Efficient Fully Secure (Hierarchical) Predicate Encryption for Conjunctions, Disjunctions and k-CNF/DNF formulae

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.

November 11, 2010

Abstract

Predicate encryption is an important cryptographic primitive that has found wide applications as it allows for fine-grained key management. In a predicate encryption scheme for a class $\mathbb{P}$ of predicates, the owner of the master secret key can derive a secret key Sk_P for any predicate $P \in \mathbb{P}$. Similarly, when encrypting plaintext M, the sender can specify an attribute vector $\vec{x}$ for the ciphertext Ct. Then, key Sk_P can decrypt all ciphertexts Ct with attribute vector $\vec{x}$ such that $P(\vec{x}) = 1$.

In this paper, we give *fully secure* implementations for Conjunctions (also called Hidden Vector Encryption in the literature), Disjunctions and k-CNF/DNF predicates that guarantee the security of the plaintext *and* of the attribute. Our constructions for Disjunctions and Conjunctions are linear in the number of variables. Previous fully secure constructions for Disjunction required time exponential in the number of variables while for Conjunctions the best previous construction was quadratic in the number of variables. We also give reduction of k-CNF and k-DNF formulae to Conjunctions for any constant k.

In proving the full security of our constructions, we elaborate on the recent paradigm of *dual encryption system* introduced in [Waters – Crypto 2009] resulting in a simplified proof strategy that could be of independent interest.

We also give a hierarchical version of our scheme that has as special case Anonymous HIBE.

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

*Work done while visiting the Department of Computer Science of The Johns Hopkins University.

Contents

1 Introduction and related work

Predicate encryption is an important cryptographic primitive that has been recently studied [2, 4, 5, 7] and that has found wide applications as it allows for fine-grained key management. Roughly speaking, in a predicate encryption scheme for predicate Q the owner of the master secret key Msk can derive secret key Sk_Q, for any predicate Q for a specified class of predicates. In encrypting plaintext M, the sender can specify an *attribute* vector $\vec{x}$ and the resulting ciphertext Ct can be decrypted only by using keys Sk_Q such that $Q(\vec{x}) = 1$. A predicate encryption scheme thus gives the owner of the master secret key control on which ciphertexts can be decrypted and this allows her to delegate the decryption of different types of messages (as specified by the attribute vector) to different entities. Several constructions for specific predicates have been given, starting from the *equality* predicate of [2], to the *hidden vector* predicate of [4] and to the *inner product* predicate of [7]. A construction for the larger class of monotone formulae have been by [5] and then extended to formulae by [12] (see also [8, 11]). These constructions guaranteed the security of *only* the plaintext encrypted by a ciphertext Ct and did not give any security guarantee for the attribute vector $\vec{x}$. This extra security property is very important for the applications and was guaranteed, in the *selective* model, by the constructions for specific predicates of [4, 7]. The selective model restricted the adversary to declare its challenges before seeing the public key and issuing any query. Following the recent breakthrough of [16, 9] that gave fully secure implementation of Identity Based Encryption (and of its hierarchical version), Lewko et al. [8] gave fully secure implementation for the inner product predicate.

Our results. In this paper, we concentrate on *fully secure* implementations of encryption schemes for conjunctions and disjunctions and their applications. For conjunctions we adhere to the standard terminology of *hidden vector encryption* (or HVE in short) as introduced by [4].

In a HVE scheme, the ciphertext attributes are vectors $\vec{x} = \langle x_1, \ldots, x_\ell \rangle$ over alphabet $\{0, 1\}$, keys are associated with vectors $\vec{y} = \langle y_1, \ldots, y_\ell \rangle$ over alphabet $\{0, 1, \star\}$ and we consider the $\mathsf{Match}(\vec{x}, \vec{y})$ predicate defined to be true if and only if, for all i, $y_i \neq \star$ implies $x_i = y_i$. We distinguish two security notions, that we call 0-security and 1-security (see Section 2) differing in the type of keys that the adversary can ask for. Roughly speaking, ξ-security considers adversaries that can ask keys for vectors $\vec{y}$ such that $\mathsf{Match}(\vec{x}_0, \vec{y}) = \mathsf{Match}(\vec{x}_1, \vec{y}) = \xi$ where $\vec{x}_0$ and $\vec{x}_1$ are the two challenge vectors chosen by the adversary. We observe that no scheme can be secure with respect to an adversary that has a key for a vector $\vec{y}$ such that $\mathsf{Match}(\vec{x}_0, \vec{y}) \neq \mathsf{Match}(\vec{x}_1, \vec{y})$. The notion of 0-security is known in the literature as *match revealing* security and all previous fully secure constructions are 0-secure.

We give a full secure implementation of HVE for *both* notions of security. Our secure implementations of HVE are proved fully secure under non-interactive constant sized (that is, independent of ℓ and of the running time of the adversary) assumptions on bilinear groups of composite order. We note that in our implementations the master secret key and the ciphertexts consist of ℓ group elements whereas a key for vector $\vec{y}$ has one group element for each component $y_i \neq \star$. Our encryption and key generation procedure are also efficient as they both require $O(\ell)$ group operations and no bilinear mapping evaluation. We stress that our 0-secure construction is more efficient than previous ones and that our 1-secure construction is the first to be proved 1-secure. We prove 1-security by means of a *tight* security reduction; that is, the security proof does not depend on the running time of the adversary. As we shall show below, the notion of 1-security, besides being of its own interest, is instrumental in obtaining 0-security for 3-DNF and Disjunctions.

We can extend our constructions to a general alphabet Σ, at the cost of expanding the public and master secret key by a factor proportional to $|\Sigma|$, which can be considered to be constant for most applications. The size of ciphertexts and secret keys does not depend on Σ. Finally, we stress that for our applications (see below) a binary alphabet suffices.

We then show polynomial deterministic reductions to HVE for any predicate represented by a formula in 3-CNF, or by a formula in 3-DNF, or by a disjunction. For example, we show that any formula Φ in 3-CNF with m clauses over n variables can be associated with a vector $\vec{y} \in \{0,1,\star\}^\ell$ with m non-$\star$ entries for $\ell = O(n^3)$. Moreover, we show how to encode a truth assignment z for n variables with a vector $\vec{x} \in \{0,1\}^\ell$ such that $\mathsf{Match}(\vec{x},\vec{y}) = 1$ if and only if the assignment encoded by $\vec{x}$ satisfies the formula encoded by $\vec{y}$.

We prove that our reduction for 3-CNF preserves full ξ-security and thus we can apply it to our construction of the HVE resulting in a scheme in which the key for a formula of m clauses contains exactly m group elements. In addition we prove that our reductions for 3-DNF and disjunctions complements security in the sense that if we apply our reductions to a ξ-secure scheme for HVE we obtain a $(1-\xi)$-secure scheme for 3-DNF (or for disjunctions). Finally, we mention that our reduction for disjunction is linear.

For sake of concreteness, we have chosen to present our results for 3-CNF and 3-DNF. It can be easily seen that they can be extended to k-CNF and k-DNF for any constant k.

Finally, we give a construction of Hierarchical HVE in which the holder of $\mathsf{Sk}_{\vec{y}}$ (the key for vector $\vec{y}$) can create (and give to a third party) the key for any vector $\vec{w}$ that is obtained by instantiating some of the $\star$ entries of $\vec{y}$ to 0 or 1. Also, our construction of HHVE is fully secure under non-interactive, constant sized (that is independent of ℓ and the running time of the adversary) assumptions on bilinear groups of composite order. In addition, our construction for HHVE results in keys and ciphertexts consisting of $O(\ell)$ group elements independently on the length of the delegation path. Our HHVE implies fully secure Anonymous HIBE.

Our proof strategy for full security. We achieve full security by means of a proof strategy that elaborates on the dual encryption methodology [16] and, in our opinion, results in a simplified proof and we expect there be further applications of our strategy. Let us now briefly review our strategy. The first step in our proof strategy, to achieve ξ-security, consists in projecting the public key to a different subspace in such a way to make it independent from the challenge ciphertext. Here, it is possible to view a similarity with the strategy of Peikert and Waters [13] but unlike their approach, in our case the change in the public key does not affect the distribution of the challenge ciphertext which is still created like in the real game. Then the proof proceeds to show that the secret keys do not help the adversary and, for this, our strategy forks depending on the value of ξ.

For $\xi = 0$, we do so by proving that, in the view of the adversary, the valid secret keys are indistinguishable from keys that are random in the subgroup in which the plaintext is embedded. More precisely, keys continue to be valid in the subgroup where the public key was projected, and are random in the other subgroups.

On the other hand, for $\xi = 1$ we show that the secret keys do not help the adversary by proving that, in the view of the adversary, the valid secret keys are indistinguishable from keys which do not have a component in the subgroup in which the plaintext is embedded.

Following this strategy we handle the concept of nominally semi-functional algorithms in a way that results in a simplified proof.

Related Work. An implementation of fully secure HVE can be derived from the fully secure construction of inner product of Lewko et al. [8] using the reduction of Katz et al. [7]. We point out though that the Inner Product construction of [8] has a master key of quadratic size and the key generation and the encryption algorithm suffer of an extra quadratic slowdown in the time complexity when compared to ours. Also, we notice that [8] and [7] only considered 0-security. Similar considerations can be made for the recent construction of Inner Product of [11].

We mention that Katz et al. [7] have presented a reduction of CNF to inner product that is polynomial (actually, cubic) when applied to 3-CNF formulae. By composing this reduction with the one from inner product to HVE gives a reduction that can be applied to our HVE implementation. The resulting scheme for 3-CNF still has a quadratic slowdown when compared to ours. Finally, for disjunctions, we mention that [7] have a construction that is 1-secure and exponential in the number of variables. In contrast, our 0-secure and 1-secure constructions are linear in the number of variables.

2 Hidden Vector Encryption and Boolean Satisfaction Encryption

In this section we give formal definitions for Hidden Vector Encryption (HVE) and for Boolean Satisfaction Problem and their security properties.

Following [14], for sake of simplicity we present predicate-only definitions and schemes for (H)HVE instead of full-fledged ones (see [7]). In Appendix A we will briefly discuss how to extend our schemes to the full-fledged version. Also, for (H)HVE we present our construction for the binary alphabet, but as outlined in [6], it is possible to modify the construction for larger alphabets without a penalty in the length of the key or the ciphertext.

2.1 Hidden Vector Encryption

Let $\vec{x}$ be vector of length ℓ over the alphabet $\{0, 1\}$ and $\vec{y}$ vector of the same length over the alphabet $\{0, 1, \star\}$. Define the predicate $\mathsf{Match}(\vec{x}, \vec{y}) = \text{TRUE}$ 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 only in the positions j where $y_j \neq \star$. This predicate is called Hidden Vector Encryption (henceforth, abbreviated in HVE) and was introduced in [4]. This predicate has like very special case Anonymous IBE but it has many other applications. For a full account of the applications, see [4].

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

$\mathsf{Setup}(1^\lambda, 1^\ell)$: takes as input a security parameter λ and a length parameter ℓ (given in unary), and outputs the public parameters Pk and the master secret key Msk.

$\mathsf{KeyGen}(\mathsf{Msk}, \vec{y})$: takes as input the master secret key Msk and a vector $\vec{y} \in \{0, 1, \star\}^\ell$, and outputs a secret key $\mathsf{Sk}_{\vec{y}}$.

$\mathsf{Encrypt}(\mathsf{Pk}, \vec{x})$: takes as input the public parameters Pk and a vector $\vec{x} \in \{0, 1\}^\ell$ and outputs a ciphertext Ct.

$\mathsf{Test}(\mathsf{Pk}, \mathsf{Ct}, \mathsf{Sk}_{\vec{y}})$: takes as input the public parameters Pk, a ciphertext Ct encrypting $\vec{x}$ and a secret key $\mathsf{Sk}_{\vec{y}}$ and outputs $\mathsf{Match}(\vec{x}, \vec{y})$.

Correctness of HVE. For correctness we require that for all pairs $(\mathsf{Pk}, \mathsf{Msk})$ output by $\mathsf{Setup}(1^\lambda, 1^\ell)$, it holds that for vectors $\vec{x} \in \{0, 1\}^\ell$ and $\vec{y} \in \{0, 1, \star\}^\ell$, we have that

$$\mathsf{Test}(\mathsf{Pk}, \mathsf{Encrypt}(\mathsf{Pk}, \vec{x}), \mathsf{KeyGen}(\mathsf{Msk}, \vec{y})) = \mathsf{Match}(\vec{x}, \vec{y})$$

except with negligible in λ probability.

2.2 Boolean Satisfaction Encryption

Let $\mathbb{B} = \{\mathbb{B}_n\}_{n>0}$ be a class of Boolean predicates indexed by the number n of variables. We encode truth assignment to n variables by means of length n Boolean vectors $\vec{z}$ and, for $\Phi \in \mathbb{B}_n$, we define the $\mathsf{Satisfy}$ predicate as $\mathsf{Satisfy}(\Phi, \vec{z}) = \Phi(\vec{z})$.

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

$\mathsf{Setup}(1^\lambda, 1^n)$: takes as input a security parameter λ and the number n of variables (given in unary), and outputs the public parameters Pk and the master secret key Msk.

$\mathsf{KeyGen}(\mathsf{Msk}, \Phi)$: takes as input the master secret key Msk and a formula $\Phi \in \mathbb{B}_n$ and outputs a secret key Sk_Φ.

$\mathsf{Encrypt}(\mathsf{Pk}, \vec{z})$: takes as input the public parameters Pk and a truth assignment $\vec{z}$ for n variables and outputs a ciphertext Ct.

$\mathsf{Test}(\mathsf{Pk}, \mathsf{Ct}, \mathsf{Sk}_\Phi)$: takes as input the public parameters Pk, a ciphertext Ct and a secret key Sk_Φ and outputs TRUE iff and only if the ciphertext is an encryption of a truth assignment $\vec{z}$ that satisfies Φ.

We remark that an HVE scheme is simply an encryption scheme for the class of Boolean predicates represented by a conjunction (or a 1-CNF formula). For sake of consistency of terminology with previous literature we keep the name HVE for this class of predicates.

Correctness of Boolean Satisfaction Encryption. For correctness we require that for all pairs $(\mathsf{Pk}, \mathsf{Msk})$ output by $\mathsf{Setup}(1^\lambda, 1^n)$, it holds that for any truth assignment $\vec{z}$ for n variables, for any formula $\Phi \in \mathcal{B}_n$ over n variables we have that the probability that

$$\mathsf{Test}(\mathsf{Pk}, \mathsf{Encrypt}(\mathsf{Pk}, \vec{z}), \mathsf{KeyGen}(\mathsf{Msk}, \Phi)) \neq \mathsf{Satisfy}(\Phi, \vec{z})$$

is negligible in λ.

2.3 Security definitions for HVE

We give two security notions depending on the type of queries $\mathcal{A}$ is allowed to ask. We formalize the two notions by means of security games $\mathsf{Game}_{\mathsf{Real}}(\xi)$, with $\xi \in \{0,1\}$, between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$. $\mathsf{Game}_{\mathsf{Real}}(\xi)$ consists of a Setup phase and of a Query Answering phase. In the Query Answering phase, the adversary can issue any number of Key Queries and one Challenge Construction query and at the end of this phase $\mathcal{A}$ outputs a guess. We stress that key queries can be issued by $\mathcal{A}$ even after he has received the challenge from $\mathcal{C}$. In $\mathsf{Game}_{\mathsf{Real}}(\xi)$ the adversary is restricted to queries for vectors $\vec{y}$ such that $\mathsf{Match}(\vec{y}, x_0) = \mathsf{Match}(\vec{y}, x_1) = \xi$.

More precisely, for $\xi \in \{0,1\}$, we define game $\mathsf{Game}_{\mathsf{Real}}(\xi)$ in the following way.

Setup. $\mathcal{C}$ 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. $\mathcal{C}$ starts the interaction with $\mathcal{A}$ on input Pk.

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

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

Winning Condition. Let η' 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 $\mathsf{Match}(\vec{x}_0, \vec{y}) = \mathsf{Match}(\vec{x}_1, \vec{y}) = \xi$.

We call such an adversary $\mathcal{A}$ a ξ-adversary and define its advantage $\mathsf{Adv}^{\mathcal{A},\xi}_{\mathsf{HVE}}(\lambda)$ in $\mathsf{Game}_{\mathsf{Real}}(\xi)$ to be the probability of winning minus $1/2$.

Definition 2.1 *An Hidden Vector Encryption scheme is* ξ-secure *if for all probabilistic polynomial time ξ-adversaries $\mathcal{A}$, we have that $\mathsf{Adv}^{\mathcal{A},\xi}_{\mathsf{HVE}}(\lambda)$ is a negligible function of λ.*

It is trivial to observe that no scheme can be secure if the adversary is allowed to receive a secret key that discriminates between the two challenges. For example, no HVE scheme is secure if the adversary has a key for $\vec{y}$ such that $\mathsf{Match}(\vec{y}, \vec{x}_0) = 0$ and $\mathsf{Match}(\vec{y}, \vec{x}_1) = 1$.

In our security definitions, though, we add the extra constraint that each adversary either requests keys which match *both* challenges (this is the case $\xi = 1$) or keys which match *neither* challenges (this is the case $\xi = 0$).

We share this limitation on the security model with [8] (which considered only the case of non-matching queries; that is, $\xi = 0$). To the best of our knowledge, it is an open problem to design a scheme that is secure without this extra constraint.

2.4 Security Definitions for Boolean Satisfaction Encryption

For Boolean Satisfaction encryption, we have similar games $\mathsf{Game}_{\mathsf{Real}}(\xi)$ that can be described in the following way.

