Deniable Encryption in a Quantum World

Andrea Coladangelo∗ Shafi Goldwasser† Umesh Vazirani‡

Abstract

(Sender-)Deniable encryption provides a very strong privacy guarantee: a sender who is coerced by
an attacker into “opening” their ciphertext after-the-fact is able to generate “fake” local random choices
that are consistent with any plaintext of their choice. The only known fully-efficient constructions of
public-key deniable encryption rely on indistinguishability obfuscation (iO) (which currently can only be
based on sub-exponential hardness assumptions).

In this work, we study (sender-)deniable encryption in a setting where the encryption procedure is a
quantum algorithm, but the ciphertext is classical. First, we propose a quantum analog of the classical
definition in this setting. We give a fully efficient construction satisfying this definition, assuming the
quantum hardness of the Learning with Errors (LWE) problem.

Second, we show that quantum computation unlocks a fundamentally stronger form of deniable
encryption, which we call perfect unexplainability. The primitive at the heart of unexplainability is
a quantum computation for which there is provably no efficient way, such as exhibiting the “history
of the computation”, to establish that the output was indeed the result of the computation. We give
a construction which is secure in the random oracle model, assuming the quantum hardness of LWE.
Crucially, this notion implies a form of protection against coercion “before-the-fact”, a property that is
impossible to achieve classically.

Contents

1 Introduction 2
1.1 Our Contributions ................................................................. 3
1.2 Related Work and Concepts ..................................................... 7

2 Technical Overview 7
2.1 Notation .............................................................................. 7
2.2 An unexplainable encryption scheme ........................................... 8
2.3 Security .............................................................................. 8
2.3.1 Zhandry’s compressed oracle technique ................................. 9
2.3.2 The structure of strategies that produce valid equations ............ 12
2.3.3 Extracting a claw ................................................................. 13

3 Preliminaries 14
3.1 Notation ............................................................................. 14
3.2 Deniable Encryption ................................................................. 15
3.3 Noisy Trapdoor Claw-Free Functions ........................................... 15

4 Quantum Deniability 17
4.1 Definition ........................................................................... 17
4.2 Construction ........................................................................ 17
4.3 Security ............................................................................. 18

∗UC Berkeley & Simons Institute for the Theory of Computing & qBraid. Email: andrea.coladangelo@gmail.com
†UC Berkeley & Simons Institute for the Theory of Computing. Email: shafi.goldwasser@gmail.com
‡UC Berkeley & Simons Institute for the Theory of Computing. Email: vazirani@berkeley.edu
1 Introduction

This work is motivated by the following overarching question: do local quantum computations alone provide an advantage in cryptography? In other words, is there any quantum advantage in the setting where honest parties can perform local quantum computations but are restricted to sending and storing classical information?

While many examples of quantum advantage are known which leverage quantum communication (or shared entanglement), e.g. key distribution [BB84, Eke92] and oblivious transfer [CK88, BCKM21], or that rely on storing quantum information, e.g. quantum money [Wie83], copy-protection [Aar09], and various other unclonable primitives [BDS16, BL20, CLLZ21], quantum advantage in the setting where parties rely solely on local quantum computations is essentially restricted to certifiable randomness [BCM+18].

Here, we study the notion of deniable encryption in this setting. Deniable encryption was introduced by Canetti et al. [CDNO97]. In a deniable encryption scheme, honest parties are able to generate a “fake” secret key (in the case of receiver deniability) and “fake” randomness (in the case of sender deniability) to claim that the public communication is consistent with any plaintext of their choice. This allows them to preserve the privacy of the true plaintext even if an adversary coerces them after-the-fact into disclosing their private information.

We restrict our attention to non-interactive public-key schemes, and we focus on sender-deniable encryption, namely the setting in which we only protect the sender against coercion by an attacker.

A bit more formally, a public-key encryption scheme is sender-deniable if there exists a “faking” algorithm that takes as input a pair of messages $m_0, m_1$, an encryption $c = \text{Enc}(m_0, r)$, and the randomness $r$ used in the encryption, and outputs some “fake” randomness $r'$, which should look consistent with a genuine encryption of $m_1$. More precisely, the view of an attacker who receives $m_1$, $c$, and the fake randomness $r'$, should be computationally indistinguishable from the view of an attacker who receives $m_1$, along with a genuine encryption of $m_1$, and the true randomness used.

In their original paper [CDNO97], Canetti et al. gave a construction of a deniable encryption scheme where the real and fake views are computationally indistinguishable up to inverse polynomial distinguishing advantage - we will call the distinguishing advantage the “faking probability”. More generally, they show that the size of the ciphertext grows with the inverse of the faking probability.

In a breakthrough work [SW14], Sahai and Waters gave the first construction with negligible faking probability and compact (i.e. polynomial-size) ciphertexts under the assumption that secure indistinguishability obfuscation (iO) exists. Recently, a breakthrough of Jain, Lin and Sahai [JLS21], showed how to construct iO from a few concrete computational hardness assumptions. However, the latter assumptions require sub-exponential hardness of the underlying problems. Moreover, the constructions from [JLS21] are not quantum-secure due to their use of bilinear maps. Other constructions that are based on “LWE-like” assumptions [BDGM20, WW21, GP21, DQV+21] also require sub-exponential hardness for a fundamental reason: they rely on compilers from weaker objects (like functional encryption) to iO, which seem to inherently introduce sub-exponential hardness. Finally, a recent work [AGM21] achieves deniable encryption with compact ciphertexts and negligible faking probability based on polynomial-time hardness of LWE. However, the running time of the encryption algorithm is non-polynomial, growing with the inverse of the...
faking probability. In sum, no constructions of deniable encryption from polynomial-time hardness assumptions achieve negligible faking probability and are fully efficient - namely, achieve compact ciphertexts and polynomial-time encryption and decryption. Thus, the following is an outstanding open question:

*Can fully efficient deniable encryption be based on polynomial-time hardness assumptions?*

In this work, we provide an affirmative answer to this question in the setting where encryption is a *quantum* algorithm, but the ciphertext is *classical*. For simplicity, we will refer to this as the *quantum setting*. Our constructions illustrate that quantum computation provides a fundamentally new kind of advantage for deniability: while classical deniability can only handle coercion after-the-fact (i.e. the attacker approaches the sender after she has sent her ciphertext), quantum computation can also protect against coercion before-the-fact (i.e. the attacker approaches the sender before she sends her ciphertext).

### 1.1 Our Contributions

We propose two notions of deniability in the quantum setting, namely *quantum deniability* and *unexplainability*, and we provide a construction for each. In this section, we give an overview of the two notions and the corresponding constructions. Our second construction satisfies a strong form of unexplainability that we call *perfect unexplainability*. The latter notion implies a form of protection against coercion before-the-fact.

**Quantum Deniability** Recall that the classical definition of deniability is centered around the notion of input randomness. Unfortunately, the latter is not well-defined for a quantum algorithm, since randomness can be intrinsically the result of a measurement, and depends on the basis in which the measurement is carried out. Instead, the correct analog of randomness is the *unmeasured quantum state* of the encryption algorithm.

Now, the strongest definition of quantum deniability that one could imagine would require that revealing the final quantum state of the encryption algorithm (right before measurement) to an attacker does not compromise the secrecy of the plaintext. However, this definition is clearly impossible to achieve, since one can rewind the computation to recover the input state, and hence the plaintext. Instead, in the case where the plaintext is a single bit, our definition of quantum deniability allows the sender to measure a single qubit of the final quantum state of the encryption algorithm (corresponding to one bit of the ciphertext), and requires that revealing all the other qubits to the attacker does not compromise the secrecy of the plaintext. One might expect that such a definition is still impossible to achieve. However, remarkably, there is a construction that achieves it.

The setting is somewhat subtle, so let us be a little more precise. The encryption of a single bit $b$ can be thought of as a pair $(z, \text{aux})$, where $z \in \{0, 1\}$ and $\text{aux} \in \{0, 1\}^n$ for some $n$ (recall that the ciphertext is entirely classical). The recipient (who has the secret key) can efficiently recover $b$ from $z$ using $\text{aux}^1$, while an adversary who only sees $(z, \text{aux})$ has negligible advantage in guessing $b$. For deniability, we require that if the sender’s quantum encryption algorithm were to only measure the qubit register corresponding to $z$, then the remaining final quantum state of the encryption algorithm (which includes $\text{aux}$ in superposition) still does not reveal any non-negligible information about $b$ to the adversary. Note that, in practice, encryption requires the honest sender to measure both $z$ and $\text{aux}$. The definition of quantum deniability is designed to show that even if the sender were to preserve a coherent state over as many qubits as possible – i.e. all but the single qubit corresponding to $z$ – this still does not reveal any information about the plaintext $b$ to the attacker.

**Theorem 1.** There exists a deniable encryption scheme (in the sense of the above definition), assuming the quantum hardness of LWE.

The scheme that makes the theorem true is simple, and is inspired by the use of trapdoor claw-free functions in [BCM18, Mah18].

In essence, in our encryption scheme, the public key is a choice of trapdoor claw-free function pair $(f_{k,0}, f_{k,1})$. The encryption of a single bit $b$ is a pair $(z, \text{aux})$ where $\text{aux} = (d, y)$ such that $z = b \oplus d \cdot (x_0 \oplus x_1)$, where $x$ is special in the sense that toggling it (and leaving $\text{aux}$ unchanged) toggles the plaintext, while $\text{aux}$ plays the role of auxiliary information that specifies the mapping between $z$ and the plaintext.
where \( x_0, x_1 \) are the pre-images of \( y \). In other words, the bit \( d \cdot (x_0 \oplus x_1) \) is used as a one-time pad. CPA security then follows straightforwardly from the fact that \( d \cdot (x_0 \oplus x_1) \) is a hardcore bit, i.e. that it is computationally hard to guess \( d \cdot (x_0 \oplus x_1) \) for uniformly random \( d \) and \( y \).

To prove deniability, we establish that the final quantum state of all registers conditioned on the classical outcome \( z \) still does not leak any information about \( b \).

The intuition for why quantum information is well-suited for deniable encryption is the following. Producing a valid encryption of the desired plaintext is equivalent to producing what is referred to as a valid “equation” in [BCM+18], i.e. a triple of the form \( (d \cdot (x_0 \oplus x_1), d, y) \). The honest execution of the quantum encryption algorithm does not have any input randomness, instead the algorithm must be precisely such that all registers delicately interfere in the right way to result in a valid equation (where the randomness in the equation comes from the final measurement), leaving no other trace of the plaintext in the leftover workspace.

We point out that the scheme that makes Theorem 3 true can be instantiated with any 2-to-1 (noisy) trapdoor claw-free function pair, satisfying the injective invariance property from [Mah18], which in turn can be constructed from the hardness of LWE.

Remark: One might wonder whether it is possible to formulate a definition of quantum deniability more closely aligned with the classical definition. There are two basic obstacles to this:

- Not only can quantum algorithms sample randomness by making measurements, but they can also “cover their tracks” by repeatedly performing measurements in an incompatible basis. As an example, consider a possibly more natural definition of deniability in which the attacker is only revealed the remaining quantum state after both \( z \) and \( aux \) are measured (i.e. whatever remains after the quantum encryption algorithm has produced the entire classical ciphertext). Unfortunately, such a definition would be trivial to satisfy by slightly modifying any classical encryption algorithm, and running it on a quantum computer: one can simply prescribe that the honest quantum encryption algorithm runs the classical encryption algorithm, using a register to sample the randomness, then measures this register in an incompatible basis, and finally appends the classical outcome of this measurement to the ciphertext as part of \( aux \) (decryption then ignores this part of the ciphertext). What this has accomplished is that the leftover quantum state is now essentially garbage, and trivially does not reveal anything to the attacker.

- The notion of a transcript of a computation is not well-defined for a quantum algorithm, in the sense that observing the computation at any step in general disturbs the computation.

It is somewhat fortunate that a stronger definition of quantum deniability, which does not appeal to the inner workings of the encryption algorithm, is achieved by a concrete scheme.

Unexplainable encryption The second notion that we propose, unexplainability, takes a different viewpoint on the concept of input randomness: in the classical setting, input randomness is simply a proxy for a “proof” that the ciphertext is a valid encryption of a certain plaintext. The notion of unexplainability formalizes this notion of a proof, and achieves a single definition that provides a natural common view of deniability in the classical and quantum setting.

In an unexplainable encryption scheme, it is simply impossible, except with negligible probability, for an efficient sender to “prove” after-the-fact that they encrypted a particular plaintext (thus an attacker has simply no reason to bother coercing a sender into opening their ciphertext in the first place). The crux in formalizing this definition is to formalize what it means for a sender to “prove” that they encrypted a particular plaintext. We argue that the appropriate notion of a proof is akin to that of an argument.

Definition 1 (Explainability (informal)). A public-key encryption scheme is explainable if there exists an efficient verification procedure \( \text{Verify} \), taking as input a tuple of public key, ciphertext, message, and alleged proof \((pk, c, m, w)\), such that \( \text{Verify}(pk, c, m, w) = 0 \) if the triple \((pk, c, m)\) is inconsistent. Moreover, \( \text{Verify} \) should satisfy the following:

- Completeness: there exists an efficient procedure that, on input \( m, pk \), generates \( c, w \) such that \( \text{Verify}(pk, c, m, w) \) outputs 1 with high probability.
• Soundness: no efficient procedure, on input $pk, m, m'$ with $m \neq m'$, can generate $c, w, w'$ such that $\text{Verify}(pk, c, m, w)$ and $\text{Verify}(pk, c, m', w')$ both output 1, except with negligible probability.

By contrapositive, a scheme is unexplainable if, for any such $\text{Verify}$, one of completeness or soundness fails. Thus, the sense in which a classical deniable scheme is unexplainable is that a sender can never convincingly “prove” to an attacker that they encrypted a particular plaintext (even if the sender wishes to do so honestly): this is precisely because, by the classical definition of deniability, it is always possible for a sender to efficiently generate randomness consistent with any plaintext of their choice. In other words, a classical deniable encryption scheme is unexplainable because the soundness condition above fails when one takes $\text{Verify}$ to be the procedure that interprets $w$ as the randomness used in the encryption, and simply checks that $c = \text{Enc}(pk, m; w)$.

The notion of unexplainability can be thought of as a rephrasing of deniability from a different perspective. In fact, we show that the appropriate variation on the definition of unexplainability is equivalent to deniability (we refer the reader to the full version for details). However, unlike deniability, the notion of unexplainability has a very natural extension to the quantum setting, as it is not centered around randomness, but rather, more abstractly, around the notion of a proof: without modifications, the definition above makes sense even in the quantum setting (where one may choose to allow the “proof” $w$ to be a quantum state).

Notice that for an encryption scheme with perfect decryption (i.e. one in which there is a unique plaintext consistent with a given ciphertext), the notion of a proof described above coincides with that of an NP (or QMA) proof: since we require that $\text{Verify}(pk, c, m, w) = 0$ if the triple $(pk, c, m)$ is inconsistent, then $\text{Verify}$ is precisely an NP-relation (or QMA-relation) for the language

$$L = \{x = (m, c, pk) : c \text{ is a valid encryption of } m \text{ under } pk\}.$$  

The definition then has the additional requirement of completeness, which asks that there is an efficient procedure that takes as input $pk, m$, and generates valid $c, w$.

Notice that, for encryption schemes with perfect decryption, unexplainability is impossible to achieve classically. First notice, that for a scheme with perfect decryption, the soundness condition in Definition 1 is trivially satisfied. Thus, the scheme is unexplainable if and only if for all $\text{Verify}$ (as in Definition 1) the completeness condition fails. However, the latter is contradicted by the following. Consider the NP-relation $\text{Verify}((m, c, pk), w)$ where the witness is the randomness used to encrypt, i.e. $\text{Verify}((m, c, pk), w) = 1$ if $\text{Enc}(pk, m; w) = c$, and $\text{Verify}((m, c, pk), w) = 0$ otherwise. Then, simply consider the efficient procedure that encrypts honestly, and outputs the randomness as the witness.

In this work, we show the following.

**Theorem 2** (Informal). There exists an unexplainable public-key encryption scheme with perfect decryption, with security in the quantum random oracle model (QROM), assuming the quantum hardness of LWE.

The scheme that makes the theorem true is a variation on the previous one, and is inspired by the follow-up work [BKVV20] to [BCM+18], which makes use of a random oracle.

Again, the public key is a choice of trapdoor claw-free function pair $(f_{k,0}, f_{k,1})$. The encryption of a single bit $b$ is a triple $(z, d, y)$ such that $z = b \oplus d \cdot (x_0 \oplus x_1) \oplus H(x_0) \oplus H(x_1)$, where $x_0, x_1$ are the pre-images of $y$. In other words, now the pad is the bit $d \cdot (x_0 \oplus x_1) \oplus H(x_0) \oplus H(x_1)$. The need for a random oracle stems from the need to obtain a more rigid characterization of the structure of algorithms that produce valid encryptions (we discuss at the end why obtaining a scheme from just LWE may be difficult).

We show that our scheme is unexplainable in the strongest sense: it is simply impossible, except with negligible probability, for an efficient quantum algorithm to produce both a valid encryption and a proof, not just for algorithms that run honest encryption. Using the terminology introduced earlier, we show that for any verification procedure $\text{Verify}$ such that $\text{Verify}(pk, c, m, w) = 0$ if the triple $(pk, c, m)$ is inconsistent, the completeness condition fails. We call this perfect unexplainability.

Notice that perfect unexplainability is impossible to achieve classically, since one can always take $\text{Verify}$ to be the procedure that interprets $w$ as the randomness in the encryption, and runs encryption forward to check consistency.
We find this behaviour to be quite striking, beyond its implications for deniable encryption, and we believe that it has the potential to find applications in other settings.

As a first example, our encryption scheme provides protection against coercion before-the-fact, in the following sense. In the classical world, a coercer who approaches a sender prior to sending an encrypted message can dictate which randomness shall be used by the sender when computing the encryption (and which plaintext shall be encrypted). In this way, the coercer can, at a later stage, check that the ciphertext submitted by the sender corresponds to an encryption of the desired plaintext with the prescribed randomness. This holds true even if the public key (or some other public information to be used in the encryption) is revealed after coercion. This type of coercion is a major concern for electronic elections with online encrypted votes. Our unexplainable encryption scheme prevents coercion before-the-fact since the randomness cannot be controlled, even by the sender themselves: it is instead the result of a carefully chosen measurement. Thus, in a scenario where the public key (or some other public information to be used in the encryption) is revealed after the coercion stage, a before-the-fact coercer would not be able to succeed. In particular, there is simply no way for an attacker to prescribe to the sender how to encrypt in a way that it can later verify, because this would immediately violate perfect unexplainability: it would imply that there exists some quantum algorithm that outputs ciphertexts and proofs of their validity. We discuss coercion before-the-fact in more detail in Section 5.2. As a further practical motivation, the fact that it is not possible for a sender, even if they wanted to, to prove that their ciphertext is a valid encryption of their plaintext provides a way to protect against “vote-selling”.

Lastly, efficiently checkable proofs have been at the center of the stage in complexity and cryptography in the past four decades: proofs with a variety of surprising properties have been discovered and have been the object of intense study (e.g. PCPs, zero-knowledge, succinct). Our work provides a new perspective on the notion of proofs in a quantum world by formalizing the notion of “inability to prove”: this is captured by a computational problem that can be solved efficiently, but for which it is impossible to concurrently provide an efficiently checkable proof of correctness. Classically, the transcript of a computation serves as an efficiently checkable proof that the output is correctly reported. At the heart of the new property is the fact that quantum computations, in general, lack the notion of a transcript or a “trajectory” of the computation. In this work, we identify a computational problem for which it is provably intractable to find a solution while concurrently extracting any meaningful proof that the solution is correct. Using the terminology introduced earlier, a proof is a witness to an appropriate NP (or QMA)-relation Verify.

The analysis in our security proof uses Zhandry’s compressed oracle technique [Zha19], and relies on two novel technical contributions.

(i) The first key step in the security proof is establishing that a strategy which produces valid encryptions must be close to a strategy that queries the oracle at a uniform superposition of the two pre-images (in a sense made more precise in the main text). In a bit more detail, let $H : \{0,1\}^n \rightarrow \{0,1\}$ be a uniformly random function, and let $x_0, x_1 \in \{0,1\}^n$. We prove a technical lemma that characterizes the structure of strategies that are successful at guessing $H(x_0) \oplus H(x_1)$. We use this lemma to derive a corresponding “rigidity” theorem for strategies that produce valid encryptions (equivalently, valid “equations” in the terminology of [BCM+18]). This rigidity theorem may find applications elsewhere, and may be of independent interest.

(ii) The second technical contribution is an online extraction argument which allows to extract a claw from any prover that produces a valid equation and a proof of its validity, with non-negligible probability. The extraction is online in the sense that no rewinding is required.

We point out that the scheme that makes Theorem 2 true can be instantiated with any 2-to-1 (noisy) trapdoor claw-free function pair (the injective invariance property is not needed here). We also remark that for both of our constructions (in Theorems 3 and 5) we do not require an adaptive hardcore bit property.

One final remark is in order. Although a perfectly unexplainable encryption scheme based solely on, say hardness of LWE without the use of a random oracles is desirable, we point out that it is unlikely that such a result can be achieved without any additional assumption. The reason is that it would imply a single-round message-response proof of quantumness protocol [BCM+18]. The following is the protocol:

- The verifier samples $(pk, sk)$ as in the encryption scheme, together with a message $m$ (say uniformly at random). The verifier sends $pk$ and $m$ to the prover.
• The prover returns an encryption $c$ of $m$ under public key $pk$.
• The verifier checks that $c$ decrypts to $m$.

