)-th query for vector \( \vec{y}^{(k)} = (y^{(k)}_1, \ldots, y^{(k)}_k) \). We have three cases and we let \( \text{Guess}^{A,S}_{1,k}(f,k) \) denote the event that \( S \), on input \( f \) and \( k \) and interacting with \( A \), does not abort in computing the answer to the \( k \)-th query.

- **Case B.1:** \( y^{(k)}_f = * \). \( S \) performs the same steps of Case A.2.

- **Case B.2:** \( y^{(k)}_f \neq * \) and \( y^{(k)}_j \neq \hat{b} \). \( S \) outputs \( \perp \) and aborts.
Case B.3: $y_f^{(k)} \neq \ast$ and $y_i^{(k)} = \hat{b}$. Let $S = S_{\overline{g}} \setminus \{j, h\}$, where $h$ is an index such that $y_h^{(k)} \neq \ast$. Such an index $h$ always exists since we assumed that each query contains at least two non-$\ast$ entries. Then, for each $i \in S$, $S$ chooses random $W_i \in \mathbb{G}_{p_4}$ and random $a_i', a_i'' \in \mathbb{Z}_N$ and sets

$$Y_i = g_1^{a_i'/r_{i,y_i^{(k)}}} \cdot g_2^{a_i''/v_{i,y_i^{(k)}}} \cdot W_i.$$ 

$S$ then chooses random $a_i'' \in \mathbb{Z}_N$ and $W_j, W_h \in \mathbb{G}_{p_4}$ and sets

$$Y_j = T \cdot g_2^{a_i''/v_{j,b}} \cdot W_j \quad \text{and} \quad Y_h = (A_1^0 B_4)^{-1/r_{h,y_h^{(k)}}} \cdot g_1^{s'/r_{h,y_h^{(k)}}} \cdot g_2^{-(s''+a_i'')} \cdot W_h,$$

where $s' = \sum_{i \in S} a_i'$ and $s'' = \sum_{i \in S} a_i''$.

This terminates the description of how $S$ handles the $k$-th key query.

$S$ handles the remaining $q - k$ queries as in Case A.2, independently from whether $y_f = \ast$ or $y_f \neq \ast$. More precisely, if $y_j = \hat{c}$ then $S$ has all the $r_{i,y_i}$'s and $v_{i,y_i}$'s needed. On the other hand, if this is not the case then $S$ can use $A_1^\beta \cdot C_4$ from $D$.

**Challenge Query Answering** $(\tilde{x}_0, \tilde{x}_1)$. If $\tilde{x}_0$ and $\tilde{x}_1$ coincide on the $f$-th component or $y_j^{(k)} = x_{\eta_3}$, $S$ aborts. We let $\text{Guess}_{2, S}(f, k)$ denote the event that $y_j^{(k)} \neq x_{\eta_3}$ while $S$ is interacting with $A$ on input $f$ and $k$. We observe that if $\text{Guess}_{2, S}(f, k)$ occurs, $x_{\eta_3} = \hat{c} = 1 - \hat{b}$.

If $S$ has not aborted, $S$ picks random $\eta \in \{0, 1\}$, sets $\tilde{x} = \tilde{x}_\eta$ and creates the challenge by running algorithm $\text{Encrypt}$ on input the challenge vector $\tilde{x}$, public parameters $\mathbb{G}_p'$ and randomizing the $\mathbb{G}_{p_4}$ part of all $X_i$ for $i < f$ such that $x_{0,i} \neq x_{1,i}$. More precisely, the challenge ciphertext is created as follows. $S$ chooses random $s \in \mathbb{Z}_N$. For each $i \in [f - 1]$ such that $x_{0,i} \neq x_{1,i}$, $S$ sets $r_i$ equal to a random element in $\mathbb{Z}_N$. $S$ sets $r_i = 1$ for all remaining $i$'s. For each $i \in [\ell]$, $S$ picks random $Z_i \in \mathbb{G}_{p_3}$ and sets $X_i = T_i^{r_i} \cdot Z_i$, and returns the tuple $(X_i)_{i \in [\ell]}$.

Two remarks are in order. First, if $S$ has not aborted, then it has access to all the values $T_{i,x_i}$ needed for computing an encryption of $\tilde{x}$ ($= \tilde{x}_\eta$). Second, as a sanity check, we verify that $S$ cannot test the nature of $T$ and thus break Assumption 2. Indeed to do so, $S$ should use $T$ to generate a key for $\tilde{y}$ and ciphertext for $\tilde{x}$ such that $\text{Match}(\tilde{x}, \tilde{y}) = 1$. Then, if $T = T_1$ the Test procedure will have success; otherwise, it will fail. In constructing the key, $S$ would use $T$ to construct the $j$-th component (which forces $y_j = \hat{b}$) and then it would need $r_{j,b}$ to construct the matching ciphertext. However, $S$ does not have access to this value.

The simulator $S$ described will be used to prove properties of games $\text{GBadQ}$. We can modify the simulator $S$ so that, on input $f$ and $k$, the challenge ciphertext is constructed by randomizing the $\mathbb{G}_{p_4}$ part also of the $f$-th component. The so modified simulator, that we call $S_2$, closely simulates the work of games $\text{GBadQ2}$ and will be used to prove properties of these games.

### 5.2.3 Properties of simulator $S$

We now state and prove some properties of $S$ that will be used in our security proof. We start by defining event $E_{f, G}^A$, as the event that in game $G$ the adversary $A$ declares two challenge vectors that differ in the $f$-th component. When the adversary $A$ is clear from the context we will simply write $E_{f, G}$. We use notation $E_f$ to denote $E_{f, G}$ for $G = \text{GPK}$.

We extend the definition of $E_{f, G}$ to include the game played by $A$ against the simulator $S$. Thus we denote by $E_{f, S}^A(f', k)$ the event that in the interaction between $A$ and $S$ on input $f'$ and $k$, $A$ declares two challenge vectors that differ in the $f$-th component. If $A, f'$ and $k$ are clear from
the context, we will simply write $E_{f,S}$. With this notation in place, the event “$S$ does not abort while interacting with $A$ on input $f$ and $k$” is equal to the event

$$\text{Guess}^A_{1,S}(f,k) \land \text{Guess}^A_{2,S}(f,k) \land E^A_{f,S}(f,k).$$

(1)

In addition, we observe that event $E^A_{f,S}(f,k)$ implies event $\text{Guess}^A_{1,S}(f,k)$ and similarly does event $\text{Guess}^A_{2,S}(f,k)$. We modify the challenger $C$ so that at the beginning of the interaction with $A$, $C$ picks $j$ and $b$ just like $S$ does. This modification makes the definitions of events $\text{Guess}^A_{1,G}$ and $\text{Guess}^A_{2,G}$ meaningful. Notice that, unlike the simulator $S$, the challenger never aborts its interaction with $A$ and that this modification does not affect $A$’s view. This implies, for example, that the fact that event $E^A_{f,C}$ has occurred during a game $G$ does not necessarily imply that event $\text{Guess}^A_{1,G}$ also occurs. We write $\text{Guess}^A_{2}$ as a shorthand for $\text{Guess}^A_{2,G}$ with $G = \text{GBadQ}$.

**Lemma 8.** For all $f$, $k$ and $A$, $\text{Prob}[\text{Guess}^A_{1,S}(f,k)] \geq \frac{1}{7}$.

**Proof.** It is easy to see that the probability of $\text{Guess}^A_{1,S}(f,k)$ is at least the probability that $y^{(k)}_j = \hat{b}$. Moreover, the view of $A$ up to the $k$-th key query is independent from $\hat{b}$ and $j$. Now observe that $\hat{y}^{(k)}$ has at least two non-star entry and, provided that if $j$ is one of these (which happens with probability $2/\ell$), the probability that $y^{(k)}_j = \hat{b}$ is $1/2$. \hfill $\square$

**Lemma 9.** For all $f$, $k$ and $A$, $\text{Prob}[\text{Guess}^A_{2,G}(f,k)] \geq \frac{1}{27}$ where $G = \text{GBadQ}(f,k)$.

