Why Proving HIBE Systems Secure is Difficult

Allison Lewko Brent Waters

Abstract

Proving security of Hierarchical Identity-Based Encryption (HIBE) and Attribution Based Encryption scheme is a challenging problem. There are multiple well-known schemes in the literature where the best known (adaptive) security proofs degrade exponentially in the maximum hierarchy depth. However, we do not have a rigorous understanding of why better proofs are not known. (For ABE, the analog of hierarchy depth is the maximum number of attributes used in a ciphertext.)

In this work, we define a certain commonly found checkability property on ciphertexts and private keys. Roughly the property states that any two different private keys that are both "supposed to" decrypt a ciphertext will decrypt it to the same message. We show that any simple black box reduction to a non-interactive assumption for a HIBE or ABE system that contains this property will suffer an exponential degradation of security.

1 Introduction

In recent years, there has been emerging interest in increasing the expressiveness of encryption systems in terms of targeting ciphertexts to certain groups of users. First examples included Hierarchical Identity-Based Encryption (HIBE) [HL02] and Attribute-Based Encryption (ABE) [SW05]. The early difficulty in HIBE and ABE research was to obtain systems that were provably secure under robust security definitions. Initial constructions of HIBE [GS02, CHK03, BB04a, BBG05] and ABE [SW05, GPSW06] had the drawback that their security reductions degraded *exponentially* in the depth of the hierarchy when encrypting an HIBE ciphertext or number of attributes used when creating an ABE ciphertext. For this reason, the first (standard model) security proofs were done in the selective model, a term coined by Canetti, Halevi and Katz [CHK03]. In this weaker model, an attacker (artificially) declared the challenge identity he was attacking *before* seeing the public parameters of the system.

At the time, researchers identified achieving standard (sometimes called adaptive or full) security for these systems as an important open problem. However, it was not well understood whether there existed full security reductions for the already proposed constructions without exponential decay, and if not, why. While there was general intuition about the limitations of what were called *partitioning* proofs (e.g., see discussion in [Wat09]), there was no rigorous explanation of these difficulties.

In 2009, Gentry and Halevi [GH09] gave an HIBE construction and proved it fully secure without an exponential degradation in the depth. Their construction made use of projective hash techniques from [CS02, Gen06]. One tradeoff is that it required the use of non-static or q-type assumptions to prove security where the size of the assumption grew with the number of key queries. Later, Waters [Wat09] described a new and more systematic approach to proving full security called dual system encryption. Using dual system encryption, he proved an HIBE system fully secure under simple assumptions. Dual system encryption was subsequently used to prove full security of ABE and other related systems [LOS$^+$10, OT10, LW12].

While these new proof techniques represent an advance in proving security, they still leave us with an incomplete picture about the security of the initial selectively secure constructions. *Can*

these systems only be proven selectively secure? If so, why? Coming to a better understanding is important for multiple reasons. First, the earlier systems are typically more practically efficient than the recent dual system encryption counterparts. If they could be proven fully secure, they might be more desirable to use. Second, it is valuable to have a more rigorous characterization of what properties of a construction make it difficult to prove security, as identifying these properties can potentially inspire new construction and proof methods for encryption systems.

Understanding Partitioning Proofs We organize our investigation around the goal of understanding partitioning proofs. Intuitively, these are proofs where a reduction algorithm (when creating a set of public parameters) splits ciphertext descriptors or "identities" into two disjoint sets. Those it can leverage for the challenge ciphertext (we call this the "challenge set") and those it cannot. If a certain identity x is in the challenge set, then the reduction cannot issue a private key for y if a private key for y should be allowed to decrypt a ciphertext associated with x.

We begin by asking the following two questions:

1. *Are there functionalities where a partitioning proof cannot work? (I.e. No reduction with a polynomial security loss exists.)*
2. *Under what circumstances are we stuck with a partitioning proof?*

To begin to answer the first question, we try to think of a basic case where partitioning will fail. To this end, we introduce a prefix encryption functionality. In a prefix encryption system, a private key is associated with a binary string y and a ciphertext with a binary string x. One can decrypt the ciphertext to reveal a hidden message M if and only if y is a prefix of x. The point of introducing this primitive is to describe a simple primitive which distills the core features needed for our impossibility result. HIBE and most expressive ABE systems imply prefix encryption in a straightforward way.

To be successful, any partitioning reduction algorithm must have the set of challenge ciphertext descriptors cover at least a non-negligible fraction of the descriptors, else one would almost never get chosen by an attacker. In addition, there must be some non-negligible chance that the private keys requested by the attacker do not violate this partition. Immediately, we see this cannot work with a prefix encryption system. Consider an attacker, $\mathcal{A}$ that choose a random length n string x (for security parameter n) as the challenge ciphertext. In addition, it asks for private keys for strings $y_1, \ldots, y_n$, where string y_i is the length i string that matches x in the first $i-1$ bits and is different in the last bit. This small number of private keys can be used to decrypt a ciphertext associated with *any* string *except* x. Thus, any partitioning reduction that has more than one string in its challenge set will not be able to answer all the key queries for this attacker. Consequently, its best strategy is to pick one string for its challenge set, which will match $\mathcal{A}$'s choice with only 2^{-n} probability.

Next, we want to understand what properties of a construction force us to be "stuck with" a partitioning proof, in the sense that there is nothing to be gained from considering different reduction techniques. For prefix encryption, it was problematic for a partitioning proof that a large number of ciphertexts types could be covered by a small number of keys. Intuitively, one might be stuck with a partitioning proof if any authorized key can "equally decrypt" a ciphertext. We consider prefix encryption constructions that implicitly allow a pair of efficient algorithms for respectively checking an acceptability condition of a key and a ciphertext. If a ciphertext associated with a string x is determined to be acceptable by this check, then *all* acceptable keys for any prefix y will decrypt to the *same* message (or all fail decryption). We refer to constructions that allow such decisive checks as "checkable" schemes.

Essentially, this says that all keys that should be able to decrypt an acceptable ciphertext will decrypt it the same way. It is notable that early constructions of HIBE [GS02, CHK03, BB04a, BBG05] and ABE [SW05, GPSW06] which were only proved selectively secure all have this property when instantiated under typically used prime order bilinear groups. This matches our intuition that they are in some sense stuck with partitioning proofs. However, constructions using the techniques of Gentry [Gen06] and dual system encryption do not meet this criteria. For example, in dual system encryption proofs, a normal secret key will decrypt a semi-functional ciphertext differently than a semi-functional secret key will.

Our Result In this work, we formalize this intuition by showing that there are no simple black box reductions from the full security of checkable prefix encryption schemes to non-interactive assumptions [1]. This result extends to HIBE and ABE as we show that these both can embed prefix encryption systems.

We capture our result in a somewhat similar manner to Coron [Cor02] and Hofheinz, Jaegar, and Knapp [HJK12] who showed that no unique [Cor02] or rerandomizable [HJK12] signatures can have black box proofs to non-interactive assumptions. While their focus was on showing the necessity of a polynomial loss (in the number of signature queries) for a class of signatures, we show the necessity of a drastic exponential loss of security for HIBE and ABE schemes.

At a high level, we construct an algorithm $\mathcal{B}$ that runs the reduction algorithm $\mathcal{R}$, where $\mathcal{B}$ acts as an computationally unbounded attacker. Since $\mathcal{B}$ is actually not a "real" attacker it will need to find a way to look like one.

To do this, $\mathcal{B}$ will first wait for the reduction algorithm to commit to a set of public parameters. Next, it will run $\mathcal{R}$ with the same public parameters multiple times (we specify more precisely the number of times in Section 3), each time choosing a random string x and collecting private keys $y_1, \ldots, y_n$ for the n strings that are prefixes of x except in their last bits. After each run, $\mathcal{B}$ rewinds $\mathcal{R}$ to the point where it published the system parameters. The point of these runs is to collect private key information relative to the committed public parameters. If any of these runs for a particular x value does not abort, then $\mathcal{B}$ has the private key information to decrypt a ciphertext for any string but x.