Setup. $\mathcal{C}$ runs the Setup algorithm on input the security parameter λ and the number n of variables (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 $\Phi \in \mathbb{B}_n$, $\mathcal{C}$ returns $\mathsf{KeyGen}(\mathsf{Msk}, \Phi)$.

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

Winning Condition. Let η' be $\mathcal{A}$'s output. We say that $\mathcal{A}$ *wins* the game if $\eta = \eta'$ and, for all Φ for which $\mathcal{A}$ has issued a Key Query, it holds that $\mathsf{Satisfy}(\Phi, z_0) = \mathsf{Satisfy}(\Phi, z_1) = \xi$.

We call such an adversary $\mathcal{A}$ a ξ-adversary and define its advantage $\mathsf{Adv}_{\mathbb{B}}^{\mathcal{A},\xi}(\lambda)$ to be the probability of winning minus $1/2$.

Definition 2.2 *An Encryption scheme for class $\mathbb{B}$ is ξ-*secure *if for all probabilistic polynomial time ξ-adversaries $\mathcal{A}$, we have that $\mathsf{Adv}_{\mathbb{B}}^{\mathcal{A},\xi}(\lambda)$ is a negligible function of λ.*

3 Complexity Assumptions

Composite order bilinear groups were first used in Cryptography by [3] (see also [1]). We suppose the existence of an efficient group generator algorithm $\mathcal{G}$ which takes as input the security parameter λ and outputs a description $\mathcal{I}$ of a bilinear setting. The description $\mathcal{I}$ of the bilinear setting consists of $\mathcal{I} = (N, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ where $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order N, and $\mathbf{e} : \mathbb{G}^2 \to \mathbb{G}_T$ is a map with the following properties:

1. (Bilinearity) $\forall\ g, h \in \mathbb{G}$ and $a, b \in \mathbb{Z}_N$ it holds that $\mathbf{e}(g^a, h^b) = \mathbf{e}(g, h)^{ab}$.
2. (Non-degeneracy) $\exists\ g \in \mathbb{G}$ such that $\mathbf{e}(g, g)$ has order N in $\mathbb{G}_T$.

We assume that the group descriptions of $\mathbb{G}$ and $\mathbb{G}_T$ include generators of the respective cyclic groups. We require that the group operations in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the bilinear map $\mathbf{e}$ are computable in deterministic polynomial time in λ. 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 $\mathbb{G}_m$ denote the subgroup of $\mathbb{G}$ 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 $\mathbf{e}(g, h) = 1$. This is called the *orthogonality* property and is a crucial tool in our constructions.

We are now ready to give our complexity assumptions.

The first assumption that we state is a subgroup-decision type assumption for bilinear settings with groups of order product of four primes. 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.

For a generator $\mathcal{G}$ returning bilinear settings of order product of four primes, we define the following distribution. First pick a random bilinear setting $\mathcal{I} = (N = p_1p_2p_3p_4, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ by running $\mathcal{G}(1^\lambda)$ and then pick

$$A_3 \leftarrow \mathbb{G}_{p_3},\ A_{13} \leftarrow \mathbb{G}_{p_1p_3},\ A_{12} \leftarrow \mathbb{G}_{p_1p_2},\ A_4 \leftarrow\in \mathbb{G}_{p_4},\ T_1 \leftarrow \mathbb{G}_{p_1p_3},\ T_2 \leftarrow \mathbb{G}_{p_2p_3}.$$

and set $D = (\mathcal{I}, A_3, A_4, A_{13}, A_{12})$. We define the advantage of an algorithm $\mathcal{A}$ in breaking Assumption 1 to be

$$\mathsf{Adv}_1^{\mathcal{A}}(\lambda) = |\text{Prob}[\mathcal{A}(D, T_1) = 1] - \text{Prob}[\mathcal{A}(D, T_2) = 1]|$$

Assumption 1 *We say that Assumption* 1 *holds for generator $\mathcal{G}$ if for all probabilistic polynomial-time algorithms $\mathcal{A}$, $\mathsf{Adv}_1^{\mathcal{A}}(\lambda)$ is a negligible function of λ.*

Our second assumption can be seen as the Decision Diffie-Hellman Assumption for composite order groups. Specifically, Assumption 2 posits the difficult 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.

For a generator $\mathcal{G}$ returning bilinear settings of order product of four primes, we define the following distribution. First pick a random bilinear setting $\mathcal{I} = (N = p_1p_2p_3p_4, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ by running $\mathcal{G}(1^\lambda)$ and then pick

$$A_1 \leftarrow \mathbb{G}_{p_1}, A_2 \leftarrow \mathbb{G}_{p_2}, A_3 \leftarrow \mathbb{G}_{p_3}, A_4, B_4, C_4, D_4 \leftarrow \mathbb{G}_{p_4}, \alpha, \beta \leftarrow \mathbb{Z}_{p_1}, T_2 \leftarrow \mathbb{G}_{p_1p_4}$$

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 $\mathcal{A}$ in breaking Assumption 2 to be

$$\mathsf{Adv}_2^{\mathcal{A}}(\lambda) = |\mathrm{Prob}[\mathcal{A}(D, T_1) = 1] - \mathrm{Prob}[\mathcal{A}(D, T_2) = 1]|$$

Assumption 2 *We say that Assumption 2 holds for generator $\mathcal{G}$ if for all probabilistic polynomial-time algorithms $\mathcal{A}$, $\mathsf{Adv}_2^{\mathcal{A}}(\lambda)$ is a negligible function of λ.*

Assumption 3 is a generalization of Assumption 2 in the sense it posits the difficult of deciding if two triplets sharing an element are *both* Diffie-Hellman (looking at the formal definition below, the two triplets are the one composed of elements whose $\mathbb{G}_{p_1}$ parts are respectively $(A_1^{\alpha}, A_1^{\beta}, A_1^{\alpha\beta})$ and $(A_1^{\gamma}, A_1^{\alpha\beta}, A_1^{\alpha\beta\gamma})$) given a third related Diffie-Hellman triplets (composed of elements whose $\mathbb{G}_{p_1}$ parts are respectively $(A_1^{\alpha\gamma}, A_1^{\beta}, A_1^{\alpha\beta\gamma})$). More formally, we have the following definition.

For a generator $\mathcal{G}$ returning bilinear settings of order N product of four primes, we define the following distribution. First pick a random bilinear setting $\mathcal{I} = (N = p_1p_2p_3p_4, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ by running $\mathcal{G}(1^\lambda)$ and then pick

$$A_1 \leftarrow \mathbb{G}_{p_1}, A_2 \leftarrow \mathbb{G}_{p_2}, A_3 \leftarrow \mathbb{G}_{p_3}, A_4, B_4, C_4, D_4, E_4, F_4, G_4 \leftarrow \mathbb{G}_{p_4}, \alpha, \beta, \gamma \leftarrow \mathbb{Z}_{p_1}, T_2 \leftarrow \mathbb{G}_{p_1p_4}$$

and set $T_1 = A_1^{\alpha\beta} \cdot G_4$ and $D = (\mathcal{I}, A_1, A_2, A_3, A_4, A_1^{\alpha} \cdot B_4, A_1^{\beta} \cdot C_4, A_1^{\gamma} \cdot D_4, A_1^{\alpha\gamma} \cdot E_4, A_1^{\alpha\beta\gamma} \cdot F_4)$. We define the advantage of an algorithm $\mathcal{A}$ in breaking Assumption 3 to be

$$\mathsf{Adv}_3^{\mathcal{A}}(\lambda) = |\mathrm{Prob}[\mathcal{A}(D, T_1) = 1] - \mathrm{Prob}[\mathcal{A}(D, T_2) = 1]|$$

Assumption 3 *We say that Assumption 3 holds for generator $\mathcal{G}$ if for all probabilistic polynomial-time algorithms $\mathcal{A}$, $\mathsf{Adv}_3^{\mathcal{A}}(\lambda)$ is a negligible function of λ.*

It is easy to see that Assumption 3 implies Assumption 2.

Our final assumption is, like Assumption 1, a subgroup-decision type of assumption. More formally, for a generator $\mathcal{G}$ returning bilinear settings of order N product of five primes, we define the following distribution. First pick a random bilinear setting $\mathcal{I} = (N = p_1p_2p_3p_4, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ by running $\mathcal{G}(1^\lambda)$ and then pick

$$A_2 \leftarrow \mathbb{G}_{p_2},\ A_3 \leftarrow \mathbb{G}_{p_3},\ A_4, B_4, \leftarrow \mathbb{G}_{p_4},\ A_{14}, B_{14} \leftarrow \mathbb{G}_{p_1p_4}$$

and set $T_1 = B_{14}$, $T_2 = B_4$ and $D = (\mathcal{I}, A_2, A_3, A_4, A_{14})$. We define the advantage of an algorithm $\mathcal{A}$ in breaking Assumption 4 to be

$$\mathsf{Adv}_4^{\mathcal{A}}(\lambda) = |\mathrm{Prob}[\mathcal{A}(D, T_1) = 1] - \mathrm{Prob}[\mathcal{A}(D, T_2) = 1]|$$

Assumption 4 *We say that Assumption 4 holds for generator $\mathcal{G}$ if for all probabilistic polynomial-time algorithms $\mathcal{A}$, $\mathsf{Adv}_4^{\mathcal{A}}(\lambda)$ is a negligible function of λ.*

In Appendix B, we prove that Assumption $1, 3$ and 4 hold in the generic group model.

4 Constructing 0-secure HVE

In this section we describe our construction for a 0-secure (also called match revealing) HVE scheme. We assume without loss of generality that the vectors $\vec{y}$ of the keys have at least two indices i, j such that $y_i, y_j \neq \star$.

$\mathsf{Setup}(1^\lambda, 1^\ell)$**:** The setup algorithm chooses a description of a bilinear group $\mathcal{I} = (N = p_1p_2p_3p_4, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ with known factorization by running a generator algorithm $\mathcal{G}$ on input 1^λ. The setup algorithm chooses random $g_1 \in \mathbb{G}_{p_1}$, $g_2 \in \mathbb{G}_{p_2}$, $g_3 \in \mathbb{G}_{p_3}$, $g_4 \in \mathbb{G}_{p_4}$. For $i \in [\ell]$ and $b \in \{0,1\}$, the algorithm chooses random $t_{i,b} \in Z_N$ 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 $\mathsf{Pk} = [N, g_3, (T_{i,b})_{i\in[\ell],b\in\{0,1\}}]$ and the master secret key is $\mathsf{Msk} = [g_{12}, g_4, (t_{i,b})_{i\in[\ell],b\in\{0,1\}}]$, where $g_{12} = g_1 \cdot g_2$.

$\mathsf{KeyGen}(\mathsf{Msk}, \vec{y})$**:** 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 $W_i \in \mathbb{G}_{p_4}$ (the W_i are chosen by raising g_4 to a random power) and sets

$$Y_i = g_{12}^{a_i/t_{i,y_i}} W_i.$$

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

$\mathsf{Encrypt}(\mathsf{Pk}, \vec{x})$**:** The encryption algorithm chooses random $s \in \mathbb{Z}_N$. For $i \in [\ell]$, the algorithm chooses random $Z_i \in \mathbb{G}_{p_3}$ (the Z_i are chosen by raising g_3 to a random power) and sets

$$X_i = T_{i,x_i}^s Z_i,$$

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

$\mathsf{Test}(\mathsf{Ct}, \mathsf{Sk}_{\vec{y}})$**:** The test algorithm computes $T = \prod_{i\in S_{\vec{y}}} \mathbf{e}(X_i, Y_i)$ and returns TRUE iff $T = 1$.

Correctness. It is easy to verify the correctness of the scheme.

Remark 4.1 *Let $\mathsf{Pk} = [N, g_3, (T_{i,b})_{i\in[\ell],b\in\{0,1\}}]$ and $\mathsf{Msk} = [g_1 \cdot g_2, g_4, (t_{i,b})_{i\in[\ell],b\in\{0,1\}}]$ be a pair of public parameter and master secret key output by the* Setup *algorithm and consider $\mathsf{Pk}' = [N, g_3, (T'_{i,b})_{i\in[\ell],b\in\{0,1\}}]$ and $\mathsf{Msk}' = [\hat{g}_1 \cdot g_2, g_4, (t_{i,b})_{i\in[\ell],b\in\{0,1\}}]$ with $T'_{i,b} = \hat{g}_1^{t_{i,b}} \cdot R'_{i,b}$ for some $\hat{g}_1 \in \mathbb{G}_{p_1}$ 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* $\mathsf{KeyGen}(\mathsf{Msk}, \vec{y})$ *and* $\mathsf{KeyGen}(\mathsf{Msk}', \vec{y})$ *are identical.*
2. *Similarly, for every $\vec{x} \in \{0,1\}^\ell$, the distributions* $\mathsf{Encrypt}(\mathsf{Pk}, \vec{x})$ *and* $\mathsf{Encrypt}(\mathsf{Pk}', \vec{x})$ *are identical.*

4.1 Security of our HVE scheme

In this section we prove that our HVE scheme is 0-secure. To prove security of our HVE scheme, we rely on Assumptions 1 and 2. For a probabilistic polynomial-time 0-adversary $\mathcal{A}$ which makes q queries for KeyGen, our proof of security will be structured as a sequence of $q+2$ games between $\mathcal{A}$ and a challenger $\mathcal{C}$. The first game, $\mathsf{Game}_{\mathsf{Real}}$, is the real HVE security game described in the previous section. The remaining games, called $\mathsf{Game}_0, \ldots, \mathsf{Game}_q$, are described (and shown indistinguishable) in the following sections.

In the rest of this section, when we refer to adversaries we mean 0-adversaries and when we refer to $\mathsf{Game}_{\mathsf{Real}}$ we mean $\mathsf{Game}_{\mathsf{Real}}(0)$.

4.1.1 Description of Game_0

Game_0 is like $\mathsf{Game}_{\mathsf{Real}}$, except that $\mathcal{C}$ uses g_2 instead of g_1 to construct the public parameters Pk given to $\mathcal{A}$. Specifically,

Setup. $\mathcal{C}$ chooses a description of a bilinear group $\mathcal{I} = (N = p_1p_2p_3p_4, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ with known factorization by running a generator algorithm $\mathcal{G}$ on input 1^λ. $\mathcal{C}$ 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 sets $g_{12} = g_1 \cdot g_2$. For each $i \in [\ell]$ and $b \in \{0,1\}$, $\mathcal{C}$ chooses random $t_{i,b} \in Z_N$ and $R_{i,b} \in \mathbb{G}_{p_3}$ and sets $T_{i,b} = g_2^{t_{i,b}} \cdot R_{i,b}$ and $T'_{i,b} = g_1^{t_{i,b}} \cdot R_{i,b}$. Then $\mathcal{C}$ sets $\mathsf{Pk} = [N, g_3, (T_{i,b})_{i\in[\ell],b\in\{0,1\}}]$, $\mathsf{Pk}' = [N, g_3, (T'_{i,b})_{i\in[\ell],b\in\{0,1\}}]$, and $\mathsf{Msk} = [g_{12}, g_4, (t_{i,b})_{i\in[\ell],b\in\{0,1\}}]$. Finally, $\mathcal{C}$ starts the interaction with $\mathcal{A}$ on input Pk.

Key Query Answering. On a query for vector $\vec{y}$, $\mathcal{C}$ returns the output of KeyGen on input $\vec{y}$ and Msk.

Challenge Construction. $\mathcal{C}$ picks one of the two challenge vectors provided by $\mathcal{A}$ and encrypts it with respect to public parameters Pk'.

4.1.2 Proof of indistinguishability of $\mathsf{Game}_{\mathsf{Real}}$ and Game_0

Lemma 4.2 *Suppose there exists a PPT algorithm $\mathcal{A}$ such that $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_{\mathsf{Real}}} - \mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_0} = \epsilon$. Then, there exists a PPT algorithm $\mathcal{B}$ with advantage ϵ in breaking Assumption* 1*.*

Proof. We show a PPT algorithm $\mathcal{B}$ which receives $(\mathcal{I}, A_3, A_4, A_{13}, A_{12})$ and T and, depending on the nature of T, simulates $\mathsf{Game}_{\mathsf{Real}}$ or Game_0 with $\mathcal{A}$. This suffices to prove the Lemma.

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

Key Query Answering. Whenever $\mathcal{A}$ asks to see the secret key $\mathsf{Sk}_{\vec{y}}$ associated with vector $\vec{y}$, $\mathcal{B}$ runs algorithm KeyGen on input Msk and $\vec{y}$.

Challenge Construction. The challenge is created by $\mathcal{B}$ by picking one of the two vectors provided by $\mathcal{A}$, let us call it $\vec{x}$, and by encrypting it by running the $\mathsf{Encrypt}$ algorithm on input $\vec{x}$ and Pk'.

This concludes the description of algorithm $\mathcal{B}$.

Now suppose $T \in \mathbb{G}_{p_1p_3}$, and thus it can be written as $T = h_1 \cdot h_3$ for $h_1 \in \mathbb{G}_{p_1}$ and $h_3 \in \mathbb{G}_{p_3}$. This implies that Pk received in input by $\mathcal{A}$ in the interaction with $\mathcal{B}$ has the same distribution as in $\mathsf{Game}_{\mathsf{Real}}$. Moreover, by writing A_{13} as $A_{13} = \hat{h}_1 \cdot \hat{h}_3$ for $\hat{h}_1 \in \mathbb{G}_{p_1}$ and $\hat{h}_3 \in \mathbb{G}_{p_3}$ which is possible since by assumption $A_{13} \in \mathbb{G}_{p_1p_3}$, we notice that that Pk and Pk' are as in the hypothesis of Remark 4.1 (with $g_1 = h_1$ and $\hat{g}_1 = \hat{h}_1$). Therefore the answers to key queries and the challenge ciphertext given by $\mathcal{B}$ to $\mathcal{A}$ have the same distribution as the answers and the challenge ciphertext received by $\mathcal{A}$ in $\mathsf{Game}_{\mathsf{Real}}$. We can thus conclude that, when $T \in \mathbb{G}_{p_1p_3}$, $\mathcal{C}$ has simulates $\mathsf{Game}_{\mathsf{Real}}$ with $\mathcal{A}$.