**Proof.** $\text{Guess}^A_{2,G}(f,k)$ is the event that $y^{(k)}_j \neq x_{\eta,j}$ in the game $G$ played by the challenger $C$ with $A$. It is easy to see that the probability that $C$ correctly guesses $j$ and $\hat{b}$ such that $x_{\eta,j} = \hat{c} = 1 - \hat{b}$ is at least $1/(2\ell)$, independently from the view of $A$. \hfill $\square$

**Lemma 10.** Suppose $T = T_1$. If $S$ does not abort in the computation of the answer to the $k$-th query, then $A$’s view up to the Challenge Query in the interaction with $S$ is the same as in $\text{GBadQ}(f,k-1)$. Moreover, if $S$ completes its execution without aborting, then $A$’s total view is the same as in $\text{GBadQ}(f,k-1)$.

Suppose instead that $T = T_2$. If $S$ does not abort in the computation of the answer to the $k$-th query, then $A$’s view up to the Challenge Query in the interaction with $S$ is the same as in $\text{GBadQ}(f,k)$. Moreover, if $S$ completes its execution without aborting, then $A$’s total view is the same as in $\text{GBadQ}(f,k)$.

**Proof.** For the proof of this lemma it is convenient to refer to the alternative and equivalent description of our HVE found in Section [A]. We notice that $\text{Pk}$ has the same distribution as the public parameters seen by $A$ in both games. The same holds for the answers to the first $(k-1)$ Key Queries and to the last $(q-k)$ Key Queries. Let us now focus on the answer to the $k$-th Key Query. We have two cases:

**Case 1:** $y^{(k)}_j = \ast$. Then the view of $A$ in the interaction with $S$ is independent from $T$ (see Case B.1) and, on the other hand, by definition, the two games coincide. Therefore the lemma holds in this case.

**Case 2:** $y^{(k)}_j \neq \ast$. Suppose $T = T_1 = A_1^\alpha \beta \cdot D_4$ and that $\text{Guess}^A_{1,S}(f,k)$ occurs. Therefore, $y^{(k)}_j = \hat{b}$ and $S$’s answer to the $k$-th key query has the same distributions as in $\text{GBadQ}(f,k-1)$. Indeed, we have that

$$Y_j = g_1^{\alpha^{(j_k)b}} \cdot g_2^{\alpha^{(j_k)b}} \cdot D_4 \cdot W_j$$
with $a'_j = \alpha$ and $r_{j,b} = 1/\beta$ and

$$Y_h = g_1^{-\alpha_j' - s'/r_h g_h(k)} \cdot g_2^{-(\alpha_j'' + s'')/v_h g_h(k)} \cdot \left(B_4^{-1/r_h g_h(k)} \cdot W_h\right).$$

and thus the $a'_j$’s and $a''_j$’s are random and sum up to 0.

On the other hand if $T$ is random in $\mathbb{G}_{p_1 p_4}$ and $S$ does not abort, the $\mathbb{G}_{p_4}$ parts of the $Y_i$’s are random and thus the answer to the $k$-th query of $A$ is distributed as in $\text{GBadQ}(f, k)$.

Finally, we observe that, if $S$ does not abort then the challenge ciphertext is constructed as in $\text{GBadQ}(f, k - 1)$ and $\text{GBadQ}(f, k)$.

$\square$

**Proof.**

**Lemma 11.** If Assumption 2 holds, then for $k = 1, \ldots, q$ and $f = 1, \ldots, \ell + 1$, and for all PPT adversaries $A$, $|\text{Prob}[E^A_{f,G}] - \text{Prob}[E^A_{f,H}]| \leq 2^{-\lambda}$ and $|\text{Prob}[\text{Guess}^A_{2,G}] - \text{Prob}[\text{Guess}^A_{2,H}]|$ are negligible functions of $\lambda$, for games $G = \text{GBadQ}(f, k - 1)$ and $H = \text{GBadQ}(f, k)$ or $G = \text{GBadQ2}(f, k - 1)$ and $H = \text{GBadQ2}(f, k)$.

**Proof.** Let us prove first the case when $G = \text{GBadQ}(f, k - 1)$ and $H = \text{GBadQ}(f, k)$. For sake of contradiction, suppose that $\text{Prob}[E^A_{f,G}] \geq \text{Prob}[E^A_{f,H}] + \epsilon$ for some non-negligible $\epsilon$. A similar reasoning holds $\text{Guess}^A_{2,G}$ and $\text{Guess}^A_{2,H}$. Then we can modify simulator $S$ into algorithm $B$ with a non-negligible advantage in breaking Assumption 2. Algorithm $B$ simply execute $S$’s code. By Lemma 8 event $\text{Guess}_{1,S}$ occurs with probability at least $1/\ell$ and in this case $B$ can continue the execution of $S$’s code and receive the challenge vectors from $A$. At this point, $B$ checks whether they differ in the $f$-th component. If they do, $B$ outputs 1; else $B$ outputs 0. It is easy to see that, by Lemma 10 the above algorithm has a non-negligible advantage in breaking Assumption 2.

We apply the the same reasoning to the case when $G = \text{GBadQ2}(f, k - 1)$ and $H = \text{GBadQ2}(f, k)$, considering algorithm $B$ that uses the code of simulator $S_2$ rather than that of $S$.

The proof of the following corollary is straightforward from Lemma 11 and Observations 4-6.

**Corollary 12.** For all $f = 1, \ldots, \ell + 1$ and $k = 0, \ldots, q$, and all PPT adversaries $A$, we have that, for $H = \text{GBadQ}(f, k)$ or $H = \text{GBadQ2}(f, k)$, $|\text{Prob}[E^A_{f,H}] - \text{Prob}[E^A_f]|$ and $|\text{Prob}[\text{Guess}^A_{2,H}] - \text{Prob}[\text{Guess}^A_{2,f}]|$ are negligible.

We are now ready to prove the following crucial technical lemma.

**Lemma 13.** Suppose there exists an adversary $A$ and integers $1 \leq f \leq \ell + 1$ and $1 \leq k \leq q$ such that $|\text{Adv}^A[G] - \text{Adv}^A[H]| \geq \epsilon$, where $G = \text{GBadQ}(f, k - 1)$, $H = \text{GBadQ}(f, k)$ and $\epsilon > 0$. Then, there exists a PPT algorithm $B$ with $\text{Adv}^B \geq \text{Prob}[E^B_f] \cdot \epsilon/(2 \cdot \ell^2) - \nu(\lambda)$, for a negligible function $\nu$.

**Proof.** Assume without loss of generality that $\text{Adv}^A[G] \geq \text{Adv}^A[H] + \epsilon$ and consider the following algorithm $B$. Algorithm $B$ uses simulator $S$ as a subroutine and interacts with $A$ on input integers $f$ and $k$ for which the above inequality holds, and an instance $(D, T)$ of Assumption 2. $B$ receives $A$’s output $\eta'$ and checks if $\eta = \eta'$ (recall that $\eta$ is the random bit chosen by $S$ in preparing the challenge ciphertext). If $\eta = \eta'$ then $B$ outputs 1; otherwise $B$ outputs 0. Therefore we have

$$\text{Prob}[B \text{ outputs } 1|T = T_1] = \text{Prob}[B \text{ outputs } 1|T = T_1 \text{ and } S \text{ does not abort}] \cdot \frac{\text{Prob}[S \text{ does not abort}|T = T_1]}{1}$$