Since perfect unexplainability is impossible to achieve classically, it must be that the encryption algorithm is quantum, and in particular that it cannot be replaced by a classical algorithm (otherwise there would be a way to “explain” by providing the input randomness).

Now, as originally pointed out in [BKVV20], a single-round proof of quantumness immediately implies a separation of the sampling classes $\text{BPP}$ and $\text{BQP}$. Such a separation does not seem to be implied by the hardness of LWE, as the current state-of-the-art suggests that LWE is equally intractable for classical and quantum computers.

1.2 Related Work and Concepts

In the classical setting, receiver-deniable encryption has also been studied. In the latter, an attacker coerces the receiver after-the-fact into revealing their secret key. While non-interactive receiver-deniable encryption is impossible classically, there exists a generic transformation that compiles any sender-deniable encryption scheme into a receiver-deniable one, at the cost of one additional message [CDNO97]. While we do not formalize this explicitly, such a transformation also applies to the quantum setting.

In an interactive setting, one can also consider the notion of bideniable encryption, i.e. an encryption scheme that is simultaneously sender and receiver-deniable. Classically, this setting has been considered by Canetti, Park and Poburinnaya [CPP20], who show that bideniable encryption can be realized from iO. We leave it as an open question to improve this result (i.e. realize bideniable encryption from weaker assumptions) in the quantum setting using classical communication.

Although we do not formalize this, we remark that, if one allows for quantum communication, deniable encryption (including bideniable) becomes immediate in the interactive setting (with information-theoretic security!). The reason is that sender and receiver can share an information-theoretically secure key by running a quantum key distribution protocol, and then use this key as a one-time pad. If the attacker approaches the parties before the key distribution protocol is completed, then the parties can trivially deny since no information about the message has been revealed yet. If the attacker approaches the parties after the key distribution protocol is completed, then the key is information-theoretically hidden from the attacker, and the parties can trivially find a key that is consistent with any message of their choice.

Acknowledgements

This work was carried out while A.C. was a Quantum Postdoctoral Fellow at the Simons Institute for the Theory of Computing supported by NSF QLCI Grant No. 2016245. S.G. is supported by DARPA under agreement No. HR00112020023. U.V is supported by Vannevar Bush faculty fellowship N00014-17-1-3025, MURI Grant FA9550-18-1-0161, and NSF QLCI Grant No. 2016245.

2 Technical Overview

We introduced two notions of deniability in the quantum setting along with two constructions. The first construction, which achieves the notion of quantum deniability is relatively straightforward to explain in full detail, along with a proof of security, and this is carried out in Section 4. The second construction, which achieves the notion of unexplainability, and its proof of security are more involved. In this section, we give an overview of the latter construction, and the proof of security.

2.1 Notation

In this overview, we let $\{(f_{k,0}, f_{k,1})\}_k$ be a family of trapdoor claw-free function pairs. We assume that $f_{k,0}, f_{k,1}$ are injective with identical range, and that they map $n$-bit strings to $m$-bit strings. In our actual scheme, we will instead use noisy trapdoor claw-free functions, since these can be constructed from LWE. However, this distinction is immaterial for the purpose of this overview.
2.2 An unexplainable encryption scheme

The construction is simple, and is inspired by the “proof of quantumness” construction in [BKVV20]. To obtain an encryption scheme, the idea is to use the “hardcore” bit in their construction as a one-time pad.

In more detail, the public key in the encryption scheme is a choice of trapdoor claw-free function \( k \), and the secret key is a trapdoor \( t_k \).

- **Encryption:** To encrypt a bit \( m \), under public key \( k \), Compute a triple \( (z, d, y) \), where \( d \in \{0, 1\}^n \) and \( y \in \mathcal{Y} \), and \( z = d \cdot (x^y_0 \oplus x^y_1) \oplus H(x^y_0) \oplus H(x^y_1) \). This can be done as follows:
  - Create the uniform superposition over \( n + 1 \) qubits \( \sum_{b \in \{0, 1\}, x \in \{0, 1\}^n} |b\rangle |x\rangle \). Then, compute \( f_{k,0} \) and \( f_{k,1} \) in superposition, controlled on the first qubit. The resulting state is
    \[
    \sum_{b \in \{0, 1\}, x \in X} |b\rangle |x\rangle |f_{k,b}(x)\rangle .
    \]
  - Measure the image register, and let \( y \) be the outcome. As a result, the state has collapsed to:
    \[
    \frac{1}{\sqrt{2}} (|0\rangle |x^0_0\rangle + |1\rangle |x^1_1\rangle ) .
    \]
  - Query the phase oracle for \( H \), to obtain:
    \[
    \frac{1}{\sqrt{2}} ((-1)^{H(x_0)} |0\rangle |x_0\rangle + (-1)^{H(x_1)} |1\rangle |x_1\rangle ) .
    \]
  - Apply the Hadamard gate to all \( n + 1 \) registers, and measure. Parse the measurement outcome as \( z|d \) where \( z \in \{0, 1\} \) and \( d \in \{0, 1\}^n \).

The ciphertext is \( c = (m \oplus z, d, y) \).

- **Decryption:** On input \( c = (\hat{z}, d, y) \), use the trapdoor \( t_k \) to compute the pre-images \( x^y_0, x^y_1 \). Output \( \hat{z} \oplus d \cdot (x^y_0 \oplus x^y_1) \oplus H(x^y_0) \oplus H(x^y_1) \).

The actual scheme will be a parallel repetition of this, i.e. the plaintext \( m \) is encrypted many times using the single-shot scheme described here. However, for the purpose of this overview, we will just consider the single-shot scheme.

2.3 Security

It is straightforward to see that the scheme satisfies CPA security. This essentially follows from a regular “hardcore bit” property, satisfied by the trapdoor claw-free function family (more details in Section 4.3). In this overview, we focus on outlining how the scheme satisfies (a strong version of) unexplainability.

Our main result is that this scheme has the property that, although it is possible to encrypt, it is not possible to simultaneously produce both a valid encryption of a desired plaintext \( m \) and a “proof” or a “certificate” \( w \) that the ciphertext indeed is an encryption of \( m \). We refer to this as perfect unexplainability.

More precisely, we show that, for any efficient algorithm \( \text{Verify} \) taking as input a tuple \((pk, c, m, w)\), such that \( \text{Verify}(pk, c, m, w) = 0 \) if the triple \((pk, c, m)\) is inconsistent, the following holds: for any efficient algorithm \( P \), for any \( m \),

\[
\Pr[\text{Verify}(pk, c, m, w) = 1 : (c, w) \leftarrow P(pk, m)] = \negl(n) .
\] (1)

Notice that perfect unexplainability is impossible to achieve with a classical encryption scheme, because one can always have \( w \) play the role of the randomness in the encryption, and have \( \text{Verify} \) be the algorithm that simply outputs \( 1 \) if \( \text{Enc}(pk, m; w) = c \), and \( 0 \) otherwise.

A consequence of this observation is that perfect unexplainability is an even stronger property than a “proof of quantumness”: if an encryption scheme is unexplainable, then it must be that producing valid encryptions (with high enough probability) is something that only a quantum computer can do (otherwise,
there would exist a classical algorithm $Enc$ that encrypts successfully, and, just like before, we can set $Verify$ to be the procedure that interprets $w$ as randomness, runs $Enc$ with this randomness, and checks consistency with the ciphertext).

For simplicity, in the rest of this discussion, we will take the plaintext to be $m = 0$, so that producing a valid ciphertext is equivalent to producing a valid equation, i.e. a triple $(z, d, y)$, such that $z = d \cdot (x_0^y \oplus x_1^y) \oplus H(x_0^y) \oplus H(x_1^y)$.

So, what is the intuition for why the property above holds for our scheme? At a high level, the intuition is that in order to produce a valid equation, one has to query the oracle on a superposition of the two pre-images, and have the two branches interfere in just the right way, so as to produce $z, d$ which satisfy the equation. This delicate process of interference between the two branches essentially requires that nothing is “left behind” in the workspace.

The difficulty with formalizing this intuition is that it assumes a global view of the entire computation. Whereas unlike in the classical setting, we cannot in general define the trajectory which in this case corresponds to the sequence of oracle queries that led to a particular configuration. To do so, we make use of Zhandry’s compressed oracle technique, which leverages properties of uniformly random oracles. It turns out that the cryptographic assumptions interact very cleanly with the compressed oracle formalism. This provides a way of carrying key quantities from our classical intuition over into natural formalizations in the quantum context.

This reduction involves two broad steps:

(i) We establish a rigidity theorem which formalizes the intuition that an algorithm $P$ which is successful at producing valid equations must query the oracle on a superposition of both pre-images, in a sense that we will make more precise below. Concisely, we appeal to Zhandry’s compressed oracle technique for “recording queries” [Zha19], which formalizes the idea that, when the oracle is uniformly random, there is a meaningful way to record the queries made by the algorithm efficiently, in a way that is well-defined and does not disrupt the run of the algorithm. In a compressed oracle simulation, the quantum state of the algorithm at any point is in a superposition over databases of queried inputs. What we establish is that, if $P$ produces a valid equation with high probability, then the quantum state of $P$ right before measurement of the equation must be close to a uniform superposition (with the appropriate phase) of two branches: one on which the first pre-image was queried, and one in which the second was queried.

(ii) We leverage (i) to construct an algorithm that extracts a claw. One crucial observation here is that, since $Verify$ never accepts an inconsistent tuple $(pk, c, m)$, then it must be the case that whenever $Verify$ accepts, it must itself have queried at a superposition of both pre-images, in the same sense as in the rigidity theorem in point (i). At a high level, the extraction algorithm will eventually be the following:

- Run $P$, followed by a measurement to obtain $z, d, y$, and an appropriate measurement of the database register, hoping to find a pre-image of $y$;
- Then, run $Verify$ on the leftover state, and conditioned on “accept”, measure the database register, hoping to find the other pre-image of $y$.

It is not a priori clear that this leads to successfully recovering a claw (in particular the measurement of the database right after running $P$, may disrupt things in a way that $Verify$ no longer accepts). What we show is that if, to begin with, $Verify$ accepts with high enough probability, then this extraction strategy works.

In the rest of the section, we discuss the elements of the security proof in a bit more detail.

### 2.3.1 Zhandry’s compressed oracle technique

In this subsection, we give an exposition of Zhandry’s compressed oracle technique, both because it is a building block in our proof of security, and also to encourage its broader use. A reader who is familiar with the technique should feel free to skip this subsection.
Let \( H : \{0,1\}^n \rightarrow \{0,1\} \) be a fixed function. For simplicity, in this overview we restrict ourselves to considering boolean functions (since this is also the relevant case for our scheme).

While classically it is always possible to record the queries of the algorithm, in a way that is undetectable to the algorithm itself, this is not possible in general in the quantum case. The issue arises because the quantum algorithm can *query in superposition*. We illustrate this with an example.

Consider an algorithm that prepares the state \( \frac{1}{\sqrt{2}}(|x_0\rangle + |x_1\rangle) |y\rangle \), and then makes an oracle query to \( H \).

The state after the query is:
\[
\frac{1}{\sqrt{2}} |x_0\rangle |y \oplus H(x_0)\rangle + \frac{1}{\sqrt{2}} |x_1\rangle |y \oplus H(x_1)\rangle
\]
(2)

Suppose we additionally “record” the query made, i.e. we copy the queried input into a third register. Then the state becomes:
\[
\frac{1}{\sqrt{2}} |x_0\rangle |y \oplus H(x_0)\rangle |x_0\rangle + \frac{1}{\sqrt{2}} |x_1\rangle |y \oplus H(x_1)\rangle |x_1\rangle
\]
(3)

Now, suppose that \( H(x_0) = H(x_1) \), then it is easy to see that, in the case where we didn’t record queries, the state of the first register after the query is exactly \( \frac{1}{\sqrt{2}}(|x_0\rangle + |x_1\rangle) \). On the other hand, if we recorded the query, then the third register is now entangled with the first, and as a result the state of the first register is no longer \( \frac{1}{\sqrt{2}}(|x_0\rangle + |x_1\rangle) \) (it is instead a mixed state). Thus, recording queries is not possible in general without disturbing the state of the oracle algorithm.

Does this mean that all hope of recording queries is lost in the quantum setting? It turns out, perhaps surprisingly, that there is a way to record queries when \( H \) is a *uniformly random* oracle.

When thinking of an algorithm that queries a uniformly random oracle, it is useful to purify the quantum state of the algorithm via an oracle register (which keeps track of the function that is being queried). An oracle query is then a unitary that acts in the following way on a standard basis element of the query register (where we omit writing normalizing constants):
\[
|x\rangle |y\rangle \sum_H |H\rangle \mapsto \sum_H |x\rangle |y \oplus H(x)\rangle |H\rangle
\]

It is well-known that, up to applying a Hadamard gate on the \( y \) register before and after a query, this oracle is equivalent to a “phase oracle”, which acts in the following way:
\[
|x\rangle |y\rangle \sum_H |H\rangle \mapsto \sum_H (-1)^{y \cdot H(x)} |x\rangle |y\rangle |H\rangle
\]
(4)

Now, to get a better sense of what is happening with each query, let’s be more concrete about how we represent \( H \) using the qubits in the oracle register.

A natural way to represent \( H \) is to use \( 2^n \) qubits, with each qubit representing the output of the oracle at one input, where we take the inputs to be ordered lexicographically. In other words, if \( |H\rangle = |t\rangle \), where \( t \in \{0,1\}^{2^n} \), then this means that \( H(x_i) = t_i \), where \( x_i \) is the \( i \)-th \( n \)-bit string in lexicographic order. Using this representation, notice that
\[
\frac{1}{\sqrt{2^n}} \sum_H |H\rangle = |+\rangle \otimes 2^n
\]

Now, notice that we can write the RHS of (4) as
\[
|x\rangle |y\rangle \sum_H (-1)^{y \cdot H(x)} |H\rangle
\]

i.e. we can equivalently think of the phase in a phase oracle query as being applied to the oracle register.

Thus, when a phase oracle query is made on a standard basis vector of the query register \( |x\rangle |y\rangle \), all that happens is:
\[
\sum_H |H\rangle \mapsto \sum_H (-1)^{y \cdot H(x)} |H\rangle
\]

Notice that, using the representation for \( H \) that we chose above, the latter transformation is:
• When $y = 0$, 
\[ |+\rangle^\otimes 2^n \mapsto |+\rangle^\otimes 2^n. \]

• When $y = 1$, 
\[ |+\rangle^\otimes 2^n \mapsto |+\rangle \cdots |+\rangle_{i-1} |-\rangle_i |+\rangle_{i+1} \cdots |+\rangle, \]
where $i$ is such that $x$ is the $i$-th string in lexicographic order.

In words, the query does not have any effect when $y = 0$, and the query flips the appropriate $|+\rangle$ to a $|-\rangle$ when $y = 1$. Then, when we query on a general state $\sum_{x,y} \alpha_{xy} |x\rangle |y\rangle$, the state after the query can be written as:
\[ \sum_{x,y} \alpha_{xy} |x\rangle |y\rangle |D_{xy}\rangle, \]
where $D_{xy}$ is the all $|+\rangle$ state, except for a $|-\rangle$ corresponding to $x$ if $y = 1$.

The crucial observation now is that all of these branches are orthogonal, and thus it makes sense to talk about "the branch on which a particular query was made": the state of the oracle register reveals exactly the query that has been made on that branch. More generally, after $q$ queries, the state will be in a superposition of branches on which at most $q$ of the $|+\rangle$'s have been flipped to $|-\rangle$'s. These locations correspond exactly to the queries that have been made.

Moreover, the good news is that there is a way to keep track of the recorded queries efficiently: one does not need to store all of the (exponentially many) $|+\rangle$'s, but it suffices to keep track only of the locations that have flipped to $|-\rangle$ (which is at most $q$). If we know that the oracle algorithm makes at most $q$ queries, then we need merely $n \cdot q$ qubits to store the points that have been queried. We will refer to the set of queried points as the database. Formally, there is a well-defined isometry that maps a state on $2^n$ qubits where $q$ of them are in the $|-\rangle$ state, and the rest are $|+\rangle$, to a state on $n \cdot q$ qubits, which stores the $q$ points corresponding to the $|-\rangle$'s in lexicographic order.

Let $D$ denote an empty database of queried points. Then a query to a uniformly random oracle can be thought of as acting in the following way:

\[
\begin{align*}
|x\rangle |y\rangle |D\rangle &\mapsto |x\rangle |y\rangle |D\rangle, & \text{if } y = 0 \\
|x\rangle |y\rangle |D\rangle &\mapsto |x\rangle |y\rangle |D \cup \{x\}\rangle, & \text{if } y = 1.
\end{align*}
\]

Such a way of implementing a uniformly random oracle is referred to as a compressed phase oracle simulation [Zha19]. Formally, the fact that the original and the compressed oracle simulations are identical from the point of view of the oracle algorithm (which does not have access to the oracle register) is because at any point in the execution of the algorithm, the states in the two simulations are both purifications of the same mixed state on the algorithm’s registers.

We point out that there are two properties of a uniformly random oracle that make a compressed oracle simulation possible:

• The query outputs at each point are independently distributed, which means that the state of the oracle register is always a product state across all of the $2^n$ qubits.

• Each query output is uniformly distributed. This is important because in general $\alpha |0\rangle + \beta |1\rangle \neq \alpha |0\rangle - \beta |1\rangle$ unless $|\alpha| = |\beta|$.

Notice that the above compressed oracle simulation does not explicitly keep track of the value of the function at the queried points (i.e. a database is just a set of queried points). In the following slight variation on the compressed oracle simulation, also from [Zha19], a database is instead a set of pairs $(x, w)$ representing a queried point and the value of the function at that point. This variation will be more useful for our analysis.

Here $D$ is a database of pairs $(x, v)$, which is initially empty. A query acts as follows on a standard basis element $|x\rangle |y\rangle |D\rangle$:

• If $y = 0$, do nothing.

• If $y = 1$, check if $D$ contains a pair of the form $(x, v)$ for some $v$.
If it does not, add $(x, |\rangle)$ to the database, where by this we formally mean: $D \mapsto \sum_v (-1)^v |D \cup (x, v)\rangle$

If it does, apply the unitary that removes $(x, |\rangle)$ from the database.

One way to understand this compressed simulation is that our database representation only keeps track of pairs $(x, |\rangle)$ (corresponding to the queried points), and it does not keep track of the other unqueried points, which in a fully explicit simulation would correspond to $|+\rangle$'s. One can think of the outputs at the unqueried points as being “compressed” in this succinct representation.

It is easy to see that the map above can be extended to a well-defined unitary. In the rest of this overview, we will take this to be our compressed phase oracle. For an oracle algorithm $A$, we will denote by $A^{CPhO}$ the algorithm $A$ run with a compressed phase oracle.

### 2.3.2 The structure of strategies that produce valid equations

Let $P$ be an efficient prover, which takes as input a choice $k$ of claw-free function pair, and outputs an equation $(z, d, y)$. Suppose $P$ outputs a valid equation, i.e. $z = d \cdot (x_0^y \oplus x_1^y) \oplus H(x_0^y) \oplus H(x_1^y)$, with high probability. Suppose we run a compressed phase oracle simulation $P^{CPhO}$. We argue that the state of $P^{CPhO}$ right before measurement, conditioned on output $(z, d, y)$, must be such that:

- Almost all the weight is on databases containing exactly one of the two pre-images of $y$.
- Up to a phase, the weights on databases containing $x_0^y$ and $x_1^y$ are approximately symmetrical.

In this subsection, we provide some intuition as to why this is true.

In general, we can write the state of $P^{CPhO}$ right before measurement of the output as:

$$\sum_{z,d,y,w,D} \alpha_{z,d,y,w,D} |z\rangle |d\rangle |y\rangle |w\rangle |D\rangle,$$

for some $\alpha_{z,d,y,w,D}$, where the $z, d, y$ registers correspond to the output equation, the $w$ register is a work register (which includes the query registers), and the $D$ register is the database register. Here $D$ is a database of pairs $(x, v)$.

This expression can be simplified in a couple of ways. First, up to negligible weight, no database can contain both pre-images, since otherwise this would yield an efficient algorithm to extract a claw. Hence, we can write the state as a superposition over $z, d, y$ and over databases that either: do not contain any pre-image $y$, they contain only $x_0^y$, or they contain only $x_1^y$:

$$\sum_{z,d,y,w,D,0} \alpha_{z,d,y,w,D,0} |z\rangle |d\rangle |y\rangle |w\rangle \sum_{v_0} (-1)^{v_0} |D \cup (x_0^y, v_0)\rangle$$

$$+ \sum_{z,d,y,w,D,1} \alpha_{z,d,y,w,D,1} |z\rangle |d\rangle |y\rangle |w\rangle \sum_{v_1} (-1)^{v_1} |D \cup (x_1^y, v_1)\rangle$$

$$+ \sum_{z,d,y,w,D,b} \beta_{z,d,y,w,D,b} |z\rangle |d\rangle |y\rangle |w\rangle |D\rangle \tag{5}$$

for some $\alpha_{z,d,y,w,D,b}, \beta_{z,d,y,w,D}$. (Recall that the reason for the presence of the phases $(-1)^{v_0}$ and $(-1)^{v_1}$ is that, by definition of the compressed oracle simulation, the output at each queried point in the database is in a $|\rangle$ state).