Finally, $\mathcal{B}$ will request a challenge ciphertext for a new random string z. If $x \neq z$ for some x used in a prior run where $\mathcal{B}$ successfully collected keys, then $\mathcal{B}$ has a private key that allows it to decrypt the challenge ciphertext and act as an attacker. If $\mathcal{R}$ is an efficient reduction, it will then break the assumption with non-negligible advantage. We can generalize this to reductions that run the attacker a polynomial number of times in sequence, but like [Cor02, HJK12] we do not cover reductions that concurrently run executions of the attack algorithm.

Future Directions Multiple interesting questions arise from this work. Perhaps the most exciting direction is to see if limitations of our impossibility result can lead to new proof techniques in the positive direction. For example, in the course of this work we discovered that one can build prefix encryption from any IBE scheme. The proof is an easy hybrid reduction. This construction lies outside of impossibility result since two keys for *different* prefixes y and y' of some string z might decrypt a (malformed) ciphertext to different values. This is different than dual system encryption techniques, which rely on giving a different key structure for the same key value. A parallel goal is of course to strengthen our impossibility results. An natural target is to see if either our impossibility results can be extended to handle reductions that run attack algorithms concurrently or alternatively if building reductions that run attack algorithms

[1] The restriction to non-interactive assumptions is natural and arguably necessary. Any scheme can be proven secure under the (possibly interactive) assumption that it is secure. The work of [BSW07] essentially does this, but with the mitigating factor of proving generic group security.

concurrently can be leveraged for new positive results. By expanding our knowledge from both ends of the spectrum, we can hope to get a more complete understanding of the space of possible security proofs for functional encryption systems.

Another direction is to examine how recent selectively secure lattice HIBE [CHKP10, ABB10] constructions fit into this framework. These constructions allow some form of key rerandomization in that an algorithm can sample a new short basis, however, the "quality" of this basis is not as good as the original and in general higher quality private keys are not reachable from lower quality private keys. One possibility is that this quality of key difference can be leveraged to prove full security of these existing schemes.

2 Preliminaries

2.1 Prefix Encryption

We present the functionality of prefix encryption as the simplest functionality that captures the core structure of hierarchical identity-based encryption. Essentially, we strip off the usual trappings of HIBE schemes that are not relevant to our purposes. In particular, we do not require explicit delegation capabilities, and we do not use "identity vectors" with large sets of potential values for each coordinate. Instead, keys and ciphertexts in a prefix encryption scheme will be associated with binary strings, and a key will be able to decrypt a ciphertext if and only if the binary string associated to the key is a prefix of the binary string associated to the ciphertext. We observe that such a functionality can be easily derived from any HIBE scheme by designating fixed identities in each coordinate to play the role of "0" and "1".

We formally define a Prefix Encryption scheme as having the following algorithms:

Setup$(\lambda) \to \text{PP}, \text{MSK}$ The setup algorithm takes in the security parameter λ and outputs the public parameters PP and a master secret key MSK.

Encrypt$(x, M, \text{PP}) \to \text{CT}$ The encryption algorithm takes in a binary string x, a message M, and the public parameters PP. It outputs a ciphertext CT.

KeyGen$(\text{MSK}, y) \to \text{SK}$ The key generation algorithm takes in the master secret key MSK and a binary string y. It outputs a secret key SK.

Decrypt$(\text{CT}, \text{SK}) \to M$ The decryption algorithm takes in a ciphertext CT and a secret key SK. If the binary string y of the secret key is a prefix of the binary string x of the ciphertext, it outputs the message M.

As we will study how security reductions behave as the binary strings involved grow longer, we will allow public parameters to specify a maximum length, q, for the indexing strings of the keys and ciphertexts. Our lower bound on the provable security degradation as an exponential function of the maximum string length will only apply to schemes that are suitably "checkable." In order to define this precisely, we will restrict our consideration to schemes can be augmented with two additional algorithms:

CTCheck$(\text{PP}, \text{CT}, x) \to \{\text{True}, \text{False}\}$ The ciphertext checking algorithm takes in public parameters PP, a ciphertext CT, and a binary string x. It outputs either True or False.

KeyCheck$(\mathrm{PP}, \mathrm{SK}, y) \rightarrow \{\mathrm{True}, \mathrm{False}\}$ The key checking algorithm takes in public parameters PP, a secret key SK, and a binary string y. It outputs either True or False.

We note that these additional algorithms are required to be efficient (just like the more standard algorithms above). We also require them to be deterministic.

For correctness, we require that $\mathrm{CTCheck}(\mathrm{PP}, \mathrm{CT}, x)$ outputs True whenever PP is honestly generated and CT is an honestly generated ciphertext for x from PP. Similarly, we require that $\mathrm{KeyCheck}(\mathrm{PP}, \mathrm{SK}, y)$ outputs True whenever PP, MSK are honestly generated and SK is an honestly generated key for y from MSK.

Definition 1. *We say a prefix encryption scheme is* ***checkable*** *if for any* $\mathrm{PP}, \mathrm{CT}, x, \mathrm{SK}_1, y_1, \mathrm{SK}_2, y_2$ *such that CTCheck*$(\mathrm{PP}, \mathrm{CT}, x) = \mathrm{True}$*, KeyCheck*$(\mathrm{PP}, \mathrm{SK}_1, y_1) = \mathrm{True}$*, KeyCheck*$(\mathrm{PP}, \mathrm{SK}_2, y_2) = \mathrm{True}$*, and* y_1, y_2 *are both prefixes of* x*, then Decrypt*$(\mathrm{CT}, \mathrm{SK}_1) =$ *Decrypt*$(\mathrm{CT}, \mathrm{SK}_2)$*.*

2.1.1 Security Definition

We now define full security for a prefix encryption scheme in terms of the following game between a challenger and an attacker. This is essentially the definition of full IND-CPA security for HIBE schemes, but the case of prefix encryption is a bit simpler as there is no need to track the delegation of keys. The game proceeds in the following phases:

Setup Phase The challenger runs $\mathrm{Setup}(\lambda)$ to produce MSK and PP. It gives PP to the attacker.

Key Query Phase I The attacker adaptively chooses binary strings y and queries the challenger for corresponding secret keys. For each queried string y, the challenger runs $\mathrm{KeyGen}(\mathrm{MSK}, y)$ to produce a secret key SK, which it gives to the attacker.

Challenge Phase The attacker declares to equal length messages M_0, M_1, and a binary string x. It is required that for all strings y queried in the previous phase, y is **not** a prefix of x. The challenger chooses a uniformly random bit $b \in \{0, 1\}$ and creates a ciphertext CT by running $\mathrm{Encrypt}(x, M_b, \mathrm{PP})$. It gives CT to the attacker.

Key Query Phase II This is the same as the first key query phase, except that any queried y must not be a prefix of the challenge string x.

Guess The attacker submits a guess b' for the bit b.

Definition 2. *We define the advantage of an attacker in this game to be* $|Pr[b = b'] - \frac{1}{2}|$*. We say an algorithm* $\mathcal{A}$ (t, ϵ, q)*-breaks a prefix encryption scheme if it runs in time* t*, achieves advantage* ϵ*, and makes at most* q *total key queries. We say a prefix encryption scheme is secure if no algorithm* (t, ϵ, q)*-breaks for parameters* t, q, ϵ *where* t, q *are polynomial in the security parameter and* ϵ *is non-negligible.*

The weaker notion of selective security would be obtained by modifying the security game above by having the attacker declare the binary string x for the challenge at the very beginning of the game, *before* seeing the public parameters.