14
By [1], we have
\[
\text{Prob}[\mathcal{S} \text{ does not abort } | T = T_1] = \text{Prob}[E_{f,S} \land \text{Guess}_{1,S} \land \text{Guess}_{2,S} | T = T_1] \\
= \text{Prob}[\text{Guess}_{1,S} | T = T_1] \cdot \text{Prob}[E_{f,S} \land \text{Guess}_{2,S} | \text{Guess}_{1,S} \land T = T_1].
\]
Now observe that event \text{Guess}_{1,S} is determined before \mathcal{S} uses \text{T} and thus
\[
\text{Prob}[\text{Guess}_{1,S} | T = T_1] = \text{Prob}[\text{Guess}_{1,S}].
\]
Moreover, by Lemma [10], if event \text{Guess}_{1,S} occurs and \text{T} = T_1, the view of \mathcal{A} up to Challenge Query is equal to the view of \mathcal{A} in game \text{G} and thus
\[
\text{Prob}[E_{f,S} \land \text{Guess}_{2,S} | \text{Guess}_{1,S} \land \text{T} = T_1] = \text{Prob}[E_{f,G} \land \text{Guess}_{2,G}]
\]
whence
\[
\text{Prob}[\mathcal{S} \text{ does not abort } | T = T_1] = \text{Prob}[\text{Guess}_{1,S}] \cdot \text{Prob}[\text{Guess}_{2,G} \land E_{f,G}] \\
= \text{Prob}[\text{Guess}_{1,S}] \cdot \text{Prob}[\text{Guess}_{2,G}] \cdot \text{Prob}[E_{f,G}] \\
\text{ (\text{Guess}_{2,G} \text{ and } E_{f,G} \text{ are independent})}
\]
Finally, if \text{T} = T_1 and \mathcal{S} does not abort, then, by Lemma [10], \mathcal{A}'s view is exactly as in game \text{G}, and thus the probability that \mathcal{B} outputs 1 is equal to the probability that \mathcal{A} wins in game \text{G}. We can thus rewrite Eq. [2] as
\[
\text{Prob}[\mathcal{B} \text{ outputs } 1 | T = T_1] = \text{Prob}[\mathcal{A} \text{ wins in } \text{G}] \cdot \text{Prob}[\text{Guess}_{1,S}] \cdot \text{Prob}[\text{Guess}_{2,G}] \cdot \text{Prob}[E_{f,G}]
\]
A similar reasoning yields
\[
\text{Prob}[\mathcal{B} \text{ outputs } 1 | T = T_2] = \text{Prob}[\mathcal{A} \text{ wins in } \text{H}] \cdot \text{Prob}[\text{Guess}_{1,S}] \cdot \text{Prob}[\text{Guess}_{2,H}] \cdot \text{Prob}[E_{f,H}]
\]
By using Corollary [12], Lemma [8] and Lemma [9] we can conclude that there exists a negligible function \nu such that we have
\[
\text{Adv}_{2}^B = \text{Prob}[\text{Guess}_{1,S}] \cdot \text{Prob}[\text{Guess}_{2}] \cdot \text{Prob}[E_f] \cdot \left( \text{Prob}[\mathcal{A} \text{ wins in } \text{G}] - \text{Prob}[\mathcal{A} \text{ wins in } \text{H}] \right) - \nu(\lambda)
\geq \frac{\epsilon}{2\ell^2} \cdot \text{Prob}[E_f] - \nu(\lambda).
\]
\[
\square
\]
The followingLemma can be proved by referring to simulator \mathcal{S}_2. We omit further details since the proof is essentially the same as the one of Lemma [13].

**Lemma 14.** Suppose there exists an adversary \mathcal{A} and integers 1 \leq f \leq \ell + 1 and 1 \leq k \leq q such that \text{Adv}^\mathcal{A}[G] - \text{Adv}^\mathcal{A}[H] \geq \epsilon, where \text{G} = \text{GBadQ2}(f, k - 1), \text{H} = \text{GBadQ2}(f, k) and \epsilon > 0. Then, there exists a PPT algorithm \mathcal{B} with \text{Adv}^\mathcal{B}_{2} \geq \text{Prob}[E_f] \cdot \epsilon/(2 \cdot \ell^2) - \nu(\lambda), for a negligible function \nu.

### 5.2.4 The advantage of \mathcal{A} in GPK

In this section we prove that, under Assumption 2, every PPT adversary \mathcal{A} has a negligible advantage in GPK = \text{GBadCh}(1) by proving that it is computationally indistinguishable from \text{GBadCh}(\ell + 1) that, by Observation [7], gives no advantage to any adversary.

**Proof.** Let \text{E}_{f,f}' denote the event that during the execution of \text{GBadCh}(f') adversary \mathcal{A} outputs two challenge vectors that differ in the \text{f}-th component. For an event \text{E}, we define the advantage \text{Adv}^\mathcal{A}[G|\text{E}] of \mathcal{A} in \text{G} conditioned on event \text{E} as \text{Adv}^\mathcal{A}[G|\text{E}] = \text{Prob}[\mathcal{A} \text{ wins in } \text{G}|\text{E}] - \frac{1}{2}.
Observation 15. For all PPT adversaries $A$ and all $1 \leq f \leq \ell$, we have that
\[ \text{Adv}^A[\text{GBadCh}(f)|\neg E_{f,f}] = \text{Adv}^A[\text{GBadCh}(f+1)|\neg E_{f,f+1}]. \]

Proof. By definition of GBadCh, if the two challenge vectors coincide in the $f$-th component, then the views of $A$ in GBadCh$(f)$ and GBadCh$(f+1)$ are the same. \hfill \Box

Observation 16. For all PPT adversaries $A$ and all $1 \leq f \leq \ell$, we have that
\[ \text{Prob}[E^A_{f,f}] = \text{Prob}[E^A_{f,f+1}]. \]

Proof. The view of $A$ in GBadCh$(f)$ up to the Challenge Query is independent from $f$. \hfill \Box

Lemma 17. If Assumption 2 holds, then, for any PPT adversary $A$, $\text{Adv}^A[\text{GPK}]$ is negligible. Specifically, if there is an adversary $A$ with $\text{Adv}^A[\text{GPK}] = \epsilon$ then there exists an adversary $B$ against Assumption 2 such that $\text{Adv}^B_2 \geq \frac{\epsilon^2}{2q\ell^3} - \nu(\lambda)$, for some negligible function $\nu$.

Proof. Let $A$ be a PPT adversary such that $\text{Adv}^A[\text{GPK}] \geq \epsilon$. Since GPK = GBadCh$(1)$ and $\text{Adv}^A[\text{GBadCh}(\ell+1)] = 0$ there must exist $f \in [\ell]$ such that
\[ |\text{Adv}^A[\text{GBadCh}(f)] - \text{Adv}^A[\text{GBadCh}(f+1)]| \geq \epsilon' = \epsilon/\ell. \] (3)

Now recall that GBadCh$(f) = \text{GBadQ}(f,0)$ and GBadCh$(f+1) = \text{GBadQ}(f,0)$. Thus, there exists $k, 1 \leq k \leq q$ such that
\[ |\text{Adv}^A[G] - \text{Adv}^A[H]| \geq \epsilon'/(2q) \]
where $G = \text{GBadQ}(f,k)$ and $H = \text{GBadQ}(f,k-1)$ or where $G = \text{GBadQ}(f,k)$ and $H = \text{GBadQ}(f,k-1)$. Then by Lemma 13 in the former case, and by Lemma 14 in the latter, we can construct an adversary $B$ against Assumption 2, such that
\[ \text{Adv}^B_2 \geq \frac{\epsilon}{4q\ell^3} \cdot \text{Prob}[E_f] - \nu(\lambda) \]

Now it remains to estimate $\text{Prob}[E_f]$. Notice that we can write
\[ \text{Adv}^A[\text{GBadCh}(f)] = \text{Prob}[E_{f,f}] \cdot \text{Adv}^A[\text{GBadCh}(f)|E_{f,f}] + \text{Prob}[\neg E_{f,f}] \cdot \text{Adv}^A[\text{GBadCh}(f)|\neg E_{f,f}], \]

and
\[ \text{Adv}^A[\text{GBadCh}(f+1)] = \text{Prob}[E_{f,f+1}] \cdot \text{Adv}^A[\text{GBadCh}(f+1)|E_{f,f+1}] + \text{Prob}[\neg E_{f,f+1}] \cdot \text{Adv}^A[\text{GBadCh}(f+1)|\neg E_{f,f+1}], \]

and, by combining Equation 3 and Observations 15 and 16, we obtain
\[ \text{Prob}[E_f] \cdot |\text{Adv}^A[\text{GBadCh}(f)|E_{f,f}] - \text{Adv}^A[\text{GBadCh}(f+1)|E_{f,f+1}]| \geq \epsilon'. \]

Since no advantage is greater than $1/2$, we can conclude that $\text{Prob}[E_f] \geq 2 \cdot \epsilon'$ and thus $B$ as advantage
\[ \text{Adv}^B_2 \geq \frac{\epsilon^2}{2q\ell^3} - \nu(\lambda) \] \hfill \Box

16
5.3 Wrapping up

In this section we state our main result.