Second, we expect intuitively that $P$ should not be able to produce valid equations if it does not query any pre-image at all. Thus, we expect that if $P$ produces a valid equation with high probability, then there should be only a small weight on the third branch in expression (5), i.e. the $\beta_{z,d,y,w,D}$ coefficients should be small. In our security proof, we formalize this intuition, and we show that any weight on the third branch contributes precisely $1/2$ to the probability of producing a valid equation, i.e. any weight on the third branch amounts to guessing an equation uniformly at random. Note that it is not a priori clear that this is true, and a more deliberate analysis is required to establish that, when calculating the probability of outputting a
valid equation, there is no interference between the branches that do not contain any pre-image, and the ones that contain one. We refer the reader to the full proof in Section 6 for more details.

So, suppose now for simplicity that \( P \) produces a valid equation with probability 1, then, from what we have discussed, up to negligible weight, the state just before measurement of the output is in a superposition of branches that contain exactly one of the two pre-images:

\[
\sum_{z,d,y,w} \alpha_{z,d,y,w,D,0} \left| z \right> \left| d \right> \left| y \right> \left| w \right> \sum_{v_0 \in \{0,1\}} (-1)^{v_0} \left| D \cup (x^y_0, v_0) \right> \\
+ \sum_{z,d,y,w} \alpha_{z,d,y,w,D,1} \left| z \right> \left| d \right> \left| y \right> \left| w \right> \sum_{v_1 \in \{0,1\}} (-1)^{v_1} \left| D \cup (x^y_1, v_1) \right>
\]  

(6)

Finally, we wish to argue that the coefficients on the two branches are uniform, i.e. \( |\alpha_{z,d,y,w,D,0}| = |\alpha_{z,d,y,w,D,1}| \) for all \( z, d, y, w, D \). For this, we again appeal to the fact that \( P \) produces a valid equation with probability 1.

What does this probability correspond to in terms of expression (6)? In order to calculate this probability, we need to first decompress the database at both pre-images. What we mean by decompressing at \( x \) is the following:

- If the database already contains \( x \), then do nothing.
- If the database does not contain \( x \), then add \((x, |+)\) to it.

The reason why this makes sense is the following. Recall that points that are not present in the compressed database correspond to \(|+\)'s in the fully explicit “uncompressed” database. Since the condition for a valid equation depends on the value at both pre-images, we need to keep track of these values, and uncompress at the two pre-images in order to talk about the probability of a valid equation.

The state after decompressing at \( x^y_0 \) and \( x^y_1 \) is:

\[
\sum_{z,d,y,w} \alpha_{z,d,y,w,D,0} \left| z \right> \left| d \right> \left| y \right> \left| w \right> \sum_{v_0,v_1} (-1)^{v_0} \left| D \cup (x^y_0, v_0) \cup (x^y_1, v_1) \right> \\
+ \sum_{z,d,y,w} \alpha_{z,d,y,w,D,1} \left| z \right> \left| d \right> \left| y \right> \left| w \right> \sum_{v_0,v_1} (-1)^{v_1} \left| D \cup (x^y_0, v_0) \cup (x^y_1, v_1) \right>
\]  

(7)

Now, by the equivalence of the regular and compressed oracle simulations, we have that the probability that \( P \) outputs a valid equation is equal to:

\[
\Pr[z = d \cdot (x^y_0 + x^y_1) \oplus v_0 \oplus v_1]
\]

In order for this probability to be close to 1, it must be that, for any \( z, d, y, w, D \), the amplitudes of the two branches \( \alpha_{z,d,y,w,D,0} \) and \( \alpha_{z,d,y,w,D,1} \) interfere precisely constructively when \( z = d \cdot (x^y_0 \oplus x^y_1) \oplus v_0 \oplus v_1 \) and interfere destructively otherwise. Hence, they have to be equal up to an appropriate phase.

### 2.3.3 Extracting a claw

We now provide some intuition about how the structure that we derived in the previous subsection can be leveraged to extract a claw.

Throughout this subsection, let \( P \) be such that, on input a choice of trapdoor claw-free function pair \( k \), it simultaneously produces valid equations and proofs which are accepted with high probability, i.e. it outputs \((z, d, y, w)\) such that \( \text{Verify}(k, z, d, y, w) = 1 \) with high probability. For concreteness we take this probability to be 9/10 in this overview.

The first key observation is that, because by definition \( \text{Verify} \) never accepts an invalid equation (or except with negligible probability), then it must be the case that whenever \( \text{Verify} \) accepts, it must itself have queried at a superposition of both pre-images in the following more precise sense.
For a choice of claw-free functions \( k \), denote by \( \text{Verify}_k \) the algorithm \( \text{Verify} \) where we fix the choice of claw-free functions to be \( k \). Let \( A \) be any efficient oracle algorithm that, on input a choice of claw-free functions, outputs tuples \((z, d, y, w)\). Suppose we run a compressed oracle simulation of \( A \), followed by \( \text{Verify} \), i.e. we run \( \left( \text{Verify}_k \circ A(k) \right)^{\text{CPhO}} \). Let the state right before measurement of \( \text{Verify} \)'s output be:

\[
\alpha |0\rangle |\phi_0\rangle + \beta |1\rangle |\phi_1\rangle,
\]

where the first qubit is the output qubit of \( \text{Verify} \), and \(|\phi_0\rangle, |\phi_1\rangle\) are some states on the remaining registers (including the database register). Then, except with negligible probability over the choice of \( k \), the following holds: if \( \beta \) is non-negligible, then \(|\phi_1\rangle\) has weight only on databases containing either \( x_0^y \) or \( x_1^y \), and moreover, the weights for each pre-image are approximately equal. Such structure on the action of \( \text{Verify} \) follows from a similar argument as in the previous subsection, combined with the fact that \( \text{Verify} \) never accepts an invalid equation.

Even with this observation in hand, it is not a priori clear how we can extract a claw: ideally, we would\( \alpha \) like to say that, because of the observation above, if we were to run \( \left( \text{Verify}_k \circ P(k) \right)^{\text{CPhO}} \) twice and measure the database register, there would be a noticeable chance of obtaining distinct pre-images of some \( y \). However, the issue is that if we run the computation twice, nothing guarantees that we will obtain the same \( y \) both times. In fact, if one thinks about the honest strategy for producing valid equations, each \( y \) has only an exponentially small probability of being the outcome.

To overcome this issue, the key observation is that, if \( P \) is successful with high enough probability, then there is a way to extract a claw with noticeable probability in a single run! The extraction algorithm is the following:

1. Run \( P^{\text{CPhO}}(k) \), and measure the output registers to obtain \( z, d, y \). Moreover, check if the database register at this point contains a pre-image of \( y \). If so, measure it. Denote this by \( x_0^y \).
2. Run \( \text{Verify}_k^{\text{CPhO}} \) on the leftover state from the previous step, and measure the output register. Conditioned on “accept”, measure the database register. If the database contains \( x_0^y \), output the claw \((x_0^y, x_1^y)\).

The idea behind this algorithm is that, thanks to the first observation, the state conditioned on obtaining “accept” in step (ii) has weight only on databases containing either \( x_0^y \) or \( x_1^y \), and moreover, the weights for each pre-image are approximately equal. This implies that, conditioned on observing “accept” in step (ii), the final measurement of the algorithm is guaranteed to produce one of the two pre-images approximately uniformly at random.

Now, notice that the algorithm already has a constant probability of obtaining one of the two pre-images in step (i) (assuming \( P \) succeeds with probability \( \frac{9}{10} \)). To see this, notice that if this weren’t the case, then \( P \) would be producing valid equations at most with probability close to \( \frac{1}{3} \) (since this is the probability of producing a valid equation when the database register does not contain any of the two pre-images), and therefore the probability that \( \text{Verify} \) accepts \( P \)'s output would also be at most close to \( \frac{1}{3} \) (instead of being \( \frac{9}{10} \)). Thus, what is left to show is that the probability of obtaining “accept” in step (ii) conditioned on finding a pre-image in step (i) is still noticeable.

It is not a priori clear that this is the case. However, what we show is that if, to begin with, \( \text{Verify} \) accepts \( P \)'s output with high enough probability (in fact any probability non-negligibly higher than 1/2), then this extraction strategy works.

We remark that for our actual construction, we will want that, for any efficient \( P \), the probability of simultaneously producing a valid equation and a proof is negligible (not just smaller than \( \frac{1}{2} + \) negligible). Thus, our encryption scheme will be a parallel repetition of the single-shot encryption scheme described in this overview, i.e. the encryption of a single bit will consist of many single-shot encryptions of that bit.

3 Preliminaries

3.1 Notation

We use the acronyms PPT and QPT for probabilistic polynomial time and quantum polynomial time respectively.
For a classical probabilistic algorithm $\mathcal{A}$, we write $\mathcal{A}(x;r)$ to denote running $\mathcal{A}$ on input $x$, with input randomness $r$.

For a finite set $S$, we use $x \leftarrow S$ to denote uniform sampling of $x$ from the set $S$. We denote $[n] = \{1, 2, \ldots, n\}$. We denote by $\text{Bool}(n)$ the set of functions from $n$ bits to 1.

### 3.2 Deniable Encryption

We recall the classical notion of sender-deniable encryption.

**Definition 2.** A public-key encryption scheme $(\text{Gen}, \text{Enc}, \text{Dec})$ is said to be deniable if there exists a PPT algorithm $\text{Fake}$ such that, for any messages $m_0, m_1$:

$$(\text{pk}, \text{Enc}(\text{pk}, m_1; r), m_1, r) \approx_c (\text{pk}, \text{Enc}(\text{pk}, m_0; r), m_1, r')$$

where $r$ is uniformly random, $\text{pk}$ is sampled according to $\text{Gen}$, and $r' \leftarrow \text{Fake}(\text{pk}, c, m_0, m_1, r)$.

### 3.3 Noisy Trapdoor Claw-Free Functions

In this section we introduce the notion of noisy trapdoor claw-free functions (NTCFs). This section is taken almost verbatim from [BKVV20]. Let $\mathcal{X}, \mathcal{Y}$ be finite sets and $\mathcal{K}$ a set of keys. For each $k \in \mathcal{K}$ there should exist two (efficiently computable) injective functions $f_{k,0}, f_{k,1}$ that map $\mathcal{X}$ to $\mathcal{Y}$, together with a trapdoor $t_k$ that allows efficient inversion from $(b, y) \in \{0, 1\} \times \mathcal{Y}$ to $f_{k,b}^{-1}(y) \in \mathcal{X} \cup \{\perp\}$. For security, we require that for a randomly chosen key $k$, no polynomial time adversary can efficiently compute $x_0, x_1 \in \mathcal{X}$ such that $f_{k,0}(x_0) = f_{k,1}(x_1)$ (such a pair $(x_0, x_1)$ is called a claw).

Unfortunately, we do not know how to construct such ‘clean’ trapdoor claw-free functions. Hence, as in the previous works [BCM+18, Mah18, BKVV20], we will use ‘noisy’ version of the above notion. For each $k \in \mathcal{K}$, there exist two functions $f_{k,0}, f_{k,1}$ that map $\mathcal{X}$ to a distribution over $\mathcal{Y}$.

**Definition 3 (NTCF family).** Let $\lambda$ be a security parameter. Let $\mathcal{X}$ and $\mathcal{Y}$ be finite sets. Let $\mathcal{K}_\mathcal{X}$ be a finite set of keys. A family of functions

$$F = \{f_{k,b} : \mathcal{X} \to \mathcal{D}_\mathcal{Y}\}_{k \in \mathcal{K}_\mathcal{X}, b \in \{0, 1\}}$$

is called a noisy trapdoor claw free (NTCF) family if the following conditions hold:

1. **Efficient Function Generation.** There exists an efficient probabilistic algorithm $\text{GEN}_F$ which generates a key $k \in \mathcal{K}_\mathcal{X}$ together with a trapdoor $t_k$:

$$(k, t_k) \leftarrow \text{GEN}_F(1^\lambda).$$

2. **Trapdoor Injective Pair.**

   (a) **Trapdoor:** There exists an efficient deterministic algorithm $\text{INV}_F$ such that with overwhelming probability over the choice of $(k, t_k) \leftarrow \text{GEN}_F(1^\lambda)$, the following holds:

   $$\text{for all } b \in \{0, 1\}, x \in \mathcal{X} \text{ and } y \in \text{Supp}(f_{k,b}(x)), \text{INV}_F(t_k, b, y) = x.$$

   (b) **Injective pair:** For all keys $k \in \mathcal{K}_\mathcal{X}$, there exists a perfect matching $\mathcal{R}_k \subseteq \mathcal{X} \times \mathcal{X}$ such that $f_{k,0}(x_0) = f_{k,1}(x_1)$ if and only if $(x_0, x_1) \in \mathcal{R}_k$.

3. **Efficient Range Superposition.** For all keys $k \in \mathcal{K}_\mathcal{X}$ and $b \in \{0, 1\}$ there exists a function $f_{k,b}^\prime : \mathcal{X} \to \mathcal{D}_\mathcal{Y}$ such that the following hold.

   (a) For all $(x_0, x_1) \in \mathcal{R}_k$ and $y \in \text{Supp}(f_{k,b}^\prime(x_b))$, $\text{INV}_F(t_k, b, y) = x_b$ and $\text{INV}_F(t_k, b \oplus 1, y) = x_{b \oplus 1}$.

   (b) There exists an efficient deterministic procedure $\text{CHK}_F$ that, on input $k, b \in \{0, 1\}, x \in \mathcal{X}$ and $y \in \mathcal{Y}$, returns 1 if $y \in \text{Supp}(f_{k,b}^\prime(x))$ and 0 otherwise. Note that $\text{CHK}_F$ is not provided the trapdoor $t_k$. 

15
(c) For every \( k \) and \( b \in \{0, 1\} \),
\[
\mathbb{E}_{x \in \mathcal{X}} \left[ H^2(f_{k,b}(x), f'_{k,b}(x)) \right] \leq \mu(\lambda).
\]
for some negligible function \( \mu \). Here \( H^2 \) is the Hellinger distance. Moreover, there exists an efficient procedure \( \text{SAMP}_F \) that on input \( k \) and \( b \in \{0, 1\} \) prepares the state
\[
\frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \sqrt{(f'_{k,b}(x))(y)} |x\rangle |y\rangle.
\] (8)

4. **Claw-Free Property.** For any PPT adversary \( \mathcal{A} \), there exists a negligible function \( \text{negl}(\cdot) \) such that the following holds:
\[
\Pr[(x_0, x_1) \in \mathcal{R}_k : (k, t_k) \leftarrow \text{GEN}_F(1^\lambda), (x_0, x_1) \leftarrow \mathcal{A}(k)] \leq \text{negl}(\lambda)
\]

The next two definitions are taken from [Mah18].

**Definition 4** (Trapdoor Injective Function Family). Let \( \lambda \) be a security parameter. Let \( \mathcal{X} \) and \( \mathcal{Y} \) be finite sets. Let \( \mathcal{K}_G \) be a finite set of keys. A family of functions
\[
\mathcal{G} = \{ g_{k,b} : \mathcal{X} \to \mathcal{D}_\mathcal{Y} \}_{b \in \{0, 1\}, k \in \mathcal{K}_G}
\]
is called a trapdoor injective family if the following conditions hold:

1. **Efficient Function Generation.** There exists an efficient probabilistic algorithm \( \text{GEN}_\mathcal{G} \) which generates a key \( k \in \mathcal{K}_\mathcal{G} \) together with a trapdoor \( t_k \):
\[
(k, t_k) \leftarrow \text{GEN}_\mathcal{G}(1^\lambda).
\]

2. **Disjoint Trapdoor Injective Pair.** For all keys \( k \in \mathcal{K}_\mathcal{G} \), for all \( b, b' \in \{0, 1\} \) and \( x, x' \in \mathcal{X} \), if \( (b, x) \neq (b', x') \), \( \text{Supp}(g_{k,b}(x)) \cap \text{Supp}(g_{k,b'}(x')) = \emptyset \). Moreover, there exists an efficient deterministic algorithm \( \text{INV}_F \) such that for all \( b \in \{0, 1\} \), \( x \in \mathcal{X} \) and \( y \in \text{Supp}(g_{k,b}(x)) \), \( \text{INV}_F(t_k, y) = (b, x) \).

3. **Efficient Range Superposition.** For all keys \( k \in \mathcal{K}_\mathcal{G} \) and \( b \in \{0, 1\} \)
   - (a) There exists an efficient deterministic procedure \( \text{CHK}_\mathcal{G} \) that, on input \( k, b \in \{0, 1\} \), \( x \in \mathcal{X} \) and \( y \in \mathcal{Y} \), outputs 1 if \( y \in \text{Supp}(g_{k,b}(x)) \) and 0 otherwise. Note that \( \text{CHK}_\mathcal{G} \) is not provided the trapdoor \( t_k \).
   - (b) There exists an efficient procedure \( \text{SAMP}_\mathcal{G} \) that on input \( k \) and \( b \in \{0, 1\} \) returns the state
\[
\frac{1}{|\mathcal{X}|} \sum_{x \in \mathcal{X}, y \in \mathcal{Y}} \sqrt{(g_{k,b}(x))(y)} |x\rangle |y\rangle.
\] (9)

**Definition 5** (Injective Invariance). A noisy trapdoor claw-free family \( \mathcal{F} \) is injective invariant if there exists a trapdoor injective family \( \mathcal{G} \) such that:

1. The algorithms \( \text{CHK}_\mathcal{F} \) and \( \text{SAMP}_\mathcal{F} \) are the same as the algorithms \( \text{CHK}_\mathcal{G} \) and \( \text{SAMP}_\mathcal{G} \).

2. For all quantum polynomial-time procedures \( \mathcal{A} \), there exists a negligible function \( \mu(\cdot) \) such that
\[
\Pr_{(k,t_k) \leftarrow \text{GEN}_\mathcal{F}(1^\lambda)}[\mathcal{A}(k) = 0] - \Pr_{(k,t_k) \leftarrow \text{GEN}_\mathcal{G}(1^\lambda)}[\mathcal{A}(k) = 0] \leq \mu(\lambda)
\] (10)

**Lemma 1** ([BCM+18], [Mah18]). Assuming the quantum hardness of LWE, there exists an injective invariant NTCF family.
4 Quantum Deniability

4.1 Definition

We assume without loss of generality that the (quantum) encryption algorithm \( \text{Enc} \) consists of an efficiently implementable isometry \( \widetilde{\text{Enc}} \) from register \( M \), consisting of \( m(\lambda) \) qubits, to register \( N \), consisting of \( n(\lambda) \) qubits, followed by a standard basis measurement of the output ciphertext register \( C \) (consisting of a subset of the qubits in \( N \)).

**Definition 6 (Quantum Deniability).** Let \( l : \mathbb{N} \rightarrow \mathbb{N} \) be polynomially bounded. A public-key encryption scheme \((\text{Gen}, \text{Enc}, \text{Dec})\) is \( l \)-deniable if the following holds. There exists a QPT algorithm \( \text{Fake} \), and a sub-register \( R \) of \( C \) consisting of \( l(\lambda) \) qubits, such that the following holds for any \( m_0, m_1 \):

\[
\left( \text{pk}, m_1, (\text{Meas}_R \otimes I_{C \setminus R} \otimes \text{Fake}_{N \setminus C}) \widetilde{\text{Enc}}(\text{pk}, m_0) \right) \approx_c \left( \text{pk}, m_1, (\text{Meas}_R \otimes I_{N \setminus R}) \widetilde{\text{Enc}}(\text{pk}, m_1) \right)
\]

where \( \text{Meas}_R \) is the quantum channel that corresponds to a standard basis measurement of register \( R \), and \( \text{pk} \) is sampled according to \( \text{Gen} \).

To keep the definition general, we included a faking algorithm \( \text{Fake} \) which is allowed to act on the leftover quantum state of the encryption algorithm. This more naturally parallels the classical definition of deniability. However, we will see that our scheme satisfies an even stronger notion of deniability, whereby the leftover state does not need any additional processing (before it gets handed to the attacker) i.e. \( \text{Fake} \) can be taken to be the identity.

4.2 Construction

We describe encryption of a single bit. This can, of course, be extended in parallel to encryptions of any number of bits.

Let \( X, Y, K \) be finite sets. Let \( F = \{f_{k,b} : X \rightarrow D_Y\}_{k \in K, b \in \{0,1\}} \) be a family of noisy trapdoor claw-free functions (which exists assuming the quantum hardness of LWE [BCM+18]). Let \( f'_{k,b} : X \rightarrow Y \) be functions satisfying the efficient range superposition property of Definition 3. For \( x \in X \), denote by \( \text{BitDecomp}(x) \) its bit decomposition.

**Construction 1.**

- \( \text{Gen}(1^\lambda) \rightarrow (\text{pk}, \text{sk}): \)

  - \( \text{Run} (k, t_k) \leftarrow \text{Gen}_F(1^\lambda) \). Output \( (\text{pk}, \text{sk}) = (k, t_k) \).