Let us discuss now the case $T \in \mathbb{G}_{p_2p_3}$. In this case, Pk provided by $\mathcal{B}$ has the same distribution as the public parameters produced by $\mathcal{C}$ in Game_0. Therefore, $\mathcal{C}$ is simulating Game_0 for $\mathcal{A}$.

This concludes the proof of the lemma. □

4.1.3 Description of Game_k, for $1 \leq k \leq q$

Each of these games is like Game_0, except that the first k key queries issued by $\mathcal{A}$ are answered with keys whose $\mathbb{G}_{p_1}$ parts are random. The remaining key queries (that is, from the $(k+1)$-st to the q-th) are answered like in the previous game. The $\mathbb{G}_{p_2}$ parts of all the answers to key queries are like in Game_0. More precisely, in Game_k, the Setup phase and the Challenge Construction are like in Game_0 and the Key Query phase is the following.

Key Query Answering. $\mathcal{C}$ answers the first k key queries in the following way. On input vector $\vec{y}$, for $i \in S_{\vec{y}}$, $\mathcal{C}$ chooses random $a_i, c_i \in \mathbb{Z}_N$ under the constraint that $\sum_{i \in S_{\vec{y}}} a_i = 0$ and random $W_i \in \mathbb{G}_{p_4}$. $\mathcal{C}$ sets, for $i \in S_{\vec{y}}$,

$$Y_i = g_1^{c_i} \cdot g_2^{a_i/t_{i,y_i}} \cdot W_i.$$

The remaining $q - k$ queries are answered like in Game_0.

4.1.4 Proof of indistinguishability of Game_{k-1} and Game_k

Lemma 4.3 *Suppose there exists a PPT algorithm $\mathcal{A}$ such that $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_{k-1}} - \mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_k} = \epsilon$. Then, there exists a PPT algorithm $\mathcal{B}$ with advantage at least $\epsilon/(2\ell)$ in breaking Assumption* 2.

Proof. We show a PPT algorithm $\mathcal{B}$ which receives $(\mathcal{I}, A_1, A_2, A_3, A_4, A_1^{\alpha} \cdot B_4, A_1^{\beta} \cdot C_4)$ and T and, depending on the nature of T, simulates Game_{k-1} or Game_k with $\mathcal{A}$. This suffices to prove the Lemma.

$\mathcal{B}$ starts by guessing the index j such that the j-th bit $y_j^{(k)}$ of the k-th query $\vec{y}^{(k)}$ is different from $\star$ and different from the j-th bit x_j of the challenge vectors provided by $\mathcal{A}$ that $\mathcal{C}$ uses to construct the challenge ciphertext. Notice that the probability that $\mathcal{B}$ correctly guesses j and $y_j^{(k)}$ is at least $1/(2\ell)$, independently from the view of $\mathcal{A}$. Notice that, if during the simulation this is not the case, then $\mathcal{B}$ aborts the simulation and fails. We next describe and prove the correctness of the simulation under the assumption that $\mathcal{B}$'s initial guess is correct. Notice that if the initial guess is correct x_j and $y_j^{(k)}$ are uniquely determined and it holds that $x_j = 1 - y_j^{(k)}$.

Setup. $\mathcal{B}$ sets $g_1 = A_1$, $g_2 = A_2$, $g_3 = A_3$, $g_4 = A_4$ and $g_{12} = A_1 \cdot A_2$. For each $i \in [\ell] \setminus \{j\}$ and $b \in \{0,1\}$, $\mathcal{B}$ chooses random $t_{i,b} \in \mathbb{Z}_N$ and $R_{i,b} \in \mathbb{G}_{p_3}$, and sets $T_{i,b} = g_2^{t_{i,b}} \cdot R_{i,b}$. Moreover, $\mathcal{B}$ chooses random $t_{j,x_j} \in Z_N, R_{j,x_j} \in \mathbb{G}_{p_3}$, $r_{j,y_j^{(k)}} \in Z_N$ and $R_{j,y_j^{(k)}} \in \mathbb{G}_{p_3}$ and sets

$$T_{j,x_j} = g_2^{t_{j,x_j}} \cdot R_{j,x_j} \qquad T_{j,y_j^{(k)}} = g_2^{r_{j,y_j^{(k)}}} \cdot R_{j,y_j^{(k)}}.$$

Notice that by assumption $x_j \neq y_j^{(k)}$. $\mathcal{B}$ then sets $\mathsf{Pk} = [N, g_3, (T_{i,b})_{i\in[\ell],b\in\{0,1\}}]$.

In addition, for each $i \in [\ell] \setminus \{j\}$ and $b \in \{0,1\}$ and $\mathcal{B}$ chooses random $R'_{i,b} \in \mathbb{G}_{p_3}$ and sets $T'_{i,b} = g_1^{t_{i,b}} \cdot R'_{i,b}$. Moreover $\mathcal{B}$ chooses random R_{j,x_j} and sets $T'_{j,x_j} = g_1^{t_{x,x_j}} \cdot R'_{j,x_j}$. The value of $T'_{j,y_j^{(k)}}$ remains unspecified. As we shall see below, in answering key queries, $\mathcal{B}$ will implicitly set $T'_{j,y_j^{(k)}} = g_1^{1/\beta} \cdot R'_{j,y_j^{(k)}}$ for a random $R'_{j,y_j^{(k)}} \in \mathbb{G}_{p_3}$.

$\mathcal{B}$ starts the interaction with $\mathcal{A}$ on input Pk. Notice that Pk has the same distribution as the public parameters seen by $\mathcal{A}$ in Game_{k-1} and Game_k.

Key Query Answering. For the first $k-1$ queries $\mathcal{B}$ proceeds as follows. Let $\vec{y}$ be the input vector. For $i \in S_{\vec{y}}$, $\mathcal{B}$ chooses random a_i such that $\sum_{i\in S_{\vec{y}}} a_i = 0$, random $z_i \in \mathbb{Z}_N$, and random $W_i \in \mathbb{G}_{p_4}$. Then, for $i \in S_{\vec{y}} \setminus \{j\}$, $\mathcal{B}$ computes

$$Y_i = g_1^{z_i} \cdot g_2^{a_i/t_{i,y_i}} \cdot W_i.$$

If $y_j = x_j$ then $\mathcal{B}$ sets

$$Y_j = g_1^{z_j} \cdot g_2^{a_j/t_{j,y_j}} \cdot W_j$$

otherwise if $y_j \neq \star$ then $\mathcal{B}$ sets

$$Y_j = g_1^{z_j} \cdot g_2^{a_j/r_{j,y_j}} \cdot W_j.$$

Also notice that the first $k-1$ answers produced by $\mathcal{B}$ have the same distribution as the first $k-1$ answers seen by $\mathcal{A}$ in Game_{k-1} and Game_k.

Let us now describe how $\mathcal{B}$ answers the k-th query the vector $\vec{y}^{(k)}$. Let h be an index such that $h \neq j$ and $y_h^{(k)} \neq \star$; such an index always exists by our assumption that all keys are for vectors with at least two entries different from $\star$. Also we remind the reader that $y_j^{(k)} = 1 - x_j$.

Let $S = S_{\vec{y}} \setminus \{j, h\}$. For each $i \in S$, $\mathcal{B}$ chooses random $a_i \in \mathbb{Z}_N$ and $W_i \in \mathbb{G}_{p_4}$ and sets

$$Y_i = g_{12}^{a_i/t_{i,y_i^{(k)}}} \cdot W_i.$$

$\mathcal{B}$ chooses random $a'_j \in \mathbb{Z}_N$ and $W_j, W_h \in \mathbb{G}_{p_4}$ and sets

$$Y_j = T \cdot g_2^{a'_j/r_{j,y_j^{(k)}}} \cdot W_j$$

and

$$Y_h = (A_1^\alpha B_4)^{-1/t_{h,y_h^{(k)}}} \cdot g_1^{-s/t_{h,y_h^{(k)}}} \cdot g_2^{-(s+a_j)/t_{h,y_h^{(k)}}} \cdot W_h,$$

where $s = \sum_{i \in S} a_i$. This terminates the description of how $\mathcal{B}$ handles the k-th key query. Let us now verify that when $T = A_1^{\alpha\beta} \cdot D_4$ then $\mathcal{B}$'s answer to the k-th key query is like in Game_{k-1}. By our settings, we have that

$$Y_j = g_1^{\alpha / t'_{j,y_j^{(k)}}} \cdot g_2^{a'_j / r_{j,y_j^{(k)}}} \cdot D_4 \cdot W_j$$

with $t'_{j,y_j^{(k)}} = 1/\beta$. By the Chinese Remainder Theorem, there exists $a_j \in Z_N$ such that

$$\begin{aligned} a_j &\equiv \alpha \mod p_1 \\ a_j &\equiv a'_j \mod p_2 \end{aligned}$$

and $t_{j,y_j^{(k)}} \in Z_N$ such that

$$\begin{aligned} t_{j,y_j^{(k)}} &\equiv t'_{j,y_j^{(k)}} \mod p_1 \\ t_{j,y_j^{(k)}} &\equiv r_{j,y_j^{(k)}} \mod p_2 \end{aligned}$$

and thus Y_j and Y_h can be written as

$$Y_j = g_{12}^{a_j / t_{j,y_j^{(k)}}} \cdot (D_4 \cdot W_j) \quad \text{and} \quad Y_h = g_{12}^{a_h / t_{h,y_h^{(k)}}} \cdot W_h$$

where $a_h = -(s + a_j)$. Therefore we have that $\sum_{i \in S_{\vec{y}}} a_i = 0$ and we can conclude that the answer to the k-th query of $\mathcal{A}$ is distributed as in Game_{k-1}.

On the other hand if T is random in $\mathbb{G}_{p_1 p_4}$ then the $\mathbb{G}_{p_1}$ parts of the Y_i's are random and thus the answer to the k-th query of $\mathcal{A}$ is distributed as in Game_k.

Let us now briefly discuss how $\mathcal{B}$ handles the l-th key queries for $l = k+1, \ldots, q$. First of all notice that if the j-th bit of the l-th query vector is equal to x_j then $\mathcal{B}$ has all the t_{i,y_i}'s needed for running algorithm KeyGen. On the other hand, if this is not the case then, by the previous settings, $t_{j,y_j} \equiv 1/\beta \mod p_1$. Therefore $\mathcal{B}$ can use $A_1^{\beta} \cdot C_4 = g_1^{1/t_{j,y_j}} \cdot C_4$ that is part of the instance of Assumption 2 that $\mathcal{B}$ has to break. We can conclude that the answers provided by $\mathcal{B}$ to $\mathcal{A}$'s last $q - k$ queries have the same distribution as in Game_k and Game_{k-1}.

Challenge Construction. The challenge is created by running algorithm $\mathsf{Encrypt}$ on input the randomly chosen challenge vector $\vec{x}$ and public parameters Pk'. Notice that under the assumption that $\mathcal{B}$ has correctly guessed x_j and thus $x_j = 1 - y_j^{(k)}$, Pk' contains all the values needed for computing an encryption of $\vec{x}$. Also notice, that the challenge ciphertext is distributed exactly like in Game_{k-1} and Game_k.

□

4.1.5 Game_q gives no advantage

We observe that in Game_q the $\mathbb{G}_{p_1}$ part of the challenge ciphertext is the only one depending on η. However, the elements of the public parameters Pk given as input to the adversary in Game_q have no $\mathbb{G}_{p_1}$ part and moreover the answer to the key queries have random and independent $\mathbb{G}_{p_1}$ part. Therefore we can conclude that for all adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_q} = 0$. We have thus proved.

Theorem 4.4 *If Assumptions* 1 *and* 2 *hold for generator $\mathcal{G}$, then the HVE scheme presented is* 0*-secure (also called match revealing secure).*

5 Constructing 1-secure HVE

In this section we describe our construction for a 1-secure HVE scheme.

$\mathsf{Setup}(1^\lambda, 1^\ell)$**:** The setup algorithm chooses a description of a bilinear group $\mathcal{I} = (N = p_1p_2p_3p_4, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ with known factorization by running a generator algorithm $\mathcal{G}$ on input 1^λ. The setup algorithm chooses random $g_1 \in \mathbb{G}_{p_1}$, $g_2 \in \mathbb{G}_{p_2}$, $g_3 \in \mathbb{G}_{p_3}$, $g_4 \in \mathbb{G}_{p_4}$. For $i \in [\ell]$ and $b \in \{0,1\}$, the algorithm chooses random $t_{i,b} \in Z_N$, random $v_i \in Z_N$ and random $R_{i,b} \in \mathbb{G}_{p_3}$ and sets $T_{i,b} = g_1^{t_{i,b}} \cdot g_4^{v_i} \cdot R_{i,b}$.

The public parameters are $\mathsf{Pk} = [N, g_3, (T_{i,b})_{i\in[\ell],b\in\{0,1\}}]$ and the master secret key is $\mathsf{Msk} = [g_{12}, g_4, (t_{i,b})_{i\in[\ell],b\in\{0,1\}}, (v_i)_{i\in[\ell]}]$, where $g_{12} = g_1 \cdot g_2$.

$\mathsf{KeyGen}(\mathsf{Msk}, \vec{y})$**:** Let $S_{\vec{y}}$ be the set of indices i such that $y_i \neq \star$. For $i \in S_{\vec{y}}$, the key generation algorithm chooses random $a_i \in \mathbb{Z}_N$ under the constraint that $\sum_{i\in S_{\vec{y}}} a_i = 0$. For $i \in S_{\vec{y}}$, the algorithm sets

$$Y_i = g_{12}^{a_i/t_{i,y_i}} g_4^{a_i/v_i}.$$

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

$\mathsf{Encrypt}(\mathsf{Pk}, \vec{x})$**:** The encryption algorithm chooses random $s \in \mathbb{Z}_N$. For $i \in [\ell]$, the algorithm chooses random $Z_i \in \mathbb{G}_{p_3}$, sets $X_i = T_{i,x_i}^s Z_i$, and returns the tuple $(X_i)_{i\in[\ell]}$.

$\mathsf{Test}(\mathsf{Ct}, \mathsf{Sk}_{\vec{y}})$**:** The test algorithm computes $T = \prod_{i\in S_{\vec{y}}} \mathbf{e}(X_i, Y_i)$ and returns TRUE iff $T = 1$.

Correctness. It is easy to verify the correctness of the scheme.

Remark 5.1 *Let* $\mathsf{Pk} = [N, g_3, (T_{i,b})_{i\in[\ell],b\in\{0,1\}}]$ *and* $\mathsf{Msk} = [g_1 \cdot g_2, g_4, (t_{i,b})_{i\in[\ell],b\in\{0,1\}}, (v_i)_{i\in[\ell]}]$ *be a pair of public parameter and master secret key output by the* Setup *algorithm and consider* $\mathsf{Pk}' = [N, g_3, (T'_{i,b})_{i\in[\ell],b\in\{0,1\}}]$ *and* $\mathsf{Msk}' = [\hat{g}_1 \cdot g_2, g_4, (t_{i,b})_{i\in[\ell],b\in\{0,1\}}, (v_i)_{i\in[\ell]}]$ *with* $T'_{i,b} = \hat{g}_1^{t_{i,b}} \cdot g_4^{v_i} \cdot R'_{i,b}$ *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* $\mathsf{KeyGen}(\mathsf{Msk}, \vec{y})$ *and* $\mathsf{KeyGen}(\mathsf{Msk}', \vec{y})$ *are identical.*
2. *Similarly, for every* $\vec{x} \in \{0,1\}^\ell$*, the distributions* $\mathsf{Encrypt}(\mathsf{Pk}, \vec{x})$ *and* $\mathsf{Encrypt}(\mathsf{Pk}', \vec{x})$ *are identical.*

5.1 Security of our HVE scheme

To prove that our HVE scheme is 1-secure, we rely on static Assumptions 1 and 4. For a probabilistic polynomial-time 1-adversary $\mathcal{A}$ our proof of security will be structured as a sequence of 2 games between $\mathcal{A}$ and a challenger $\mathcal{C}$. The first game, $\mathsf{Game}_{\mathsf{Real}}(1)$, is the real HVE security game described in the previous section. The remaining games, called $\mathsf{Game}_0, \mathsf{Game}_1$, are described (and shown indistinguishable) in the following sections.

In the rest of this section, when we refer to adversaries we mean 1-adversaries and when we refer to $\mathsf{Game}_{\mathsf{Real}}$ we mean $\mathsf{Game}_{\mathsf{Real}}(1)$.