**Theorem 18.** If Assumption 1 and 2 hold, then the HVE scheme described in Section 4 is secure (in the sense of Definition 4).

**Proof.** Use Lemma 3 and Lemma 17.

6 Conclusions and future directions

In this paper we have presented the first fully secure construction for a predicate encryption that is secure against unrestricted adversarial key queries by giving a construction for HVE. Our work shows that there is no need to restrict the querying power of the adversary to obtain a fully secure construction for a predicate encryption. We base our proof of security on natural hardness assumptions on groups of composite orders. Our proof technique stems from the dual system encryption methodology of Waters [19] which is augmented with a careful design of the intermediate security games based on observations on the relationship between the challenge ciphertexts and matching and non-matching queries.

We see two complementary directions for future work. The first one concentrates on the design of more efficient schemes and this can be achieved by attacking on two fronts: reducing the number of primes in the order of the underlying bilinear group and by reducing the size of keys and ciphertexts. Along the second direction, one would like to design fully secure unrestricted-query schemes for richer classes of predicates. The first that comes to mind is the orthogonality predicate (called inner product predicate in the literature [11, 12, 14]); the main and more challenging open problem in this area tough remains to have a general result about predicates with poly-size circuits.

7 Acknowledgements

The work of the authors has been supported in part by the European Commission through the EU ICT program under Contract ICT-2007-216676 ECRYPT II, and by the italian Ministry of University and Research Project PRIN 2008 PEPPER: Privacy and Protection of Personal Data (prot. 2008SY2PH4).
References


A  An alternative description of our HVE

In this section we give an alternative albeit equivalent description of the HVE algorithms that will make our proof of security simpler.

We start from the simple observation that the exponent arithmetic is performed modulo the order of the base. For sake of concreteness, let us look at the KeyGen algorithm that sets $Y_i$ equal to

$$Y_i = (g_1 \cdot g_2)^{a_i/t_i,y_i} \cdot W_i,$$

for $a_i, t_i, y_i \in \mathbb{Z}_N$. This is equivalent to computing $Y_i$ as

$$Y_i = g_1^{a'_i/r_i,y_i} \cdot g_2^{a''_i/v_i,y_i} \cdot W_i$$

for $a'_i, a''_i, r_i, y_i, v_i, y_i \in \mathbb{Z}_N$ satisfying the following system of modular equations

$$a'_i \equiv a_i \pmod{p_1} \quad \quad r_i \equiv t_i \pmod{p_1}$$

$$a''_i \equiv a_i \pmod{p_2} \quad \quad v_i \equiv t_i \pmod{p_2}$$

Conversely, computing $Y_i = g_1^{a'_i/r_i,b} \cdot g_2^{a''_i/v_i,b}$ for $a'_i, a''_i, r_i, b, v_i, b \in \mathbb{Z}_N$ is equivalent to computing $g_1^{a_i/t_i,b} \cdot g_2^{a''_i/v_i,b}$ for $a_i, t_i, b \in \mathbb{Z}_N$ satisfying the above system of modular equations (in the unknowns $a_i$ and $t_i,b$). By the Chinese remainder theorem the above systems have solutions in $\mathbb{Z}_N$ provided that $N$ is a multiple of $p_1 \cdot p_2$. Moreover, for all values of $r_i,b$ and $v_i,b$, and of $a'_i$ and $a''_i$ the systems above have the same number of solutions. Therefore, the distributions of $Y_i = g_1^{a_i/t_i,b}$ for random $a_i, t_i, b \in \mathbb{Z}_N$ and the distribution of $Y_i = g_1^{a'_i/r_i,b} \cdot g_2^{a''_i/v_i,b}$ for random $a'_i, a''_i, r_i, b, v_i, b \in \mathbb{Z}_N$ coincide.

In view of the above observation, we can describe the Setup and KeyGen algorithms in the following way.

Setup($1^\lambda, 1^\ell$): The setup algorithm chooses a description of a bilinear group $\mathcal{I} = (N = p_1 p_2 p_3 p_4, \mathbb{G}, \mathbb{G}_T, e)$ with known factorization by running a generator algorithm $\mathcal{G}$ on input $1^\lambda$. The setup algorithm chooses random $g_1 \in \mathbb{G}_{p_1}, g_2 \in \mathbb{G}_{p_2}, g_3 \in \mathbb{G}_{p_3}, g_4 \in \mathbb{G}_{p_4}$ and, for $i \in [\ell]$ and $b \in \{0,1\}$, random $r_i,b, v_i, b \in \mathbb{Z}_N$ and random $R_i,b \in \mathbb{G}_{p_4}$ and sets $T_i,b = g_1^{r_i,b} \cdot R_i,b$.

The public parameters are $\mathcal{PK} = [N, g_3, (T_i,b)_{i \in [\ell], b \in \{0,1\}}]$ and the master secret key is $\mathcal{MSK} = [g_{12}, g_4, (r_i,b, v_i,b)_{i \in [\ell], b \in \{0,1\}}]$, where $g_{12} = g_1 \cdot g_2$.

KeyGen($\mathcal{MSK}, \vec{y}$): Let $S_{\vec{y}}$ be the set of indices $i$ such that $y_i \neq \ast$. The key generation algorithm chooses random $a'_i \in \mathbb{Z}_N$ for $i \in S_{\vec{y}}$ and random $a''_i \in \mathbb{Z}_N$ for $i \in S_{\vec{y}}$ under the constraint that $\sum_{i \in S_{\vec{y}}} a'_i = \sum_{i \in S_{\vec{y}}} a''_i = 0$. For $i \in S_{\vec{y}}$, the algorithm chooses random $W_i \in \mathbb{G}_{p_4}$ and sets

$$Y_i = g_1^{a'_i/r_i,y_i} \cdot g_2^{a''_i/v_i,y_i} \cdot W_i.$$

The algorithm returns the tuple $(Y_i)_{i \in S_{\vec{y}}}$.