2.2 Hierarchical Identity-Based Encryption

For completeness, we now present the relevant definitions for HIBE schemes. We use $\vec{I} = (I_1, \ldots, I_k)$ to denote an identity vector.

A HIBE scheme is composed of the following algorithms:

Setup$(\lambda) \to \text{PP}, \text{MSK}$ The setup algorithm takes in the security parameter λ and outputs the public parameters PP and a master secret key MSK.

Encrypt$(\vec{I}, M, \text{PP}) \to \text{CT}$ The encryption algorithm takes in an identity vector $\vec{I}$, a message M, and the public parameters PP. It outputs a ciphertext CT.

KeyGen$(\text{MSK}, \vec{I}) \to \text{SK}$ The key generation algorithm takes in the master secret key MSK and an identity vector $\vec{I}$. It outputs a secret key SK for $\vec{I}$.

Delegate$(\text{PP}, \text{SK}, I_{k+1}) \to \text{SK}'$ The delegation algorithm takes in the public parameters, a secret key for an identity vector $\vec{I} = (I_1, \ldots, I_k)$, and a new component I_{k+1} to be added to the identity vector. It produces a new secret key SK$'$ for the identity $(I_1, \ldots, I_k, I_{k+1})$.

Decrypt$(\text{CT}, \text{SK}) \to M$ The decryption algorithm takes in a ciphertext CT and a secret key SK. If the identity vector of the secret key is a prefix of the identity vector of the ciphertext, it outputs the message M.

As we will study how security reductions behave as the identity vectors involved grow longer, we will allow public parameters to specify a maximum length, q, for the identity vectors associated with the keys and ciphertexts.

Similarly to our definitions for Prefix Encryption schemes, we consider HIBE schemes equipped with two additional algorithms:

CTCheck$(\text{PP}, \text{CT}, \vec{I}\ ) \to \{\text{True}, \text{False}\}$ The ciphertext checking algorithm takes in public parameters PP, a ciphertext CT, and an identity vector $\vec{I}$. It outputs either True or False.

KeyCheck$(\text{PP}, \text{SK}, \vec{I}\ ) \to \{\text{True}, \text{False}\}$ The key checking algorithm takes in public parameters PP, a secret key SK, and an identity vector $\vec{I}$. It outputs either True or False.

We note that these additional algorithms are required to be efficient (just like the more standard algorithms above). We also require them to be deterministic.

For correctness, we require that $\text{CTCheck}(\text{PP}, \text{CT}, \vec{I})$ outputs True whenever PP is honestly generated and CT is an honestly generated ciphertext for $\vec{I}$ from PP. Similarly, we require that $\text{KeyCheck}(\text{PP}, \text{SK}, \vec{I})$ outputs True whenever PP, MSK are honestly generated and SK is an honestly generated key for $\vec{I}$ from MSK.

Definition 3. *We say a HIBE scheme is* ***checkable*** *if for any* $\text{PP}, \text{CT}, \vec{I}^*, \text{SK}_1, \vec{I}^1, \text{SK}_2, \vec{I}^2$ *such that* $\mathit{CTCheck}(\text{PP}, \text{CT}, \vec{I}^*) = \mathit{True}$, $\mathit{KeyCheck}(\text{PP}, \text{SK}_1, \vec{I}^1) = \mathit{True}$, $\mathit{KeyCheck}(\text{PP}, \text{SK}_2, \vec{I}^2) = \mathit{True}$, *and* $\vec{I}^1, \vec{I}^2$ *are both prefixes of* $\vec{I}^*$, *then* $\mathit{Decrypt}(\text{CT}, \text{SK}_1) = \mathit{Decrypt}(\text{CT}, \text{SK}_2)$.

2.2.1 Security Definition

We present the full security definition for a HIBE scheme that is given in [SW08]. We define the following game between a challenger and an attacker.

Setup Phase The challenger runs Setup(λ) to produce MSK and PP. It gives PP to the attacker. The challenger also initializes a set $S = \emptyset$. (As the game progresses, S will hold the secret keys that the challenger has produced but not given out.)

Key Query Phase I The attacker adaptively makes three kinds of queries: Create, Delegate, and Reveal. In a Create query, the attacker supplies an identity vector $\vec{I}$. The challenger creates a key for $\vec{I}$ by running the KeyGen algorithm with MSK and $\vec{I}$ as input. It places the resulting key in the set S and gives the attacker a reference to it (note that it does not give the key itself to the attacker). In a Delegate query, the attacker specifies a reference to a key in S and an identity component I'. The challenger takes the key in S, which corresponds to some identity vector $\vec{I}$ and runs the delegation algorithm on it to create a new key for the identity $\vec{I} : I'$ (this denotes the vector created by appending I' to $\vec{I}$). The new key is placed in S, and the attacker is given a reference to it. In a Reveal query, the attacker supplies a reference to a key in S and the challenger gives the corresponding key to the attacker and removes it from S.

Challenge Phase The attacker declares two equal length messages M_0, M_1, and an identity vector $\vec{I}^*$. It is required that none of the revealed keys in the previous phase have identity vectors that are prefixes of $\vec{I}^*$. The challenger chooses a uniformly random bit $b \in \{0, 1\}$ and creates a ciphertext CT by running Encrypt$(\vec{I}^*, M_b, \text{PP})$. It gives CT to the attacker.

Key Query Phase II This is the same as the first key query phase, except that any revealed key must not be a prefix of the challenge vector $\vec{I}^*$.

Guess The attacker submits a guess b' for the bit b.

Definition 4. *We define the advantage of an attacker in this game to be* $|Pr[b = b'] - \frac{1}{2}|$. *We say an algorithm* $\mathcal{A}$ (t, ϵ, q)*-breaks a HIBE scheme if it runs in time* t*, achieves advantage* ϵ*, and makes at most* q *total key queries. We say a HIBE scheme is secure if no algorithm* (t, ϵ, q)*-breaks for parameters* t, q, ϵ *where* t, q *are polynomial in the security parameter and* ϵ *is non-negligible.*

We note that for schemes where the delegation algorithm produces keys that are identically distributed to keys produced directly by the key generation algorithm, this definition is equivalent to a simpler definition that only allows the usual reveal queries. This is because the delegation lineage of a key is now irrelevant for its distribution. Since it suffices for our purposes to ignore the delegation algorithm, we could just as easily work with the simpler definition, but we prefer to present the full definition here for completeness.

The weaker notion of selective security is obtained by modifying the security game above by having the attacker declare the vector $\vec{I}^*$ for the challenge at the very beginning of the game, *before* seeing the public parameters.

2.3 Non-interactive Decisional Problems and Simple Black Box Reductions

We now formally define the kinds of decisional problems and reductions we will consider. We start by describing the non-interactive decisional problems we allow:

Definition 5. *A non-interactive decisional problem* $\Pi = (C, \mathcal{D})$ *is described by a set* C *and a distribution* $\mathcal{D}$ *on* C*. We refer to* C *as the set of challenges, and each* $c \in C$ *is associated with a bit* $b(c) \in \{0, 1\}$*. We say that an algorithm* $\mathcal{A}$ (ϵ, t)*-solves* Π *if* $\mathcal{A}$ *runs in time* t *and*

$$Pr[\mathcal{A}(c) = b(c) : c \xleftarrow{\mathcal{D}} C] \geq \frac{1}{2} + \epsilon.$$

Here, $c \xleftarrow{\mathcal{D}} C$ *denotes that* c *is chosen randomly from* C *according to the distribution* $\mathcal{D}$*.*

Decisional problems used as cryptographic hardness assumptions are actually families of such problems, parameterized by a security parameter λ. Below, we will abuse notation mildly and write only Π while λ is implicit. We will write $poly(\lambda)$ and $neg(\lambda)$ to denote functions that are polynomial functions of λ and negligible functions in λ, respectively.