5.1.1 Description of Game_0

Game_0 is like $\mathsf{Game}_{\mathsf{Real}}$, except that $\mathcal{C}$ uses g_2 instead of g_1 to construct the public parameters Pk given to $\mathcal{A}$. Specifically,

Setup. $\mathcal{C}$ chooses random $g_1 \in \mathbb{G}_{p_1}, g_2 \in \mathbb{G}_{p_2}, g_3 \in \mathbb{G}_{p_3}, g_4 \in \mathbb{G}_{p_4}$. For $i \in [\ell]$ and $b \in \{0,1\}$, $\mathcal{C}$ chooses random $t_{i,b} \in Z_N$, random $v_i \in Z_N$ and random $R_{i,b} \in \mathbb{G}_{p_3}$ and sets $T_{i,b} = g_2^{t_{i,b}} \cdot g_4^{v_i} \cdot R_{i,b}$ and $T'_{i,b} = g_1^{t_{i,b}} \cdot g_4^{v_i} \cdot R_{i,b}$. Then $\mathcal{C}$ sets $\mathsf{Pk} = [N, g_3, (T_{i,b})_{i\in[\ell],b\in\{0,1\}}]$ and $\mathsf{Pk}' = [N, g_3, (T'_{i,b})_{i\in[\ell],b\in\{0,1\}}]$. $\mathcal{C}$ starts the interaction with $\mathcal{A}$ on input Pk.

Key Query Answering. On a query for vector $\vec{y}$, $\mathcal{C}$ returns the output of KeyGen on input $\vec{y}$ and $\mathsf{Msk} = [g_{12}, g_4, (t_{i,b})_{i\in[\ell],b\in\{0,1\}}, (v_i)_{i\in[\ell]}]$, where $g_{12} = g_1 \cdot g_2$.

Challenge Construction. $\mathcal{C}$ picks one of the two challenge vectors provided by $\mathcal{A}$ and encrypts it with respect to public parameters Pk'.

5.1.2 Proof of indistinguishability of $\mathsf{Game}_{\mathsf{Real}}$ and Game_0

Lemma 5.2 *Suppose there exists a PPT algorithm $\mathcal{A}$ such that $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_{\mathsf{Real}}} - \mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_0} = \epsilon$. Then, there exists a PPT algorithm $\mathcal{B}$ with advantage ϵ in breaking Assumption* 1.

PROOF. We show a PPT algorithm $\mathcal{B}$ which receives $(\mathcal{I}, A_3, A_4, A_{13}, A_{12})$ and T and, depending on the nature of T, simulates $\mathsf{Game}_{\mathsf{Real}}$ or Game_0 with $\mathcal{A}$. This suffices to prove the Lemma.

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

Key Query Answering. Whenever $\mathcal{A}$ asks to see the secret key $\mathsf{Sk}_{\vec{y}}$ associated with vector $\vec{y}$, $\mathcal{B}$ runs algorithm KeyGen on input Msk and $\vec{y}$.

Challenge Construction. The challenge is created by $\mathcal{B}$ by picking one of the two vectors provided by $\mathcal{A}$, let us call it $\vec{x}$, and by encrypting it by running the $\mathsf{Encrypt}$ algorithm on input $\vec{x}$ and Pk'.

Let us write A_{13} as $A_{13} = \hat{h}_1 \cdot \hat{h}_3$ for $\hat{h}_1 \in \mathbb{G}_{p_1}$ and $\hat{h}_3 \in \mathbb{G}_{p_3}$. Now suppose $T \in \mathbb{G}_{p_1p_3}$, and thus $T = h_1 \cdot h_3$ for $h_1 \in \mathbb{G}_{p_1}$ and $h_3 \in \mathbb{G}_{p_3}$. Notice that Pk and Pk' are as in the hypothesis of Remark 5.1 (with $g_1 = h_1$ and $\hat{g}_1 = \hat{h}_1$) and thus the challenge given by $\mathcal{C}$ to $\mathcal{A}$ has the same distribution as an encryption of $\vec{x}$ with Pk. We can thus conclude that in this case $\mathcal{C}$ has simulated $\mathsf{Game}_{\mathsf{Real}}$ with $\mathcal{A}$.

Let us discuss now the case $T \in \mathbb{G}_{p_2p_3}$. In this case the public parameters Pk provided by $\mathcal{B}$ have the same distribution as the public parameters produced by $\mathcal{C}$ in Game_0. Therefore, $\mathcal{C}$ is simulating Game_0 for $\mathcal{A}$.

This concludes the proof of the lemma. □

5.1.3 Description of Game_1

This game is like Game_0, except that in the answers provided by $\mathcal{C}$ the key queries. Specifically the queries are answered without the $\mathbb{G}_{p_1}$ part. The $\mathbb{G}_{p_2}$ part of all answers is like in Game_0. Specifically, we have.

Setup. Like in Game_0.

Query answering. $\mathcal{C}$ answers the queries in the following way. On input vector $\vec{y}$, for $i \in S_{\vec{y}}$, $\mathcal{C}$ chooses random $a_i, b_i \in \mathbb{Z}_N$ under the constraint that $\sum_{i\in S_{\vec{y}}} a_i = \sum_{i\in S_{\vec{y}}} b_i = 0$. $\mathcal{C}$ sets, for $i \in S_{\vec{y}}$,

$$Y_i = g_2^{a_i/t_{i,y_i}} \cdot g_4^{b_i/v_i} \cdot W_i.$$

Challenge Construction. Like in Game_0.

5.1.4 Proof of indistinguishability of Game_0 and Game_1

Lemma 5.3 *Suppose there exists a PPT algorithm $\mathcal{A}$ such that $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_0} - \mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_1} = \epsilon$. Then, there exists a PPT algorithm $\mathcal{B}$ with advantage at least ϵ in breaking Assumption* 4.

Proof. We show a PPT algorithm $\mathcal{B}$ which receives $(\mathcal{I}, A_2, A_3, A_4, A_{14})$ and T and, depending on the nature of T, simulates Game_0 or Game_1 with $\mathcal{A}$. This suffices to prove the Lemma.

Setup. $\mathcal{B}$ sets $g_2 = A_2$, $g_3 = A_3$, $g_4 = A_4$. For $i \in [\ell]$ and $b \in \{0,1\}$, $\mathcal{B}$ chooses random $t_{i,b}, v_i \in \mathbb{Z}_N$ and $R_{i,b} \in \mathbb{G}_{p_3}$, and sets $T_{i,b} = g_2^{t_{i,b}} \cdot g_4^{v_i} \cdot R_{i,b}$. These settings determine public parameters $\mathsf{Pk} = [N, g_3, (T_{i,b})_{i\in[\ell],b\in\{0,1\}}]$.

$\mathcal{B}$ starts the interaction with $\mathcal{A}$ on input Pk. Notice that Pk has the same distribution as the public parameters seen by $\mathcal{A}$ in Game_0 and Game_1.

Key Query Answering. $\mathcal{B}$ computes the answer to query for vector $\vec{y}$ in the following way. For $i \in S_{\vec{y}}$, $\mathcal{B}$ chooses random $a_i \in \mathbb{Z}_N$ subject to $\sum_{i\in S_{\vec{y}}} a_i = 0$ and sets

$$Y_i = g_2^{a_i/t_{i,y_i}} \cdot T^{a_i/v_i}$$

Now suppose $T = B_{14}$ and write $T = g_1 g_4^c$ for some $g_1 \in \mathbb{G}_{p_1}$ and $c \in \mathbb{Z}_{p_4}$. By our setting we have

$$Y_i = g_1^{a_i/v_i} \cdot g_2^{a_i/t_{i,y_i}} \cdot g_4^{ca_i/v_i}$$

which implicitly sets $t_{i,y_i} \equiv v_i \mod p_1$. It is now easy to see that the answer to the query is distributed as in Game_0. On the other hand, if $T = B_4$ then the key does not contain the $\mathbb{G}_{p_1}$ part and thus the answer to the query of $\mathcal{A}$ is distributed as in Game_1.

Notice that, since $\mathcal{A}$ is a 1-adversary, then for every query vector $\vec{y}$ that $\mathcal{A}$ can submit, it holds that for each $i \in [\ell]$, $y_i = \star$ or $y_i = x_{0,i} = x_{1,i}$. Therefore, during the simulation for each $i \in [\ell]$, $\mathcal{B}$ need only to determine $t_{i,x_{0,i}} = t_{i,x_{1,i}}$ and it does so by setting it congruent to $v_i \mod p_1$.

Challenge Construction. For $i \in [\ell]$, $\mathcal{B}$ chooses random $Z_i \in \mathbb{G}_{p_3}$. Then $\mathcal{B}$ computes the following. For $i \in [\ell]$,

$$X_i = A_{14}^{v_i} \cdot Z_i$$

Finally notice that by writing $A_{14} = (g_1 \cdot g_4^c)^s$, the challenge ciphertext is distributed exactly like in Game_0 and Game_1.

□

5.1.5 Game_1 gives no advantage

We observe that the $\mathbb{G}_{p_1}$ part of the challenge ciphertext is the only one depending on η. However, the elements of the public parameters Pk given as input to the adversary in Game_1 have no $\mathbb{G}_{p_1}$ part and the same hold also for the answers to the key queries. Therefore we can conclude that for all adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_1} = 0$. We have thus proved.

Theorem 5.4 *If Assumptions* 1 *and* 4 *hold for generator* $\mathcal{G}$*, then the HVE scheme presented is* 1-*secure.*

5.2 Merging the schemes

It is easy to see that our 1-secure HVE scheme can be extended to a scheme that is also 0-secure. This can be done by using a bilinear instance of order product of five primes and by using the new subgroup to randomize the secret keys. The proof of the 0-security of the resulting scheme is very similar to that provided in section 4. We point out that this does not mean that the resulting scheme is secure against adversaries that can request *both* matching and non-matching queries.

6 Hierarchical HVE

We start by giving the definition of Hierarchical HVE (see also [15],[10]). Given $\vec{y}, \vec{w} \in \{0,1,\star\}^\ell$, we say that $\vec{w}$ is a *delegation* of $\vec{y}$, in symbols $\vec{w} \prec \vec{y}$, iff for each $i \in [\ell]$ we have $y_i = \star$ or $y_i = w_i$. For example $\langle 1,0,1,\star\rangle \prec \langle 1,0,\star,\star\rangle$. Notice that $\prec$ imposes a partial order on $\{0,1,\star\}^\ell$.

A Hierarchical HVE scheme (HHVE) consists of five efficient algorithms (Setup, $\mathsf{Encrypt}$, KeyGen, Test, $\mathsf{Delegate}$). The semantics of Setup, $\mathsf{Encrypt}$, KeyGen and Test are identical to those given for HVE. The delegation algorithm has the following semantics.

$\mathsf{Delegate}(\mathsf{Pk}, \mathsf{Sk}_{\vec{y}}, \vec{y}, \vec{w})$: takes as input $\vec{y}, \vec{w} \in \{0,1,\star\}^\ell$ such that $\vec{w} \prec \vec{y}$ and secret key $\mathsf{Sk}_{\vec{y}}$ for $\vec{y}$ and outputs secret key $\mathsf{Sk}_{\vec{w}}$ for $\vec{w}$.

Correctness of HHVE. We require that for pairs $(\mathsf{Pk}, \mathsf{Msk})$ output by $\mathsf{Setup}(1^\lambda, 1^\ell)$, for all $y \in \{0,1,\star\}^\ell$, keys $\mathsf{Sk}_{\vec{y}}$ computed by KeyGen on input Msk, for all $\vec{w} \prec \vec{y}$ and all delegation paths $\vec{w} = \vec{w}_n \prec \vec{w}_{n-1} \prec \ldots \prec \vec{w}_0 = \vec{y}$ of length $n \geq 0$ with $\mathsf{Sk}_{\vec{w}_i} = \mathsf{Delegate}(\mathsf{Pk}, \mathsf{Sk}_{\vec{w}_{i-1}}, \vec{w}_{i-1}, \vec{w}_i)$ for $i \in [n]$, and all $\vec{x} \in \{0,1\}^\ell$ we have that the probability that

$$\mathsf{Test}(\mathsf{Pk}, \mathsf{Encrypt}(\mathsf{Pk}, \vec{x}), \mathsf{Sk}_{\vec{w}}) \neq \mathsf{Match}(\vec{x}, \vec{w})$$

is negligible in λ.

6.1 Security definition for HHVE

Our security definition follows [15] and requires that no PPT adversary $\mathcal{A}$ has non-negligible advantage over 1/2 in game $\mathsf{Game}_{\mathsf{Real}}$ against a challenger $\mathcal{C}$. $\mathsf{Game}_{\mathsf{Real}}$ consists of the Setup Phase followed by a Query Phase. The Query Phase consists of several Key Queries and one Challenge Construction Query. We stress that the Challenge Construction Query is not necessarily the last query of the Query Phase. More precisely, we have the following game.

Setup. $\mathcal{C}$ 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. $\mathcal{C}$ starts the interaction with $\mathcal{A}$ on input Pk.

Key Query Answering. Key queries can be of three different types. $\mathcal{C}$ answers these queries in the following way. $\mathcal{C}$ starts by initializing the set S of private keys that have been created but not yet given to $\mathcal{A}$ equal to $\emptyset$.

- *Create.* To make a *Create* query, $\mathcal{A}$ specifies a vector $\vec{y} \in \{0, 1, \star\}^\ell$. In response, $\mathcal{C}$ creates a key for $\vec{y}$ by running the KeyGen algorithm on input Msk and $\vec{y}$. $\mathcal{C}$ adds this key to the set S and gives $\mathcal{A}$ only a reference to it, not the actual key.

- *Delegate.* To make a *Delegate* query, $\mathcal{A}$ specifies a reference to a key $\mathsf{Sk}_{\vec{y}}$ in the set S and a vector $\vec{w} \in \{0, 1, \star\}^\ell$ such that $\vec{w} \prec \vec{y}$. In response, $\mathcal{C}$ makes a key for $\vec{w}$ by executing the $\mathsf{Delegate}$ algorithm on input Pk, $\mathsf{Sk}_{\vec{y}}$, $\vec{y}$ and $\vec{w}$. $\mathcal{C}$ adds this key to the set S and again gives $\mathcal{A}$ only a reference to it, not the actual key.

- *Reveal.* To make a *Reveal* query, $\mathcal{A}$ specifies an element of the set S. $\mathcal{C}$ gives the corresponding key to $\mathcal{A}$ and removes it from the set S. We note that $\mathcal{A}$ needs no longer make any delegation queries for this key because it can run the $\mathsf{Delegate}$ algorithm on the revealed key by itself.

Challenge Construction. To make a Challenge Construction query, $\mathcal{A}$ specifies a pair $\vec{x}_0, \vec{x}_1 \in \{0, 1\}^\ell$. $\mathcal{C}$ answers by picking random $\eta \in \{0, 1\}$ and returning $\mathsf{Encrypt}(\mathsf{Pk}, \vec{x}_\eta)$.

At the end of the game, $\mathcal{A}$ outputs a guess η' for η. We say that $\mathcal{A}$ *wins* if $\eta = \eta'$ and for all $\vec{y}$ for which $\mathcal{A}$ has seen a secret key, it holds that $\mathsf{Match}(\vec{x}_0, \vec{y}) = \mathsf{Match}(\vec{x}_1, \vec{y}) = 0$. The *advantage* $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{HHVE}}(\lambda)$ of $\mathcal{A}$ is defined to be the probability that $\mathcal{A}$ wins the game minus 1/2. We are now ready for the following definition.

Definition 6.1 *A Hierarchical Hidden Vector Encryption scheme is secure if for all probabilistic polynomial time adversaries $\mathcal{A}$, we have that $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{HHVE}}(\lambda)$ is a negligible function of λ.*

Notice that in our security definition the adversary can see keys only for vectors $\vec{y}$ that do not match either of the two challenges. We share this limitation with [8].

6.2 Our construction for HHVE

In this section we describe our construction for an HHVE scheme. As in HVE, we assume without loss of generality that the vectors $\vec{y}$ of the keys have at least two indices i, j such that $y_i, y_j \neq \star$.

$\mathsf{Setup}(1^\lambda, 1^\ell)$: The setup algorithm chooses a description of a bilinear group $\mathcal{I} = (N = p_1 p_2 p_3 p_4, \mathbb{G}, \mathbb{G}_T, \mathbf{e})$ with known factorization and random $g_1 \in \mathbb{G}_{p_1}$, $g_2 \in \mathbb{G}_{p_2}$, $g_3, R \in \mathbb{G}_{p_3}$, $g_4 \in \mathbb{G}_{p_4}$ and sets

$g_{12} = g_1 \cdot g_2$. Then, for each $i \in [\ell]$ and $b \in \{0,1\}$, the setup algorithm chooses random $t_{i,b} \in Z_N$ and $R_{i,b} \in \mathbb{G}_{p_3}$ and sets $T_{i,b} = g_1^{t_{i,b}} \cdot R_{i,b}$.

The public parameters are $\mathsf{Pk} = \left[N, g_3, g_4, g_1 \cdot R, (T_{i,b})_{b\in\{0,1\},i\in[\ell]}\right]$ and the master secret key is $\mathsf{Msk} = \left[g_{12}, (t_{i,b})_{b\in\{0,1\},i\in[\ell]}\right]$.

$\mathsf{KeyGen}(\mathsf{Msk}, \vec{y})$: Let $S_{\vec{y}}$ be the set of indices i such that $y_i \neq \star$. For each $i \in [\ell]$, the key generation algorithm chooses random $a_i \in Z_N$ such that $\sum_{i\in[\ell]} a_i = 0$ and random $R_i \in \mathbb{G}_{p_4}$. For $i \notin S_{\vec{y}}$ and $b \in \{0,1\}$, the algorithm chooses random $R_{i,b} \in \mathbb{G}_{p_4}$. Then for each $i \in [\ell]$, the key generation algorithm sets

$$Y_i = \begin{cases} g_{12}^{a_i/t_{i,y_i}} \cdot R_i, & \text{for } \; i \in S_{\vec{y}}; \\ g_{12}^{a_i} \cdot R_i, & \text{for } \; i \notin S_{\vec{y}}; \end{cases}$$

and, for each $i \notin S_{\vec{y}}$ and $b \in \{0,1\}$,

$$D_{i,b} = g_{12}^{a_i/t_{i,b}} \cdot R_{i,b}.$$

Finally the key generation algorithm returns the key $\mathsf{Sk}_{\vec{y}} = \left[(Y_i)_{i\in[\ell]}, (D_{i,b})_{i\notin S_{\vec{y}},b\in\{0,1\}}\right]$.