B  Correctness of our HVE scheme

Lemma 19. The HVE scheme presented in Section 2 is correct.
Equation 4 follows by definition of the $X_i$’s and $Y_i$’s. Equation 5 follows from Equation 4 by definition of the $T_{i,x_i}$ and by the orthogonality property of bilinear groups of composite order discussed in Section 3. Equation 6 follows from Equation 5 since, if $\text{Match}(\vec{x}, \vec{y}) = 1$, then, for each $i \in S_{\vec{y}}$, $x_i = y_i$. Equation 7 follows from Equation 6 by the bilinear property of $e$. Equation 8 follows from Equation 7 by simple algebraic manipulations. Equation 9 follows from Equation 8 since by construction we have $\sum_{i \in S_{\vec{y}}} a_i = 0$. Finally, notice that if the value obtained by this computation is 1 (the identity of the target group) then, by definition, the $\text{Test}$ procedure returns $\text{TRUE}$ as expected. Suppose now that $\text{Match}(\vec{x}, \vec{y}) = 0$. Let $A$ be the set of positions $i$ such that $x_i = y_i$, and $B = S_{\vec{y}} \setminus A$. By definition we have that $\text{Test}(\text{Ct}, \text{Sk}_{\vec{y}})$ computes the following product

$$\prod_{i \in S_{\vec{y}}} e(X_i, Y_i) = e(g_1, g_1)^{\sum_{i \in A} a_i} \cdot e(g_1, g_1)^{\sum_{i \in B} a_i w_i}$$

Equation 10 follows by some of the facts already observed in the discussion of Equations 4-8, by definition of $A$ and $B$, and by setting $w_i = \frac{t_i \cdot x_i}{t_i \cdot y_i}$. Since that with very high probability $s \neq 0 \mod \mathbb{Z}_N$ then, by the fact that $e$ is non-degenerate, such a product equals 1 (the unity of the target group) if and only if

$$\sum_{i \in A} a_i + \sum_{i \in B} a_i w_i = 0.$$ 

Equation 11 follows by some of the facts already observed in the discussion of Equations 4-8, by definition of $A$ and $B$, and by setting $w_i = \frac{t_i \cdot x_i}{t_i \cdot y_i}$. Since that with very high probability $s \neq 0 \mod \mathbb{Z}_N$ then, by the fact that $e$ is non-degenerate, such a product equals 1 (the unity of the target group) if and only if

$$\sum_{i \in S_{\vec{y}}} a_i = 0$$

By recalling that $\sum_{i \in S_{\vec{y}}} a_i = 0$ and by the fact that $A$ and $B$ are a partition of $S_{\vec{y}}$, from Equation 11 follows that

$$\sum_{i \in S_{\vec{y}}} a_i + \sum_{i \in B} a_i (w_i - 1) = \sum_{i \in B} a_i (w_i - 1).$$

If $i \in B$, then $x_i \neq y_i, y_i \neq \ast$, and thus, without loss of generality, $w_i = \frac{t_i \cdot x_i}{t_i \cdot y_i}$ is distributed randomly in $\mathbb{Z}_N$ with very high probability since that $t_i \cdot 0$ and $t_i \cdot 1$ are chosen randomly and independently from $\mathbb{Z}_N$. Hence, since that the set $B$ is of cardinality exponentially smaller than $N$, with very high probability for each $i \in B$, $w_i - 1$ is distributed independently and randomly in $\mathbb{Z}_N$. From the fact that $\text{Match}(\vec{x}, \vec{y}) = 0$ it follows that the set $B$ is non-empty and so, from the fact
that the $w_i$'s are distributed randomly and independently in $\mathbb{Z}_N$, it follows that Equation 12 is different from 0 mod $N$ with very high probability. Thus, with very high probability, the product computed by the Test procedure is different from the identity of the target group and so with the same probability it returns FALSE as expected.

\[ \square \]

C  Generic security of our Complexity Assumptions

We now prove that, if factoring is hard, our two complexity assumptions hold in the generic group model. We adopt the framework of [11] to reason about assumptions in bilinear groups $\mathbb{G}, \mathbb{G}_T$ of composite order $N = p_1 p_2 p_3 p_4$. We fix generators $g_{p_1}, g_{p_2}, g_{p_3}, g_{p_4}$ of the subgroups $\mathbb{G}_{p_1}, \mathbb{G}_{p_2}, \mathbb{G}_{p_3}, \mathbb{G}_{p_4}$ and thus each element of $x \in \mathbb{G}$ can be expressed as $x = g_{p_1}^{a_1} g_{p_2}^{a_2} g_{p_3}^{a_3} g_{p_4}^{a_4}$, for $a_i \in \mathbb{Z}_{p_i}$. For sake of ease of notation, we denote element $x \in \mathbb{G}$ by the tuple $(a_1, a_2, a_3, a_4)$.

We do the same with elements in $\mathbb{G}_T$ (with the respect to generator $e(g_{p_1}, g_{p_2})$) and will denote elements in that group as bracketed tuples $[a_1, a_2, a_3, a_4]$. We use capital letters to denote random variables and reuse random variables to denote relationships between elements. For example, $X = (A_1, B_1, C_1, D_1)$ is a random element of $\mathbb{G}$, and $Y = (A_2, B_1, C_2, D_2)$ is another random element that shares the same $\mathbb{G}_{p_2}$ part.

We say that a random variable $X$ is dependent from the random variables $\{A_i\}$ if there exists $\lambda_i \in \mathbb{Z}_N$ such that $X = \sum \lambda_i A_i$ as formal random variables. Otherwise, we say that $X$ is independent of $\{A_i\}$. We state the following theorems from [11].

**Theorem 20** (Theorem A.1 of [11]). Let $N = \prod_{i=1}^m p_i$ be a product of distinct primes, each greater than $2^\lambda$. Let $\{X_i\}$ be random variables over $\mathbb{G}$ and $\{Y_i\}, T_1$ and $T_2$ be random variables over $\mathbb{G}_T$. Denote by $t$ the maximum degree of a random variable and consider the following experiment in the generic group model:

Algorithm $\mathcal{A}$ is given $N, \{X_i\}, \{Y_i\}$ and $T_b$ for random $b \in \{0, 1\}$ and outputs $b' \in \{0, 1\}$. $\mathcal{A}$’s advantage is the absolute value of the difference between the probability that $b = b'$ and 1/2.

Suppose that $T_1$ and $T_2$ are independent of $\{Y_i\} \cup \{e(X_i, X_j)\}$. Then if $\mathcal{A}$ performs at most $q$ group operations and has advantage $\delta$, then there exists an algorithm that outputs a nontrivial factor of $N$ in time polynomial in $\lambda$ and the running time of $\mathcal{A}$ with probability at least $\delta - O(q^2 t/2^\lambda)$.

**Theorem 21** (Theorem A.2 of [11]). Let $N = \prod_{i=1}^m p_i$ be a product of distinct primes, each greater than $2^\lambda$. Let $\{X_i\}, T_1, T_2$ be random variables over $\mathbb{G}$ and let $\{Y_i\}$ be random variables over $\mathbb{G}_T$, where all random variables have degree at most $t$.

Let $N = \prod_{i=1}^m p_i$ be a product of distinct primes, each greater than $2^\lambda$. Let $\{X_i\}, T_1$ and $T_2$ be random variables over $\mathbb{G}$ and let $\{Y_i\}$ be random variables over $\mathbb{G}_T$. Denote by $t$ the maximum degree of a random variable and consider the same experiment as the previous theorem in the generic group model.

Let $S := \{i \mid e(T_1, X_i) \neq e(T_2, X_i)\}$ (where inequality refers to inequality as formal polynomials). Suppose each of $T_1$ and $T_2$ is independent of $\{X_i\}$ and furthermore that for all $k \in S$ it holds that $e(T_1, X_k)$ is independent of $\{B_i\} \cup \{e(X_i, X_j)\} \cup \{e(T_1, X_j)\}_{i \neq k}$ and $e(T_2, X_k)$ is independent of $\{B_i\} \cup \{e(X_i, X_j)\} \cup \{e(T_2, X_i)\}_{i \neq k}$. Then if there exists an algorithm $\mathcal{A}$ issuing at most $q$ instructions and having advantage $\delta$, then there exists an algorithm that outputs a nontrivial factor of $N$ in time polynomial in $\lambda$ and the running time of $\mathcal{A}$ with probability at least $\delta - O(q^2 t/2^\lambda)$.