We next define the type of reductions we will address. We do not consider reductions in full generality - instead we restrict our consideration to black box reductions that satisfy additional requirements. Namely, we require simple reductions that only run the attacker once in a straight line fashion - meaning that the reduction simulates the security game exactly once with the attacker, who it interacts with as a black box. Note that this does not allow the reduction to rewind the attacker or supply its randomness, etc.

Definition 6. *An algorithm $\mathcal{R}$ is a simple $(t, \epsilon, q, \delta, t')$-reduction from a decisional problem Π to breaking the security of a prefix encryption scheme $Prefix$ if, when given black box access to any attacker $\mathcal{A}$ that (t, ϵ, q)-breaks the scheme $Prefix$, the algorithm $\mathcal{R}$ (δ, t')-solves the problem Π after simulating the security game once for $\mathcal{A}$.*

We note that all of the security reductions given for prior HIBE and ABE schemes are simple reductions in the sense of Definition 6.

2.4 Obtaining Prefix Encryption from HIBE

Given a HIBE scheme with algorithms Setup_{HIBE}, KeyGen_{HIBE}, Encrypt_{HIBE}, Delegate_{HIBE}, and Decrypt_{HIBE}, we will derive a prefix encryption scheme with algorithms Setup_{Pre}, KeyGen_{Pre}, Encrypt_{Pre}, and Decrypt_{Pre}. To accomplish this, we only require that there are at least two possible values for each component of the identity vectors allowed in the HIBE scheme.

We let $\text{Setup}_{Pre} := \text{Setup}_{HIBE}$. We then suppose that $\{I_1^0, I_1^1\}, \{I_2^0, I_2^1\}, \ldots, \{I_q^0, I_q^1\}$ are sets of values such that taking any combination $(I_1^{b_1}, I_2^{b_2}, \ldots, I_q^{b_q})$ for bits $b_1, \ldots, b_q \in \{0, 1\}$ forms a valid identity vector (and $I_j^0 \neq I_j^1$ for all j). We define KeyGen_{Pre} to generate a key for a binary string $y = (y_1, y_2, \ldots, y_k)$ for $k \leq q$ by running KeyGen_{HIBE} on the identity $(I_1^{y_1}, I_2^{y_2}, \ldots, I_k^{y_k})$. We similarly define Encrypt_{Pre} to encrypt to a binary vector $x = (x_1, \ldots, x_j)$ by running Encrypt_{HIBE} to encrypt to $(I_1^{x_1}, \ldots, I_j^{x_j})$. We can then set $\text{Decrypt}_{Pre} = \text{Decrypt}_{HIBE}$.

We now observe that if we start with a checkable HIBE, then the derived prefix encryption scheme will also be checkable:

Lemma 7. *If Setup_{HIBE}, KeyGen_{HIBE}, Encrypt_{HIBE}, Delegate_{HIBE}, and Decrypt_{HIBE} is a checkable HIBE scheme, than Setup_{Pre}, KeyGen_{Pre}, Encrypt_{Pre}, and Decrypt_{Pre} obtained from it as described above is a checkable prefix encryption scheme.*

Proof. We define the CTCheck algorithm for our prefix scheme as follows. It first translates the binary vector input into the corresponding identity vector via the encoding described above. Then it runs the CTCheck algorithm of the HIBE scheme on the same ciphertext and parameters on this identity vector. Similarly, the KeyCheck algorithm for our prefix scheme will simply translate the binary vector to the corresponding identity vector and run the KeyCheck algorithm for the HIBE scheme. It is clear that the required correctness properties of these algorithms are retained. Since the decryption algorithm of the prefix scheme simply runs the decryption algorithm of the HIBE scheme, the fact that the HIBE scheme is checkable implies that the derived prefix scheme is also checkable. □

Finally, we observe that simple security reductions for the initial HIBE scheme can be translated into simple security reductions for the derived prefix encryption scheme:

Lemma 8. *If $\mathcal{R}_{HIBE}$ is a simple $(t, \epsilon, q, \delta, t')$-reduction from a decisional problem Π to breaking the security of a HIBE encryption scheme, then we can obtain from $\mathcal{R}$ a new reduction $\mathcal{R}_{Pre}$ that is a simple $(t, \epsilon, q, \delta, t')$-reduction from the same decisional problem Π to breaking the security of the derived prefix encryption scheme.*

Proof. We derive $\mathcal{R}_{Pre}$ as follows. When given black box access to an attacker $\mathcal{A}$ on the prefix encryption scheme, $\mathcal{R}_{Pre}$ runs $\mathcal{R}$ and simulates an attacker $\mathcal{A}'$ on the HIBE scheme. It does this by translating a key request made by $\mathcal{A}$ for a binary vector y into a key request for the corresponding identity vector $\vec{I}$. It similarly translates the challenge vector declared by $\mathcal{A}$ and forward these translations to $\mathcal{R}$. When $\mathcal{R}$ produces an output, $\mathcal{R}_{Pre}$ copies this output as its own. Note that the running time of $\mathcal{R}_{Pre}$ is precisely the running time of $\mathcal{R}$. □

3 Main Result

We now prove our main result, establishing that any polynomial time simple black box reduction between the security of a checkable prefix encryption scheme and a hard, non-interactive decisional problem can only achieve an advantage that degrades exponentially in q, where q is the maximum string length of the scheme.

Essentially, we leverage the fact that the reduction can be run to obtain secret keys and then be rewound to "forget" these keys were produced. We can then use the secret keys obtained during the first runs of the reduction to simulate a successful attacker against a different challenge in a final run. The checking algorithms play a pivotal role in ensuring that the unorthodox manner in which these keys are obtained does not compromise their effectiveness. Intuitively, for keys and ciphertexts that pass the (publicly computable) checks, the result of a successful decryption is guaranteed to be independent of the origins of the key.

It is interesting to consider what happens if one tries to apply such techniques to more complicated reductions. A first example would be reductions that sequentially run the attacker a bounded number of times. In such a case, our result should extend easily via an application of the union bound, analogously to the extensions in [Cor02, HJK12]. However, it is not clear how to extend our argument to reductions that may run interleaved instances of the attacker, using concurrency in an arbitrary way. We observe that the arguments in [Cor02, HJK12] also do not address this case.

Theorem 9. *Let $Prefix$ = (Setup, Encrypt, KeyGen, Decrypt, CTCheck, KeyCheck) denote a checkable prefix encryption scheme, and let $\Pi(\lambda)$ denote a decisional problem such that no algorithm running in time $t = poly(\lambda)$ can obtain an advantage that is non-negligible in λ. Then any simple $(t, \epsilon, q, \delta, t')$-reduction $\mathcal{R}$ from Π to the security of $Prefix$ with $t = poly(\lambda)$, $t' = poly(\lambda)$ must have a value of δ such that δ vanishes exponentially as a function of q (up to terms that are negligible in λ).*

Proof. We let $Prefix$ = (Setup, Encrypt, KeyGen, Decrypt, CTCheck, KeyCheck) denote a checkable prefix encryption scheme. We suppose that $\mathcal{R}$ is a simple $(t, \epsilon, q, \delta, t')$-reduction from a decisional problem Π to breaking the security of this prefix encryption scheme. We now design an algorithm $\mathcal{B}$ to solve Π.

A Hypothetical Attacker We first define a hypothetical attacker $\mathcal{A}$ that (t, ϵ, q)-breaks the security of the prefix encryption scheme for some time t. $\mathcal{A}$ proceeds as follows: it first receives PP as input (we assume this also implicitly includes λ). It chooses a random binary string x of length q. In the first key query phase, it requests keys for strings $y_1, \ldots, y_q$ where each y_i is the binary string of length i formed by taking the first $i-1$ bits of x and then the opposite of the i^{th}

bit of x. Note that each y_i is not a prefix of x. It receives the corresponding keys $\mathrm{SK}_1, \ldots, \mathrm{SK}_q$ from the challenger. For each, it runs $\mathrm{KeyCheck}(\mathrm{PP}, \mathrm{SK}_i, y_i)$. If any of these checks outputs False, it quits.