$\mathsf{Encrypt}(\mathsf{Pk}, \vec{x})$: The encryption algorithm chooses random $s \in \mathbb{Z}_N$ and $Z \in \mathbb{G}_{p_3}$ and, for each $i \in [\ell]$, random $Z_i \in \mathbb{G}_{p_3}$. The algorithm sets $X_0 = (g_1 R)^s \cdot Z$ and, for each $i \in [\ell]$, $X_i = (T_{i,x_i})^s Z_i$. The algorithm returns the ciphertext

$$\mathsf{Ct} = \left[X_0, (X_i)_{i\in[\ell]}\right].$$

We stress that, unlike the HVE, a ciphertext for HHVE contains element X_0.

$\mathsf{Test}(\mathsf{Ct}, \mathsf{Sk}_{\vec{y}})$: The test algorithm computes

$$T = \mathbf{e}(X_0, \prod_{i\notin S_{\vec{y}}} Y_i) \cdot \prod_{i\in S_{\vec{y}}} \mathbf{e}(X_i, Y_i).$$

It returns TRUE if $T = 1$, FALSE otherwise.

$\mathsf{Delegate}(\mathsf{Pk}, \mathsf{Sk}_{\vec{y}}, \vec{y}, \vec{w})$: On input a secret key $\mathsf{Sk}_{\vec{y}} = \left[(Y'_i)_{i\in[\ell]}, (D'_{i,b})_{i\notin S_{\vec{y}},b\in\{0,1\}}\right]$ for vector $\vec{y}$, the delegation algorithm chooses random $z \in \mathbb{Z}_N$. For $i \in S_{\vec{w}}$, the algorithm chooses random $R_i \in \mathbb{G}_{p_4}$ and, for $i \notin S_{\vec{w}}$ and $b \in \{0,1\}$, random $R_{i,b} \in \mathbb{G}_{p_4}$.

The delegation algorithm for $i \in S_{\vec{w}}$ computes Y_i as

$$Y_i = \begin{cases} Y_i'^{z} R_i, & \text{if } y_i \neq \star; \\ D'^{z}_{i,w_i} R_i, & \text{if } y_i = \star. \end{cases}$$

Finally, for $i \notin S_{\vec{w}}$ and $b \in \{0,1\}$, the delegation algorithm sets

$$D_{i,b} = D'^{z}_{i,b} R_{i,b},$$

and returns the key $\mathsf{Sk}_{\vec{w}} = \left[(Y_i)_{i\in[\ell]}, (D_{i,b})_{i\notin S_{\vec{w}},b\in\{0,1\}}\right]$.

Notice that, for vectors $\vec{y}$ and $\vec{w}$ such that $\vec{w} \prec \vec{y}$, the distribution of the key $\mathsf{Sk}_{\vec{w}}$ for $\vec{w}$ output by KeyGen on input Msk and $\vec{w}$, and the distribution of the key for $\vec{w}$ output by $\mathsf{Delegate}$ on input

Pk, a key $\mathsf{Sk}_{\vec{y}}$, $\vec{y}$ and $\vec{w}$ coincide. Therefore, the correctness of the scheme for keys generated by KeyGen is sufficient for proving correctness for every key.

Furthermore, any delegation path starts from a secret key for a vector $\vec{y}$ created by running KeyGen. For any such a $\vec{y}$ and for any delegation path $\vec{w} = \vec{w}_0 \prec \vec{w}_1 \prec \ldots \prec \vec{w}_{n-1} \prec \vec{w}_n = \vec{y}$, the distribution of the keys for $\vec{w}$, obtained by following the delegation path, is identical to the distribution of the keys for the same vector obtained by delegation directly from $\vec{y}$.

Notice also that the distributions of $(\mathsf{Sk}_{\vec{y}}, \mathsf{Sk}_{\vec{w}})$ when $\mathsf{Sk}_{\vec{w}}$ is generated by KeyGen or by Delegate do differ. This makes the security proof more involved.

Correctness It is easy to verify the correctness of the scheme.

Remark 6.2 *Notice that the observations of the Remark 4.1 apply also to the pairs* $(\mathsf{Pk}, \mathsf{Msk})$ *and* $(\mathsf{Pk}', \mathsf{Msk}')$ *output by the* KeyGen *algorithm of the* HHVE *scheme and differing in the base* g_1 *and* $\hat{g}_1$ *of the* $\mathbb{G}_{p_1}$ *part.*

6.3 Security of our HHVE scheme

To prove security of our HHVE scheme, we rely on static Assumptions 1 and 3. For a probabilistic polynomial-time adversary $\mathcal{A}$ which makes q *Reveal* key queries, our proof of security will be structured as a sequence of $q+2$ games between $\mathcal{A}$ and a challenger $\mathcal{C}$. The first game, $\mathsf{Game}_{\mathsf{Real}}$, is the real HHVE security game described in the previous section. The remaining games, called $\mathsf{Game}_0, \ldots, \mathsf{Game}_q$, are described (and shown indistinguishable) in the following.

Description of Game_0. Game_0 is like $\mathsf{Game}_{\mathsf{Real}}$, except that $\mathcal{C}$ uses g_2 instead of g_1 to construct the public parameters Pk given to $\mathcal{A}$. Specifically,

Setup. $\mathcal{C}$ chooses random $g_1 \in \mathbb{G}_{p_1}, g_2 \in \mathbb{G}_{p_2}, R, g_3 \in \mathbb{G}_{p_3}, g_4 \in \mathbb{G}_{p_4}$. For $i \in [\ell]$ and $b \in \{0,1\}$, $\mathcal{C}$ chooses random $t_{i,b} \in Z_N$ and $R_{i,b} \in \mathbb{G}_{p_3}$ and sets $T_{i,b} = g_2^{t_{i,b}} \cdot R_{i,b}$ and $T'_{i,b} = g_1^{t_{i,b}} \cdot R_{i,b}$, Then $\mathcal{C}$ sets $\mathsf{Pk} = [N, g_3, g_4, g_1 \cdot R, (T_{i,b})_{b\in\{0,1\}, i\in[\ell]}]$ and $\mathsf{Pk}' = [N, g_3, g_4, g_1 \cdot R, (T'_{i,b})_{b\in\{0,1\}, i\in[\ell]}]$. $\mathcal{C}$ starts the interaction with $\mathcal{A}$ on input Pk.

Key Query Answering. Like in $\mathsf{Game}_{\mathsf{Real}}$.

Challenge construction. $\mathcal{C}$ picks one of the two challenge vectors provided by $\mathcal{A}$ and encrypts it with respect to public parameters Pk'.

6.3.1 Proof of indistinguishability of $\mathsf{Game}_{\mathsf{Real}}$ and Game_0

Lemma 6.3 *Suppose there exists a PPT algorithm $\mathcal{A}$ such that $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_{\mathsf{Real}}} - \mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_0} = \epsilon$. Then, there exists a PPT algorithm $\mathcal{B}$ with advantage ϵ in breaking Assumption* 1.

PROOF. We show a PPT algorithm $\mathcal{B}$ which receives $(\mathcal{I}, A_3, A_4, A_{13}, A_{12})$ and T and, depending on the nature of T, simulates $\mathsf{Game}_{\mathsf{Real}}$ or Game_0 with $\mathcal{A}$. This suffices to prove the Lemma.

Setup. $\mathcal{B}$ starts by constructing public parameters Pk and Pk' in the following way. $\mathcal{B}$ sets $g_3 = A_3, g_{12} = A_{12}, g_4 = A_4$ and, for $i \in [\ell]$ and $b \in \{0,1\}$, $\mathcal{B}$ 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 $\mathsf{Pk} = [N, g_3, g_4, A_{13}, (T_{i,b})_{i\in[\ell], b\in\{0,1\}}]$, $\mathsf{Msk} = [g_{12}, (t_{i,b})_{i\in[\ell], b\in\{0,1\}}]$, $\mathsf{Pk}' = [N, g_3, g_4, A_{13}, (T'_{i,b})_{i\in[\ell], b\in\{0,1\}}]$, and $S = \emptyset$ and starts the interaction with $\mathcal{A}$ on input Pk.

Notice that Pk has the same distribution of the public parameters seen by $\mathcal{A}$ in $\mathsf{Game}_{\mathsf{Real}}$ and Game_0.

Key Query Answering. $\mathcal{B}$ handles the $\mathcal{A}$'s queries in the following way.

- *Create*. $\mathcal{B}$ handles a *Create* query on vector $\vec{y}$ by setting $\mathsf{Sk}_{\vec{y}} = \mathsf{KeyGen}(\mathsf{Msk}, \vec{y})$. Then, $\mathcal{B}$ add $\mathsf{Sk}_{\vec{y}}$ to S an gives $\mathcal{A}$ a reference to $\mathsf{Sk}_{\vec{y}}$.

- *Delegate*. $\mathcal{B}$ handles a *Delegate* query on secret key $\mathsf{Sk}_{\vec{y}}$, vector $\vec{y}$ and $\vec{w}$ such that $\vec{w} \prec \vec{y}$, by setting $\mathsf{Sk}_{\vec{w}} = \mathsf{Delegate}(\mathsf{Pk}, \mathsf{Sk}_y, \vec{y}, \vec{w})$. Then, $\mathcal{B}$ add $\mathsf{Sk}_{\vec{w}}$ to S an gives $\mathcal{A}$ a reference to $\mathsf{Sk}_{\vec{w}}$.

- *Reveal*. $\mathcal{B}$ simply removes the requested secret key from S and gives it to $\mathcal{A}$.

Challenge Construction. The challenge is created by $\mathcal{B}$ by picking one of the two vectors provided by $\mathcal{A}$, let us call it $\vec{x}$, and by encrypting it by running the $\mathsf{Encrypt}$ algorithm on input $\vec{x}$ and Pk'.

Let us write A_{13} as $A_{13} = \hat{h}_1 \cdot \hat{h}_3$ for $\hat{h}_1 \in \mathbb{G}_{p_1}$ and $\hat{h}_3 \in \mathbb{G}_{p_3}$.

Now suppose $T \in \mathbb{G}_{p_1 p_3}$, and thus $T = h_1 \cdot h_3$ for $h_1 \in \mathbb{G}_{p_1}$ and $h_3 \in \mathbb{G}_{p_3}$. Notice that Pk and Pk' are as in the hypothesis of Remark 6.2 (with $g_1 = h_1$ and $\hat{g}_1 = \hat{h}_1$) and thus the challenge given by $\mathcal{C}$ to $\mathcal{A}$ has the same distribution as an encryption of $\vec{x}$ with Pk. We can thus conclude that in this case $\mathcal{C}$ has simulated $\mathsf{Game}_{\mathsf{Real}}$ with $\mathcal{A}$.

Let us discuss now the case $T \in \mathbb{G}_{p_2 p_3}$. In this case the public parameters Pk provided by $\mathcal{B}$ have the same distribution as the public parameters produced by $\mathcal{C}$ in Game_0. Therefore, $\mathcal{C}$ is simulating Game_0 for $\mathcal{A}$.

This concludes the proof of the lemma. $\square$

Description of Game_k for $1 \leq k \leq q$. The Setup Phase and the Challenge Construction query of each of these games are like in Game_0. The first k *Reveal* queries issued by $\mathcal{A}$ are instead answered by $\mathcal{C}$ by returning keys whose $\mathbb{G}_{p_1}$ parts are random. All remaining *Reveal* queries are answered like in Game_0. We stress that the $\mathbb{G}_{p_2}$ parts of all answers are like in Game_0.

More precisely, the Key Query are handled by $\mathcal{C}$ in the following way. $\mathcal{C}$ starts by initializing the set S to the empty set and the *query counter* v and the *reveal query counter* Rv equal to 0.

- *Create*$(\vec{y})$: $\mathcal{C}$ increments v and, for each $i \in [\ell]$, chooses random $a_{\mathsf{v},i} \in \mathbb{Z}_N$ such that $\sum_{i=1}^{\ell} a_{\mathsf{v},i} = 0$ and adds the tuple $(\mathsf{v}, \vec{y}, (a_{\mathsf{v},1}, \ldots, a_{\mathsf{v},\ell}))$ to the set S.

  $\mathcal{C}$ returns v to $\mathcal{A}$.

- *Delegate*$(\mathsf{v}', \vec{w})$: For *Delegate* key query on vector $\vec{w}$, $\mathcal{C}$ increments v and adds the tuple $(\mathsf{v}, \vec{w}, \mathsf{v}')$ to the set S.

  $\mathcal{C}$ returns v to $\mathcal{A}$.

- *Reveal*(v'): Suppose entry v' in S refers to key $\mathsf{Sk}_{\vec{w}}$ which is the the result of a delegation path $\vec{w} = \vec{w}_0 \prec \vec{w}_1 \prec \ldots \prec \vec{w}_n = \vec{y}$ of length $n \geq 0$ starting from key $\mathsf{Sk}_{\vec{y}}$ created as result of the v''-th *Create* key query.

  $\mathcal{C}$ chooses random $z \in \mathbb{Z}_N$ and, for each $i \in [\ell]$, random $c_i \in \mathbb{Z}_N$ and $R_i \in \mathbb{G}_{p_4}$. Moreover for each $i \notin S_{\vec{w}}$ and $b \in \{0, 1\}$, $\mathcal{C}$ chooses random $R_{i,b} \in \mathbb{G}_{p_4}$.

$\mathcal{C}$ increments Rv. If $\mathsf{Rv} \leq k$, then for each $i \in [\ell]$, $\mathcal{C}$ sets

$$Y_i = \begin{cases} g_1^{c_i} \cdot g_2^{z a_{\mathsf{v}'',i}/t_{i,w_i}} \cdot R_i, & \text{if} \quad i \in S_{\vec{w}}; \\ g_1^{c_i} \cdot g_2^{z a_{\mathsf{v}'',i}} \cdot R_i, & \text{if} \quad i \notin S_{\vec{w}}; \end{cases}$$

and, for each $i \notin S_{\vec{w}}$ and for each $b \in \{0,1\}$, $\mathcal{C}$ sets

$$D_{i,b} = g_1^{c_i} \cdot g_2^{z a_{\mathsf{v}'',i}/t_{i,b}} \cdot R_{i,b}.$$

If instead $\mathsf{Rv} > k$, then for each $i \in [\ell]$, $\mathcal{C}$ sets

$$Y_i = \begin{cases} g_{12}^{z a_{\mathsf{v}'',i}/t_{i,w_i}} \cdot R_i, & \text{if} \quad i \in S_{\vec{w}}; \\ g_{12}^{z a_{\mathsf{v}'',i}} \cdot R_i, & \text{if} \quad i \notin S_{\vec{w}}; \end{cases}$$

and, for each $i \notin S_{\vec{w}}$ and for each $b \in \{0,1\}$, $\mathcal{C}$ sets

$$D_{i,b} = g_{12}^{z a_{\mathsf{v}'',i}/t_{i,b}} \cdot R_{i,b}.$$

Finally, $\mathcal{C}$ returns the key $\mathsf{Sk}_{\vec{w}}$ consisting of the Y_i's and the $D_{i,b}$'s.

6.3.2 Proof of indistinguishability of Game_{k-1} and Game_k

Lemma 6.4 *Suppose there exists a PPT algorithm $\mathcal{A}$ such that $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_{k-1}} - \mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_k} = \epsilon$. Then, there exists a PPT algorithm $\mathcal{B}$ with advantage at least $\epsilon/(2\ell)$ in breaking Assumption* 3.

Proof. We show a PPT algorithm $\mathcal{B}$ which receives $(\mathcal{I}, A_1, A_2, A_3, A_4, A_1^{\alpha} \cdot B_4, A_1^{\beta} \cdot C_4, A_1^{\gamma} \cdot D_4, A_1^{\alpha\gamma} \cdot E_4, A_1^{\alpha\beta\gamma} \cdot F_4)$ and T and, depending on the nature of T, simulates Game_{k-1} or Game_k with $\mathcal{A}$. This suffices to prove the Lemma.

$\mathcal{B}$ starts by guessing the index j such that the j-th bit $w_j^{(k)}$ of the k-th *Reveal* query $\vec{w}^{(k)}$ is different from $\star$ and different from the j-th bit x_j of the challenge vectors provided by $\mathcal{A}$ that $\mathcal{B}$ uses to construct the challenge ciphertext. Notice that such an index j always exists and that the probability that $\mathcal{B}$ correctly guesses j and $w_j^{(k)}$ (and thus $x_j = 1 - w_j^{(k)}$) is at least $1/(2\ell)$, independently from the view of $\mathcal{A}$. Notice that, if during the simulation this is not the case, then $\mathcal{B}$ aborts the simulation and fails. We next describe and prove the correctness of the simulation under the assumption that $\mathcal{B}$'s initial guess is correct.

Setup. $\mathcal{B}$ sets $g_1 = A_1$, $g_2 = A_2$, $g_3 = A_3$, $g_4 = A_4$ and $g_{12} = A_1 \cdot A_2$.