- \( \text{Enc}(m, \text{pk}) \rightarrow c: \)

  - On input \( m \in \{0,1\} \), and \( \text{pk} = k \), run \( \text{Samp}_F(k, \cdot) \) on a uniform superposition of \( b \)'s, to obtain the state

    \[
    \frac{1}{\sqrt{|X|}} \sum_{b \in \{0,1\}, x \in X} \sqrt{f'_{k,b}(x)(y)} |b \rangle |x \rangle |y \rangle ,
    \]

    where we assume that \( x \) and \( y \) are represented by their bit decomposition. We assume without loss of generality that \( \text{Samp}_F \) returns any auxiliary register to the \( |0 \rangle \) state. \(^2\)

  - Measure the image register, and let \( y \) be the outcome. As a result, the state has collapsed to:

    \[
    \frac{1}{\sqrt{2}} (|0 \rangle |x_0 \rangle + |1 \rangle |x_1 \rangle),
    \]

    where \( x_0, x_1 \in X \) are the unique elements such that \( y \) is in the support of \( f'_{k,b}(x_b) \).

---

\(^2\)Since the output of \( \text{Samp}_F \) on the output registers is a pure state, one can always have \( \text{Samp}_F \) coherently “uncompute” on all registers except those containing the output.
Let \( n \) be the length of \( \text{BitDecomp}(x_0) \). Apply a Hadamard gate to all of the remaining registers, and measure. Parse the measurement outcome as \( z \| d \) where \( z \in \{0, 1\} \) and \( d \in \{0, 1\}^n \).
Output \( c = (z, d, y) \).

- Dec\((c, t_k) \rightarrow m:\)
  - Let \( c = (z', d, y) \). For \( b \in \{0, 1\} \), run Inv\(_F\)(\( t_k, b, y \)) to obtain pre-images \( x_0 \) and \( x_1 \).
  - Output \( m = z' \oplus d \cdot (\text{BitDecomp}(x_0) \oplus \text{BitDecomp}(x_1)) \).

Fake\((m', \text{aux})\): Given as input \( m' \) (the desired plaintext to be claimed), and a quantum state \( \text{aux} \) (the leftover workspace after encryption), output \( (m', \text{aux}) \), i.e. output the leftover workspace unchanged.

**Theorem 3.** Assuming the quantum hardness of LWE, the scheme of construction 1 is CPA-secure and 1-deniable (as in Definition 6).

Correctness of the encryption scheme is straightforward to verify. Hence, we focus on security in the next section.

### 4.3 Security

From now on, for ease of notation, when referring to the bit-decomposition of \( x \), we simply write \( \text{BitDecomp}(x) \) when the context is clear. We denote by \( x_0^y \) and \( x_1^y \) the two pre-images of \( y \).

**CPA security** CPA security follows straightforwardly from the following “hardcore bit” property satisfied by \( F \), which says that any quantum polynomial-time adversary \( A \) has negligible advantage in the following game between a challenger and \( A \):

- The challenger samples \( k \), and runs \( \text{SAMP}_F(k, \cdot) \) on a uniform superposition of \( b \)'s to obtain the state
  \[
  \frac{1}{\sqrt{|X|}} \sum_{b \in \{0, 1\}, x \in X} \sqrt{f_{k,b}(x)(y)} |b\rangle |x\rangle |y\rangle .
  \]
- Measures the last register to get \( y \).
- Then, applies a Hadamard gate on the first two registers to get \( z, d \) such that \( z = d \cdot (x_0 \oplus x_1) \), where \( x_0 \) and \( x_1 \) are the pre-images of \( y \). Sends \( y, d \) to \( A \).
- \( A \) returns \( z' \).

\( A \) wins if \( z = z' \).

To see that this “hardcore bit” property holds, suppose for a contradiction that there was a quantum polynomial time \( A \) breaking this property. Then the following algorithm \( A' \) recovers a claw. We use the notation \( A(y, d) \) to denote the output of \( A \) on input \( y, d \).

- On input \( k \), run \( \text{SAMP}_F(k, \cdot) \) on a uniform superposition of \( b \)'s to obtain the state
  \[
  \frac{1}{\sqrt{|X|}} \sum_{b \in \{0, 1\}, x \in X} \sqrt{f'_{k,b}(x)(y)} |b\rangle |x\rangle |y\rangle .
  \]
- Measure the last two registers to obtain \( x, y \) where \( y \in \text{Supp}(f'_{k,b}(x)) \) for some \( b \in \{0, 1\} \).
- Run the “quantum Goldreich-Levin” extraction algorithm using \( A(y, \cdot) \) as an oracle [AC02]. Recall that the “quantum Goldreich-Levin” extraction algorithm makes a single query to \( A(y, \cdot) \). Let \( s \) be the output of this extraction algorithm.
- Output \( (x, x \oplus s) \) as the claw.
Since \( \mathcal{A}(y, d) \) guesses \( d \cdot (x_0 \oplus x_1) \) with non-negligible advantage, the “quantum Goldreich-Levin” extraction algorithm outputs \( s = x_0 \oplus x_1 \) with non-negligible probability, which results in \( \mathcal{A}' \) outputting a claw with non-negligible probability.

Note that the reduction works even if \( \mathcal{A} \) is non-uniform with quantum advice, since \( \mathcal{A}' \) makes a single query to \( \mathcal{A} \), so there is no issue with rewinding.

With this hardcore bit property in hand, CPA security is immediate: notice that the distribution \((pk, Enc(pk, 0))\) is simply \((k, z, d, y)\), such that \( z = d \cdot (x_0^y \oplus x_1^y) \), where \( k, d, z, y \) are sampled exactly from the distribution of the hardcore bit property. On the other hand, the distribution \((pk, Enc(pk, 1))\) is \((k, z, d, y)\), such that \( z \neq d \cdot (x_0^y \oplus x_1^y) \), where \( k, z, d, y \) have the same distribution as before, except that \( z \) is flipped. Clearly, distinguishing the two distributions is precisely equivalent to guessing the value of \( d \cdot (x_0^y \oplus x_1^y) \), which is the hardcore bit game above.

**Deniability** We will now show that Construction 1 satisfies deniability, according to Definition 6. In particular, we will show that, for Construction 1, one can take the algorithm \( \text{Fake} \) to be the identity.

Using the notation of Definition 6, the output of the encryption algorithm \( \text{Enc} \), before measurement of the output, is a state on register \( \mathcal{N} \), a sub-register of which, \( \mathcal{C} \), is eventually measured to produce the classical ciphertext. Notice that for \( \text{Enc} \) as defined in Construction 1, all qubits in \( \mathcal{N} \setminus \mathcal{C} \) are eventually returned to the state \(|0\rangle \). Thus, in order to prove 1-deniability, all that is left to show is that there is a subset \( \mathcal{R} \), of size 1, of the qubits of \( \mathcal{C} \), such that for any \( m_0, m_1 \in \{0, 1\} \),

\[
(pk, m_1, \text{Meas}_R \otimes I_{\mathcal{C}\setminus\mathcal{R}}([\tilde{\text{Enc}}(m_0)]_{\mathcal{C}})) \approx_c (pk, m_1, \text{Meas}_R \otimes I_{\mathcal{C}\setminus\mathcal{R}}([\tilde{\text{Enc}}(m_1)]_{\mathcal{C}})),
\]

(11)

where recall that \( \tilde{\text{Enc}} \) denotes \( \text{Enc} \) excluding the final measurement, and we are denoting by \([\tilde{\text{Enc}}(m)]_{\mathcal{C}}\) the restriction of the state \( \tilde{\text{Enc}}(m) \) to the register \( \mathcal{C} \).

We take \( \mathcal{R} \) to be the qubit corresponding to the first bit of the ciphertext. Then, up to replacing \( f_{k,b} \) with \( f'_{k,b} \) (which affects the distribution at most negligibly - see Lemma 2.0.1 in [BKVV20] relating Hellinger distance to trace distance), conditioned on the outcome of the measurement being \( z \), the LHS and the RHS of (11) are respectively,

\[
\left( k, m_1, z, \sum_{x, d, y: d \cdot (x_0^y \oplus x_1^y) = z \oplus m_0} \sqrt{(f_{k,0}(x))(y)} |d| |y \right)
\]

and

\[
\left( k, m_1, z, \sum_{x, d, y: d \cdot (x_0^y \oplus x_1^y) = z \oplus m_1} \sqrt{(f_{k,0}(x))(y)} |d| |y \right)
\]

where \( x_0^y \) and \( x_1^y \) denote the pre-images of \( y \) (and we omit writing \( \text{BitDecomp} \) when it is clear from the context). Then, it is easy to see that the two distributions are computationally indistinguishable if and only if:

\[
\left( k, \sum_{x, d, y: d \cdot (x_0^0 \oplus x_1^0) = 0} \sqrt{(f_{k,0}(x))(y)} |d| |y \right) \approx_c \left( k, \sum_{d, y: d \cdot (x_0^1 \oplus x_1^1) = 1} \sqrt{(f_{k,0}(x))(y)} |d| |y \right)
\]

(12)

where we omit normalization constants.

To argue that these two states are indistinguishable, we appeal to the injective invariance property. Recall that by the latter, there exists a trapdoor injective family

\[
\mathcal{G} = \{ g_{k,b} : \mathcal{X} \to \mathcal{D}_y \}_{k \in K_\mathcal{G}, b \in \{0, 1\}}
\]

(as in Definition 4) such that

(i) \( \text{CHK}_\mathcal{X} = \text{CHK}_\mathcal{G} \) and \( \text{SAMP}_\mathcal{X} = \text{SAMP}_\mathcal{G} \).

(ii) It is computationally hard to distinguish whether \( k \leftarrow \text{GEN}_\mathcal{X} \) or \( k \leftarrow \text{GEN}_\mathcal{G} \).
Now, suppose for a contradiction that there was a distinguisher $D$ for the two (families of) states in (12). Then, there exists a distinguisher $D'$ that breaks property (ii) of injective invariance:

- $D'$ receives $k$ as input.
- $D'$ runs $\overline{\text{Enc}(pk,0)}$ with $pk = k$, and then measures the first qubit. Let $z$ be the outcome. (note that this step is well-defined since $\text{SAMP}_F = \text{SAMP}_G$).
- $D'$ gives $k$ and the leftover state to $D$, which returns a guess $z'$. If $z' = z$ (i.e. the distinguisher for the two states succeeded), $D'$ guesses that $k$ was sampled from $\text{GEN}_F$. Otherwise, it guesses that it was sampled from $\text{GEN}_G$.

The reason why $D'$ has non-negligible advantage is the following. In the case that $k \leftarrow \text{GEN}_F$, the leftover states conditioned on $z = 0$ and $z = 1$ are (up to the negligible error incurred by replacing $f'_{k,b}$ with $f_{k,b}$) those on the LHS and the RHS of (12) respectively. On the other hand, when $k \leftarrow \text{GEN}_G$, it is easy to see that the state that results from applying $\overline{\text{Enc}(pk,0)}$ and measuring the first qubit is independent of the outcome $z$. In particular, the leftover state is simply

$$\sum_{x,d,y \in \{0,1\}} \sum \sqrt{(g_{k,b}(x))(y) | d \rangle | y \rangle}.$$ 

So the probability that $D$ guesses $z$ correctly in this case is exactly $\frac{1}{2}$.

5 Unexplainable Encryption

In this section, we formally introduce the notions of unexplainable encryption, and perfectly unexplainable encryption. In 5.2, we discuss the notion of coercion before-the-fact in more detail, and its relationship to perfect unexplainability. In 5.3, we discuss in detail the relationship between unexplainable encryption and the standard definition of deniable encryption.

5.1 Definition

In the following definition, $\text{Explain}$ takes as input a public key $pk$, a message $m$, and outputs a ciphertext $c$ and a witness $w$; $\text{Verify}$ takes as input a public key $pk$, a ciphertext $c$, a message $m$, a witness $w$, and outputs a bit; $\text{FakeExplain}$ takes as input a public key $pk$, a pair of messages $m,m'$, and outputs a ciphertext $c$, and two witnesses $w,w'$. In the rest of the section, we use the notation $\text{Pr}_{pk}$ to mean that the probability is over sampling a $pk$ from the the generation algorithm $\text{Gen}(1^\lambda)$ (where the security parameter $\lambda$ is omitted from the notation).

**Definition 7** (Unexplainable Encryption). We say that a public-key encryption scheme $(\text{Gen}, \text{Enc}, \text{Dec})$ is unexplainable if the following holds. Let $\text{Verify}$ and $\text{Explain}$ be any (non-uniform) QPT algorithms such that, for all $pk,c,m,w$, $\text{Verify}(pk,c,m,w) = 0$, except with exponentially small probability, if $c$ is not in the support of the distribution of $\text{Enc}(pk,m)$.

Suppose $\text{Verify}, \text{Explain}$ satisfy the following completeness condition: there exists a non-negligible function $\gamma$, such that, for any $m$, for all $\lambda$, 

$$\text{Pr}_{pk}[\text{Verify}(pk,c,m,w) = 1 : (c,w) \leftarrow \text{Explain}(pk,m)] \geq \gamma(\lambda).$$

Then, there exist a polynomial-time algorithm $\text{FakeExplain}$ and a non-negligible function $\text{negl}$, such that, for any distinct messages $m,m'$, for all $\lambda$:

$$\text{Pr}_{pk}[\text{Verify}(pk,c,m,w) = 1 : c,w \leftarrow \text{Explain}(pk,m)] - \text{Pr}_{pk}[\text{Verify}(pk,c,m',w') = 1 : c,w \leftarrow \text{Explain}(pk,m), w' \leftarrow \text{FakeExplain}(pk,m,m',c,w)] = \text{negl}(\lambda)$$ (13)
We believe that this definition naturally captures the ideal of privacy desired in a deniable setting. Moreover, since this definition does not explicitly refer to input randomness, it straightforwardly generalizes to the setting of a quantum encryption algorithm (with classical ciphertexts): the only modifications are that \( Enc \) is a quantum polynomial-time algorithm, and that \( w \) and \( w' \) are (possibly entangled) quantum states.

We remark that in Definition 7 we restricted to considering \( \text{Verify} \) such that \( \text{Verify}(pk, c, m, w) = 0 \), with probability 1, whenever \( c \) is not in the support of the distribution of \( Enc(pk, m) \). One could relax this requirement slightly (and thus obtain a stronger definition) by asking that \( \Pr[\text{Verify}(pk, c, m, w) = 0] \) be exponentially small (in the security parameter) whenever \( c \) is not in the support of the distribution of \( Enc(pk, m) \). We will show that our scheme still satisfies this definition.

The following is a special case of unexplainable encryption. In words, an encryption scheme is perfectly unexplainable if there does not exist a pair of efficient algorithms \( \text{Verify} \) and \( \text{Explain} \) (where \( \text{Verify}(pk, c, m, w) = 0 \) if \( c \) is not in the range of \( Enc(pk, m; -) \)), for which the completeness condition holds.

**Definition 8** (Perfectly Unexplainable Encryption). A public key encryption scheme \( (\text{Gen}, \text{Enc}, \text{Dec}) \) is said to be perfectly unexplainable if the following holds. Let \( \text{Explain} \) and \( \text{Verify} \) be any (non-uniform) QPT algorithms such that, for all \( pk, c, m, w \), \( \text{Verify}(pk, c, m, w) = 0 \), except with exponentially small probability, if \( c \) is not in the support of the distribution of \( Enc(pk, m) \). Then, for any \( m \), there exists a negligible function \( \text{negl} \), such that for any \( \lambda \),

\[
\Pr_{pk}[\text{Verify}(pk, c, m, w) = 1 : (c, w) \leftarrow \text{Explain}(pk, m)] = \text{negl}(\lambda). \tag{14}
\]

At first glance, perfect unexplainability seems unattainable. In fact, it is clear that a classical encryption scheme cannot be perfectly unexplainable: one can always take \( \text{Explain} \) to be the algorithm that encrypts honestly and outputs the ciphertext together with the randomness used (i.e. the witness is the randomness), and take \( \text{Verify} \) to be the algorithm that runs encryption forward and checks consistency. In contrast, the quantum encryption scheme that we will describe in Section 6 will be perfectly unexplainable.

We emphasize that perfect unexplainability is the strongest sense in which a scheme can be unexplainable: condition (13) in Definition 7 can be replaced by something as demanding as we desire, but this doesn’t matter because, for a perfectly unexplainable encryption scheme, completeness is never satisfied, and thus any replacement of condition (13) would be vacuously satisfied. We point out that one could even relax the quantifier over \( m \) in the completeness condition to “\( \exists m \)”. Our scheme from Section 6 would still satisfy this stronger definition.

### 5.2 Coercion before-the-fact

While protecting against coercion before-the-fact is very desirable in practice, to the best of our knowledge, a corresponding notion has not been previously formalized (most likely due to the fact that protecting against coercion before-the-fact is impossible to achieve classically in a cryptographic sense). Here, we propose a formal definition, and we observe that this is essentially equivalent to the notion of perfect unexplainability from Definition 8.

In a coercion before-the-fact scenario, an attacker, who has in mind a message \( m \), wishes to prescribe to the sender how she should encrypt later in a way that:

- The resulting ciphertext decrypts to \( m \) with overwhelming probability.
- There is an efficient procedure for the attacker to verify that the sender’s ciphertext (which the attacker obtains by intercepting) will decrypt to \( m \) with overwhelming probability.

First, notice that there is no hope of protecting against coercion before-the-fact in a model where the attacker can approach the sender before she sends her ciphertext, and knows all of the information that will be available to the sender at the time of encryption (e.g. the public key). In fact, in such a model, the attacker can simply generate a genuine encryption \( c \) of the desired message \( m \), and prescribe that the sender’s ciphertext later be exactly \( c \). So, instead we consider the scenario where the public key is not known to the attacker at the time when he approaches the sender. Such a model captures an online election where citizens encrypt their votes before sending them to the government using a public key encryption scheme, and the public key is announced publicly only on election day. The attacker is allowed to approach the
sender any time before election day (i.e. any time before the public key is announced). More generally, one can consider a model where some additional information (not necessarily the public key) is revealed to the sender just before she encrypts. For simplicity, we restrict ourselves to formalizing the setting where the public key is the only information that is not known to the attacker at the time of coercion.

We first formally define the notion of a coercion before-the-fact attack. An encryption scheme then protects against coercion before-the-fact if no such attack exists.

**Definition 9** (Coercion before-the-fact attack). Let \((\text{Gen, Enc, Dec})\) be a public-key encryption scheme with classical ciphertexts (where \(\text{Dec}\) is a classical deterministic algorithm). A coercion before-the-fact attack is a pair \((\text{Enc'} \text{, Verify})\) where:

- \(\text{Enc'}(pk, m) \rightarrow c\) is a non-uniform QPT algorithm
- \(\text{Verify}(c, m) \rightarrow \text{accept/reject}\) is a non-uniform QPT algorithm

They satisfy:

- (Completeness of verification) For any \(m, \lambda\),
  \[
  \Pr[\text{Verify}(\text{Enc'}(pk, m), m) = \text{accept} : (sk, pk) \leftarrow \text{Gen}(1^\lambda)] = 1 .
  \]
- (Soundness of verification) There exists \(C > 0\), such that the following holds for all \(c, m, \lambda\):
  \[
  \Pr[\text{Dec}(sk, c) \neq m \land \text{Verify}(pk, c, m) = \text{accept} : (sk, pk) \leftarrow \text{Gen}(1^\lambda)] \leq 2^{-\lambda C} .^3
  \]

**Definition 10.** We say that an encryption scheme protects against coercion before-the-fact if no coercion before-the-fact attacks exists.

It is straightforward to see that a perfectly unexplainable encryption scheme protects against coercion before-the-fact. This is because a coercion before-the-fact attack provides a way to encrypt in a way that can be later verified.

**Theorem 4.** A perfectly unexplainable encryption scheme protects against coercion before-the-fact.

**Proof.** Let \((\text{Gen, Enc, Dec})\) be a perfectly unexplainable encryption scheme (satisfying the non-uniform version of Definition 8). Suppose for a contradiction that a coercion before-the-fact attack \((\text{Enc'}, \text{Verify})\) existed.

Define \(\text{Verify}'\) to be the non-uniform algorithm that on input \(pk, c, m, w, w'\), runs \(\text{Verify}(pk, c, m)\) and ignores \(w\). Let \(\text{Explain}\) be the non-uniform algorithm that, on input \(pk, m\), runs \(c \leftarrow \text{Enc'}(pk, m)\), and outputs \(c, w^*,\) for some fixed \(w^*\). By the completeness and soundness of the coercion before-the-fact attack, it follows that the pair \((\text{Explain, Verify}')\) contradicts perfect unexplainability.

Definition 9 only considers attacks where \(\text{Enc'}\) and \(\text{Verify}\) are non-uniform with classical advice. More generally, one could also consider attacks where \(\text{Enc'}\) and \(\text{Verify}\) have quantum advice. In particular, this advice could be in the form of an entangled state over two registers corresponding to the advice of \(\text{Enc'}\) and \(\text{Verify}\) respectively. This captures the scenario where an attacker coerces the sender before-the-fact by giving to the sender half of some entangled state, and prescribing what the encryption operation should be, i.e. \(\text{Enc'}\). The verification of the sender’s intercepted ciphertext then makes use of the other half of the entangled state.

It is not difficult to see that a slightly more general version of the definition of perfect unexplainability, where \(\text{Explain}\) and \(\text{Verify}\) are allowed to be non-uniform with entangled quantum advice, implies this more general notion of protection against coercion before-the-fact. Such a notion of perfect unexplainability holds assuming the quantum hardness of \(\text{LWE}\) against QPT algorithm with polynomial quantum advice. Our security proof in Section 6 goes through unchanged (since our extraction algorithm does not involve any rewinding).