We apply Theorem 21 to prove the security of our assumptions in the generic group model.
Assumption 1. We can express this assumption as:

\[ X_1 = (0, 0, 1, 0), \quad X_2 = (A_1, 0, A_3, 0), \quad X_3 = (B_1, 0, B_3, 0), \quad X_4 = (0, 0, 0, 1) \]

and

\[ T_1 = (Z_1, 0, Z_3, 0), \quad T_2 = (0, Z_2, Z_3, 0). \]

It is easy to see that \( T_1 \) and \( T_2 \) are both independent of \( \{X_i\} \) because, for example, \( Z_3 \) does not appear in the \( X_i \)’s. Next, we note that for this assumption we have \( S = \{2, 3\} \), and thus, considering \( T_1 \) first, we obtain the following tuples:

\[ C_{1,2} = e(T_1, X_2) = [Z_1 A_1, 0, Z_3 A_3, 0], \quad C_{1,3} = e(T_1, X_3) = [Z_1 B_1, 0, Z_3 B_3, 0]. \]

It is easy to see that \( C_{1,k} \) with \( k \in \{2, 3\} \) is independent of \( \{e(X_i, X_j)\} \cup \{e(T_1, X_i)\}_{i \neq k} \). Analogous arguments apply for the case of \( T_2 \). Thus the independence requirements of Theorem 21 are satisfied and Assumption 1 is generically secure, assuming it is hard to find a nontrivial factor of \( N \).

Assumption 2. We can express this assumption as:

\[ X_1 = (1, 0, 0, 0), \quad X_2 = (0, 1, 0, 0), \quad X_3 = (0, 0, 1, 0), \quad X_4 = (0, 0, 0, 1), \quad X_5 = (A, 0, 0, B_4), \quad X_6 = (B, 0, 0, C_4) \]

and

\[ T_1 = [AB, 0, 0, D_4], \quad T_2 = [Z_1, 0, 0, Z_4]. \]

We note that \( D_4 \) and \( Z_1 \) do not appear in \( \{X_i\} \) and thus \( T_1 \) and \( T_2 \) are both independent from them. Next, we note that for this assumption we have \( S = \{1, 4, 5, 6\} \), and thus, considering \( T_1 \) first, we obtain the following tuples:

\[ C_{1,1} = e(T_1, X_1) = [AB, 0, 0, 0], \quad C_{1,4} = e(T_1, X_4) = [0, 0, 0, D_4] \]
\[ C_{1,5} = e(T_1, X_5) = [A^2 B, 0, 0, D_4 B_4], \quad C_{1,6} = e(T_1, X_6) = [A B^2, 0, 0, D_4 C_4] \]

It is easy to see that \( C_{1,k} \) with \( k \in \{4, 5, 6\} \) is independent of \( \{e(X_i, X_j)\} \cup \{e(T_1, X_i)\}_{i \neq k} \). For \( C_{1,1} \), we observe that the only way to obtain an element whose first component contains \( AB \) is by computing \( e(X_5, X_6) = [AB, 0, 0, B_4 C_4] \) but then there is no way to generate an element whose fourth component is \( B_4 C_4 \) and hence no way to cancel that term.

Analogous arguments apply for the case of \( T_2 \).

Thus the independence requirement of Theorem 21 is satisfied and Assumption 2 is generically secure, assuming it is hard to find a nontrivial factor of \( N \).

D Full-Fledged HVE

A Full-Fledged HVE scheme is a tuple of four efficient probabilistic algorithms \((\text{Setup}, \text{Encrypt}, \text{KeyGen}, \text{Decrypt})\) with the following semantics.

\text{Setup}(\lambda, \ell): \text{ takes as input a security parameter } \lambda \text{ and a length parameter } \ell \text{ (given in unary), and outputs the public parameters } Pk \text{ and the master secret key } Msk.

\text{KeyGen}(Msk, \vec{y}): \text{ takes as input the master secret key } Msk \text{ and a vector } \vec{y} \in \{0, 1, \star\}^\ell \text{, and outputs a secret key } Sk_{\vec{y}}.
 Encrypt(Pk, \vec{x}, M): takes as input the public parameters Pk and a vector \vec{x} \in \{0, 1\}^\ell, and a message M in some associated message space. It outputs a ciphertext Ct.

Decrypt(Pk, Ct, Sk_y): takes as input the public parameters Pk, a ciphertext Ct encrypting attribute vector \vec{x} and message M, and a secret key Sk_y. It outputs plaintext M'.

Correctness. For correctness we require that for pairs (Pk, Msk) output by Setup(1^\lambda, 1^\ell), it holds that for any vectors \vec{x} \in \{0, 1\}^\ell and for any plaintext M and for any \vec{y} \in \{0, 1, \ast\}^\ell, we have that

\text{Decrypt}(Pk, \text{Encrypt}(Pk, \vec{x}, M), \text{KeyGen}(Msk, \vec{y})) = M

is negligibly in \lambda close to 1 if Match(\vec{x}, \vec{y}) = 1 and negligibly in \lambda close to 0 if Match(\vec{x}, \vec{y}) = 0.

D.1 Security definition

We define security by means of the following game GReal played between an adversary \mathcal{A} and a challenger \mathcal{C}.

Setup. \mathcal{C} runs the Setup algorithm on input the security parameter \lambda and the length parameter \ell (given in unary) to generate public parameters Pk and master secret key Msk. \mathcal{C} starts the interaction with \mathcal{A} on input Pk.

Key Query Answering. Upon receiving a query for vector \vec{y}, \mathcal{C} returns KeyGen(Msk, \vec{y}).

Challenge Construction. Upon receiving the pair ((\vec{x}_0, M_0), (\vec{x}_1, M_1)), \mathcal{C} picks random \eta \in \{0, 1\} and returns Encrypt(Pk, \vec{x}_\eta, M_\eta).

Winning Condition. Let \eta' be \mathcal{A}'s output. We say that \mathcal{A} wins the game if \eta = \eta' and for all \vec{y} for which \mathcal{A} has issued a Key Query, it holds Match(\vec{x}_0, \vec{y}) = Match(\vec{x}_1, \vec{y}). Moreover, if \mathcal{A} has issued a Key Query for a vector \vec{y} such that Match(\vec{x}_0, \vec{y}) = Match(\vec{x}_1, \vec{y}) = 1, then M_0 must be equal to M_1.

We define the advantage Adv^A(\lambda) of \mathcal{A} in GReal as the probability of winning minus 1/2 and for security we require the advantage to be negligible.

D.2 Full-fledged scheme for HVE

It is easy to extend our scheme for HVE to the full-fledged case in the following way. In the schemes for Hidden Vector Encryption we add the value \Omega = e(g_1, g_1)^z for a random z to the public key and add z to the master secret key. In constructing the secret keys, we choose that the \alpha_i's so that they sum up to z (instead of summing up to 0). In the encryption for a message M \in \mathbb{G}_T, we add the element \Omega = M \cdot \Omega^s, where s is the same random values used to compute the other components of the ciphertext. Then it is easy to see that the blinding factor \Omega^z can be obtained from the keys and the ciphertext. For completeness, we present the formal description of the full-fledged scheme. The associated message space of the following full-fledged scheme is the target group \mathbb{G}_T. As for the predicate-only scheme of Section 4, we assume that all vectors associated with the key have at least two non-star positions.

Setup(1^\lambda, 1^\ell): The setup algorithm chooses a description of a bilinear group \mathcal{I} = (N = p_1p_2p_3p_4, \mathbb{G}, \mathbb{G}_T, e) with known factorization by running a generator algorithm \mathcal{G} on input 1^\lambda. The setup algorithm chooses random \vec{z} \in \mathbb{Z}_N, g_1 \in \mathbb{G}_{p_1}, g_2 \in \mathbb{G}_{p_2}, g_3 \in \mathbb{G}_{p_3}, g_4 \in \mathbb{G}_{p_4}, and, for i \in [\ell] and b \in \{0, 1\}, random \vec{t}_{i,b} \in \mathbb{Z}_N and random \vec{R}_{i,b} \in \mathbb{G}_{p_3} and sets \vec{T}_{i,b} = g_1^{\vec{t}_{i,b}} \cdot \vec{R}_{i,b}. Furthermore, it sets
\[\Omega = e(g_1, g_1)^z.\] The public parameters are \(\text{Pk} = [N, g_3, \Omega, (T_i, b)_{i \in [\ell], b \in \{0, 1\}}]\) and the master secret key is \(\text{Msk} = [g_{12}, g_4, (t_i, b)_{i \in [\ell], b \in \{0, 1\}}]\), where \(g_{12} = g_1 \cdot g_2.\)

**KeyGen(Msk, \bar{y}):** Let \(S_{\bar{y}}\) be the set of indices \(i\) such that \(y_i \neq \ast\). The key generation algorithm chooses random \(a_i \in \mathbb{Z}_N\) for \(i \in S_{\bar{y}}\) under the constraint that \(\sum_{i \in S_{\bar{y}}} a_i = z\). For \(i \in S_{\bar{y}}\), the algorithm chooses random \(W_i \in G_{p_4}\) and sets \(Y_i = g_{a_i/t_i, y_i}^{a_i} \cdot W_i\).