$\mathcal{B}$ chooses random $R \in \mathbb{G}_{p_3}$ and, for $i \in [\ell] \setminus \{j\}$ and $b \in \{0,1\}$, $\mathcal{B}$ chooses random $t_{i,b} \in \mathbb{Z}_N$ and $R_{i,b} \in \mathbb{G}_{p_3}$. Then $\mathcal{B}$ sets $T_{i,b} = g_2^{t_{i,b}} \cdot R_{i,b}$. Moreover, $\mathcal{B}$ chooses random $t_{j,1-w_j^{(k)}}, r_{j,w_j^{(k)}} \in Z_N$ and $R_{j,1-w_j^{(k)}}, R_{j,w_j^{(k)}} \in \mathbb{G}_{p_3}$ and sets $T_{j,1-w_j^{(k)}} = g_2^{t_{j,1-w_j^{(k)}}} \cdot R_{j,1-w_j^{(k)}}$ and $T_{j,w_j^{(k)}} = g_2^{r_{j,w_j^{(k)}}} \cdot R_{j,w_j^{(k)}}$. These settings determine public parameters $\mathsf{Pk} = [N, g_3, g_4, g_1 R, (T_{i,b})_{i \in [\ell], b \in \{0,1\}}]$.

In addition, for $i \in [\ell] \setminus \{j\}$ and $b \in \{0,1\}$ and for $i = j$ and $b = 1 - w_j^{(k)}$, $\mathcal{B}$ chooses random $R'_{i,b} \in \mathbb{G}_{p_3}$ and sets $T'_{i,b} = g_1^{t_{i,b}} \cdot R'_{i,b}$. The value of $T'_{j,w_j^{(k)}}$ remains unspecified. As we shall see below, $\mathcal{B}$ will implicitly set $T'_{j,w_j^{(k)}} = g_1^{1/\beta} \cdot R'_{j,w_j^{(k)}}$ for some random $R'_{j,w_j^{(k)}} \in \mathbb{G}_{p_3}$.

$\mathcal{B}$ starts the interaction with $\mathcal{A}$ on input Pk. Notice that Pk has the same distribution as the public parameters seen by $\mathcal{A}$ in Game_{k-1} and Game_k.

Key Query Answering. $\mathcal{B}$ handles the *Create* and *Delegate* queries as prescribed in Game_{k-1} and Game_k. For the *Reveal* queries $\mathcal{B}$ proceeds as follow.

$\mathcal{B}$ handles the first $k-1$ *Reveal* key queries as follow. Suppose $\mathcal{A}$ requests the key for $\vec{w}$ belonging to a delegation path, of length greater or equal to 0, which starts from the v-th *Create* key query. $\mathcal{B}$ chooses random $z \in \mathbb{Z}_N$ and, for $i \in [\ell]$, random $c_i \in \mathbb{Z}_N$ and $R_i \in \mathbb{G}_{p_4}$ and, for $i \notin S_{\vec{w}}$ and $b \in \{0,1\}$, random $R_{i,b} \in \mathbb{G}_{p_4}$. Then $\mathcal{B}$ computes, for each $i \in [\ell] \setminus \{j\}$,

$$Y_i = \begin{cases} g_1^{c_i} \cdot g_2^{z a_{v,i}/t_{i,w_i}} \cdot R_i, & \text{if} \quad i \in S_{\vec{w}}; \\ g_1^{c_i} \cdot g_2^{z a_{v,i}} \cdot R_i, & \text{if} \quad i \notin S_{\vec{w}}; \end{cases}$$

and, for each $i \neq j$ and $i \notin S_{\vec{w}}$

$$D_{i,b} = g_1^{c_i} \cdot g_2^{z a_{v,i}/t_{i,b}} \cdot R_{i,b}.$$

Moreover, if $j \in S_{\vec{w}}$ then $\mathcal{B}$ sets

$$Y_j = \begin{cases} g_1^{c_j} \cdot g_2^{z a_{v,j}/r_{j,w_j^{(k)}}} R_j, & \text{if } w_j = w_j^{(k)}; \\ g_1^{c_j} \cdot g_2^{z a_{v,j}/t_{j,x_j}} R_j, & \text{if } w_j = x_j. \end{cases}$$

If instead $j \notin S_w$ then $\mathcal{B}$ sets

$$Y_j = g_1^{c_j} \cdot g_2^{z a_{v,j}} \cdot R_j, \quad D_{j,w_j^{(k)}} = g_1^{c_j} \cdot g_2^{z a_{v,j}/r_{j,w_j^{(k)}}} \cdot R_{j,w_j^{(k)}}, \quad D_{j,1-w_j^{(k)}} = g_1^{c_j} \cdot g_2^{z a_{v,j}/t_{j,x_j}} \cdot R_{j,x_j},$$

This terminates the description of how $\mathcal{B}$ handles the first $k-1$ *Reveal* queries. We observe that the $\mathbb{G}_{p_1}$ parts of the keys returned in the first $k-1$ *Reveal* queries are random, whereas the $\mathbb{G}_{p_2}$ parts are correctly computed. Therefore, we can conclude that the answer to the first $k-1$ *Reveal* queries have the same distribution as in games Game_{k-1} and Game_k.

Let us now describe how $\mathcal{B}$ handles the k-th *Reveal* key query on a vector $\vec{w}^{(k)}$ belonging to a delegation path, possibly of length 0, which starts from the v-th *Create* key query. Let h be an index such that $h \neq j$ and $w_h^{(k)} \neq \star$. Such an index always exists by our assumption that all keys are for vectors with at least two entries different from $\star$.

$\mathcal{B}$ chooses random $z \in \mathbb{Z}_N$ and, for each $i \in [\ell]$, random $R_i \in \mathbb{G}_{p_4}$ and, for each $i \neq j, h$ such that $i \notin S$ and $b \in \{0,1\}$, random $R_{i,b} \in \mathbb{G}_{p_4}$. Then, for each $i \in [\ell] \setminus \{j,h\}$, $\mathcal{B}$ sets

$$Y_i = \begin{cases} g_{12}^{z a_{v,i}/t_{i,w_i}} R_i, & \text{if} \quad i \in S_{\vec{w}^{(k)}}; \\ g_{12}^{z a_{v,i}} R_i, & \text{if} \quad i \notin S_{\vec{w}^{(k)}}. \end{cases}$$

Moreover for $i \neq \{j,h\}$ such that $i \notin S_{\vec{w}^{(k)}}$ and $b \in \{0,1\}$, $\mathcal{B}$ sets

$$D_{i,b} = g_{12}^{z a_{v,i}/t_{i,b}} \cdot R_{i,b}.$$

Finally, $\mathcal{B}$ computes $s = \sum_{i \in [\ell] \setminus \{j,h\}} a_{v,i}$ and sets

$$Y_j = T^z \cdot g_2^{z a_{v,j}/r_{j,w_j^{(k)}}} \cdot R_j$$

and

$$Y_h = (A_1^{\alpha} B_4)^{-z/t_{h,w_h^{(k)}}} \cdot g_1^{-zs/t_{h,w_h^{(k)}}} \cdot g_2^{-z(s+a_j)/t_{h,w_h^{(k)}}} \cdot R_h.$$

This terminates the description of how $\mathcal{B}$ handles the k-th *Reveal* query.
Suppose now that $T = A_1^{\alpha\beta} \cdot G_4$ and thus by our settings, we have that

$$Y_j = g_1^{z\alpha\beta} \cdot g_2^{za_{v,j}/r_{j,w_j^{(k)}}} \cdot G_4 \cdot R_j.$$

By the Chinese Remainder Theorem, there exists $a_j \in Z_N$ such that

$$\begin{aligned} a_j &\equiv \alpha \mod p_1 \\ a_j &\equiv a_{v,j} \mod p_2 \end{aligned}$$

and $t_{j,w_j^{(k)}} \in Z_N$ such that

$$\begin{aligned} t_{j,w_j^{(k)}} &\equiv 1/\beta \mod p_1 \\ t_{j,w_j^{(k)}} &\equiv r_{j,w_j^{(k)}} \mod p_2. \end{aligned}$$

We stress that $\mathcal{B}$ does not know a_j and $t_{j,w_j^{(k)}} \in Z_N$ and does not need these values to perform its computation. By setting, $a_i = a_{v,i}$ for $i \neq j, h$, $a_h = -(s + a_j)$ and by the definition of a_j, we can write Y_j and Y_h as

$$Y_j = g_{12}^{za_j/t_{j,w_j^{(k)}}} \cdot (G_4 \cdot R_j) \quad \text{and} \quad Y_h = g_{12}^{za_h/t_{h,w_h^{(k)}}} \cdot R_h.$$

Therefore, all the exponents of g_{12} are equal to the exponents of the key produced by v-th *Create* query multiplied by the common a value z and we can conclude that in this case the answer to the k-th query of $\mathcal{A}$ is distributed as in Game_{k-1}.

On the other hand if T is random in $\mathbb{G}_{p_1p_4}$ then the $\mathbb{G}_{p_1}$ parts of the Y_i's are random and independent thus the answer to the k-th *Reveal* query of $\mathcal{A}$ is distributed as in Game_k.

Let us now describe how $\mathcal{B}$ handles the remaining $(q - k)$ *Reveal* queries. We fix notation by denoting with $\vec{w}$ the vector for which $\mathcal{A}$ asks to reveal the key and we suppose that the key is part of a delegation path that starts from the key created by the v-th *Create* key query. We distinguish two cases depending on whether the key of the k-th *Reveal* key query is derived from the same *Create* key query and start from the the case in which it is not.

In this case, $\mathcal{B}$ chooses random $z \in \mathbb{Z}_N$ and, for each $i \in [\ell]$, $\mathcal{B}$ chooses random $R_i \in \mathbb{G}_{p_4}$. Furthermore for each $i \neq j$ such that $i \in S_{\vec{w}}$ and for each $b \in \{0, 1\}$, $\mathcal{B}$ chooses random $R_{i,b} \in \mathbb{G}_{p_4}$. Then $\mathcal{B}$ computes

$$Y_i = \begin{cases} g_{12}^{za_{v,i}/t_{i,w_i}} \cdot R_i, & \text{if } i \in S_{\vec{w}}; \\ g_{12}^{za_{v,i}} \cdot R_i, & \text{if } i \notin S_{\vec{w}}; \end{cases}$$

and, for each $i \neq j$ such that $i \notin S_{\vec{w}}$ and $b \in \{0, 1\}$, $\mathcal{B}$ computes

$$D_{i,b} = g_{12}^{za_{v,i}/t_{i,b}} \cdot R_{i,b}.$$

For index j, $\mathcal{B}$ sets

$$Y_j = \begin{cases} \left(A_1^{\beta} C_4\right)^{za_{v,j}} g_2^{za_{v,j}/r_{j,w_j^{(k)}}} R_j, & \text{if } w_j = w_j^{(k)}; \\ g_{12}^{za_{v,j}/t_{j,1-w_j^{(k)}}} R_j, & \text{if } w_j = 1 - w_j^{(k)}; \\ g_{12}^{za_{v,j}} \cdot R_j, & \text{if } w_j = \star; \end{cases}$$

Finally, if $w_j = \star$ then $\mathcal{B}$ sets

$$D_{j,w_j^{(k)}} = \left(A_1^{\beta} C_4\right)^{za_{v,j}} \cdot g_2^{za_{v,j}/r_{j,w_j^{(k)}}} R_{j,w_j^{(k)}},\ D_{j,1-w_j^{(k)}} = g_{12}^{za_{v,j}/t_{j,1-w_j^{(k)}}} \cdot R_{j,x_j}.$$

First of all, we stress that $\mathcal{B}$ can carry out the above computation, since all needed values have either been chosen by $\mathcal{B}$ as part of the setup or, like $\mathcal{A}_1^{\beta} \cdot C_4$, are part of the challenge instance that $\mathcal{B}$ is trying to solve. By the settings above, by the definition of $t_{j,w_j^{(k)}}$ and a_j, by setting $a_i = a_{v,i}$ for all $i \neq j$, we can write Y_i for each $i \in [\ell]$ as

$$Y_i = \begin{cases} g_{12}^{\frac{za_i}{t_{i,w_i}}} \cdot R_i', & \text{if } \ w_i \in \{0,1\}; \\ g_{12}^{za_i} \cdot R_i', & \text{if } \ w_i = \star; \end{cases}$$

for a random $R_i' \in \mathbb{G}_{p_4}$. Specifically, $R_i' = R_i$ except for index j and for the case $w_j = w_j^{(k)}$. In this latter case we have $R_j' = R_j \cdot C_4^{za_{v,j}}$. By observing that the a_i sum up to 0, and that they all appear multiplied by the same random value z, we can conclude that the Y_i's are distributed as in Game_{k-1} and Game_k. A similar reasoning shows that the $D_{i,b}$'s are also correctly distributed.

Let us now conclude the proof by discussing the case in which the *Reveal* key query for $\vec{w}$ is for a key whose starting point in the delegation path corresponds to the v-th *Create* key query and it is the same as the one of the k-th *Reveal* key query.

Fix an index $h \neq j$. $\mathcal{B}$ chooses random $z \in \mathbb{Z}_N$ and, for $i \in [\ell]$ and $b \in \{0,1\}$, random $R_i, R_{i,b} \in \mathbb{G}_{p_4}$. Then, for each $i \neq j, h$, $\mathcal{B}$ sets

$$Y_i = \begin{cases} (A_1^{\gamma} D_4)^{za_{v,i}/t_{i,w_i}} \cdot g_2^{za_{v,i}/t_{i,w_i}} \cdot R_i, & \text{if } i \in S_{\vec{w}}; \\ (A_1^{\gamma} D_4)^{za_{v,i}} g_2^{za_{v,i}} R_i, & \text{if } i \notin S_{\vec{w}}; \end{cases}$$

and, for $i \neq j, h$ such that $i \notin S_{\vec{w}}$, $\mathcal{B}$ sets for $b \in \{0,1\}$

$$D_{i,b} = (A_1^{\gamma} D_4)^{za_{v,i}/t_{i,b}} \cdot g_2^{za_{v,i}/t_{i,b}} \cdot R_{i,b}.$$

Moreover, $\mathcal{B}$ computes Y_j as

$$Y_j = \begin{cases} \left(A_1^{\alpha\beta\gamma} F_4\right)^{z} \cdot g_2^{za_{v,j}/r_{j,w_j^{(k)}}} \cdot R_j, & \text{if } \ w_j = w_j^{(k)}; \\ (A_1^{\alpha\gamma} E_4)^{z/t_{j,1-w_j^{(k)}}} \cdot g_2^{za_{v,j}/t_{j,1-w_j^{(k)}}} \cdot R_j, & \text{if } w_j = 1 - w_j^{(k)}; \\ (A_1^{\alpha\gamma} E_4)^{z} \cdot g_2^{za_{v,j}} \cdot R_j, & \text{if } w_j = \star. \end{cases}$$

and, if $w_j = \star$, $\mathcal{B}$ computes

$$\begin{aligned} D_{j,w_j^{(k)}} &= (A_1^{\alpha\beta\gamma} F_4)^{z} \cdot g_2^{za_{v,j}/r_{j,w_j^{(k)}}} \cdot R_{j,w_j^{(k)}} \\ D_{j,1-w_j^{(k)}} &= (A_1^{\alpha\gamma} E_4)^{z/t_{j,1-w_j^{(k)}}} \cdot g_2^{za_{v,j}/t_{j,1-w_j^{(k)}}} \cdot R_{j,1-w_j^{(k)}}. \end{aligned}$$

Finally, $\mathcal{B}$ sets $s = \sum_{i \in [\ell] \setminus \{j,h\}} a_{v,i}$, and computes

$$Y_h = \begin{cases} (A_1^{\alpha\gamma} E_4)^{-z/t_{h,w_h}} \cdot (A_1^{\gamma} C_4)^{-zs/t_{h,w_h}} \cdot g_2^{-z(s+a_{v,j})/t_{h,w_h}} \cdot R_h, & \text{if } \; h \in S_{\vec{w}}; \\ (A_1^{\alpha\gamma} E_4)^{-z} \cdot (A_1^{\gamma} C_4)^{-zs} \cdot R_h, & \text{if } \; h \notin S_{\vec{w}}. \end{cases}$$

and, if $h \notin S_{\vec{w}}$, $\mathcal{B}$ computes

$$D_{h,b} = (A_1^{\alpha\gamma} E_4)^{-z/t_{h,b}} \cdot (A_1^{\gamma} C_4)^{-zs/t_{h,b}} \cdot g_2^{-z(s+a_{v,j})/t_{h,b}} \cdot R_{h,b}$$

for $b \in \{0,1\}$. This concludes the description of how $\mathcal{B}$ replies to the *Reveal* key queries and we stress that $\mathcal{B}$ can carry out the prescribed computation as all needed values either have been chosen by $\mathcal{B}$ in the setup or are part of the challenge for Assumption 3 that $\mathcal{B}$ is trying to break. Let us now verify that the answer provided by $\mathcal{B}$ is correct also in this case.

By the Chinese Remainder Theorem, there exists $z' \in \mathbb{Z}_N$ such that

$$\begin{aligned} z' \quad &\equiv z \cdot \gamma \bmod \; p_1 \\ z' \quad &\equiv z \quad \bmod \; p_2 \end{aligned}$$

Again we stress that $\mathcal{B}$ does not need to know z' to perform its computation. By the above settings, by the definition of $t_{j,w_j^{(k)}}$ and a_j, by setting $a_i = a_{v,i}$ for all $i \neq j, h$, and by setting $a_h = \sum_{i \neq h} a_j$, we can write, for each $i \in [\ell]$, Y_i as

$$Y_i = \begin{cases} g_{12}^{z' a_i / t_{i,w_i}} \cdot R_i', & \text{if } i \in S_{\vec{w}}; \\ g_{12}^{z' a_i} \cdot R_i', & \text{if } i \notin S_{\vec{w}}; \gamma \end{cases}$$

for some random $R_i' \in \mathbb{G}_{p_4}$. Therefore, the exponents of g_{12} are the same as the ones of the key created as effect of v-th *Create* key query multiplied by a common value z'. As similar reasoning holds for the $D_{i,b}$'s and we can therefore conclude that the answer provided by $\mathcal{B}$ has the same distribution as in Game_k and Game_{k-1}.