---

^3The only reason why the upper bound in the soundness condition is exponentially small rather than being zero is that, for a scheme with imperfect decryption, the classical attack that prescribes the input randomness and checks consistency of the intercepted ciphertext with the prescribed randomness has completeness 1 but exponentially small soundness, rather than 0.
5.3 The relationship between deniable and unexplainable encryption

While the definition of unexplainable encryption that we introduced in Section 5.1 seems related to deniability, its exact relationship to it is not entirely clear. The reason why the two definitions are difficult to formally compare is that the classical definition of deniability is, in some sense, centered around the input randomness, while the latter does not appear at all in unexplainability. In this section, we shed light on the relationship between unexplainability and deniability, by introducing a variation of unexplainability in the classical setting, and showing that it is equivalent to deniability. Essentially, we show that deniability can be equivalently phrased from the point of view of unexplainability.

The syntax of the algorithms is slightly different from that of Definition 7: Explain takes as input a public key \( pk \), a message \( m \), randomness \( r \), and outputs some witness \( w \); Verify takes as input a public key \( pk \), a ciphertext \( c \), a message \( m \), a witness \( w \), and outputs a bit; FakeExplain takes as input a public key \( pk \), a pair of messages \( m, m' \), randomness \( r \), and outputs a witness \( w \). We emphasize that the following definition is entirely classical.

**Definition 11** (Unexplainable Encryption (deniability-style)). A public key encryption scheme \((\text{Gen, Enc, Dec})\) is said to be classically unexplainable if the following holds. For any PPT algorithm \( \text{Explain} \), there exists a PPT algorithm \( \text{FakeExplain} \) such that the following holds for any PPT algorithm \( \text{Verify} \) such that, for all \( pk, c, m, w \), \( \text{Verify}(pk, c, m, w) = 0 \) (with probability 1) if \( c \) is not in the range of \( \text{Enc}(pk, m; \cdot) \).

Suppose \( \text{Verify, Explain} \) satisfy the following completeness condition: there exists a non-negligible function \( \gamma \), such that, for any \( m \), for any \( \lambda \),

\[
\Pr_{pk,r}[\text{Verify}(pk, \text{Enc}(pk, m; r), m, w) = 1 : w \leftarrow \text{Explain}(pk, m, r)] = \gamma(\lambda).
\]

Then, for any distinct messages \( m, m' \), for any \( \lambda \), we have:

\[
\begin{align*}
\Pr_{pk,r}[\text{Verify}(pk, \text{Enc}(pk, m; r), m, w) & = 1 : w \leftarrow \text{Explain}(pk, m, r)] \\
& - \Pr_{pk,r}[\text{Verify}(pk, \text{Enc}(pk, m'; r), m, w) = 1 : w \leftarrow \text{FakeExplain}(pk, m, m', r)]
\end{align*}
\]

\( = \text{neg}(\lambda) \quad (15) \)

**Lemma 2.** A classical encryption scheme is sender-deniable (as in Definition 2) if and only if it is classically unexplainable (according to Definition 11).

In light of this equivalence, one can view Definition 7 from the previous section as a natural generalization of Definition 11, and hence of deniability, to the quantum setting, where no input randomness is a priori present.

We remark that the order of quantification over algorithms is different in Definitions 7 and 11: in the former, we required that \( \forall \text{Explain}, \text{Verify} \) satisfying completeness, \( \exists \text{FakeExplain} \) such that \((15)\) is satisfied.; in the latter, we require that \( \forall \text{Explain}, \exists \text{FakeExplain} \) such that \( \forall \text{Verify} \) (for which \( \text{Verify, Explain} \) satisfy completeness), \( (15) \) is satisfied. Although we find the former order of quantification to be more natural, the latter is what is necessary for the equivalence with the classical definition of deniability. We point out that the choice of order of quantification does not affect whether our scheme from Section 6 satisfies the definition. This is because the order of quantification does not affect perfect unexplainability.

**Proof of Lemma 2.** \( \Rightarrow \): Suppose \( (\text{Gen, Enc, Dec}) \) is sender-deniable. Then there exists an algorithm \( \text{Fake} \) as in the definition of deniability.

Let \( \text{Explain} \) be any QPT algorithm (with syntax as in the definition of unexplainability). We define the following algorithm \( \text{FakeExplain} \):

- On input \((pk, m, m', r)\), run \( r' \leftarrow \text{Fake}(pk, m, m', r) \).
- Run \( w \leftarrow \text{Explain}(pk, m', r') \). Output \( w \).

Since the encryption scheme is sender-deniable, by definition, the following two distributions are computationally indistinguishable for any \( m, m' \), where \( pk \) is sampled from \( \text{Gen} \), \( r \) is uniformly random, and \( r' \leftarrow \text{Fake}(pk, m, m', r) \):

\[
(pk, \text{Enc}(pk, m; r), m, r) \approx_r (pk, \text{Enc}(pk, m'; r), m, r')
\]

Consider then the following distinguishing algorithm \( D \) for the two distributions:

\[
\begin{align*}
\Pr_{pk,r}[D(pk, \text{Enc}(pk, m; r), m, r) = 1 : r \leftarrow \text{Verify}(pk, \text{Enc}(pk, m; r), m, r)] \\
& - \Pr_{pk,r}[D(pk, \text{Enc}(pk, m'; r), m, r) = 1 : r \leftarrow \text{Verify}(pk, \text{Enc}(pk, m'; r), m, r)]
\end{align*}
\]

\( = \text{neg}(\lambda) \quad (16) \)
• On input $(pk, c, m, r)$, run $w \leftarrow \text{Explain}(pk, m, r)$.
• Then, run $\text{Verify}(pk, c, m, w)$, and output the outcome.

By (16),
\[ \left| \Pr_{pk} [D(pk, \text{Enc}(pk, m; r), m, r) = 1] - \Pr_{pk} [D(pk, \text{Enc}(pk, m'; r), m, r') = 1] \right| = \negl(\lambda). \]

Then, notice that the first term on the LHS is precisely equal to
\[ \Pr_{pk, r} [\text{Verify}(pk, \text{Enc}(pk, m; r), m, w) = 1 : w \leftarrow \text{Explain}(pk, m, r)], \]
while the second term, using the definition of $\text{FakeExplain}$, is equal to
\[ \Pr_{pk, r} [\text{Verify}(pk, \text{Enc}(pk, m'; r), m, w) = 1 : w \leftarrow \text{FakeExplain}(pk, m, m', r)]. \]

Since this holds for any $m, m'$, this establishes precisely that the scheme is unexplainable.

“$\Rightarrow$”: Let $(\text{Gen}, \text{Enc}, \text{Dec})$ be a strongly unexplainable encryption scheme. Now, consider the following algorithm $\text{Explain}$:
• On input $(pk, m, r)$ output $(c, w)$ where $c = \text{Enc}(pk, m; r)$ and $w = r$.

By definition of strong unexplainability, for any QPT $\text{Explain}$, so in particular for $\text{Explain}$ as defined above, there exists a QPT algorithm $\text{FakeExplain}$ such that for any QPT $\text{Verify}$ (such that the pair $\text{Verify}, \text{Explain}$ satisfies completeness), for any distinct messages $m, m'$, we have
\[ \left| \Pr_{pk, r} [\text{Verify}(pk, \text{Enc}(pk, m; r), m, w) = 1 : w \leftarrow \text{Explain}(pk, m, r)] \\
- \Pr_{pk, r} [\text{Verify}(pk, \text{Enc}(pk, m'; r), m, w) = 1 : w \leftarrow \text{FakeExplain}(pk, m, m', r)] \right| = \negl(\lambda) \quad (17) \]

Now, suppose for a contradiction that the encryption scheme is not deniable. Recall, that our particular choice of algorithm $\text{Explain}$ outputs randomness. Then, since $\text{FakeExplain}$ takes inputs of the form $(pk, m, m', r)$ and also outputs some randomness $r'$, the latter has the correct syntax for a faking algorithm in a deniable encryption scheme. Since our scheme is not deniable by hypothesis, then there must exist a pair of distinct messages $m_+, m'_+$, and an efficient distinguisher $D$ for the two distributions $(pk, \text{Enc}(pk, m_+; r), m_+, r) = 1$ and $(pk, \text{Enc}(pk, m'_+; r), m_+, r')$ where $r' \leftarrow \text{Fake}(pk, m_+, m'_+, r)$. Without loss of generality, assume that, for all $\lambda$,
\[ \Pr_{pk, r} [D(pk, \text{Enc}(pk, m_+; r), m_+, r) = 1] \geq \Pr_{pk, r} [D(pk, \text{Enc}(pk, m'_+; r), m_+, r') = 1] + \delta(\lambda). \quad (18) \]

where $\delta$ is a non-negligible function.

Now, consider the algorithm $\text{Verify}$, defined as follows:
• On input $pk, c, m, w$, check that $\text{Enc}(pk, m; w) = c$. If this is not the case, output 0. Otherwise, proceed to the next step.
• If $m \neq m_+$, output 1, otherwise, run $D(pk, c, m, w)$, and output the outcome.

First, we argue that $\text{Verify}$ and $\text{Explain}$ as defined above satisfy the completeness condition.

Notice that the probability on the LHS of (18) is precisely
\[ \Pr_{pk, r} [\text{Verify}(pk, \text{Enc}(pk, m; r), m, w) = 1 : w \leftarrow \text{Explain}(pk, m, r)], \]
and the probability on the RHS of (18) is precisely
\[ \Pr_{pk, r} [\text{Verify}(pk, \text{Enc}(pk, m'; r), m, w) = 1 : w \leftarrow \text{FakeExplain}(pk, m, m', r)]. \]

Hence, (18) contradicts (17).
6 A Perfectly Unexplainable Encryption Scheme

In this section, we describe a public-key encryption scheme, and prove that it is both CPA-secure and perfectly unexplainable (as in Definition 8).

6.1 Construction

Let \( \mathcal{X}, \mathcal{Y}, \mathcal{K} \) be finite sets. Let \( \mathcal{F} = \{ f_{k,b} : \mathcal{X} \to \mathcal{Y} \}_{k \in \mathcal{K}, b \in \{0,1\}} \) be a family of noisy trapdoor claw-free functions (which exists assuming the quantum hardness of LWE [BCM+18]). Let \( f'_{k,b} : \mathcal{X} \to \mathcal{Y} \) be functions satisfying the efficient range superposition property of Definition 3.

The scheme that we describe in this section is a parallel repeated version of the scheme described in the technical overview (Section 2.3.1). Here, by parallel repetition we mean that the same plaintext \( m \) is encrypted \( L \) times (with the same public key), where \( L \) is polynomial in the security parameter. Without parallel repetition, we are only able to show that the LHS of Equation (14) is upper bounded by \( \frac{1}{2} + \text{negl} \). With parallel repetition, we will be able to improve this to \( \text{negl} \).

Construction 2. Parameters: \( L = \text{poly}(\lambda) \).

- \( \text{Gen}(1^\lambda) \to (pk, sk) \):
  - \( \text{Run} (k, t_k) \leftarrow \text{GEN}_F(1^\lambda) \). Output \( (pk, sk) = (k, t_k) \).
- \( \text{Enc}(m, pk) \to c \):
  - For \( i \in [L] \), do the following:
    * On input \( m \in \{0,1\} \), and \( pk = k \), run \( \text{SAMP}_F(k, \cdot) \) on a uniform superposition of \( b \)'s, to obtain the state
      \[
      \frac{1}{\sqrt{|\mathcal{X}|}} \sum_{b \in \{0,1\}, x \in \mathcal{X}} \sqrt{f'_{k,b}(x)} (y) |b| |x| |y|,
      \]
      where we assume that \( x \) and \( y \) are represented by their bit decomposition. We assume without loss of generality that \( \text{SAMP}_F \) that any auxiliary register is returned to the \( |0 \rangle \) state. \(^4\)
    * Measure the image register, and let \( y_i \in \mathcal{Y} \) be the outcome. As a result, the state has collapsed to:
      \[
      \frac{1}{\sqrt{2}} (|0 \rangle |x_0 \rangle + |1 \rangle |x_1 \rangle),
      \]
      where \( x_0, x_1 \in \mathcal{X} \) are the unique elements such that \( y_i \) is in the support of \( f'_{k,b}(x_b) \).
    * Query the phase oracle for \( H \), to obtain:
      \[
      \frac{1}{\sqrt{2}} ((-1)^{H(\text{BitDecomp}(x_0))} |0 \rangle |x_0 \rangle + (-1)^{H(\text{BitDecomp}(x_1))} |1 \rangle |x_1 \rangle).\]
      * Let \( n \) be the length of \( \text{BitDecomp}(x_0) \). Apply a Hadamard gate to all of the remaining registers, and measure. Parse the measurement outcome as \( z_i || d_i \) where \( z_i \in \{0,1\} \) and \( d_i \in \{0,1\}^n \).
      Let \( z'_i := z_i \oplus m \).
      - Let \( z' := z'_1 \ldots z'_L \), \( d := d_1 \ldots d_L \), \( y = y_1 \ldots y_L \). Output \( c = (z', d, y) \).
- \( \text{Dec}(c, t_k) \to m \):
  - Let \( c = (z', d, y) \). Parse \( z' \) as \( z' = z'_1 \ldots z'_L \). Similarly for \( d \) and \( y \).
  - For \( i \in [L] \):
    * For \( b \in \{0,1\} \), run \( \text{INV}_F(t_k, b, y_i) \) to obtain pre-images \( x'_0 \) and \( x'_1 \).
    * Let \( m_i = z'_i \oplus d_i (\text{BitDecomp}(x'_0) \oplus \text{BitDecomp}(x'_1)) \oplus H(\text{BitDecomp}(x'_0)) \oplus H(\text{BitDecomp}(x'_1)) \).

\(^4\)Since the output of \( \text{SAMP}_F \) on the output registers is a pure state, one can always have \( \text{SAMP}_F \) coherently “uncompute” on all registers except does containing the output.
– If \( m_1 = \ldots = m_L \), output \( m_1 \), otherwise output \( \perp \).

**Theorem 5.** The scheme of Construction 2 is a CPA-secure perfectly unexplainable encryption scheme (as in Definition 8) in the quantum random oracle model (QROM), assuming the quantum hardness of LWE.

The rest of Section 6 is dedicated to proving Theorem 5. For simplicity, we focus on the uniform version of Definition 8. However, our construction also similarly satisfies the non-uniform version, assuming the hardness of LWE against non-uniform quantum adversaries.

CPA security is proven analogously to the proof for Construction 1. Thus, we focus on perfect unexplainability.

In Section 6.2, we introduce Zhandry’s compressed oracle technique. In Section 6.3, we prove technical results about the structure of strategies guessing the xor of two query outputs. In Section 6.4, we specialize these results to the concrete setting of guessing “equations”. In 6.5, we show how this structure can be used to extract a claw.

### 6.2 Compressed oracles

In this subsection, we formally introduce Zhandry’s technique for recording queries [Zha19]. This section is loosely based on the explanation in [Zha19]. For a slightly more informal treatment, which carries most of the essence, we suggest starting from Section 2.3.1 in the technical overview.

**Standard and Phase Oracles** The quantum random oracle, which is the quantum analogue of the classical random oracle, is typically presented in one of two variations: as a *standard* or as a *phase* oracle.

The standard oracle is a unitary acting on three registers: an \( n \)-qubit register representing the input to the function, an \( m \)-qubit register for writing the response, and a \( m2^n \)-qubit register representing the truth table of the queried function \( H : \{0,1\}^n \rightarrow \{0,1\}^m \). The algorithm that queries the standard oracle has access to the first two registers, while the third register, the oracle’s state, is hidden from the algorithm except by making queries. The standard oracle unitary acts in the following way on standard basis states:

\[
|x\rangle|y\rangle|H\rangle \mapsto |x\rangle|y \oplus H(x)\rangle|H\rangle.
\]

For a uniformly random oracle, the oracle register is initialized in the uniform superposition \( \frac{1}{\sqrt{m2^n}} \sum_{H} |H\rangle \). This initialization is of course equivalent to having the oracle register be in a completely mixed state (i.e. a uniformly chosen \( H \)). This equivalence can be seen by just tracing out the oracle register. We denote the standard (uniformly random) oracle unitary by StO. Moreover, for an oracle algorithm \( A \), we will denote by \( A^{\text{StO}} \) the algorithm \( A \) interacting with the standard oracle, implemented as above.

The phase oracle formally gives a different interface to the algorithm making the queries, but is equivalent to the standard oracle up to Hadamard gates. It again acts on three registers: an \( n \)-qubit register for the input, an \( m \)-qubit “phase” register, and a \( m2^n \)-qubit oracle register. It acts in the following way on standard basis states:

\[
|x\rangle|s\rangle|H\rangle \mapsto (-1)^{s \cdot H(x)} |x\rangle|s\rangle|H\rangle.
\]

For a uniformly random oracle, the oracle register is again initialized in the uniform superposition. One can easily see that the standard and phase oracles are equivalent up to applying a Hadamard gate on the phase register before and after a query. We denote the phase oracle unitary by PhO. Moreover, for an oracle algorithm \( A \), we will denote by \( A^{\text{PhO}} \) the algorithm \( A \) interacting with the phase oracle.

**Compressed oracle** The compressed oracle technique, introduced by Zhandry [Zha19], is an equivalent way of implementing a quantum random oracle which (i) is efficiently implementable, and (ii) keeps track of the queried inputs in a meaningful way. This paragraph is loosely based on the explanation in [Zha19].

In a compressed oracle, the oracle register does not represent the full truth table of the queried function. Instead, it represents a *database* of queried inputs, and the values at those inputs. More precisely, if we have an upper bound \( t \) on the number of queries, a database \( D \) is represented as an element of the set \( S^t \) where \( S = (\{0,1\}^n \cup \{\perp\}) \times \{0,1\}^m \). Each value in \( S \) is a pair \((x,y)\): if \( x \neq \perp \), then the pair means that the value of the function at \( x \) is \( y \), which we denote by \( D(x) = y \); and if \( x = \perp \), then the pair is not currently
used, which we denote by $D(x) = \perp$. Concretely, let $l \leq t$. Then, for $x_1 < x_2 < \ldots < x_t$ and $y_1, \ldots, y_l$, the database representing $D(x_i) = y_i$ for $i \in [l]$, with the other $t-l$ points unspecified, is represented as

$$\left( (x_1, y_1), (x_2, y_2), \ldots, (x_t, y_t), (\perp, 0^m), \ldots, (\perp, 0^m) \right)$$

where the number of $(\perp, 0^m)$ pairs is $t-l$. We emphasize that in this database representation, the pairs are always ordered lexicographically according to the input value, and the $(\perp, 0^m)$ pairs are always at the end.

In order to define precisely the action of a compressed oracle query, we need to introduce some additional notation.

Let $|D|$ denote the number of pairs $(x, y)$ in database $D$ with $x \neq \perp$. Let $t$ be an upper bound on the number of queries. Then, for a database $D$ with $|D| < t$ and $D(x) = \perp$, we write $D \cup (x, y)$ to denote the new database obtained by deleting one of the $(\perp, 0^m)$ pairs, and by adding the pair $(x, y)$ to $D$, inserted at appropriate location (to respect the lexicographic ordering of the input values).

We also define a “decompression” procedure. For $x \in \{0, 1\}^n$, $\text{StdDecomp}_x$ is a unitary operation on the database register. If $D(x) = \perp$, it adds a uniform superposition over all pairs $(x, y)$ (i.e. it “uncompressed” at $x$). Otherwise, if $D$ is specified at $x$, and the corresponding $y$ register is in a uniform superposition, $\text{StdDecomp}$ removes $x$ and the uniform superposition from $D$. If $D$ is specified at $x$, and the corresponding $y$ register is in a state orthogonal to the uniform superposition, then $\text{StdDecomp}$ acts as the identity. More precisely,

- For $D$ such that $D(x) = \perp$ and $|D| < t$,

$$\text{StdDecomp}_x |D\rangle = \frac{1}{\sqrt{2^n}} \sum_y |D \cup (x, y)\rangle$$

- For $D$ such that $D(x) = \perp$ and $D = t$,

$$\text{StdDecomp}_x |D\rangle = |D\rangle$$

- For $D$ such that $D(x) = \perp$ and $|D| < t$,

$$\text{StdDecomp}_x \left( \frac{1}{\sqrt{2^n}} \sum_y (-1)^{z \cdot y} |D \cup (x, y)\rangle \right) = \begin{cases} 
\frac{1}{\sqrt{2^n}} \sum_y (-1)^{z \cdot y} |D \cup (x, y)\rangle & \text{if } z \neq 0 \\
|D\rangle & \text{if } z = 0
\end{cases} \quad (19)$$

Note that we have specified the action of $\text{StdDecomp}_x$ on an orthonormal basis of the database register (with a bound of $t$ on the size of the database). Moreover, it is straightforward to verify that $\text{StdDecomp}_x$ maps this orthonormal basis to another orthonormal basis, and is thus a well-defined unitary. Note moreover that applying $\text{StdDecomp}_x$ gives the identity. Let $\text{StdDecomp}$ be the related unitary acting on all the registers $x, y, D$ which acts in the following way:

$$\text{StdDecomp} |x, y\rangle \otimes |D\rangle = |x, y\rangle \otimes \text{StdDecomp}_x |D\rangle .$$

So far, we have considered a fixed upper bound on the number of queries. However, one of the advantages of the compressed oracle technique is that an upper bound on the number of queries does not need to be known in advance. To handle a number of queries that is not fixed, we defined the procedure $\text{Increase}$ which simply increases the upper bound on the size of the database by initializing a new register in the state $|\perp, 0^n\rangle$, and appending it to the end. Formally, $\text{Increase} |x, y\rangle \otimes |D\rangle \mapsto |x, y\rangle \otimes |D\rangle |\perp, 0^n\rangle$.