Next, the attacker $\mathcal{A}$ declares two messages M_0, M_1 (we suppose these are fixed, distinct messages) and x as the challenge string. It receives the ciphertext CT from the challenger. It then runs $\mathrm{CTCheck}(\mathrm{PP}, \mathrm{CT}, x)$. If this outputs False, it quits. Otherwise, it samples SK^* uniformly from the set of all values of SK such that $\mathrm{KeyCheck}(\mathrm{PP}, \mathrm{SK}, x_i) = \text{True}$ for any prefix x_i of x. (Of course, this step may not be efficient.) After obtaining SK^*, it decrypts CT with SK^*. If the result is $M_{b'}$ for some $b' \in \{0, 1\}$, it guesses b' with probability $\frac{1}{2} + \epsilon$ and guesses the opposite with probability $\frac{1}{2} - \epsilon$. If the result is not M_0 or M_1, it guesses randomly.

For ease of analysis we will view the hypothetical attacker's set of coins as drawn from a space $Z \times F$. The set Z is the set of possible choices of the challenge string x, and we let F denote the set of all other random coins used.

We now verify that attacker $\mathcal{A}$ has advantage ϵ in the real security game. In this case, since the public parameters and ciphertext are honestly generated, then SK^* properly decrypts the challenge ciphertext, and hence the result will always be M_b. $\mathcal{A}$ then guesses b correctly with probability $\frac{1}{2} + \epsilon$.

Using the Reduction We are assuming that the reduction $\mathcal{R}$ runs the attacker once in a straight-line fashion (e.g. no rewinding). We now create an algorithm $\mathcal{B}$ to solve Π by using $\mathcal{R}$. (Note that $\mathcal{B}$ can rewind $\mathcal{R}$: we just do not allow $\mathcal{R}$ to rewind the attacker.)

$\mathcal{B}$ first receives a problem instance c, which it gives as input to $\mathcal{R}$. $\mathcal{R}$ then outputs public parameters PP. Now $\mathcal{B}$ will simulate the hypothetical attacker described above as follows. First, it will run $\mathcal{R}$ several times in an attempt to collect secret keys. Then it will use the collected keys to simulate the attacker on a new run of $\mathcal{R}$.

More precisely, we let τ be a parameter to be specified later (it will be polynomial in the string length q and the security parameter). $\mathcal{B}$ will choose τ independent random binary strings $x^1, x^2, \ldots, x^\tau$ of length q. It will then query keys for strings $y_1^1, \ldots, y_q^1$ derived from x^1 as described above (note this behavior is identical to the hypothetical attacker $\mathcal{A}$). After receiving each key, it runs the KeyCheck algorithm. If this check ever outputs False, then $\mathcal{B}$ considers this run to be an "aborting run". In addition, $\mathcal{B}$ receives a challenge ciphertext CT. If the CTCheck algorithm run on CT returns false, then it is is also considered to be an "aborting run."[2] If the run was not aborting, then $\mathcal{B}$ successfully received a corresponding key SK_i^1 for each i from 1 to q such that $\mathrm{KeyCheck}(\mathrm{PP}, \mathrm{SK}_i^1, y_i^1) = \text{True}$. It then stores these $\mathrm{SK}_1^1, \ldots, \mathrm{SK}_q^1$ values.

Next, it rewinds the reduction $\mathcal{R}$ to the point just after it output the public parameters. It will then run $\mathcal{R}$ again (using fresh random coins) and querying keys for strings $y_1^2, \ldots, y_q^2$ derived from x_2. It continues in this way until it has run $\mathcal{R}$ exactly τ times on these same PP. If *all* τ runs were aborting runs, then $\mathcal{B}$ stops and guesses randomly. Otherwise, it continues.

Next, it chooses a new random binary string z of length q. If $z = x^i$ for any i from 1 to τ, then $\mathcal{B}$ stops and guesses randomly. Otherwise, it runs $\mathcal{R}$ one more time on these same PP with fresh random coins, querying keys for strings $w_1, \ldots, w_q$ derived from z. Upon receiving each key for $w_1, \ldots, w_q$, it runs the KeyCheck algorithm as before. If any of these checks fail, it stops and guesses randomly. Otherwise, $\mathcal{B}$ submits the fixed, distinct messages M_0, M_1 and the challenge string z to the reduction. $\mathcal{B}$ receives CT in return. It runs the CTCheck algorithm. If this check fails, $\mathcal{B}$ stops and guesses randomly. If the check passes, it fixes and index j from

[2] We observe that for the purposes of collecting private keys, it is not important for the reduction algorithm to return a valid challenge ciphertext. However, we choose to require this to maintain a uniform definition of an "aborting run" in our analysis.

1 to τ such that the j^{th} run was not aborting. Then, it considers the unique y_i^j that is a prefix of z (note that the index i is defined as the first bit where z and x^j differ).

$\mathcal{B}$ now decrypts CT with SK_i^j. If the result is $M_{b'}$ for some $b' \in \{0,1\}$, it guesses b' with probability $\frac{1}{2}+\epsilon$ and guesses the opposite with probability $\frac{1}{2}-\epsilon$. If the result is not M_0 or M_1, it guesses randomly. It gives b' to $\mathcal{R}$, and finally copies the output of $\mathcal{R}$ as its own output.

It is crucial to observe here that $\mathcal{B}$ is decrypting the challenge ciphertext with a secret key that may not be equivalently distributed to the key that the hypothetical attacker $\mathcal{A}$ would use. Nonetheless, since decryption only occurs when the key SK_i^j and the challenge ciphertext CT have passed their respective checks, it must be the case that the decryption of CT by SK_i^j produces the *same result* as decryption of CT by any other acceptable key, hence $\mathcal{B}$ correctly simulates the decryption output that $\mathcal{A}$ would obtain, despite the fact that it is not simulating the proper key distribution.

Analyzing Algorithm $\mathcal{B}$ We recall that C denotes the set of challenges. We let R denote the set of possible random coins chosen by $\mathcal{R}$ for a single run. We introduce the following notation for the coins used by $\mathcal{B}$ during its final run of the reduction algorithm $\mathcal{R}$. Recall, that in a single run the hypothetical attacker's coins is draw from a space $Z \times F$, Z is the choice of possible challenge strings and F is the set of other coins used. For the final run, we let $z \in Z$ and $f \in F$ denote the simulated choice of these coins.

Fixing $c \in C$, $r \in R$, $z \in Z$, and $f \in F$, we define that the tuple (c, r, z, f) belongs to the event W if running the reduction once with this c and these coins r and an attacker using coins z, f results in all the key and ciphertext checks passing and the reduction correctly solving the challenge. (I.e. W is the set of coins for the final run where the final run does not abort and it gives the correct answer.)

We partition the tuples $(c, r, z, f) \in W$ into two disjoint sets. For notational convenience, we split $r \in R$ into substrings r_1 and r_2 such that r_1 are the coins used to determine PP and r_2 are the remaining coins used by the reduction. We let U denote the set of tuples in W such that, fixing c and r_1, replacing the remaining coins for $\mathcal{R}$ and the attacker with freshly sampled coins results in a non-aborting run with probability $\geq \rho$ (where ρ is a threshold we will specify later). We let V denote the set of tuples in W such that this results in a non-aborting run with probability $< \rho$. Note that by definition, W is a disjoint union of U and V. Hence $\mathbb{P}[W] = \mathbb{P}[U] + \mathbb{P}[V]$.

Note that any two runs that share the same c and r_1 coins also share the same challenge and public parameters generated by the reduction. This is the point to which $\mathcal{B}$ rewinds when conducting multiple runs. We can think of these are being "neighboring" sets of runs. Intuitively, we are partitioning the set W into the set U where a neighbor of $u \in U$ is more likely to be non-aborting and the set V where a neighbor of $v \in V$ is less likely to be non-aborting.