Challenge construction. $\mathcal{B}$ creates the challenge ciphertext by running algorithm $\mathsf{Encrypt}$ on input one randomly chosen challenge vector $\vec{x}$ provide by $\mathcal{A}$ and public parameters Pk'. Notice that under the assumption that $\mathcal{B}$ has correctly guessed $w_j^{(k)}$ we have that $x_j \neq w_j^{(k)}$, and this Pk' contains all the values needed for computing an encryption of $\vec{x}$. Therefore the challenge ciphertext is distributed exactly like in Game_{k-1} and Game_k. □

6.3.3 Game_q gives no advantage

We observe that in Game_q the $\mathbb{G}_{p_1}$ part of the challenge ciphertext is the only one depending on η. In addition notice that the $g_1 \cdot R_3$ is the only component of the public parameters which contains a $\mathbb{G}_{p_1}$ part but it is independent from η. Thus, it gives no advantage to the adversary and moreover the answer to the key queries have random and independent $\mathbb{G}_{p_1}$ part. Therefore we can conclude that for all adversaries $\mathcal{A}$, $\mathsf{Adv}^{\mathcal{A}}_{\mathsf{Game}_q} = 0$. We have thus proved.

Theorem 6.5 *If Assumptions* 1 *and* 3 *hold, then our HHVE scheme is secure.*

7 Reductions

7.1 Reducing 3-CNF to HVE.

In this section we show how to construct a ξ-secure Encryption scheme for the class of Boolean predicates that can be expressed as a 3-CNF formula from a ξ-secure HVE scheme. We consider formulae Φ in 3-CNF over n variables in which each clause $C \in \Phi$ contains exactly 3 distinct variables. We call such a clause *admissible* and denote by $\mathbb{C}_n$ the set of all admissible clauses over the n variables $x_1, \ldots, x_n$ and set $M_n = |\mathbb{C}|$. Notice that $M_n = \Theta(n^3)$. We also fix a canonical ordering $C_1, \ldots, C_{M_n}$ of the clauses in $\mathbb{C}_n$.

Let $\mathcal{H} = (\mathsf{Setup}_{\mathcal{H}}, \mathsf{KeyGen}_{\mathcal{H}}, \mathsf{Encrypt}_{\mathcal{H}}, \mathsf{Test}_{\mathcal{H}})$ be an HVE scheme and construct a 3-CNF scheme $\mathsf{3CNF} = (\mathsf{Setup}_{\mathsf{3CNF}}, \mathsf{KeyGen}_{\mathsf{3CNF}}, \mathsf{Encrypt}_{\mathsf{3CNF}}, \mathsf{Test}_{\mathsf{3CNF}})$ in the following way:

$\mathsf{Setup}_{\mathsf{3CNF}}(1^\lambda, 1^n)$: the algorithm runs the $\mathsf{Setup}_{\mathcal{H}}$ algorithm on input 1^λ and 1^{M_n} and returns its output.

$\mathsf{KeyGen}_{\mathsf{3CNF}}(\mathsf{Msk}, \Phi)$: For a formula $\Phi \in \mathbb{C}_n$, the key generation algorithm constructs vector $\vec{y} \in \{0, 1, \star\}^{M_n}$ by setting, for each $i \in \{1, \ldots, M_n\}$,

$$y_i = \begin{cases} 1, & \text{if } C_i \in \Phi; \\ \star, & \text{otherwise.} \end{cases}$$

We denote this transformation by $y = \mathsf{FEncode}(\Phi)$. Then the key generation algorithm returns $\mathsf{Sk}_\Phi = \mathsf{KeyGen}_{\mathcal{H}}(\mathsf{Msk}, \vec{y})$.

$\mathsf{Encrypt}_{\mathsf{3CNF}}(\mathsf{Pk}, \vec{z})$: 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:

$$x_i = \begin{cases} 1, & \text{if } C_i \text{ is satisfied by } \vec{z}; \\ 0, & \text{if } C_i \text{ is not satisfied by } \vec{z}. \end{cases}$$

We denote this transformation by $\vec{x} = \mathsf{AEncode}(\vec{z})$. Then the encryption algorithm returns

$$\mathsf{Ct} = \mathsf{Encrypt}_{\mathcal{H}}(\mathsf{Pk}, \vec{x}).$$

$\mathsf{Test}_{\mathsf{3CNF}}(\mathsf{Sk}_\Phi, \mathsf{Ct})$: The algorithm runs the $\mathsf{Test}_{\mathcal{H}}$ algorithm on input Sk_Φ and Ct and returns its output.

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

ξ-security. Let us now verify that the reduction preserves ξ-security, for $\xi = 0, 1$. Let $\mathcal{A}$ be a ξ-adversary for $\mathsf{3CNF}$ that tries to break the scheme for n variables and consider the following adversary $\mathcal{B}$ for $\mathcal{H}$ that uses $\mathcal{A}$ as a subroutine and tries to break a $\mathcal{H}$ with $\ell = M_n$ by interacting with challenger $\mathcal{C}$. $\mathcal{B}$ receives a public key Pk for $\mathcal{H}$ and passes it to $\mathcal{A}$ (notice that a randomly chosen public key for $\mathcal{H}$ has the same distribution of a randomly chosen public key for $\mathsf{3CNF}$). Whenever $\mathcal{A}$ asks for the key for formula Φ, $\mathcal{B}$ constructs $\vec{y} = \mathsf{FEncode}(\Phi)$ and asks $\mathcal{C}$ for a key $\mathsf{Sk}_{\vec{y}}$ for $\vec{y}$ and returns it to $\mathcal{A}$. When $\mathcal{A}$ asks for a challenge by providing truth assignments $\vec{z}_0$ and $\vec{z}_1$,

$\mathcal{B}$ simply computes $\vec{x}_0 = \mathsf{AEncode}(\vec{z}_0)$ and $\vec{x}_1 = \mathsf{AEncode}(\vec{z}_1)$ and gives the pair $(\vec{x}_0, \vec{x}_1)$ to $\mathcal{C}$. $\mathcal{B}$ then returns the challenge ciphertext obtained from $\mathcal{C}$ to $\mathcal{A}$. Finally, $\mathcal{B}$ outputs $\mathcal{A}$'s guess.

We observe that $\mathcal{B}$'s simulation is perfect. Indeed, we have that if for all $\mathcal{A}$'s queries Φ we have that $\mathsf{Satisfy}(\Phi, \vec{z}_0) = \mathsf{Satisfy}(\Phi, \vec{z}_1) = \xi$, then all $\mathcal{B}$'s queries $\vec{y}$ to $\mathcal{C}$ also have the property $\mathsf{Match}(\vec{y}, \vec{x}_0) = \mathsf{Match}(\vec{y}, \vec{x}_1) = \xi$. We can thus conclude that $\mathcal{B}$'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 7.1 *If Assumption* 1 *and* 2 *hold for generator* $\mathcal{G}$ *then there exists a* 0*-secure encryption scheme for the class of predicates that can be represented by* 3*-CNF formulae.*

If Assumption 1 *and* 4 *hold for generator* $\mathcal{G}$ *then there exists a* 1*-secure encryption scheme for the class of predicates that can be represented by* 3*-CNF formulae.*

Hierarchical HVE and Hierarchical 3-CNF. In a Hierarchical 3-CNF scheme the owner of the secret key for formula Φ can derive secret keys for formulae that can be obtained from Φ by adding extra clauses. The derivation does not need the master secret key. A similar reduction shows that one can construct 3-HCNF Encryption schemes starting from Hierarchical HVE. We omit further details.

7.2 Reducing 3-CNF 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 $\mathcal{H} = (\mathsf{Setup}_{\mathcal{H}}, \mathsf{KeyGen}_{\mathcal{H}}, \mathsf{Encrypt}_{\mathcal{H}}, \mathsf{Test}_{\mathcal{H}})$ be an HVE scheme and construct the predicate-only scheme $\vee = (\mathsf{Setup}_{\vee}, \mathsf{KeyGen}_{\vee}, \mathsf{Encrypt}_{\vee}, \mathsf{Test}_{\vee})$ for disjunctions in the following way:

$\mathsf{Setup}_{\vee}(1^\lambda, 1^n)$: the algorithm runs the $\mathsf{Setup}_{\mathcal{H}}$ algorithm on input 1^λ and 1^n.

$\mathsf{KeyGen}_{\vee}(\mathsf{Msk}, C)$: For a clause C, the key generation algorithm constructs vector $\vec{y} \in \{0, 1, \star\}^n$ in the following way. Let $\vec{w}$ be a truth assignment to the n variables that does not satisfy clause C. We stress that all such truth assignments give the same truth values to the variables appearing (in positive or negated form) in C. For each $i \in \{1, \ldots, n\}$, the algorithms sets:

$$y_i = \begin{cases} w_i, & \text{if the } i\text{-th variable appears in } C; \\ \star, & \text{otherwise.} \end{cases}$$

We denote this transformation by $\vec{y} = \mathsf{CEncode}(C)$. Then the key generation algorithm returns $\mathsf{Sk}_C = \mathsf{KeyGen}_{\mathcal{H}}(\mathsf{Msk}, \vec{y})$.

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

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

Correctness. Correctness follows from the observation that for a clause C and assignment $\vec{z}$, $\mathsf{Satisfy}(\vec{z}, C) = 1$ if and only if $\mathsf{Match}(\vec{z}, \mathsf{CEncode}(C)) = 0$.

Security. It is easy to see that if $\mathcal{H}$ is $(1-\xi)$-secure then $\vee$ is ξ-secure. Indeed notice that we can transform any ξ-adversary $\mathcal{A}$ for $\vee$ into a $(1-\xi)$-adversary $\mathcal{B}$ for $\mathcal{H}$ in the obvious way and that any ξ-query of $\mathcal{A}$ for a key for $\vee$ can be answered by making a $(1-\xi)$-query for $\mathcal{H}$. By applying the above reduction to the 0-secure and 1-secure HVE construction of the previous sections we obtain the following theorem.

Theorem 7.2 *If Assumption* 1 *and* 4 *hold for generator* $\mathcal{G}$ *then there exists a* 0*-secure encryption scheme for the class of predicates that can be represented by a disjunction.*

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

7.3 Reducing 3-DNF to 3-CNF

We observe that if Φ is a predicate represented by a 3-DNF formula then its negation $\bar{\Phi}$ can be represented by a 3-CNF formula. Therefore let $\mathsf{3CNF} = (\mathsf{Setup}_{\mathsf{3CNF}}, \mathsf{KeyGen}_{\mathsf{3CNF}}, \mathsf{Encrypt}_{\mathsf{3CNF}}, \mathsf{Test}_{\mathsf{3CNF}})$ and consider the following scheme $\mathsf{3DNF} = (\mathsf{Setup}_{\mathsf{3DNF}}, \mathsf{KeyGen}_{\mathsf{3DNF}}, \mathsf{Encrypt}_{\mathsf{3DNF}}, \mathsf{Test}_{\mathsf{3DNF}})$. The setup algorithm $\mathsf{Setup}_{\mathsf{3DNF}}$ is the same as $\mathsf{Setup}_{\mathsf{3CNF}}$. The key generation algorithm $\mathsf{Setup}_{\mathsf{3DNF}}$ for predicate Φ represented by a 3-DNF simply invokes the key generation algorithm $\mathsf{Setup}_{\mathsf{3CNF}}$ for $\bar{\Phi}$ that can be represented by a 3-CNF formula. The encryption algorithm $\mathsf{Encrypt}_{\mathsf{3DNF}}$ is the same as $\mathsf{Encrypt}_{\mathsf{3CNF}}$. The test algorithm $\mathsf{Test}_{\mathsf{3DNF}}$ on input ciphertext Ct and key for 3-DNF formula Φ (that is actually a for 3-CNF formula $\bar{\Phi}$) thus $\mathsf{Test}_{\mathsf{3CNF}}$ on Ct and the key and complements the result. Correctness can be easily argued. We notice that this reduction however does not preserve ξ-security but rather complements it. More precisely, for proving ξ-security we can easily see that any ξ-adversary for $\mathsf{3DNF}$ can be used to construct a $(1-\xi)$-adversary for $\mathsf{3CNF}$.

By combining the above reduction with the construction given by Theorem 7.1.

Theorem 7.3 *If Assumption* 1 *and* 2 *hold for generator* $\mathcal{G}$ *then there exists a* 1*-secure encryption scheme for the class of predicates represented by* 3*-DNF formulae.*

If Assumption 1 *and* 4 *hold for generator* $\mathcal{G}$ *then there exists a* 0*-secure encryption scheme for the class of predicates represented by* 3*-DNF formulae.*

8 Open problems and future work

We leave as a future work the implementation of a symmetric-key version of our encryption schemes (see for example [14]).

We proved the full security in two models: in the case in which the adversary can request keys which do not satisfy both the challenges (0-security, which is the only notion considered in [8] and subsequent works) and in the case in which the adversary can request keys which satisfy both the challenges (1-security). It would be interesting to have a construction that is secure against adversaries allowed to request keys which either satisfy both challenges or satisfy neither (match concealing model).

We gave a tight reduction that does not depend on the running time (and number of queries q) for the case of our 1-secure scheme. It is an open problem the design of a scheme with a tight security reduction for 0-secure schemes.

References

[1] Dan Boneh. Bilinear groups of composite order. In Tsuyoshi Takagi, Tatsuaki Okamoto, Eiji Okamoto, and Takeshi Okamoto, editors, *Pairing-Based Cryptography - Pairing 2007, First International Conference. Prooceedings*, volume 4575 of *Lecture Notes in Computer Science*, pages 39–56, Tokyo, Japan, July 2–4, 2007. Springer-Verlag, Berlin, Germany.

[2] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522, Interlaken, Switzerland, May 2–6, 2004. Springer-Verlag, Berlin, Germany.

[3] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 325–341, Cambridge, MA, USA, February 10–12, 2005. Springer-Verlag, Berlin, Germany.

[4] Dan Boneh and Brent Waters. Conjunctive, subset and range queries on encrypted data. In Salil P. Vadhan, editor, *TCC 2007: 4th Theory of Cryptography Conference*, volume 4392 of *Lecture Notes in Computer Science*, pages 535–554, Amsterdam, The Netherlands, February 21–24, 2007. Springer-Verlag, Berlin, Germany.

[5] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-Based Encryption for Fine-Grained Access Control for Encrypted Data. In *ACM CCS 06: 13th Conference on Computer and Communications Security*, pages 89–98, Alexandria, VA, USA, October 30 - November 3, 2006. ACM Press.

[6] Vincenzo Iovino and Giuseppe Persiano. Hidden-vector encryption with groups of prime order. In Steven D. Galbraith and Kenneth G. Paterson, editors, *Pairing-Based Cryptography - Pairing 2008, Second International Conference. Prooceedings*, volume 5209 of *Lecture Notes in Computer Science*, pages 75–88, Egham, UK, September 1–3, 2008. Springer-Verlag, Berlin, Germany.

[7] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate Encryption Supporting Disjunction, Polynomial Equations, and Inner Products. In Nigel Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 146–162, Istanbul, Turkey, April 13–17, 2008. Springer-Verlag, Berlin, Germany.

[8] Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, pages 62–91, French Riviera, France, May 10 –June 3, 2010. Springer-Verlag, Berlin, Germany.

[9] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure hibe with short ciphertexts. In Daniele Micciancio, editor, *TCC 2010: 7th Theory of Cryptography Conference*, volume 5978 of *Lecture Notes in Computer Science*, pages 455–479, Zurich, Switzerland, February 9–11, 2010. Springer-Verlag, Berlin, Germany.

[10] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 214–231, Tokyo, Japan, December 6–10, 2009. Springer-Verlag, Berlin, Germany.

[11] Tatsuaki Okamoto and Katsuyuki Takashima. Fully Secure Functional Encryption with General Relations from the Decisional Linear Assumption. In Tal Rabin, editor, *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 191–208, Santa Barbara, CA, USA, August 15–19, 2010. Springer-Verlag, Berlin, Germany.

[12] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson, editors, *ACM CCS 07: 14th Conference on Computer and Communications Security*, pages 195–203, Alexandria, VA, USA, October 28 - 31, 2007. ACM Press.

[13] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 187–196, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.

[14] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In Omer Reingold, editor, *TCC 2009: 6th Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 457–473, San Francisco, CA, USA, 2009. Springer-Verlag, Berlin, Germany.

[15] Elaine Shi and Brent Waters. Delegating capabilities in predicate encryption systems. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *Automata, Languages and Programming: 35rd International Colloquium*, volume 5126 of *Lecture Notes in Computer Science*, pages 560–578, Reykjavik, Iceland, July 7–11, 2008. Springer-Verlag, Berlin, Germany.

[16] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In Shai Halevi, editor, *Advances in Cryptology – CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 619–636, Santa Barbara, CA, USA, August 16–20, 2009. Springer-Verlag, Berlin, Germany.

A From Predicate-only to Full-Fledged Schemes

A.1 Full-Fledged Encryption schemes

A *Full-Fledged Encryption scheme* for the class $\mathbb{B}$ of Boolean predicates is a tuple of four efficient probabilistic algorithms (Setup, $\mathsf{Encrypt}$, KeyGen, $\mathsf{Decrypt}$) with the following semantics.