The algorithm returns the tuple \((Y_i)_{i \in S_{\bar{y}}}\). Here we use the fact that \(S_{\bar{y}}\) has size at least 2.

**Encrypt(Pk, \bar{x}, M):** The encryption algorithm chooses random \(s \in \mathbb{Z}_N\). For \(i \in [\ell]\), the algorithm chooses random \(Z_i \in G_{p_3}\) and sets \(X_i = T_{i, x_i}^s \cdot Z_i\). Furthermore, it computes \(W = M \cdot \Omega^{-s}\) and returns the tuple \((W, (X_i)_{i \in [\ell]})\).

**Decrypt(Ct, Sk_{\bar{y}}):** The decryption algorithm computes \(T = \prod_{i \in S_{\bar{y}}} e(X_i, Y_i)\). It returns \(M' = W \cdot T\).

By adding a MAC (or some parity check code) to the plaintext \(M\), it is possible to detect invalid decryptions. We omit the proof of correctness and security for this scheme since they are analogous to those for the predicate-only scheme of Section 4.

---

### E Reductions

In this section we show how to construct an encryption scheme for the class of Boolean predicates that can be expressed as a \(k\)-CNF or \(k\)-DNF formula and disjunctions from an HVE scheme.

We first start by giving formal definitions for the Boolean Satisfaction Problem and its security properties.

#### E.1 Boolean Satisfaction Encryption

Let \(\mathbb{B} = \{\mathbb{B}_n\}_{n \geq 0}\) be a class of Boolean predicates indexed by the number \(n\) of variables. We define the Satisfy predicate as \(\text{Satisfy}(\Phi, \bar{z}) = \Phi(\bar{z})\) for \(\bar{z} \in \{0, 1\}^n\).

An *Encryption scheme for class \(\mathbb{B}\)* is a tuple of four efficient probabilistic algorithms \((\text{Setup}, \text{Encrypt}, \text{KeyGen}, \text{Test})\) with the following semantics.

- **Setup(1^\lambda, 1^n):** takes as input a security parameter \(\lambda\) and the number \(n\) of variables, and outputs the public parameters \(\text{Pk}\) and the master secret key \(\text{Msk}\).

- **KeyGen(Msk, \Phi):** takes as input the master secret key \(\text{Msk}\) and a formula \(\Phi \in \mathbb{B}_n\) and outputs a secret key \(\text{Sk}_\Phi\).

- **Encrypt(Pk, \bar{z}):** takes as input the public parameters \(\text{Pk}\) and a truth assignment \(\bar{z}\) for \(n\) variables and outputs a ciphertext \(\text{Ct}\).

- **Test(Pk, Ct, Sk_\Phi):** takes as input the public parameters \(\text{Pk}\), a ciphertext \(\text{Ct}\) and a secret key \(\text{Sk}_\Phi\) and outputs \text{TRUE} iff and only if the ciphertext is an encryption of a truth assignment \(\bar{z}\) that satisfies \(\Phi\).
**Correctness of Boolean Satisfaction Encryption.** We require that for all pairs (Pk, Msk) ← Setup(1^λ, 1^n), it holds that for any truth assignment \( \vec{z} \) for \( n \) variables, for any formula \( \Phi \in \mathbb{B}_n \) over \( n \) variables we have that the probability that \( \text{Test}(Pk, \text{Encrypt}(Pk, \vec{z}), \text{KeyGen}(Msk, \Phi)) \neq \text{Satisfy}(\Phi, \vec{z}) \) is negligible in \( \lambda \).

### E.2 Security definitions for Boolean Satisfaction Encryption

For Boolean Satisfaction encryption, we have a game similar to that of HVE. GReal can be described in the following way.

**Setup.** \( C \) runs the Setup algorithm, (Pk, Msk) ← Setup(1^λ, 1^n). Then \( C \) starts the interaction with \( A \) on input Pk.

**Key Query Answering.** For \( \Phi \in \mathbb{B}_n \), \( C \) returns KeyGen(Msk, \Phi).

**Challenge Construction.** Upon receiving the pair (\( \vec{z}_0, \vec{z}_1 \)) of truth assignments over \( n \) variables, \( C \) picks random \( \eta \in \{0, 1\} \) and returns Encrypt(Pk, \( \vec{z}_\eta \)).

**Winning Condition.** Let \( \eta' \) be \( A \)'s output. We say that \( A \) wins the game if \( \eta = \eta' \) and, for all \( \Phi \) for which \( A \) has issued a Key Query, it holds that \( \text{Satisfy}(\Phi, \vec{z}_0) = \text{Satisfy}(\Phi, \vec{z}_1) \).

We define the advantage \( \text{Adv}_A^A(\lambda) \) of \( A \) in GReal to be the probability of winning minus 1/2.

**Definition 22.** An Encryption scheme for class \( \mathbb{B} \) is secure if for all PPT adversaries \( A \), we have that \( \text{Adv}_B^A(\lambda) \) is a negligible function of \( \lambda \).

### E.3 Reducing \( k \)-CNF to HVE

We consider formulae \( \Phi \) in \( k \)-CNF, for constant \( k \), over \( n \) variables in which each clause \( C \in \Phi \) contains exactly \( k \) distinct variables. We call such a clause **admissible** and denote by \( C_n \) the set of all admissible clauses over the \( n \) variables \( x_1, \ldots, x_n \) and set \( M_n = |C| \). Notice that \( M_n = \Theta(n^k) \).

We also fix a canonical ordering \( C_1, \ldots, C_{M_n} \) of the clauses in \( C_n \).

Let \( \mathcal{H} = (\text{Setup}_{\mathcal{H}}, \text{KeyGen}_{\mathcal{H}}, \text{Encrypt}_{\mathcal{H}}, \text{Test}_{\mathcal{H}}) \) be an HVE scheme and construct a \( k \)-CNF scheme \( k\text{CNF} = (\text{Setup}_{k\text{CNF}}, \text{KeyGen}_{k\text{CNF}}, \text{Encrypt}_{k\text{CNF}}, \text{Test}_{k\text{CNF}}) \) as follows:

**Setup_{kCNF}(1^λ, 1^n):** The algorithm returns the output of Setup_{\mathcal{H}}(1^λ, 1^{M_n}).

**KeyGen_{kCNF}(Msk, \Phi):** For a \( k \)-CNF formula \( \Phi \), the key generation algorithm constructs vector \( \vec{y} \in \{0, 1, *\}^{M_n} \) by setting, for each \( i \in \{1, \ldots, M_n\} \), \( y_i = 1 \) if \( C_i \in \Phi \); \( y_i = * \) otherwise. We denote this transformation by \( y = \text{FEncode}(\Phi) \). Then the key generation algorithm returns \( \text{Sk}_\Phi = \text{KeyGen}_{\mathcal{H}}(\text{Msk}, \vec{y}) \).

**Encrypt_{kCNF}(Pk, \vec{z}):** The algorithm constructs vector \( \vec{x} \in \{0, 1\}^{M_n} \) in the following way: For each \( i \in \{1, \ldots, M_n\} \) the algorithms sets \( x_i = 1 \) if \( C_i \) is satisfied by \( \vec{z} \); \( x_i = 0 \) if \( C_i \) is not satisfied by \( \vec{z} \). We denote this transformation by \( \vec{x} = \text{AEncode}(\vec{z}) \). Then the encryption algorithm returns \( \text{Ct} = \text{Encrypt}_{\mathcal{H}}(Pk, \vec{x}) \).