Claim 10. $\mathbb{P}[V] < \rho$.

The proof of this claim follows in a similar vein to the heavy row lemma [OO98].

Proof. Given c, r_1, we can define $p(c, r_1)$ to be the probability of a non-aborting run when independent random values of r_2, z, f are chosen and $p'(c, r_1)$ to be the probability of a non-aborting and correct run when independent random values of r_2, z, f are chosen. Then we observe:

$$\mathbb{P}[V] = \sum_{c,r_1 \text{ s.t. } p'(c,r_1)<\rho} \mathbb{P}[c, r_1]p'(c, r_1) \leq \sum_{c,r_1 \text{ s.t. } p(c,r_1)<\rho} \mathbb{P}[c, r_1]p(c, r_1) < \rho \sum_{c,r_1} \mathbb{P}[c, r_1] < \rho.$$

□

We define the event A to be the collection of tuples (c, r, z, f) such that an aborting run is produced (here, we consider an aborting run to include any key check or ciphertext check failure). We note that A is disjoint from W. We let S denote the event that the reduction solves the challenge correctly.

Claim 11. *If Π is computationally hard, then $\mathbb{P}[A]\left|\left(\mathbb{P}[S|A] - \frac{1}{2}\right)\right| = negl(\lambda)$.*

Proof. Suppose that $\mathbb{P}[A]\left(\mathbb{P}[S|A] - \frac{1}{2}\right) = \epsilon' > 0$. We then define the following algorithm $\mathcal{B}'$ to solve Π. $\mathcal{B}'$ chooses random coins for the attacker and runs $\mathcal{R}$ once until either an abort occurs or it reaches the end where the attacker should provide a response. If an abort occurs, then $\mathcal{B}'$ copies the output of the reduction as its own. Otherwise, it guesses randomly.

The success probability of $\mathcal{B}'$ is

$$\frac{1}{2}(1 - \mathbb{P}[A]) + \mathbb{P}[A]\mathbb{P}[S|A] = \frac{1}{2} + \mathbb{P}[A]\left(\mathbb{P}[S|A] - \frac{1}{2}\right) = \frac{1}{2} + \epsilon'.$$

Thus, we must have $\epsilon' = negl(\lambda)$ if Π is computationally hard. The case when $\mathbb{P}[A]\left(\frac{1}{2} - \mathbb{P}[S|A]\right) = \epsilon' > 0$ is analogous, except that $\mathcal{B}'$ should flip the output of the reduction in the case of an abort. □

We observe that the success probability of the reduction (with one run of the hypothetical attacker) is

$$= \mathbb{P}[A]\mathbb{P}[S|A] + \mathbb{P}[W] = \frac{1}{2} + \delta.$$

Combining this with Claim 10 and Claim 11, we see that

$$\frac{1}{2} \cdot \mathbb{P}[A] + \mathbb{P}[U] \geq \frac{1}{2} + \delta - \rho - negl(\lambda). \tag{1}$$

We let X_i, F_i denote the sets of possible coins for the attacker that $\mathcal{B}$ will use during the i^{th} run of $\mathcal{R}$, and we let R_2^i denote the set of possible coins the reduction will use for the i^{th} run. For each i, we define A_i to be the event that $(c, r_1, r_2^i, x_i, f_i)$ produces an aborting run. We define E_i to be the event that $z = x_i$. We let $\overline{A_i}$ and $\overline{E_i}$ denote their complements.

We now consider the probability that $\mathcal{B}$ solves the decisional problem Π. We observe that this is:

$$\geq \frac{1}{2} \cdot \mathbb{P}[A] + \sum_{(c,r,z,f) \in U} \mathbb{P}[c, r, z, f] \cdot \mathbb{P}\left[\bigcup_{i=1}^{\tau} \overline{A_i} \cap \overline{E_i} \mid c, r, z, f\right]. \tag{2}$$

We consider a tuple $(c, r, z, f) \in U$. We observe

$$\mathbb{P}\left[\bigcup_{i=1}^{\tau} \overline{A_i} \cap \overline{E_i} \mid c, r, z, f\right] \geq 1 - \mathbb{P}\left[\bigcup_{i=1}^{\tau} E_i | c, r, z, f\right] - \mathbb{P}\left[\bigcap_{i=1}^{\tau} A_i | c, r, z, f\right].$$

By the union bound, $\mathbb{P}\left[\bigcup_{i=1}^{\tau} E_i | c, r\right] \leq \tau 2^{-q}$. Since the events A_i are independent once c, r, z, f are fixed, we have $\mathbb{P}\left[\bigcap_{i=1}^{\tau} A_i | c, r, z, f\right] \leq (1 - \rho)^{\tau}$ (here we have also used that $(c, r, z, f) \in U$). Thus,

$$\mathbb{P}\left[\bigcup_{i=1}^{\tau} \overline{A_i} \cap \overline{E_i} \mid c, r, z, f\right] \geq 1 - \tau 2^{-q} - (1 - \rho)^{\tau}.$$

Combining this with (2), we see that $\mathcal{B}$ solves the decisional problem Π with probability

$$\geq \frac{1}{2} \cdot \mathbb{P}[A] + \mathbb{P}[U](1 - \tau 2^{-q} - (1 - \rho)^{\tau}).$$

Considering (1), we see this is

$$\geq \frac{1}{2} + \delta - \rho - negl(\lambda) - \tau 2^{-q} - (1-\rho)^{\tau}.$$

Hence, if we set $\rho = \frac{\delta}{4}$, the advantage of $\mathcal{B}$ is at least

$$\frac{3}{4}\delta - negl(\lambda) - \tau 2^{-q} - \left(1 - \frac{\delta}{4}\right)^{\tau}. \tag{3}$$

We now set $\tau = \frac{\lambda}{\delta}$. We observe that $\left(1 - \frac{\delta}{4}\right)^{\frac{1}{\delta}}$ is upper bounded by a constant strictly less than 1, since $\lim_{n\to\infty}\left(1 - \frac{1}{n}\right)^n = \frac{1}{e}$. Hence we see that (3) is $= \frac{3}{4}\delta - \frac{\lambda}{\delta}2^{-q} - negl(\lambda)$. This shows that δ must be exponentially small as a function of q when Π is computationally hard. □

4 Implications for Existing Constructions

Our result can be applied to explain why the first HIBE schemes that were proven secure in the standard model relied on the weaker notion of selective security. Of course, one can easily translate selective security into full security for the same schemes while incurring an exponential loss in terms of the hierarchy depth, as we have shown to be inherent for checkable schemes when using a typical class of reductions.

As an illustrative example, we show that the selectively secure HIBE scheme of Boneh and Boyen [BB04a] is checkable. We first review the scheme. Below, λ denotes the security parameter and q denotes the maximum depth. The scheme will be constructed in a bilinear group G of prime order p. We will assume that identities $\vec{I}$ are vectors of length $\leq q$ whose components are elements of $\mathbb{Z}_p$ and that messages M are elements of G_T.