Now, define the unitaries $\text{CStO}'$ and $\text{CPhO}'$ acting in the following way:

$$\text{CStO}' |x, y\rangle \otimes |D\rangle = |x, y \oplus D(x)\rangle \otimes |D\rangle$$

$$\text{CPhO}' |x, y\rangle \otimes |D\rangle = (-1)^{x \cdot D(x)} |x, y\rangle \otimes |D\rangle \quad (20)$$
Finally, we define the compressed standard and phase oracles $C\text{StO}$ and $C\text{PhO}$ as:

$$C\text{StO} = \text{StdDecomp} \circ C\text{StO}' \circ \text{StdDecomp} \circ \text{Increase}$$

$$C\text{PhO} = \text{StdDecomp} \circ C\text{PhO}' \circ \text{StdDecomp} \circ \text{Increase}$$

For an oracle algorithm $A$, we denote by $A^{C\text{StO}}$ (resp. $A^{C\text{PhO}}$), the algorithm $A$ run with the compressed standard (resp. phase) oracle, implemented as described above. The following lemma establishes that regular and compressed oracles are equivalent.

**Lemma 3** ([Zha19]). For any oracle algorithm $A$, and any input state $|\psi\rangle$, $\Pr[A^{\text{StO}}(|\psi\rangle) = 1] = \Pr[A^{C\text{StO}}(|\psi\rangle) = 1]$. Similarly, for any oracle algorithm $B$, $\Pr[B^{\text{PhO}}(|\psi\rangle) = 1] = \Pr[B^{C\text{PhO}}(|\psi\rangle) = 1]$.

In the rest of the paper, we choose to work with phase oracles and compressed phase oracles. Moreover, to use a more suggestive name, we will denote the compressed phase oracle $C\text{PhO}$ by $C\text{PhO}$.

### 6.3 Guessing the XOR of two query outputs

In this section, we formalize the main technical ingredient that will allow us to characterize the structure of strategies that produce valid equations. We do this via a few related technical lemmas. Lemma 4 is a general theorem which relates the probability of guessing the xor of the query outputs at two points, to the structure of a strategy that achieves such probability of guessing. We will not use Lemma 4 directly in our proof of security, but we choose to include it for its generality, as it may be of independent interest. Lemmas 5 and 6 are more specialized, and they establish properties of strategies in certain regimes of guessing probability. We will employ the latter lemmas in our security proof.

Let $n, L \in \mathbb{N}$. In this section, we consider oracle algorithms querying functions in $\text{Bool}(n)$ (i.e. $n$ bit input, and single bit output).

We introduce some notation. When running a compressed oracle simulation, we denote by $O$ the database register. For convenience, we use the following notation for the database register: for a database $D$ (of pairs in $\{0,1\}^n \times \{0,1\}$), and a subset $X \subseteq \{0,1\}^n$, we write:

$$|D \cup X| := \sum_{w \in \{0,1\}^n : x \in X} (-1)^{w_i} |D \cup \{(x, w_x) : x \in X\}|.$$

To simplify further, we write $|D \cup x| = |D \cup \{x\}|$. For $x \in \{0,1\}^n$, we concisely write $D \ni x$ (resp. $D \not\ni x$) to mean that $D$ contains (resp. does not contain) a pair $(x, v)$ for some $v$.

In the following lemma, $P$ is an oracle algorithm acting on an $L$-qubit register $T$, the output register; $W$, a work register; $X$ and $E$, the query registers (where $X$ is an $n$-qubit register, and $E$ is a 1-qubit register). For ease of notation, we will write $|t, x, e, w⟩ := |t⟩|x⟩|e⟩|w⟩$.

**Lemma 4.** Let $n, L \in \mathbb{N}$. For $i \in [L]$, $b \in \{0,1\}$, let $x^i_b \in \{0,1\}^n$. Let $P$ be an oracle algorithm, and $|\psi\rangle$ a state. Let $\epsilon > 0$. Suppose

$$\Pr_{H \leftarrow \text{Bool}(n)} \left[ t_i = H(x^i_0) \otimes H(x^i_1) \text{ for all } i \in [L] : t \leftarrow P^H |\psi\rangle \right] \geq 1 - \epsilon.$$

Let $|\psi_{\text{final}}\rangle$ be the state of $P^{C\text{PhO}} |\psi\rangle$ just before measurement of the output. Then, for any fixed $i$, there exists a (not necessarily normalized) $|\psi'_{\text{final}}\rangle$ such that:

$$|\psi'_{\text{final}}\rangle = \sum_{t \in \{0,1\}^L, x, e, w} \alpha_{t, x, e, w, D} |t, x, e, w⟩ \sum_{b \in \{0,1\}} (-1)^{b_i} |D \cup x^i_b⟩ + \sum_{t \in \{0,1\}^L, x, e, w} \beta_{t, x, e, w, D} |t, x, e, w⟩ \left( |D⟩ + (-1)^{t_i} |D \cup \{x^i_0, x^i_1\}⟩ \right)$$

for some $\alpha_{t, x, e, w, D}$ and $\beta_{t, x, e, w, D}$, and

$$|||\psi_{\text{final}}⟩ - |\psi'_{\text{final}}⟩||^2 \leq 2\epsilon.$$
It is straightforward to see that the above also implies that, upon normalization, $|\psi'_\text{final}\rangle$ is $O(\sqrt{\epsilon})$-close to $|\psi_{\text{final}}\rangle$ in Euclidean distance.

**Proof of Lemma 4.** Fix $i$. Then, we can write the state of $\mathcal{P}^{\text{CPhO}}|\psi\rangle$, just before measurement of the output as

$$
|\psi_{\text{final}}\rangle = \sum_{t \in \{0,1\}^t} \sum_{x,e,w,D,b \in \{0,1\}} \alpha_{t,x,e,w,D,b} |t, x, e, w \rangle |D \cup x^i_b\rangle + \sum_{t \in \{0,1\}^t} \beta_{t,x,e,w,D} |t, x, e, w \rangle |D \cup \{x^i_0, x^i_1\}\rangle + \sum_{t \in \{0,1\}^t} \gamma_{t,x,e,w,D} |t, x, e, w \rangle |D\rangle
$$

for some $\alpha_{t,x,e,w,D,b}$, $\beta_{t,x,e,w,D}$ and $\gamma_{t,x,e,w,D}$.

By the equivalence of the standard and compressed oracle simulations (Lemma 3), we have that, for any $i$, $\Pr_{H \leftarrow \text{Bool}(n)}[t_i = H(x^i_0) \oplus H(x^i_1) : t_i \leftarrow P^H|\psi\rangle]$ is equal to $\Pr[t_i = D(x^i_0) \oplus D(x^i_1)]$ where $t_i$ is distributed as in a compressed oracle simulation. In particular, this is precisely the same as $\Pr[t_i = v_0 \oplus v_1]$, where $t_i = v_0 \oplus v_1$ are distributed as in the following process:

- Run $\mathcal{P}^{\text{CPhO}}|\psi\rangle$, and obtain $t$.
- Apply $\text{StdDecomp}_{x^i_0}$ followed by $\text{StdDecomp}_{x^i_1}$ to the leftover state.
- Measure the database register to obtain the values $v_0$ and $v_1$ at $x^i_0$ and $x^i_1$ respectively. Output $t, v_0, v_1$.

Now,

$$
\text{StdDecomp}_{x^i_0} \circ \text{StdDecomp}_{x^i_1}|\psi_{\text{final}}\rangle = \frac{1}{2} \sum_{t \in \{0,1\}^t} \sum_{x,e,w,D,b \in \{0,1\}} \sum_{v_0,v_1 \in \{0,1\}} \alpha_{t,x,e,w,D,b} (-1)^{v_0} |t, x, e, w \rangle |D \cup \{(x^i_0, v_0), (x^i_1, v_1)\}\rangle
$$

$$
+ \frac{1}{2} \sum_{t \in \{0,1\}^t} \sum_{x,e,w,D} \beta_{t,x,e,w,D} (-1)^{v_0+v_1} |t, x, e, w \rangle |D \cup \{(x^i_0, v_0), (x^i_1, v_1)\}\rangle
$$

$$
+ \frac{1}{2} \sum_{t \in \{0,1\}^t} \sum_{x,e,w,D} \gamma_{t,x,e,w,D} |t, x, e, w \rangle |D \cup \{(x^i_0, v_0), (x^i_1, v_1)\}\rangle
$$

Then,

$$
\epsilon \geq \Pr_{H \leftarrow \text{Bool}(n)}[t_j \neq H(x^i_0) \oplus H(x^i_1) \text{ for some } j \in [L] : t_j \leftarrow P^H|\psi\rangle]
$$

$$
\geq \Pr_{H \leftarrow \text{Bool}(n)}[t_i \neq H(x^i_0) \oplus H(x^i_1) : t_i \leftarrow P^H|\psi\rangle]
$$

$$
= \frac{1}{4} \sum_{t \in \{0,1\}^t} \sum_{x,e,w,D,b \in \{0,1\}^t} \sum_{v_0,v_1 \in \{0,1\}} \left| \sum_b (-1)^{v_0} \alpha_{t,x,e,w,D,b} + (-1)^{v_0+v_1} \beta_{t,x,e,w,D} + \gamma_{t,x,e,w,D} \right|^2
$$

$$
= \frac{1}{4} \sum_{t \in \{0,1\}^t} \sum_{x,e,w,D,b \in \{0,1\}^t} \sum_{v_0,v_1 \in \{0,1\}} \left| \sum_b (-1)^{v_0+v_1} \alpha_{t,x,e,w,D,b} + (-1)^v \beta_{t,x,e,w,D} + \gamma_{t,x,e,w,D} \right|^2,
$$

(26)
where the first line is by the hypothesis of the lemma, and the third line follows from calculating \( \Pr[t_i \neq v_0 \oplus v_1] \) in expression (25).

We can further simplify expression (26) as:

\[
(26) = \frac{1}{2} \sum_{t \in \{0,1\}^L, x, e, w} |\sum_b (-1)^b \alpha_{t,x,e,w,D,b}|^2 \\
+ \frac{1}{2} \sum_{t \in \{0,1\}^L, x, e, w} |(-1)^{t_i} \beta_{t,x,e,w,D} + \gamma_{t,x,e,w,D}|^2 \\
= \frac{1}{2} \sum_{t \in \{0,1\}^L, x, e, w} |\alpha_{t,x,e,w,D,0} - (-1)^{t_i} \alpha_{t,x,e,w,D,1}|^2 \\
+ \frac{1}{2} \sum_{t \in \{0,1\}^L, x, e, w} |\gamma_{t,x,e,w,D} - (-1)^{t_i} \beta_{t,x,e,w,D}|^2, \\
\tag{27}
\]

where the first equality is a crucial part of the calculation, which follows from the following simple fact: for any complex numbers \( \beta, \gamma \),

\[
|\beta + \gamma|^2 + |\beta - \gamma|^2 = 2|\beta|^2 + 2|\gamma|^2.
\]

Now, consider the (not necessarily normalized) state \( |\psi_{\text{final}}\rangle \) which is defined as in (35), except we additionally set \( \alpha_{t,x,e,w,D,0} = (-1)^{t_i} \alpha_{t,x,e,w,D,1} \) for all \( t, x, e, w, D \), and \( \gamma_{t,x,e,w,D} = (-1)^{t_i} \beta_{t,x,e,w,D} \). Then, notice that

\[
|| |\psi_{\text{final}}\rangle - |\psi_{\text{final}}\rangle ||^2 = \sum_{t \in \{0,1\}^L, x, e, w} |\alpha_{t,x,e,w,D,0} - (-1)^{t_i} \alpha_{t,x,e,w,D,1}|^2 + \sum_{t \in \{0,1\}^L, x, e, w} |\gamma_{t,x,e,w,D} - (-1)^{t_i} \beta_{t,x,e,w,D}|^2 \\
\leq 2\epsilon. \\
\tag{28}
\]

where the last line follows from Equations (26) and (27).

\[\square\]

Lemmas 5 and 6 below are technical lemmas, which will be useful in the security proof. In words, Lemma 5 says the following. As earlier, for \( i \in [L] \), let \((x_0^i, x_1^i) \in \{0,1\}^n \times \{0,1\}^n\). Consider a state resulting from a compressed oracle simulation of an algorithm acting on the registers \( T, X, E, W \), as introduced earlier. Suppose that the state only has weight on databases such that, for at least \( k \) of the indices \( i \), the database contains neither \( x_0^i \) nor \( x_1^i \). Then, \( \Pr[t_i = D(x_0^i) \oplus D(x_1^i)] \) for all \( i \) is at most \( 2^{-k} \). In other words, for each index \( i \) such that neither \( x_0^i \) nor \( x_1^i \) are present in the database, the probability of correctly guessing all of the xor’s correctly, drops by a factor of \( 2 \).

In order to state this lemma formally, in the form that will be useful later, we introduce some additional notation. Denote by \( S_{\text{comp}} \) the set of all (normalized) states of the form:

\[
\sum_{t,x,e,w,X \subseteq \{0,1\}^n} \alpha_{t,x,e,w,X} |t, x, e, w\rangle \langle D_X|.
\]

These are states that can be reached by running a compressed oracle simulation. Next, for a set \( X = \{ x_b^i \in \{0,1\}^n : i \in [L], b \in \{0,1\} \} \), define the projector \( \Pi_{\text{no-claw}(X)} := \sum_D D \hat{2} \{ x_b^i \} |D\rangle \langle D| \). Denote by \( S_{\text{no-claw}(X)} \) the set of (normalized) states \( |\psi\rangle \) on registers TXEO such that \( \Pi_{\text{no-claw}(X)} |\psi\rangle = |\psi\rangle \). This is the set of states that have zero weight on databases containing a “claw”. Further, for \( l \in [L] \), define the projector

\[
\Pi_{X, l} := \sum_{D; \{ t : D \cap \{ x_b^i \} \neq \emptyset \} \leq l} |D\rangle \langle D|.
\]
Denote by $S_{X, \leq l}$ the set of (normalized) states $|\psi\rangle$ on registers TXEWO such that $\Pi_{X, \leq l} |\psi\rangle = |\psi\rangle$. In words, this is the set of states with weight only on databases which, for at most $l$ of the indices in $[L]$, contain any of $x_0^i$ or $x_1^i$. Let $\text{DecompAll}$ be the map that "decompresses" at every input, i.e.

$$\text{DecompAll} := \bigotimes_{x \in \{0, 1\}^n} \text{StdDecomp}_x.$$ 

Finally, define

$$\Pi_{\text{valid}} := \sum_{t, D, \forall i, t_i = D(x_0^i) \oplus D(x_1^i)} |t\rangle \langle t| \otimes |D\rangle \langle D|.$$ 

**Lemma 5.** Let $n, L \in \mathbb{N}$. Let $X = \{x_0^i \in \{0, 1\}^n : i \in [L], b \in \{0, 1\}\}$. Let $|\psi\rangle \in S_{\text{comp}} \cap S_{\text{no-claw}(X)} \cap S_{X, \leq l}$. Then,

$$\|\Pi_{\text{valid}} \text{DecompAll} |\psi\rangle \|^2 \leq 2^{-L+l}.$$ 

**Proof.** First, for a set $A$, we write $\text{StdDecomp}_A := \bigotimes_{x \in A} \text{StdDecomp}_x$. Then,

$$\|\Pi_{\text{valid}} \text{DecompAll} |\psi\rangle \|^2 = \|\Pi_{\text{valid}} \circ (\text{StdDecomp}_{\{0, 1\}^n \setminus X} \otimes \text{StdDecomp}_X) |\psi\rangle \|^2$$

Now, notice that $\Pi_{\text{valid}}$ and $\text{StdDecomp}_{\{0, 1\}^n \setminus X}$ commute, because $\Pi_{\text{valid}}$ acts trivially on all database elements other than those in $X$. Thus,

$$\|\Pi_{\text{valid}} \text{DecompAll} |\psi\rangle \|^2 = \|\text{StdDecomp}_{\{0, 1\}^n \setminus X} \circ \Pi_{\text{valid}} \circ \text{StdDecomp}_X |\psi\rangle \|^2$$

where the last line is simply because $\text{StdDecomp}_{\{0, 1\}^n \setminus X}$ is a unitary.

Now, since $|\psi\rangle \in S_{\text{comp}} \cap S_{\text{no-claw}(X)}$, we can write

$$|\psi\rangle = \sum_{t \in \{0, 1\}^L, x, e, w} \sum_{D \cap X = \emptyset} \sum_{I \subseteq [L]} \sum_{b = (b_i : i \in I)} \sum_{v_0, v_1 \in \{0, 1\}, t \in [L]} \alpha_{t,x,e,w,D,I,b} |t, x, e, w, D \cup \{x_0^i : i \in I\}\rangle |D\rangle |t, x_0^i, v_0^i\rangle |t, x_1^i, v_1^i\rangle |D\rangle.$$ 

Then,

$$\text{StdDecomp}_X |\psi\rangle = \sum_{t \in \{0, 1\}^L, x, e, w} \sum_{D \cap X = \emptyset} \sum_{I \subseteq [L]} \sum_{b = (b_i : i \in I)} \sum_{v_0, v_1 \in \{0, 1\}, t \in [L]} \alpha_{t,x,e,w,D,I,b} \sum_{v_0^i, v_1^i \in \{0, 1\}} (-1)^{\sum_i v_0^i} |t, x, e, w\rangle |D\rangle |t, x_0^i, v_0^i\rangle |t, x_1^i, v_1^i\rangle |D\rangle.$$

We can then calculate

$$\|\Pi_{\text{valid}} \text{StdDecomp}_X |\psi\rangle \|^2 = \sum_{t \in \{0, 1\}^L, x, e, w} \sum_{D \cap X = \emptyset} \sum_{I \subseteq [L]} \sum_{b = (b_i : i \in I)} \sum_{v_0, v_1 \in \{0, 1\}, t \in [L]} \sum_{v_0^i, v_1^i \in \{0, 1\}} (-1)^{\sum_i v_0^i} \alpha_{t,x,e,w,D,I,b} |t, x, e, w, D, I, b\rangle |D\rangle |t, x_0^i, v_0^i\rangle |t, x_1^i, v_1^i\rangle |D\rangle.$$ 

We can rewrite (29) as

$$\sum_{t \in \{0, 1\}^L, x, e, w} \sum_{D \cap X = \emptyset} \sum_{I \subseteq [L]} \sum_{b = (b_i : i \in I)} (-1)^{\sum_i v_0^i} \sum_{v_0^i, v_1^i \in \{0, 1\}} (-1)^{\sum_i v_0^i} \alpha_{t,x,e,w,D,I,b} |t, x, e, w, D, I, b\rangle |D\rangle |t, x_0^i, v_0^i\rangle |t, x_1^i, v_1^i\rangle |D\rangle.$$ 

$$\sum_{t \in \{0, 1\}^L, x, e, w} \sum_{D \cap X = \emptyset} 2^L \sum_{I \subseteq [L]} \sum_{b = (b_i : i \in I)} (-1)^{\sum_i v_0^i} \alpha_{t,x,e,w,D,I,b} |t, x, e, w, D, I, b\rangle |D\rangle |t, x_0^i, v_0^i\rangle |t, x_1^i, v_1^i\rangle |D\rangle,$$
for any complex numbers $\alpha_i, \beta_N$, \[
\sum_{i \in [M]} \sum_{j \in [N]} (-1)^{w_{j,i}} \beta_j = M \cdot (|\beta_1|^2 + \cdots + |\beta_N|^2).
\]

Verifying this fact is a straightforward calculation. Then, to be explicit, to obtain (30), we apply the latter as desired.

Proof. In general, for any $i \in [M]$, let $w_{1,i}, \ldots, w_{N,i} \in \{0,1\}$, be such that: for any $s \neq t \in [N]$, \{\{i \in [M] : w_{s,i} = w_{t,i}\} = M/2. Then, for any complex numbers $\beta_1, \ldots, \beta_N$, \[
\sum_{i \in [M]} \sum_{j \in [N]} (-1)^{w_{j,i}} \beta_j = M \cdot (|\beta_1|^2 + \cdots + |\beta_N|^2).
\]

Putting everything together, we have shown that the RHS of (30) is upper bounded by \[
\frac{1}{2^{2L}} \sum_{t \in \{0,1\}^{L \times x,e,w}} 2^L \cdot 2^L \sum_{I \subseteq [L]} \sum_{b \in \{b_i : i \in I\}} |\alpha_{t,x,e,w,D,I,b}|^2
\]
\[
= \frac{1}{2^{L-2}} \sum_{I \subseteq [L]} \sum_{t \in \{0,1\}^{L \times x,e,w}} \sum_{b \in \{b_i : i \in I\}} |\alpha_{t,x,e,w,D,I,b}|^2
\]
\[
= \frac{1}{2^{L-2}} \cdot \|\psi\|^2 = \frac{1}{2^{L-2}}.
\]

As desired.

The next lemma is essentially an abstraction of Lemma 4, stated in a form that will be useful for our security proof later.