$\mathsf{Setup}(1^\lambda, 1^n, 1^k)$: takes as input a security parameter λ, the number n of variables, and the length of the plaintext (given in unary) and outputs the public parameters Pk and the master secret key Msk.

$\mathsf{KeyGen}(\mathsf{Msk}, \Phi)$: takes as input the master secret key Msk and a formula $\Phi \in \mathbb{B}_n$ and outputs a secret key Sk_Φ.

$\mathsf{Encrypt}(\mathsf{Pk}, \vec{z}, M)$: takes as input the public parameters Pk, a truth assignment $\vec{z}$ for n variables and plaintext $M \in \{0,1\}^k$ and outputs a ciphertext Ct.

$\mathsf{Decrypt}(\mathsf{Pk}, \mathsf{Ct}, \mathsf{Sk}_\Phi)$: takes as input the public parameters Pk, a ciphertext Ct for truth assignment $\vec{z}$ and plaintext M and a secret key Sk_Φ outputs plaintext M'.

Correctness of Boolean Satisfaction Encryption. For correctness we require that for all pairs $(\mathsf{Pk}, \mathsf{Msk})$ output by $\mathsf{Setup}(1^\lambda, 1^n, 1^k)$, it holds that for any truth assignment $\vec{z}$ for n variables, for any plaintex $M \in \{0,1\}^k$ and for any formula $\Phi \in \mathcal{B}_n$ over n variables we have that the probability that

$$\mathsf{Decrypt}(\mathsf{Pk}, \mathsf{Encrypt}(\mathsf{Pk}, \vec{z}), \mathsf{KeyGen}(\mathsf{Msk}, \Phi)) = M$$

is negligibly in λ close to 1 if $\vec{z}$ satisfies Φ and negligibly in λ close to 0 if $\vec{z}$ does not satisfy Φ.

A.2 Security definitions

We define ξ-security for $\xi = 0, 1$, by means of the following game $\mathsf{Game}_{\mathsf{Real}}(\xi)$ played between a ξ-adversary and a challenger $\mathcal{C}$¿

Setup. $\mathcal{C}$ runs the Setup algorithm on input the security parameter λ, the number n of variables and the length k of the plaintext, 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 $\Phi \in \mathbb{B}_n$, $\mathcal{C}$ returns $\mathsf{KeyGen}(\mathsf{Msk}, \Phi)$.

Challenge Construction. Upon receiving pairs $((\vec{z}_0, M_0), (\vec{z}_1, M_1))$ of truth assignments over n variables and k-bit messages, $\mathcal{C}$ picks random $\eta \in \{0,1\}$ and returns $\mathsf{Encrypt}(\mathsf{Pk}, \vec{z}_\eta, M_\eta)$.

Winning Condition. Let η' be $\mathcal{A}$'s output. We say that $\mathcal{A}$ *wins* the game if $\eta = \eta'$ and, for all Φ for which $\mathcal{A}$ has issued a Key Query, it holds that $\mathsf{Satisfy}(\Phi, z_0) = \mathsf{Satisfy}(\Phi, z_1) = \xi$.

We call such an adversary $\mathcal{A}$ a ξ-adversary and define its advantage $\mathsf{Adv}_{\mathbb{B}}^{\mathcal{A},\xi}(\lambda)$ to be the probability of winning minus 1/2.

Definition A.1 *An Encryption scheme for class* $\mathbb{B}$ *is* ξ-secure *if for all probabilistic polynomial time* ξ*-adversaries* $\mathcal{A}$*, we have that* $\mathsf{Adv}_{\mathbb{B}}^{\mathcal{A},\xi}(\lambda)$ *is a negligible function of* λ*.*

A.3 Extending conjunctions

It is easy to extend our scheme for HVE (and HHVE) to the full-fledged case in the following way. In the schemes for (Hierarchical) Hidden Vector Encryption we add the value $\Omega = \mathbf{e}(g_1, g_1)^z$ 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 a_i's so that they sum up to z (instead of summing up to 0). In the encryption for a message M, 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 the blinding factor Ω^s can be obtained from the keys and the ciphertext. The security of the scheme then requires an extra assumption in the target group. A similar modification works for 3-CNF. We omit further details.

A.4 Full-fledged schemes for Disjunctions and k-DNF

We now outline a general construction of a full fledged encryptions scheme for Disjunctions based on predicate only construction for 2-DNF. Let $\mathsf{2DNF} = (\mathsf{Setup}_{\mathsf{2DNF}}, \mathsf{KeyGen}_{\mathsf{2DNF}}, \mathsf{Encrypt}_{\mathsf{2DNF}}, \mathsf{Test}_{\mathsf{2DNF}})$ be a predicate only encryption scheme for 2-DNF and let us consider the following full-fledged scheme $\mathsf{DISJ} = (\mathsf{Setup}_{\mathsf{DISJ}}, \mathsf{KeyGen}_{\mathsf{DISJ}}, \mathsf{Encrypt}_{\mathsf{DISJ}}, \mathsf{Decrypt}_{\mathsf{DISJ}})$ for disjunctions.

$\mathsf{Setup}_{\mathsf{DISJ}}(1^\lambda, 1^n, 1^k)$ returns $(\mathsf{Msk}, \mathsf{Pk}) = \mathsf{Setup}_{\mathsf{2DNF}}(1^\lambda, 1^{n+k})$. Msk is thus a secret key ofr $n + k$ variables that we denote by $x_1, \ldots, x_n$ and $m_1, \ldots, m_k$.

$\mathsf{KeyGen}_{\mathsf{DISJ}}(\mathsf{Msk}, \Phi)$ on input master secret key Msk and formula Φ over n variables $x_1, \ldots, x_n$ computes, for each $i \in [k]$, $\mathsf{Sk}_i^0 = \mathsf{KeyGen}_{\mathsf{2DNF}}(\mathsf{Msk}, \Phi_i^0)$ and $\mathsf{Sk}_i^1 = \mathsf{KeyGen}_{\mathsf{2DNF}}(\mathsf{Msk}, \Phi_i^1)$ where $\Phi_i^0 = \Phi \wedge \bar{m}_i$ and $\Phi_i^1 = \Phi \wedge m_i$. Finally, the algorithm returns $\mathsf{Sk}_\Phi = (\mathsf{Sk}_1^0, \mathsf{Sk}_1^1, \ldots, \mathsf{Sk}_k^0, \mathsf{Sk}_k^1)$. We stress that if predicate Φ is represented by a disjunction then, for all $i \in [k]$, predicates Φ_i^0 and Φ_i^1 can be represented by 2-DNF formulae.

$\mathsf{Encrypt}_{\mathsf{DISJ}}(\mathsf{Pk}, \vec{z}, M)$ constructs vector z' of length $n+k$ by concatenating the n bits of $\vec{z}$ and the k bits of M and returns $\mathsf{Ct} = \mathsf{Encrypt}_{\mathsf{2DNF}}(\mathsf{Pk}, z')$.

$\mathsf{Decrypt}_{\mathsf{DISJ}}(\mathsf{Pk}, \mathsf{Ct}, \mathsf{Sk}_\Phi)$ computes the i-th bit M_i of the plaintext in the following way.

If $\mathsf{Test}_{\mathsf{2DNF}}(\mathsf{Pk}, \mathsf{Ct}, \mathsf{Sk}_i^b) = 1$ for some $b \in \{0, 1\}$ then $M_i = b$; else $M_i = \perp$.

We argue correctness by means of the following easy observations. First of all, no assignment z' to $n + k$ variables can satisfy both Φ_i^0 and Φ_i^1 and thus the decryption algorithm is well defined. Then observe that if z' satisfies Φ_i^b then it must be the case that variable $m_i = b$ and thus we have that the i-th bit $M_i = b$. Finally, observe that if z' satisfies neither Φ_i^0 nor Φ_i^1 then it must be the case that z does not satisfy Φ and in this case the decryption algorithm rightly fails.

Lemma A.2 *If* $\mathsf{2DNF}$ *is a* 0*-secure encryption scheme for* 2*-DNF then* DISJ *is a* 0*-secure encryption scheme for Disjunctions.*

Proof. Suppose for sake of contradiction that there exists a probabilistic polynomial-time 0-adversary $\mathcal{A}$ for DISJ with a non-negligible advantage in $\mathsf{Game}_{\mathsf{Real}}(0)$ for DISJ. Then we construct a 0-adversary $\mathcal{B}$ for $\mathsf{2DNF}$ with a non-negligible advantage in $\mathsf{Game}_{\mathsf{Real}}(0)$ for $\mathsf{2DNF}$.

$\mathcal{B}$ receives public key $\mathsf{Pk}_{\mathsf{2DNF}}$ from challenger $\mathcal{C}$ and runs $\mathcal{A}$ on input $\mathsf{Pk}_{\mathsf{DISJ}} = \mathsf{Pk}_{\mathsf{2DNF}}$. Notice that $\mathsf{Pk}_{\mathsf{2DNF}}$ is obtained by running $\mathsf{Setup}_{\mathsf{2DNF}}$ on input 1^λ and $n + k$ and thus, by the definition of algorithm $\mathsf{Setup}_{\mathsf{DISJ}}$, $\mathsf{Pk}_{\mathsf{DISJ}}$ has the same distribution of the input of $\mathcal{A}$ in $\mathsf{Game}_{\mathsf{Real}}(0)$ of DISJ. To answer $\mathcal{A}$'s key query for disjunction Φ, $\mathcal{B}$ issues $2k$ queries for predicates Φ_i^0 and Φ_i^1 for $i \in [k]$ and construct Sk_Φ by concatening the replies to its $2k$ queries. Also in this case, by the definition of $\mathsf{KeyGen}_{\mathsf{DISJ}}$, the replies to $\mathcal{A}$'s queries have the right distribution. When $\mathcal{B}$ receives challenge pairs $((\vec{z}_0, M_0), (\vec{z}_1, M_1))$ from $\mathcal{A}$, $\mathcal{B}$ constructs z_0' by concatenating $\vec{z}_0$ and M_0 and z_1' by concatenating $\vec{z}_1$ and M_1 and issues a challenge construction query for (z_0', z_1'). Upon receving the challenge ciphertext Ct for z_η' then $\mathcal{B}$ passes it to $\mathcal{A}$. Notice that, by the definition of $\mathsf{Encrypt}_{\mathsf{DISJ}}$, Ct is an encryption of M_η with attribute $\vec{z}_\eta$. Finally, $\mathcal{B}$ outputs $\mathcal{A}$'s guess η' for η. This terminates the description of $\mathcal{B}$.

The proof is completed by the observation that if $\mathcal{A}$'s queries for a key for disjunction Φ is such that $\mathsf{Satisfy}(\Phi, \vec{z}_0) = \mathsf{Satisfy}(\Phi, \vec{z}_1) = 0$ then it holds that for all $i \in [k]$ and $b \in \{0,1\}$ it holds $\mathsf{Satisfy}(\Phi_i^b, z_0') = \mathsf{Satisfy}(\Phi_i^b, z_1') = 0$. Therefore we can conclude that the advantage of $\mathcal{B}$ in $\mathsf{Game}_{\mathsf{Real}}(0)$ for $\mathsf{2DNF}$ is the same as the advantage of $\mathcal{A}$ in $\mathsf{Game}_{\mathsf{Real}}(0)$ for DISJ. □

By combining the previous lemma with Theorem 7.3, we obtain the following theorem.

Theorem A.3 *If Assumption* 1 *and* 4 *hold for generator* $\mathcal{G}$ *then there exists a* 0*-secure full-fledged encryption scheme for the class of predicates represented by disjunctions.*

B Generic Security of Our Complexity Assumptions

We now prove that, if factoring is hard, our four complexity assumptions hold in the generic group model. Since Assumption 3 implies Assumption 2, it suffices to prove generic validity for Assumption $1, 3$ and 4 only. We adopt the framework of [7] to reason about assumptions in bilinear groups $\mathbb{G}, \mathbb{G}_T$ of composite order $N = p_1p_2p_3p_4$. 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 $\mathbf{e}(g_{p_i}, g_{p_i})$) 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 $X = \sum_i \lambda_i A_i$ as formal random variables. Otherwise, we say that X is *independent* of $\{A_i\}$. We state the following theorems from [7].

Theorem B.1 (Theorem A.2 of [7]) *Let* $N = \prod_{i=1}^m p_i$ *be a product of distinct primes, each greater than* 2^λ. *Let* $\{X_i\}, T_1, T_2$ *be random variables over* $\mathbb{G}$ *and let* $\{Y_i\}$ *be random variables over* 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^λ. *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 $S := \{i \mid \mathbf{e}(T_1, X_i) \neq \mathbf{e}(T_2, X_i)\}$ *(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* $\mathbf{e}(T_1, X_k)$ *is independent of* $\{B_i\} \cup \{\mathbf{e}(X_i, X_j)\} \cup \{\mathbf{e}(T_1, X_i)\}_{i \neq k}$ *and* $\mathbf{e}(T_2, X_k)$ *is independent of* $\{B_i\} \cup \{\mathbf{e}(X_i, X_j)\} \cup \{\mathbf{e}(T_2, X_i)\}_{i \neq k}$. *Then if there exists an algorithm* $\mathcal{A}$ *issuing at most* q

instructions and having advantage δ, then there exists an algorithm that outputs a nontrivial factor of N in time polynomial in λ and the running time of $\mathcal{A}$ with probability at least $\delta - \mathcal{O}(q^2 t/2^\lambda)$.

We apply these theorems to prove the security of our assumptions in the generic group model.

Assumption 1. We can express this assumption as:

$$X_1 = (0,0,1,0),\ X_2 = (A_1, 0, A_3, 0),\ X_3 = (B_1, 0, B_3, 0),\ X_4 = (0,0,0,1)$$

and

$$T_1 = (Z_1, 0, Z_3, 0), T_2 = (0, Z_2, Z_3, 0).$$

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:

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

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

Assumption 3. We can express this assumption as:

$$\begin{array}{llll} X_1 = (1,0,0,0), & X_2 = (0,1,0,0), & X_3 = (0,0,1,0), & X_4 = (0,0,0,1), \\ X_5 = (A,0,0,B_4), & X_6 = (B,0,0,C_4), & X_7 = (C,0,0,D_4), & X_8 = (AC,0,0,E_4), \\ X_9 = (ABC,0,0,F_4) & & & \end{array}$$

and

$$T_1 = [AB, 0, 0, G_4], \quad T_2 = [Z_1, 0, 0, Z_4].$$

We note that G_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 $S = \{1,4,5,6,7,8,9\}$, and thus, considering T_1 first, we obtain the following tuples:

$$\begin{array}{ll} C_{1,1} = \mathbf{e}(T_1, X_1) = [AB, 0, 0, 0], & C_{1,4} = \mathbf{e}(T_1, X_4) = [0,0,0,G_4] \\ C_{1,5} = \mathbf{e}(T_1, X_5) = [A^2 B, 0, 0, G_4 B_4], & C_{1,6} = \mathbf{e}(T_1, X_6) = [AB^2, 0, 0, G_4 C_4] \\ C_{1,7} = \mathbf{e}(T_1, X_7) = [ABC, 0, 0, G_4 D_4] & C_{1,8} = \mathbf{e}(T_1, X_8) = [A^2 BC, 0, 0, G_4 E_4] \\ C_{1,9} = \mathbf{e}(T_1, X_9) = [A^2 B^2 C, 0, 0, G_4 F_4]. & \end{array}$$

It is easy to see that $C_{1,k}$ with $k \in \{4,5,9\}$ is independent of $\{\mathbf{e}(X_i, X_j)\} \cup \{\mathbf{e}(T_1, X_i)\}_{i \neq k}$.

For $C_{1,1}$, we observe that the only way to obtain an element whose first component contains AB is by computing $\mathbf{e}(A_5, A_6) = [AB, 0, 0, B_4 C_4]$ 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. Similarly for $C_{1,8}$, to obtain an element whose first component contains $A^2 BC$ the only way is by computing $\mathbf{e}(A_5, A_8) = [A^2 BC, 0, 0, B_4 F_4]$ but there is no way to cancel the fourth component $B_4 F_4$.

For $C_{1,7}$, we notice that the only way to obtain an element whose first component contains ABC is by computing $\mathbf{e}(A_1, A_9) = [ABC, 0, 0, 0]$ but then there is no way to generate an element whose fourth component is $G_4 D_4$ and hence no way to cancel that term from $C_{1,7}$.

Analogous arguments apply for the case of T_2.

Thus the independence requirement of Theorem B.1 is satisfied and Assumption 3 is generically secure, assuming it is hard to find a nontrivial factor of N.

Assumption 4. We can express this assumption as:

$$X_1 = (0, 1, 0, 0),\ X_2 = (0, 0, 1, 0),\ X_3 = (0, 0, 0, 1),\ X_4 = (A_1, 0, 0, A_4)$$

and

$$T_1 = (Z_1, 0, 0, Z_4), T_2 = (0, 0, 0, Z_4).$$

It is easy to see that T_1 and T_2 are both independent of $\{X_i\}$ because, for example, Z_4 does not appear in the X_i's. Next, we note that for this assumption we have $S = \{4\}$, and thus, considering T_1 first, we obtain the following tuples:

$$C_{1,4} = \mathbf{e}(T_1, X_4) = [Z_1 A_1, 0, 0, Z_4 A_4].$$

It is easy to see that $C_{1,4}$ is independent of $\{\mathbf{e}(X_i, X_j)\} \cup \{\mathbf{e}(T_1, X_i)\}_{i \neq k}$. An analogous arguments apply for the case of T_2. Thus the independence requirements of Theorem B.1 are satisfied and Assumption 4 is generically secure, assuming it is hard to find a nontrivial factor of N.