**Test_{kCNF}(Sk_\Phi, \text{Ct}):** The algorithm returns the output of Test_{\mathcal{H}}(Sk_\Phi, \text{Ct}).

**Correctness.** Correctness follows from the observation that for formula \( \Phi \) and assignment \( \vec{z} \), we have that \( \text{Match}(\text{AEncode}(\vec{z}), \text{FEncode}(\Phi)) = 1 \) if and only if \( \text{Satisfy}(\Phi, \vec{z}) = 1 \).

**Security.** Let \( A \) be an adversary for \( k\text{CNF} \) that tries to break the scheme for \( n \) variables and consider the following adversary \( B \) for \( \mathcal{H} \) that uses \( A \) as a subroutine and tries to break a \( \mathcal{H} \) with
\[ \ell = M_a \] by interacting with challenger \( C \). \( B \) receives a \( \mathcal{P}k \) for \( \mathcal{H} \) and passes it to \( \mathcal{A} \). Whenever \( \mathcal{A} \) asks for the key for formula \( \Phi \), \( B \) constructs \( \vec{y} = \mathcal{F}\text{Encode}(\Phi) \) and asks \( C \) for a key \( \mathcal{S}k_{\vec{y}} \) for \( \vec{y} \) and returns it to \( \mathcal{A} \). When \( \mathcal{A} \) asks for a challenge by providing truth assignments \( \vec{z}_0 \) and \( \vec{z}_1 \), \( B \) simply computes \( \vec{x}_0 = \mathcal{A}\text{Encode}(\vec{z}_0) \) and \( \vec{x}_1 = \mathcal{A}\text{Encode}(\vec{z}_1) \) and gives the pair \( (\vec{x}_0, \vec{x}_1) \) to \( C \). \( B \) then returns the challenge ciphertext obtained from \( C \) to \( \mathcal{A} \). Finally, \( B \) outputs \( \mathcal{A}'s \) guess.

First, \( B \)’s simulation is perfect. Indeed, we have that if for all \( \mathcal{A}'s \) queries \( \Phi \) we have that \( \text{Satisfy}(\Phi, \vec{z}_0) = \text{Satisfy}(\Phi, \vec{z}_1) \), then all \( B \)’s queries \( \vec{y} \) to \( C \) also have the property \( \text{Match}(\vec{y}, \vec{x}_0) = \text{Match}(\vec{y}, \vec{x}_1) \). Thus \( B \)’s advantage is the same as \( \mathcal{A}'s \). By combining the above reduction with our constructions for HVE, we have the following theorems.

**Theorem 23.** For any constant \( k > 0 \), if Assumption 1 and 2 hold for generator \( \mathcal{G} \) then there exists a secure encryption scheme for the class of predicates that can be represented by \( k \)-CNF formulae.

### E.4 Reducing Disjunctions to HVE

In this section we consider the class of Boolean predicates that can be expressed as a single disjunction. We assume without loss of generality that a disjunction does not contain a variable and its negated.

Let \( \mathcal{H} = (\text{Setup}_\mathcal{H}, \text{KeyGen}_\mathcal{H}, \text{Encrypt}_\mathcal{H}, \text{Test}_\mathcal{H}) \) be an HVE scheme and construct the predicate-only scheme \( \vee = (\text{Setup}_\vee, \text{KeyGen}_\vee, \text{Encrypt}_\vee, \text{Test}_\vee) \) for disjunctions in the following way:

- **Setup\(_\vee\)\((\lambda^1, \lambda^n)\):** the algorithm returns the output of \( \text{Setup}_\mathcal{H}(\lambda^1, \lambda^n) \).

- **KeyGen\(_\vee\)(Msk, C):** For a clause \( C \), the key generation algorithm constructs vector \( \vec{y} \in \{0, 1, *\}^n \) in the following way. Let \( \vec{w} \) be a truth assignment to the \( n \) variables that does not satisfy clause \( C \). For each \( i \in \{1, \ldots, n\} \), the algorithms sets \( y_i = w_i \) if the \( i \)th variable appears in \( C \); \( y_i = * \) otherwise. We denote this transformation by \( \vec{y} = \mathcal{C}\text{Encode}(C) \). The output is \( \mathcal{S}k_C = \text{KeyGen}_\mathcal{H}(\text{Msk}, \vec{y}) \).

- **Encrypt\(_\vee\)(Pk, \vec{z}):** The encryption algorithm returns \( \mathcal{Ct} = \text{Encrypt}_\mathcal{H}(\text{Pk}, \vec{z}) \).

- **Test\(_\vee\)(Sk\(_C\), Ct):** The algorithm returns \( 1 - \text{Test}_\mathcal{H}(\text{Sk}_C, \mathcal{Ct}) \).

**Correctness.** It follows from the observation that for a clause \( C \) and assignment \( \vec{z} \), \( \text{Satisfy}(C, \vec{z}) = 1 \) if and only if \( \text{Match}(\mathcal{C}\text{Encode}(C), \vec{z}) = 0 \).

**Security.** It is easy to see that if \( \mathcal{H} \) is secure then \( \vee \) is secure. In particular, notice that if for \( \mathcal{A}'s \) query \( C \) we have that \( \text{Satisfy}(C, \vec{z}_0) = \text{Satisfy}(C, \vec{z}_1) = \xi \in \{0, 1\} \), then for \( \mathcal{B}'s \) query \( \vec{y} = \mathcal{C}\text{Encode}(C) \) to \( C \) we have that \( \text{Match}(\vec{y}, \vec{z}_0) = \text{Match}(\vec{y}, \vec{z}_1) = 1 - \xi \).

**Theorem 24.** If Assumption 1 and 2 hold for generator \( \mathcal{G} \) then there exists a secure encryption scheme for the class of predicates that can be represented by a disjunction.

### E.5 Reducing \( k \)-DNF to \( k \)-CNF

We observe that if \( \Phi \) is a predicate represented by a \( k \)-DNF formula then its negation \( \bar{\Phi} \) can be represented by a \( k \)-CNF formula. Therefore let \( \mathcal{kCNF} = (\text{Setup}_{\mathcal{kCNF}}, \text{KeyGen}_{\mathcal{kCNF}}, \text{Encrypt}_{\mathcal{kCNF}}, \text{Test}_{\mathcal{kCNF}}) \) and consider the following scheme \( \mathcal{kDNF} = (\text{Setup}_{\mathcal{kDNF}}, \text{KeyGen}_{\mathcal{kDNF}}, \text{Encrypt}_{\mathcal{kDNF}}, \text{Test}_{\mathcal{kDNF}}) \). The setup algorithm \( \text{Setup}_{\mathcal{kDNF}} \) is the same as \( \text{Setup}_{\mathcal{kCNF}} \). The key generation algorithm \( \text{Setup}_{\mathcal{kDNF}} \) for predicate \( \Phi \) represented by a \( k \)-DNF simply invokes the key generation algorithm \( \text{Setup}_{\mathcal{kCNF}} \) for \( \Phi \)
that can be represented by a $k$-CNF formula. The encryption algorithm $\text{Encrypt}_{k\text{DNF}}$ is the same as $\text{Encrypt}_{k\text{CNF}}$. The test algorithm $\text{Test}_{k\text{DNF}}$ on input ciphertext $Ct$ and key for $k$-DNF formula $\Phi$ (that is actually a for $k$-CNF formula $\Phi$) thus $\text{Test}_{k\text{CNF}}$ on $Ct$ and the key and complements the result. Correctness and security can be easily argued as for Disjunctions.

By combining the above reduction with the construction given by Theorem 23.

**Theorem 25.** If Assumption 1 and 2 hold for generator $G$ then there exists a secure encryption scheme for the class of predicates represented by $k$-DNF formulae.