4.1 The Boneh-Boyen HIBE Construction

Setup$(\lambda, q) \rightarrow \text{MSK}, \text{PP}$ The setup algorithm chooses a bilinear group G of sufficiently large prime order p. We let g denote a generator of G and $e : G \times G \rightarrow G_T$ denote the bilinear map. The algorithm chooses a uniformly random exponent $\alpha \in \mathbb{Z}_p$ and sets $g_1 = g^{\alpha}$. The algorithm also chooses random generators $g_2, h_1, \ldots, h_q \in G$. The MSK is g_2^{α}, while the public parameters are:

$$\text{PP} := \{G, p, e, g, g_1, g_2, h_1, \ldots, h_q\}.$$

Encrypt$(M, \vec{I} = (I_1, \ldots, I_k) \rightarrow \text{CT}$ The encryption algorithm chooses a uniformly random exponent $s \in \mathbb{Z}_p$ and forms the ciphertext as:

$$\text{CT} := \left\{ M e(g_1, g_2)^s,\ g^s,\ \left(g_1^{I_1} h_1\right)^s,\ \ldots,\ \left(g_1^{I_k} h_k\right)^s \right\}.$$

KeyGen$(\vec{I} = (I_1, \ldots, I_k), \text{MSK}) \rightarrow \text{SK}_{\vec{I}}$ The key generation algorithm chooses uniformly random exponents $r_1, \ldots, r_k$ and produces a secret key for identity $\vec{I}$ as:

$$\text{SK}_{\vec{I}} := \left\{ g_2^{\alpha} \prod_{i=1}^{K} \left(g_1^{I_i} h_i\right)^{r_i},\ g^{r_1},\ \ldots,\ g^{r_k} \right\}.$$

We note that delegation here is rather natural, as one can add on a new coordinate I_{k+1} to the identity vector by sampling a new exponent $r_{k+1} \in \mathbb{Z}_p$, multiplying $\left(g_1^{I_{k+1}} h_{k+1}\right)^{r_{k+1}}$ into the first group element, and appending the extra element $g^{r_{k+1}}$ to the current key. However, we will not need to refer to delegation in order to apply our result.

Decrypt$(\mathrm{CT}, \mathrm{SK}_{\vec{I}}) \to \{M, \perp\}$ The decryption algorithm takes in a ciphertext encrypted to an identity vector $\vec{I}^* = (I_1^*, \ldots, I_j^*)$ and a secret key for an identity vector $\vec{I} = (I_1, \ldots, I_k)$. If $\vec{I}$ is not a prefix of $\vec{I}^*$, it outputs $\perp$. Otherwise, it computes the message as follows. We let $\{C, C_0, C_1, \ldots, C_j\}$ denote the elements of the ciphertext, ordered as in the description above. We let $\{K, K_1, \ldots, K_k\}$ similarly denote the elements of the secret key. Then the decryption algorithm computes:

$$M = C \cdot \frac{\prod_{i=1}^{k} e(C_i, K_i)}{e(C_0, K)}.$$

Correctness Observe that when $\vec{I}$ is a prefix of $\vec{I}^*$:

$$C \cdot \frac{\prod_{i=1}^{k} e(C_i, K_i)}{e(C_0, K)} = Me(g, g_2)^{\alpha s} \cdot \frac{\prod_{i=1}^{k} e(g_1^{I_i} h_i, g)^{r_i s}}{e(g, g_2)^{\alpha s} e\left(\prod_{i=1}^{k} (g_1^{I_i} h_i)^{r_i}, g^s\right)} = M.$$

To show this HIBE scheme is checkable, we must specify appropriate efficient algorithms for ciphertext checking and key checking. Our checking algorithms will assume that the bilinear group G comes equipped with an efficient membership test, meaning that an arbitrary object that is purported to be a group element can be efficiently checked for membership in the group. This test is assumed to be perfect (error-free).

CTCheck$(\mathrm{PP}, \mathrm{CT}, \vec{I})$ The ciphertext check algorithm first tests that PP and CT are comprised of the appropriate number of group elements (using the group membership test for G). If any of these tests fail, it outputs False. Otherwise, we let $C, C_0, C_1, \ldots, C_j$ denote the group elements comprising the ciphertext (where $\vec{I}$ has length j) and we let $g, g_1, g_2, h_1, \ldots, h_q$ denote the group elements contained in PP. It is checked that none of PP elements are the identity element. It is then checked that

$$e(C_i, g) = e(C_0, g_1^{I_i} h_i)$$

for each i from 1 to j. If any of these checks fail, the algorithm output False. Otherwise, it outputs True.

KeyCheck$(\mathrm{PP}, \mathrm{SK}_{\vec{I}}, \vec{I} = (I_1, \ldots, I_k))$ The key check algorithm tests that PP and the secret key each contain the correct number of elements, and that all the elements of both are in fact elements of the group G by performing membership tests. If any of these tests fail, the algorithm outputs $False$. Otherwise, we let $K, K_1, \ldots, K_k$ denote the group elements comprising the secret key, and we let $g, g_1, g_2, h_1, \ldots, h_q$ denote the group elements contained in PP. It is checked that none of PP elements are the identity element. Since each of $K_1, \ldots, K_k$ is an element of the cyclic group G and g is a generator, there must exists values $r_1, \ldots, r_k \in \mathbb{Z}_p$ such that $K_1 = g^{r_1}, \ldots, K_k = g^{r_k}$. It remains to check that K is properly formed with respect to these r_i's. To test this, the algorithm computes

$$A := e(g, K),\ B := e(g_1, g_2) \prod_{i=1}^{k} e(K_i, g_1^{I_i} h_i).$$

If $A = B$, the algorithm outputs True. Otherwise, it outputs $False$.

Proposition 12. *The HIBE scheme in Section 4.1 is checkable.*

Proof. We observe that the checking algorithms always output True when parameters, keys, and ciphertexts are honestly generated. Furthermore, when the public parameters and a secret key pass all of the checks, it must be the case that the secret key is correctly formed for some values of $r_1, \ldots, r_k \in \mathbb{Z}_p$. Thus, the secret key will correctly decrypt any honestly generated ciphertext. To see this, note that $A = B$ in the key check if and only if $K = g_2^{\alpha} \prod_{i=1}^{k} (g_1^{I_i} h_i)^{r_i}$ for the r_i values defined from $K_1, \ldots, K_k$. This again relies on the fact that G is a cyclic group generated by g and G_T is also a cyclic group, generated by $e(g, g)$. Hence, $A, B \in G_T$ can only be equal if there discrete logarithms base $e(g, g)$ modulo p are equal. Similarly, a ciphertext can only pass the check if it is properly formed for some value of $s \in \mathbb{Z}_p$.

Hence, for any PP that pass the checks, the set of possible secret keys that pass the key check for a given identity vector is indexed precisely by the p^k possible values of $r_1, \ldots, r_k$, and the possible ciphertexts for a given identity vector are indexed precisely by the p possible vales of s. As a consequence, we see that any two acceptable keys for authorized identity vectors decrypt any acceptable ciphertext to the same message. □

4.2 Other Schemes

The reasoning employed above to analyze the checking algorithms of the Boneh-Boyen HIBE scheme is also applicable to other schemes with similar structure. More specifically, we can apply the same kind of analysis to any scheme with perfect correctness where the sets of possible keys and ciphertexts output by the key generation and encryption algorithms are parameterized by discrete log relationships that can be tested by pairing with public group elements. Other schemes displaying these properties include the Waters IBE and HIBE schemes in [Wat05], the HIBE construction by Boneh, Boyen, and Goh in [BBG05] that achieves compact ciphertexts, the HIBE scheme of Canetti, Halevi, and Katz [CHK03], the HIBE scheme of Gentry and Silverberg [GS02], and the ABE schemes of Goyal, Pandey, Sahai, and Waters [GPSW06] and Waters [Wat11]. Thus, all of these schemes are checkable. (A checkable ABE scheme can be defined analogously to a checkable HIBE scheme, and we show in Appendix A that a checkable ABE scheme can be used to build a checkable prefix encryption scheme.)