Lemma 6. Let $n, L \in \mathbb{N}$. For $i \in [L]$, $b \in \{0,1\}$, let $x^i_b \in \{0,1\}^n$. Let $|\psi\rangle \in \mathcal{S}_\text{comp}$. Suppose \[
\|\Pi_\text{valid} \circ \text{DecompAll} |\psi\rangle \|^2 \geq 1 - \epsilon,
\]

Then, for each $i$, there exists a (not necessarily normalized) state \[
|\psi\rangle = \sum_{t,x,y,x,e,w} \alpha_{t,x,e,w,D} |t,x,e,w\rangle \left( |D \cup x^i_0\rangle + (-1)^i \ |D \cup x^i_1\rangle \right)
\]
\[
+ \sum_{t,x,e,w} \beta_{t,x,e,w,D} |t,x,e,w\rangle \left( |D\rangle + (-1)^i \ |D \cup \{x^i_0, x^i_1\}\rangle \right),
\]

for some coefficients $\alpha_{t,x,e,w,D}, \beta_{t,x,e,w,D}$, such that \[
\| |\psi\rangle - |\psi\rangle' \|^2 \leq 2\epsilon.
\]

Proof. In general, for any $i$, we can write the state $|\psi\rangle$ as \[
|\psi\rangle = \sum_{t,x,e,w} \sum_{b \in \{0,1\}} \alpha_{t,x,e,w,D,b} |t,x,e,w\rangle |D \cup x^i_b\rangle
\]
\[
+ \sum_{t,x,e,w} \beta_{t,x,e,w,D} |t,x,e,w\rangle |D \cup \{x^i_0, x^i_1\}\rangle
\]
\[
+ \sum_{t,x,e,w} \gamma_{t,x,e,w,D} |t,x,e,w\rangle |D\rangle
\]

(33)
for some $\alpha_{t,x,e,w,D,b}, \beta_{t,x,e,w,D}, \gamma_{t,x,e,w,D}$. Let

$$|\psi'\rangle = \sum_{t,x,e,w \in D \cap x_0^t, x_1^t} \alpha_{t,x,e,w,D} |t, x, e, w\rangle \left(|D \cup x_0^t\rangle + (-1)^t |D \cup x_1^t\rangle\right)$$

$$+ \sum_{t,x,e,w \in D \cap x_0^t, x_1^t} \beta_{t,x,e,w,D} |t, x, e, w\rangle \left(|D\rangle + (-1)^t |D \cup \{x_0^t, x_1^t\}\rangle\right),$$

(34)

where $\alpha_{t,x,e,w,D} := \alpha_{t,x,e,w,D,0}$.

By applying the exact same analysis as in the proof of Lemma 4, we obtain that

$$\|\psi - |\psi'\rangle\|^2 \leq 2\epsilon.$$

\[\square\]

6.4 The structure of strategies that produce valid equations

In this section, we characterize the structure of strategies that produce valid equations. The results in this section essentially follow from the results of the previous section (about the structure of strategies that guess the xor of query outputs), specialized to the setting where databases do not contain a claw. We first state and prove a somewhat clean and general lemma (Lemma 7), for the case of guessing a single equation, relating the probability of success to the structure of the strategy. Although we will not be using the latter lemma directly, we include it as it modular enough that it may be of independent interest. Then, we state two corollaries of lemmas 5 and 6, which we will use directly in the security proof later.

Throughout this section, let $\mathcal{X}, \mathcal{Y}, \mathcal{K}$ be finite sets. Let $F = \{f_{k,b} : \mathcal{X} \rightarrow \mathcal{D}_Y\}_{k \in \mathbb{K}, b \in \{0,1\}}$ be a family of noisy trapdoor claw-free functions. Let $f_{k,b} : \mathcal{X} \rightarrow \mathcal{Y}$ be functions satisfying the efficient range superposition property of Definition 3.

Let $P$ be a prover. Denote the registers that $P$ acts on as: $Z, D, Y$, corresponding to the equation; $X, E$, corresponding to the query registers; $W$, corresponding to the witness. For ease of notation, we omit keeping track of $P$’s input registers, and any auxiliary work register. Assume that $x \in \mathcal{X}$ is represented in register $X$ via its bit decomposition. Let $n$ denote the size of the bit decomposition. For ease of notation, for $x \in \mathcal{X}$, we omit writing $\text{BitDecomp}(x)$ in this section, when it is clear from the context.

All parameters implicitly depend on the security parameter $\lambda$, which we omit writing.

**Lemma 7.** Let $P$ be any non-uniform quantum polynomial-time oracle algorithm taking as input a choice of claw-free function pair $k$, and outputting a state on registers $ZDYXEW$. Suppose there exists a function $\delta$ (of the security parameter) such that:

$$\Pr[z = d \cdot (x_0^d \oplus x_1^d) \oplus H(x_0^d) \oplus H(x_1^d) : k \leftarrow F(1^\lambda), (z, d, y) \leftarrow P^H(k), H \leftarrow \text{Boo}(n)] \geq \frac{1}{2} + \delta(\lambda).$$

Let the state of $P_{\text{CPHO}}(k)$ just before measurement of the output be

$$|\psi_{\text{final}}^k\rangle = \sum_{z, d, y, x, e, w \in \{0,1\}} \sum_{D \cap x_0^d, x_1^d} \alpha_{z,d,y,x,e,w,D,b}^k |z, d, y, x, e, w\rangle |D \cup x_0^y\rangle$$

$$+ \sum_{z, d, y, x, e, w \in \{0,1\}} \beta_{z,d,y,x,e,w,D}^k |z, d, y, x, e, w\rangle |D \cup \{x_0^y, x_1^y\}\rangle$$

$$+ \sum_{z, d, y, x, e, w \in \{0,1\}} \gamma_{z,d,y,x,e,w,D}^k |z, d, y, x, e, w\rangle |D\rangle$$

(35)

Then, there exists a negligible function $\text{negl}$ such that, for all $\lambda$: 33
\[
\mathbb{E}_{k \leftarrow \mathcal{F}} \sum_{z,d,y,x,e,w, D \not
i x_0^y, x_1^y} \left| \beta_{z,d,y,x,e,w,D}^k \right|^2 = \text{negl}(\lambda) \,.
\]

\[
\mathbb{E}_{k \leftarrow \mathcal{F}} \sum_{z,d,y,x,e,w} \sum_{b \in \{0,1\}} \left| \alpha_{z,d,y,x,e,w,D,b}^k \right|^2 \geq 2\delta(\lambda) - \text{negl}(\lambda) \,.
\]

\[
\mathbb{E}_{k \leftarrow \mathcal{F}} \sum_{z,d,y,x,e,w, D \not
i x_0^y, x_1^y} \sum_{b \in \{0,1\}} \left| \alpha_{z,d,y,x,e,w,D,b}^k \right|^2 \leq 2\delta(\lambda) + \text{negl}(\lambda) \,.
\]

In words, this lemma establishes the following:

• The state of the oracle algorithm after a compressed oracle simulation has at most negligible weight on databases containing a claw. This is point (i).

• In the regime where \( \delta \) is non-negligible, there must be non-negligible weight on \( y, D \) such that \( D \) contains at least one of the two pre-images of \( y \). Moreover, on average over \( k \), there is non-negligible weight on \( z, d, y, x, e, w, D \), such that \( \alpha_{z,d,y,x,e,w,D,0}^k \) is a non-negligible fraction of \( \alpha_{z,d,y,x,e,w,D,1}^k \) and vice versa.

• In the regime where the winning probability is close to 1, almost all the weight must be on \( y, D \) such that \( D \) contains at least one of the two pre-images of \( y \). Moreover, it must be that, approximately, \( \alpha_{z,d,y,x,e,w,D,0}^k = (-1)^{z+d \cdot (x_0^y \oplus x_1^y)} \alpha_{z,d,y,x,e,w,D,1}^k \).

The proof uses ideas encountered in the previous section.

**Proof of Lemma 7.** By an analogous calculation as in the proof of Lemma 4, we have that, for all \( \lambda \),

\[
\frac{1}{2} - \delta(\lambda) \geq \text{Pr}\left[z \neq d \cdot (x_0^y \oplus x_1^y) \oplus H(x_0^y) \oplus H(x_1^y) : k \leftarrow \mathcal{F}(1^\lambda), (z, d, y) \leftarrow \mathcal{P}^H(k), H \leftarrow \text{Bool}(n)\right] = \\
\mathbb{E}_{k} \left( \frac{1}{2} \sum_{z,d,y,x,e,w, D \not
i x_0^y, x_1^y} \left| \alpha_{z,d,y,x,e,w,D,0}^k - (-1)^{z+d \cdot (x_0^y \oplus x_1^y)} \alpha_{z,d,y,x,e,w,D,1}^k \right|^2 + \frac{1}{2} \sum_{z,d,y,x,e,w, D \not
i x_0^y, x_1^y} \left| \beta_{z,d,y,x,e,w,D}^k - (-1)^{z+d \cdot (x_0^y \oplus x_1^y)} \beta_{z,d,y,x,e,w,D}^k \right|^2 \right) \,.
\]

(36)

Now, notice that there exists a negligible function \( \text{negl} \), such that, for all \( \lambda \),

\[
\mathbb{E}_{k} \sum_{z,d,y,x,e,w, D \not
i x_0^y, x_1^y} \left| \beta_{z,d,y,x,e,w,D}^k \right|^2 = \text{negl}(\lambda) \,.
\]

(37)
otherwise this would immediately yield an efficient algorithm to extract a claw. This, combined with (36), straightforwardly implies that, for all \( \lambda \),
\[
\frac{1}{2} - \delta(\lambda) + \text{neg}'(\lambda) \geq \mathbb{E}_k \left( \frac{1}{2} \sum_{z, d, y, x, e, w \in D} |\alpha_{z, d, y, x, e, w, D, 0} - (-1)^{z+d}x_0^y \alpha_{z, d, y, x, e, w, D, 1}|^2 + \frac{1}{2} \sum_{z, d, y, x, e, w \in D, x \neq x_1^0} |\gamma_{z, d, y, x, e, w, D}|^2 \right),
\]
for some other negligible function \( \text{neg}'(\lambda) \). Equation (37) also implies that, for all \( \lambda \),
\[
\mathbb{E}_k \sum_{z, d, y, x, e, w, D} |\gamma_{z, d, y, x, e, w, D}|^2 \geq 1 - \mathbb{E}_k \sum_{z, d, y, x, e, w \in \{0,1\}, D} \sum_{x \neq x_1^0} |\alpha_{z, d, y, x, e, w, D, b}|^2 - \text{neg}(\lambda)
\]
(39)
Plugging this into (38), gives
\[
\frac{1}{2} - \delta(\lambda) + \text{neg}'(\lambda) \geq \mathbb{E}_k \left( \frac{1}{2} \sum_{z, d, y, x, e, w \in D} |\alpha_{z, d, y, x, e, w, D, 0} - (-1)^{z+d}x_0^y \alpha_{z, d, y, x, e, w, D, 1}|^2 + \frac{1}{2} \sum_{z, d, y, x, e, w \in D, x \neq x_1^0} |\alpha_{z, d, y, x, e, w, D, b}|^2 - \text{neg}(\lambda) \right),
\]
(40)
Now, to prove (ii), suppose for a contradiction that there exists some polynomial \( p \) such that, for infinitely many \( \lambda \),
\[
\mathbb{E}_k \sum_{z, d, y, x, e, w \in \{0,1\}, D} \sum_{x \neq x_1^0} |\alpha_{z, d, y, x, e, w, D, b}|^2 < 2\delta(\lambda) - p(\lambda).
\]
Then, for any such \( \lambda \), plugging this into (40), and rearranging, gives
\[
\text{neg}'(\lambda) + \text{neg}(\lambda) \geq \frac{1}{2} \mathbb{E}_k \sum_{z, d, y, x, e, w \in D} |\alpha_{z, d, y, x, e, w, D, 0} - (-1)^{z+d}x_0^y \alpha_{z, d, y, x, e, w, D, 1}|^2 + 2p(\lambda),
\]
(41)
which is a contradiction.

Finally, (iii) follows similarly from (40).

\( \square \)

Next, we proceed to stating and proving two corollaries of Lemmas 5 and 6 from the previous section. In this case, we do not restrict ourselves to the case of guessing a single equation. So, registers \( \mathbf{Z}, \mathbf{D}, \mathbf{Y} \), each consist of \( L \) sub-registers, for some \( L \in \mathbb{N} \). So \( \mathbf{Y} = Y_1 \ldots Y_L \), and similarly for \( \mathbf{Z} \) and \( \mathbf{D} \). We will write \( y = y_1 \ldots y_L \) where \( y_i \) is the \( i \)-th image; \( z = z_1 \ldots z_L \), where \( z_i \in \{0,1\}; d = d_1 \ldots d_L \), where \( d_i \in \{0,1\}^n \).

We introduce some additional notation. For a subset \( X \subseteq \mathcal{X} \), we define the following state of the database register \( \mathbf{O} \),
\[
D_X := \sum_{w : \mathbf{w} \neq \mathbf{0}} (-1)^{\sum_{x \in X} w_x} |\{(x, w_x) : x \in X\} \rangle \langle \mathbf{w}|.
\]

Analogously to the previous section, let \( S_{\text{comp}} \) be the set of states, on registers \( \text{ZDYXEWO} \), that can be reached by running a compressed oracle simulation. For an NTWF key \( k \), let \( S_{k,\text{-claw}} \) be the set of states, on registers \( \text{ZDYXEWO} \), that have zero weight on \( y, D \) such that \( D \) contains a claw for \( y_i \) for some \( i \).

Further, for \( l \in [L] \), define the projector
\[
\Pi_{k,\leq l} := \sum_{y, D : |\{z : D(z, x_0^y) \neq \mathbf{0}\}| \leq l} |y\rangle \langle y| \otimes |D\rangle \langle D|_{\mathbf{O}}.
\]

35
Denote by $S_{k, \leq l}$ the set of states $|\psi\rangle$ on registers ZDYXEWO such that $\Pi_{k, \leq l} |\psi\rangle = |\psi\rangle$.

Finally, define

$$\Pi_{\text{valid}, k} := \sum_{\forall i, z, d, l, \langle x_0^y, z \rangle = D(x_0^y) \oplus D(x_1^y)} |z, d, y\rangle \langle z, d, y| \otimes |D\rangle \langle D| ,$$

where the pre-images are with respect to key $k$. In words, $\Pi_{\text{valid}, k}$ is the projector onto states for which all $L$ equations are valid, with respect to key $k$.

**Corollary 1.** Let $k$ be an NTCF key. Let $|\psi\rangle \in S_{\text{comp}} \cap S_{k, \text{no-claw}} \cap S_{k, \leq l}$. Then,

$$\|\Pi_{\text{valid}, k} \circ \text{DecompAll} |\psi\rangle\|^2 \leq 2^{-L+l}.$$

**Proof.** For a fixed $y$, let

$$\Pi_{\text{valid}, k, y} := \sum_{\forall i, z, d, l, \langle x_0^y, z \rangle = D(x_0^y) \oplus D(x_1^y)} |z, d, y\rangle \langle z, d, y| \otimes |D\rangle \langle D| .$$

Then, we can write $\Pi_{\text{valid}, k}$ as

$$\Pi_{\text{valid}, k} = \sum_{y} |y\rangle \langle y| \otimes \Pi_{\text{valid}, k, y}.$$

Now, we can write $|\psi\rangle = \sum_{y} \alpha_y |y\rangle \otimes |\phi_y\rangle$, for some $\alpha_y, |\phi_y\rangle$. Then,

$$\|\Pi_{\text{valid}, k} \circ \text{DecompAll} |\psi\rangle\|^2 = \sum_{y} |\alpha_y|^2 \|\Pi_{\text{valid}, k, y} \circ \text{DecompAll} |\phi_y\rangle\|^2 \leq (\sum_{y} |\alpha_y|^2) \cdot 2^{-L+l} = \| |\psi\rangle\|^2 \cdot 2^{-L+l} = 2^{-L+l} ,$$

where the second line follows by an application of Lemma 5, using the fact that, for each $y$, $|\phi_y\rangle$ inherits the property that the database register contains no claws, and there are at most $l$ indices such that the database contains some pre-image corresponding to each of these indices.

The following is a corollary of Lemma 6

**Corollary 2.** Let $k$ be a choice of claw-free function pair. Let $|\psi\rangle \in S_{\text{comp}} \cap S_{k, \text{no-claw}}$. Suppose

$$\|\Pi_{\text{valid}, k} \circ \text{DecompAll} |\psi\rangle\|^2 \geq 1 - \epsilon ,$$

Then, for each $i$, there exists a (not necessarily normalized) state

$$|\psi_i\rangle = \sum_{\forall i, z, d, y, x, e, w, D} \alpha_{z, d, y, x, e, w, D} |z, d, y, x, e, w\rangle \left( |D \cup x_0^y\rangle + (-1)^{z+d-(x_0^y \oplus x_1^y)} |D \cup x_1^y\rangle \right) + \sum_{\forall i, z, d, y, x, e, w, D} \beta_{z, d, y, x, e, w, D} \left( |D\rangle + (-1)^{z+d-(x_0^y \oplus x_1^y)} |D \cup \{x_0^y, x_1^y\}\rangle \right),$$

for some coefficients $\alpha_{z, d, y, x, e, w, D}, \beta_{z, d, y, x, e, w, D}$, such that

$$\| |\psi\rangle - |\psi_i\rangle \|^2 \leq 2\epsilon .$$

Notice that the above straightforwardly implies that the normalization of $|\psi_i\rangle$ is $O(\sqrt{\epsilon})$-close to $|\psi\rangle$ in Euclidean distance.
Proof. Define \( \Pi_{\text{invalid}, k} := I - \Pi_{\text{valid}, k} \). For a fixed \( y \), let

\[
\Pi_{\text{invalid}, k, y} := \sum_{z, d, D, \psi} |z, d, D\rangle \langle z, d, D| \cdot \langle D| \langle D|
\]

Then, we can write \( \Pi_{\text{invalid}, k} \) as

\[
\Pi_{\text{invalid}, k} = \sum_y |y\rangle \langle y| \otimes \Pi_{\text{invalid}, k, y}
\]

Now, we can write \( |\psi\rangle = \sum_y \gamma_y |y\rangle \otimes |\phi_y\rangle \), for some \( \gamma_y \), \( |\phi_y\rangle \). Then,

\[
\epsilon \geq \| \Pi_{\text{invalid}, k} \circ \text{DecompAll} \ |\psi\rangle \|^2 = \sum_y \gamma_y^2 \| \Pi_{\text{invalid}, k, y} \circ \text{DecompAll} \ |\phi_y\rangle \|^2.
\] (43)

Since \( |\psi\rangle \in S_{\text{comp}} \), then, for each \( y \), the state \( |\phi_y\rangle \) inherits the same property (as a state on all other registers except the \( Y \) register). Thus, we can apply Lemma 6, and deduce that, for each \( y \), for each \( i \), there exists a (not necessarily normalized) state

\[
|\phi'_y\rangle = \sum_{z, d, x, e, w, D} \alpha_{z, d, x, e, w} \langle D \cup x_0^w | D \cup x_1^w | \rangle \left( |D \cup x_0^w | D \cup x_1^w \right)
\]

\[
+ \sum_{z, d, x, e, w, D} \beta_{z, d, x, e, w} \langle D \cup (-1)^{z_i + d_i \cdot (x_0^w \oplus x_1^w)} |D \cup x_0^w \rangle \langle D \cup x_1^w | \rangle,
\] (44)

such that

\[
\| |\phi_y\rangle - |\phi'_y\rangle \|^2 \leq 2 \cdot \| \Pi_{\text{invalid}, k, y} \circ \text{DecompAll} \ |\phi_y\rangle \|^2.
\] (45)

Formally, in order to match the syntax of Lemma 6, we append an output register \( T \) to \( |\phi_y\rangle \), and apply the unitary that maps \( |0\rangle_T \rightarrow |z_i \oplus d_i \cdot (x_0^w \oplus x_1^w)\rangle_T |z, d, y\rangle \). Then, applying Lemma 6 gives precisely the above.

Now, let \( |\psi'\rangle = \sum_y \gamma_y |y\rangle \otimes |\phi'_y\rangle \). Then, plugging this into (43), gives

\[
\| |\psi\rangle - |\psi'\rangle \|^2 \leq 2\epsilon.
\]

Notice that \( |\psi'\rangle \) has the desired form. \( \square \)

6.5 Extracting a claw

In this section we complete the proof of perfect unexplainability of Construction 2, building on the machinery developed in Section 6.4.

Suppose for a contradiction that there exist algorithms \( \text{Verify} \) and \( \text{Explain} \) which violate Definition 8. Then, there exists \( m \) such that

\[
\Pr_{pk}[\text{Verify}(pk, c, m, w) = 1 : (c, w) \leftarrow \text{Explain}(pk, m)]
\]

is non-negligible.

Moreover, recall that \( \text{Verify}(pk, c, m, w) = 0 \) if \( c \) is not in the support of the distribution of \( \text{Enc}(pk, m) \) (we say that the tuple \( (pk, c, m) \) is inconsistent in this case). In this section, we will prove a slightly stronger result, and we will only assume that \( \text{Verify}(pk, c, m, w) = 0 \) with exponentially small probability (in the security parameter) if \( c \) is not in the support of the distribution of \( \text{Enc}(pk, m) \).

Without loss of generality, let us assume that \( m = 0 \), so that a valid ciphertext \( c = (z, d, y) \) simply corresponds to \( L \) valid "equations", i.e. \( z_i = d_i \cdot (x_0^w \oplus x_1^w) \oplus H(x_0^w) \oplus H(x_1^w) \) for all \( i \in [L] \).