The HIBE construction of Gentry and Halevi [GH09] does not conform to this structure and is not checkable (under some computational assumption) - this is why it can avoid exponential degradation in security as the hierarchy depth grows. The later HIBE constructions in [Wat09, LW10] and ABE constructions in [LOS+10, OT10, LW12] that are proven fully secure through the dual system encryption methodology also avoid the basic structure that leads to checkability, even though they can be viewed as alternate instantiations of the intuitive mechanisms of the prior Boneh-Boyen, Boneh-Boyen-Goh, and Goyal-Pandey-Sahai-Waters schemes. More concretely, schemes designed for dual system encryption come equipped with additional dimensions that complicate the landscape of possible keys and ciphertexts. As a consequence of this alteration to the scheme structure, they fall outside the rubric of simple discrete log relationships between pairs of elements in a prime order cyclic group that can be checked by pairing with public elements. (Some dual system constructions use composite order groups for this purpose, and some replace single group elements with larger tuples of group elements.) The additional dimensions that prevent such checks are designed to enable a simulator to produce "semi-functional" keys that still function like honestly generated keys when decrypting honestly generated ciphertexts, but behave differently when decrypting "semi-functional" ciphertexts that cannot be efficiently distinguished from honestly generated ones. This circumvents our lower bound. The situation for the lattice-based HIBE constructions [ABB10, CHKP10] and recent ABE construction [Boy13] is not clear: it would be interesting to determine if they are checkable or not. We summarize the current state of knowledge in Table 1.

Table 1: Checkability of Schemes

	Checkable	Not Checkable	Unknown
HIBE schemes	[GS02],[CHK03],[BB04a], [BB04b], [Wat05]	[GH09],[Wat09], [LW10]	[ABB10, CHKP10]
ABE schemes	[GPSW06, Wat11]	[LOS$^+$10, OT10, LW12]	[Boy13]

5 Prefix Encryption from IBE

We conclude by showing a result in the positive direction; that prefix encryption can actually be built from the simpler primitive of IBE. We prove the reduction secure relative to the IBE scheme with a polynomial loss of security. Since there are known IBE constructions [BF01, Wat05] that are both checkable and have polynomial security reductions to decision assumptions, this might at first seem like a contradiction to our main result. The catch is that our IBE to prefix encryption will not preserve the checkability property (if it existed) of the underlying IBE system.

The transformation itself is very straightforward. For a prefix encryption of message M under string x of length k, define $x^{(i)}$ as the length i prefix of x for each i. To encrypt, simply give k separate IBE encryptions of the message M to the identities $x^{(1)}, x^{(2)}, \ldots, x^{(k)}$ (encoded as strings). A private key for string y is just an IBE secret key for y. To decrypt a ciphertext for x where y is a prefix of x, the algorithm simply looks up the $|y|$-th component and uses the IBE decryption algorithm to decrypt. Security to the underlying IBE systems follows from a straightforward hybrid argument over the IBE ciphertext components comprising the prefix encryption ciphertext.

This new prefix encryption will *not* have our checkability property *even* if the underlying IBE system does. The reason is that if y, y' are secret keys for different prefixes of x, they will decrypt to different messages for certain invalid ciphertexts (and these cannot be discarded with a check). Indeed, the hybrid argument of security precisely relies on this.

This transformation gives one example of where restrictions (in this case the checkability constraint) to our negative result can point to methods for proving security. One interesting aspect of this example is that it falls in a different category than dual system encryption. Dual system encryption leveraged the ability to create two forms (normal and semi-functional) for the *same* key descriptor that would decrypt certain ciphertexts differently. Here we leverage that private keys for *different* key types might decrypt a ciphertext differently.

6 Acknowledgements

We thank the anonymous reviewers for their important points regarding our analysis.

References

[ABB10] S. Agrawal, D. Boneh, and X. Boyen. Efficient lattice (h)ibe in the standard model. In *EUROCRYPT*, pages 553–572, 2010.

[BB04a] D. Boneh and X. Boyen. Efficient selective-id secure identity based encryption without random oracles. In *EUROCRYPT*, pages 223 – 238, 2004.

[BB04b] D. Boneh and X. Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.

[BBG05] D. Boneh, X. Boyen, and E. Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, pages 440–456, 2005.

[BF01] D. Boneh and M. Franklin. Identity based encryption from the weil pairing. In *CRYPTO*, pages 213–229, 2001.

[Boy13] X. Boyen. Attribute-based encryption from lattices. In *TCC*, 2013.

[BSW07] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[CHK03] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.

[CHKP10] D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, pages 523–552, 2010.

[Cor02] J. Coron. Optimal security proofs for pss and other signature schemes. In *EUROCRYPT*, pages 272–287, 2002.

[CS02] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, pages 45–64, 2002.

[Gen06] C. Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, pages 445–464, 2006.

[GH09] C. Gentry and S. Halevi. Hierarchical identity based encryption with polynomially many levels. In *TCC*, pages 437–456, 2009.

[GPSW06] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute based encryption for fine-grained access control of encrypted data. In *ACM conference on Computer and Communications Security*, pages 89–98, 2006.

[GS02] C. Gentry and A. Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, pages 548–566, 2002.

[HJK12] D. Hofheinz, T. Jager, and E. Knapp. Waters signatures with optimal security reduction. In *Public Key Cryptography*, 2012.

[HL02] J. Horwitz and B. Lynn. Toward hierarchical identity-based encryption. In *EUROCRYPT*, pages 466–481, 2002.

[LOS+10] A. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

[LW10] A. Lewko and B. Waters. New techniques for dual system encryption and fully secure hibe with short ciphertexts. In *TCC*, pages 455–479, 2010.

[LW12] A. Lewko and B. Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In *CRYPTO*, pages 180–198, 2012.

[OO98] Kazuo Ohta and Tatsuaki Okamoto. On concrete security treatment of signatures derived from identification. In *CRYPTO*, pages 354–369, 1998.

[OT10] T. Okamoto and K. Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, pages 191–208, 2010.

[SW05] A. Sahai and B. Waters. Fuzzy identity based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[SW08] E. Shi and B. Waters. Delegating capabilities in predicate encryption systems. In *Automata, Languages and Programming*, volume 5126 of *LNCS*, pages 560–578. Springer, 2008.

[Wat05] B. Waters. Efficient identity-based ecnryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.

[Wat09] B. Waters. Dual system encryption: realizing fully secure ibe and hibe under simple assumptions. In *CRYPTO*, pages 619–636, 2009.

[Wat11] B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *PKC*, pages 53–70, 2011.

A Obtaining Prefix Encryption from ABE

We briefly show how to realize a prefix encryption scheme from Key-Policy-ABE. The construction is simply a use case of the underlying ABE system. Therefore the checkable properties are directly inherited in the prefix encryption system. We refer the reader to [GPSW06] for the formal definition of KP-ABE.

Given a Key-Policy ABE scheme with algorithms Setup_{ABE}, KeyGen_{ABE}, Encrypt_{ABE} and Decrypt_{ABE}, we will derive a prefix encryption scheme with algorithms Setup_{Pre}, KeyGen_{Pre}, Encrypt_{Pre}, and Decrypt_{Pre}.

To setup the prefix encryption system for a maximum of length q strings, run Setup_{ABE} with a universe of $2q$ attributes $i:0, i:1$ for $i = 1$ to q.

KeyGen_{Pre} generates a key for a binary string $y = (y_1, y_2, \ldots, y_k)$ for $k \leq q$ by running KeyGen_{ABE} for the policy $1 : y_1$ AND $2 : y_2 \cdots$ AND $k : y_k$.

Encrypt_{Pre} encrypts to a binary vector $x = (x_1, \ldots, x_j)$ by running Encrypt_{ABE} with attributes $1 : x_1, 2 : x_2, \ldots, j : x_j$. Finally, we can then set $\text{Decrypt}_{Pre} = \text{Decrypt}_{ABE}$.

Correctness follows directly from the semantics of KP-ABE. Moreover, since the prefix encryption is directly encoded as a use case of KP-ABE, if the KP-ABE system is checkable, its use as a prefix encryption scheme is also checkable.

It is also relatively straightforward to construct a (checkable) prefix encryption scheme from a (checkable) Ciphertext-Policy-ABE scheme.