At a high level, the proof proceeds in two steps.
(i) The first key observation is that, because by definition Verify never accepts an inconsistent tuple (or except with exponentially small probability), then it must be the case that whenever Verify accepts, it must itself have queried at a superposition of both pre-images in the following more precise sense.

For an NTCF key $k$, denote by Verify$_k$ the algorithm Verify where we fix the choice of claw-free functions to be $k$. Let $A$ be any efficient oracle algorithm that, on input a choice of claw-free functions, outputs tuples $(z, d, y, w)$. Suppose we run a compressed oracle simulation of $A$, followed by Verify, i.e. we run $(\text{Verify}_k \circ A(k))^{\text{CPhO}}$. Let the state right before measurement of Verify’s output be:

$$\alpha |0\rangle + \beta |1\rangle |\phi_1\rangle,$$

where the first qubit is the output qubit of Verify, and $|\phi_0\rangle$, $|\phi_1\rangle$ are some states on the remaining registers (including the database register). Then, except with negligible probability over the choice of $k$, the following holds: if $\beta$ is non-negligible, then $|\phi_1\rangle$ has weight only on databases containing, for each $i$, either $x_0^y$ or $x_1^y$, and moreover, the weights for each pre-image are approximately equal. A bit more precisely, we first argue that, for any state $|\psi\rangle$ which results from a compressed oracle simulation,

$$\|\langle 1 \rangle \langle 1\rangle_Q \otimes (I - \Pi_{\text{valid}})) \text{DecompAll} \circ \text{Verify}^{\text{CPhO}}_{k} |\psi\rangle \|^2$$

must be exponentially small. Otherwise, this would violate the fact that Verify accepts inconsistent tuples only with exponentially small probability. This is the content of Lemma 8.

Since $|\phi_1\rangle$ is a state resulting from a compressed oracle simulation, then, if $\beta$ is non-negligible, it must be that

$$\|\langle 1 \rangle \langle 1\rangle_Q \otimes \Pi_{\text{valid}}) \text{DecompAll} \circ \text{Verify}^{\text{CPhO}}_{k} |\phi_1\rangle \|^2 \geq 1 - O(2^{-c})$$

for some $c > 0$.

We can then appeal to Corollary 2 to deduce that $|\phi_1\rangle$ has the right structure, i.e. weight only on databases containing, for each $i$, either $x_0^y$ or $x_1^y$, with the weights for each pre-image approximately equal.

(ii) We then construct an extraction algorithm which succeeds at extracting a claw with non-negligible probability (thus giving a contradiction). To provide some intuition, let us first describe a simplified version of the algorithm in the case $L = 1$. This is only sufficient to extract a claw if Explain succeeds at convincing Verify with high probability, e.g. $9/10$ (as opposed to merely non-negligible).

(a) Run Explain$^{\text{CPhO}}(k)$, and measure the output registers to obtain $z, d, y$. Moreover, check if the database register at this point contains a pre-image of $y$. If so, measure it. Denote this by $x_0^y$.

(b) Run Verify$^{\text{CPhO}}_{k}$ on the leftover state from the previous step, and measure the output register. Conditioned on “accept”, measure the database register. If the database contains $x_0^y$, output the claw $(x_0^y, x_b^y)$.

As we also described in the technical overview, the idea behind this algorithm is that, thanks to the first observation, the state conditioned on obtaining “accept” in step (b) has weight only on databases containing either $x_0^y$ or $x_1^y$, and moreover, the weights for each pre-image are approximately equal. This implies that, conditioned on observing “accept” in step (b), the final measurement of the algorithm is guaranteed to produce one of the two pre-images approximately uniformly at random.

Now, notice that the algorithm already has a constant probability of obtaining one of the two pre-images in step (a) (assuming Explain succeeds with probability $\frac{9}{10}$). To see this, notice that if this weren’t the case, then Explain would be producing valid equations at most with probability close to $\frac{1}{2}$ (since this is the probability of producing a valid equation when the database register does not contain any of the two pre-images), and therefore the probability that Verify accepts Explain’s output would also be at most close to $\frac{1}{2}$ (instead of being $\frac{9}{10}$). Thus, what is left to show is that the probability of obtaining “accept” in step (ii) conditioned on finding a pre-image in step (i) is still noticeable (in fact, constant).

To see that this is the case, notice that, using the notation introduced earlier, the fact that Explain succeeds with probability $\frac{9}{10}$ implies that the coefficient $\beta$ is such that $|\beta|^2 > \frac{9}{10}$. The question is then:
if we look at the state after running $\text{Explain}^{C\text{PhO}}(k)$, which databases are contributing to the weight on $\beta$? The key is that databases that do not contain any pre-image can at most contribute weight $\frac{1}{2}$ (this is the content of Lemma 1 from Section 6.4, for the special case of $L = 1$). Thus at least $\frac{1}{10}$ of the weight in $|\beta|^2$ must come from databases containing one of the two pre-images. Thus, conditioned on obtaining a pre-image after step (a), the algorithm still has a constant probability of obtaining the other pre-image after the measurement in step (b).

Things are a bit more tricky if we wish to show that $\text{Explain}$ succeeds at most with negligible probability. Then, the argument above no longer works. The first step is to modify the encryption algorithm to be a parallel repetition of the single-shot algorithm, as in Construction 2. Let $L$ denote the number of repetitions. Let us try to carry out a similar argument as before. We can again deduce that $|\beta|^2$ must be non-negligible if $\text{Explain}$ succeeds with non-negligible probability. Now, paralleling our analysis in the $L = 1$ case, what we would like to establish is that there exists some (efficiently implementable) measurement $M$ of the database register which satisfies the following:

(a) The outcome of $M$ is a string $x$ (potentially a pre-image).
(b) Conditioned on outcome $x$, running $\text{Verify}$ on the post-measurement state still leads to acceptance with non-negligible probability.

While condition (b) was straightforwardly true in the $L = 1$ case, it need not be in the case of a general $L$. The reason is that for a given $y = y_1 \ldots y_L$, there are now exponentially many possible subsets of pre-images of the $y_i$. It could be the case that a very favorable interference of all of these exponentially many branches leads to a non-negligible $|\beta|^2$, but conditioned on obtaining any particular $x$, $\text{Verify}$ may only accept with exponentially small probability. Our approach to overcome this barrier is different than earlier, and is a bit more involved.

We show that there exists a sufficiently small (polynomial-size) set of measurements such that the following guarantee holds: at least one of these measurements produces a pre-image of some $y_i$ with non-negligible probability. Moreover, the post-measurement state conditioned on producing a pre-image leads to $\text{Verify}$ accepting with non-negligible probability. This is essentially the content of Lemma 9. We provide some high-level intuition before presenting the formal proof.

Consider the first image $y_1$. Consider the measurement $M_1$ that checks whether a database contains a pre-image of $y_1$ or not. We can write the state before this measurement as

$$|\phi_{\text{yes}}\rangle + |\phi_{\text{no}}\rangle,$$

where $|\phi_{\text{yes}}\rangle$ is the projection onto the former outcome, and $|\phi_{\text{no}}\rangle$ on the latter.

One of the two unnormalized post-measurement states must contribute non-negligible weight to acceptance (meaning that running $\text{Verify}$ on it will result in non-negligible amplitude on the accept outcome). In fact, suppose for a contradiction that both contributions were exponentially small, then, no matter how constructive the interference is between the two branches, the resulting weight on accept would still be exponentially small (since there are only two branches here).

Now, if we are lucky and it is $|\phi_{\text{yes}}\rangle$ that leads to a non-negligible weight on accept, then we are almost done. We simply perform measurement $M_1$. Then with non-negligible probability we will collapse to $|\phi_{\text{yes}}\rangle$. Now, $|\phi_{\text{yes}}\rangle$ is made up of two components: one with databases containing $x_0^1$, and one with databases containing $x_1^1$. One of the two (unnormalized) components must contribute non-negligible amplitude to accept. Then, if we measure to extract a pre-image, we collapse to this component with non-negligible probability.

Suppose instead that we are unlucky, and it is only $|\phi_{\text{no}}\rangle$ that contributed non-negligible weight to accept. For simplicity, let’s take this non-negligible weight to be $1/p$. Then, we can sort of iterate the argument, going to the next pre-image $y_2$. We can define the 2-bit outcome measurement $M_2$ that checks whether the database contains a pre-image of $y_1$ (this is the first bit of the outcome) and whether it contains a pre-image of $y_2$ (this is the second bit).

Notice that we can write:

$$|\phi_{\text{no}}\rangle = |\phi_{\text{no, yes}}\rangle + |\phi_{\text{no, no}}\rangle,$$

39
where $|\phi_{\text{no, yes}}\rangle$ is the component that has weight only on databases that contain a pre-image of $y_2$ (and no pre-image of $y_1$), and $|\phi_{\text{no, no}}\rangle$ has weight on databases that do not contain a pre-image of $y_2$ (and still no pre-image of $y_1$). If the (un-normalized) state $|\phi_{\text{no, yes}}\rangle$ contributes non-negligible weight to accept (upon applying Verify) then we are in good shape, since $M_2$ will collapse to $|\phi_{\text{no, yes}}\rangle$ with non-negligible probability. Otherwise, almost all of the $1/p$ contribution (except a negligible amount) must come from $|\phi_{\text{no, no}}\rangle$.

We can continue to iterate this say $L/2$ times. If we are unlucky every time, it means that $|\phi_{\text{no, no, ..., no}}\rangle$ (with $L/2$ no’s) contributes $1/p$ weight (minus a negligible quantity) to acceptance. The key observation now is that states with databases that have a constant fraction of pre-images missing can only lead to acceptance with exponentially small probability (this was formalized in Corollary 1). So this is a contradiction.

Thus, along the way, some $|\phi_{\text{no, ..., no, yes}}\rangle$ must be such that $|||\phi_{\text{no, ..., no, yes}}\rangle||$ is non-negligible, and moreover Verify $|\phi_{\text{no, ..., no, yes}}\rangle$ has non-negligible weight on accept.

Note, importantly that we do not perform all of the $L/2$ measurements that we defined simultaneously. Instead, only one measurement (picked uniformly at random) is performed. The existence of some good $|\phi_{\text{no, ..., no, yes}}\rangle$ guarantees that with $1/2^L$ probability we will be making a measurement which results in a collapse to the good $|\phi_{\text{no, ..., no, yes}}\rangle$ with non-negligible probability.

We now proceed to the formal proof. Since our construction and security are in the random oracle model, Verify and Explain are oracle algorithms.

As in the previous section, we denote the registers that Explain acts on as: $X, D, Y, W$, corresponding to the equation; $X, E, W$, corresponding to the query registers; $W$, corresponding to the witness. For ease of notation, we omit keeping track of the input registers and any auxiliary work register. Moreover, recall that we are taking $m = 0$ without loss of generality, so we omit keeping track of an $M$ register.

Verify acts on registers: $Z, D, Y, W$, the input registers, and $Q$ an output register. Again, we omit keeping track of an input register containing the public key.

When running a compressed oracle simulation, we denote by $O$ the database register.

For ease of notation we denote $Q := \text{Verify}$ and $P := \text{Explain}$.

Denote by $Q_k$ the algorithm $Q$ with fixed input the NTCF key $k$, and similarly denote by $Q_{k,z,d,y}$ the algorithm $Q$ with fixed inputs $k, z, d, y$. For $y \in \{0, 1\}^{m \times L}$, we write $y = y_1 \ldots y_L$, where $y_i \in \{0, 1\}^m$ for each $i$ (here $m$ is the size of the bit-decomposition of an element in the co-domain of $f_{k,0}, f_{k,1}$).

Recall that we denote by $\text{PhO}$ and $\text{CPhO}$ respectively the phase oracle and the compressed phase oracle (as defined in Section 6.2). Finally, using a similar notation as in the previous section, we denote by $S_{\text{comp}}$ the set of (normalized) states on registers $ZDYXEWO$ of the form:

$$\sum_{z, d, y, x, e, w, X \subseteq \{0, 1\}^m} \alpha_{z, x, e, w, X} |z, d, y, x, e, w, X\rangle |D_X\rangle.$$ 

These are states that can be reached by running a compressed oracle simulation.

For an NTCF key $k$, let $S_{k, \text{no-claw}}$ be the set of (normalized) states, on registers $ZDYXEWO$, that have zero weight on $y, D$ such that $D$ contains a claw for $y_i$ for some $i$. Let $\Pi_{\text{valid}, k}$, acting on registers $ZDYXE$, be the projector onto states, for which all $L$ equations are valid, with respect to key $k$. We will omit writing the subscript $k$ when it is clear from the context.

**Lemma 8.** There exists a constant $c > 0$ such that, for all $\lambda$, for all $|\psi\rangle \in S_{\text{comp}}$, for all $k$:

$$\|(1 - |\psi\rangle \langle \psi|) \text{DecompAll} \circ Q_k^{\text{CPhO}} |\psi\rangle \|^2 \leq 2^{-\lambda c}.$$ 

**Proof.** Suppose for a contradiction that, for all $c > 0$, there exists $\lambda$, $|\psi\rangle \in S_{\text{comp}}$, and $k$, such that:

$$\|(1 - |\psi\rangle \langle \psi|) \text{DecompAll} \circ Q_k^{\text{CPhO}} |\psi\rangle \|^2 > 2^{-\lambda c}. \tag{46}$$

Then, we will show that, for all $c > 0$, there exists $\lambda$, $z, d, y, H$, and a state $|\phi_H\rangle$ on registers $XEW$ (plus some purifying registers) such that $z \neq d \cdot (x_0^H + x_1^H) \oplus H(x_0^H) \oplus H(x_1^H)$, but

$$\|(1 - |\psi\rangle \langle \psi|) Q_k^H (|z, d, y\rangle \otimes |\phi_H\rangle) \|^2 > 2^{-\lambda c}.$$
which would contradict the fact that \( Q \) accepts inconsistent answers with exponentially small probability.

For a function \( H \in \text{Bool}(n) \), let \( \Pi_H \) be the projector onto the database corresponding to the truth table of \( H \). Then,

\[
I - \Pi_{\text{valid}} = \sum_{z,d,y} \sum_{z \neq d \neq (x_0^z \oplus x_1^z) \oplus H(x_0^y) \oplus H(x_1^y)} |z, d, y \rangle \langle z, d, y| \otimes \Pi_H.
\]

We can thus rewrite Equation (46) as saying that, for all \( c \), there exists \( \lambda \), and a state \( |\psi\rangle = \sum_{z,d,y} |z, d, y\rangle \otimes |\phi_{z,d,y}\rangle \in \mathcal{S}_{\text{comp}} \), such that

\[
\sum_{z,d,y} \sum_{z \neq d \neq (x_0^z \oplus x_1^z) \oplus H(x_0^y) \oplus H(x_1^y)} \| (\langle 1 \rangle_Q \otimes \Pi_H) \text{DecompAll} \circ Q_k^{\text{PhO}} (|z, d, y\rangle \otimes |\phi_{z,d,y}\rangle) \|^2 > 2^{-\lambda c}.
\] (47)

This implies that there exists \( d, z, y \) such that

\[
\sum_{H_c} \sum_{z \neq d \neq (x_0^z \oplus x_1^z) \oplus H(x_0^y) \oplus H(x_1^y)} \| (\langle 1 \rangle_Q \otimes \Pi_H) \text{DecompAll} \circ Q_k^{\text{PhO}} |z, d, y\rangle \otimes |\phi_{z,d,y}\rangle \text{XEW}_{O} \|^2 > 2^{-\lambda c}.
\] (48)

Let \( |\phi\rangle := |\phi_{z,d,y}\rangle \) from now on, where \( z, d, y \) satisfy the above equation.

Notice that \( \text{DecompAll} \circ Q_k^{\text{PhO}} |\psi\rangle = Q_k^{\text{PhO}} \circ \text{DecompAll} |\psi\rangle \) for any \( |\psi\rangle \in \mathcal{S}_{\text{comp}} \). Hence, the LHS of (48) is equal to

\[
\sum_{H_c} \sum_{z \neq d \neq (x_0^z \oplus x_1^z) \oplus H(x_0^y) \oplus H(x_1^y)} \| (\langle 1 \rangle_Q \otimes \Pi_H) Q_k^{\text{PhO}} \circ \text{DecompAll} |z, d, y\rangle \otimes |\phi\rangle \|^2.
\] (49)

Now, we can write \( \text{DecompAll} |\phi\rangle = \sum_{H_c} |\phi_{H}\rangle_{\text{XEW}} \otimes |H\rangle_Q \) for some \( |\phi_{H}\rangle \), where \( |H\rangle \) denote the database corresponding to the truth table of the function \( H \). Then, (49) is equal to

\[
\sum_{H_c} \sum_{z \neq d \neq (x_0^z \oplus x_1^z) \oplus H(x_0^y) \oplus H(x_1^y)} \| (\langle 1 \rangle_Q \otimes \Pi_H) Q_k^{\text{PhO}} |z, d, y\rangle \otimes |\phi_{H}\rangle \otimes |H\rangle \|^2
\] (50)

\[
= \sum_{H_c} \| (\langle 1 \rangle_Q Q_k^{\text{PhO}} |z, d, y\rangle \otimes |\phi_{H}\rangle \otimes |H\rangle \|^2
\] (51)

where the last line uses the fact that \( Q_k^{\text{PhO}} \) commutes with \( \Pi_H \). It follows straightforwardly that there exists \( H \) such that \( z \neq m \oplus d \cdot (x_0^y \oplus x_1^y) \oplus H(x_0^y) \oplus H(x_1^y) \) and

\[
\| (\langle 1 \rangle_Q Q_k^{\text{PhO}} |z, d, y\rangle \otimes |\phi_{H}\rangle \otimes |H\rangle \|^2 > 2 \cdot 2^{-\lambda c}.
\]

The latter is equivalent to

\[
\| (\langle 1 \rangle_Q Q_k^{H} |z, d, y\rangle \otimes |\phi_{H}\rangle \|^2 > 2 \cdot 2^{-\lambda c},
\]

as desired.

\[ \square \]

Now, suppose that we knew the following lemma were true. Then we will show that this yields an algorithm to extract a claw. Let \( I \subseteq [L] \), and \( b_I = (b_i : i \in I) \) where \( b_i \in \{0, 1, \perp\} \). Let \( \Pi^{I}_{y,I} \) be the projector (acting on register \( O \)) onto databases \( D_X \) such that: for \( i \in I \), if \( b_i = \perp \), then \( X \neq x_0^{y_i}, x_1^{y_i} \), otherwise \( X \) contains \( x_0^{y_i}, x_1^{y_i} \). Notice that for any \( y \) and \( I \subseteq \{0, 1\}^L \), the projective measurement \( M_{y,I} := \{ \Pi^{I}_{y,I} \}_{b_I} \) can be performed efficiently.

In the following lemma, let \( |\psi_{k}\rangle \) be the state of \( P^{\text{PhO}}(k) \) just before measurement of the output.

**Lemma 9.** There exists a non-negligible function \( g \), a sequence of sets \( J_{X} \subseteq K \), with \( |J_{X}| \geq g(\lambda) \), an efficiently generated sequence of sets \( \mathcal{S}_{\text{good},X} \) of pairs \((I, b_I)\), where \( I \subseteq [L] \), and \( b_I = (b_i : i \in I) \) where \( b_i \in \{0, 1, \perp\} \), such that the following holds for all \( \lambda \):
(a) $|S_{\text{good},\lambda}| = \text{poly}(\lambda)$.

(b) $b_i \neq 1 |I|$ for all $(I, b_i) \in S_{\text{good},\lambda}$.

(c) For $k \in J_\lambda$, there exists $(I, b_i) \in S_{\text{good},\lambda}$ such that $\|1 \langle 1 | Q C^{\text{PHO}}_k (\sum_y |y\rangle \otimes \Pi_{y,i}^{b_i}) |\psi_k\rangle\|^2 \geq g(\lambda)$.

With this lemma in hand, we can show that the following algorithm Ext succeeds at extracting a claw.

(i) On input $k$, run $P C^{\text{PHO}}(k)$ and obtain $z, d, y$.

(ii) Sample $(I, s_I) \leftarrow S_{\text{good},\lambda}$ and perform the measurement $M_{y,I}$ on the database register of the leftover state from step (i). If the outcome is $s_I$, then the database contains at least a pre-image of $y_i$ for some $i \in I$. Let $x_k^{y_i}$ be this pre-image.

(iii) Run $Q C^{\text{PHO}}_k$, and measure the database register. If the latter contains $x_k^{y_i}$ output $(x_0^{y_i}, x_1^{y_i})$.

Let $J_\lambda$ and $S_{\text{good},\lambda}$ be as in Lemma 9, and let $(I, b_i) \in S_{\text{good},\lambda}$ be such that it satisfies condition (c) in Lemma 9.

The above algorithm samples this pair with probability $\frac{1}{|S_{\text{good},\lambda}|} = \frac{1}{\text{poly}(\lambda)}$ in step (ii). There exists some negligible function $\text{negl}$, such that, for any $k \in J_\lambda$, up to $\text{negl}(\lambda)$ distance, we have

$$Q C^{\text{PHO}}_k (\sum_y |y\rangle \langle y| \otimes \Pi_{y,i}^{b_i}) |\psi_k\rangle = \alpha |0\rangle |\phi_0\rangle + \beta |1\rangle |\phi_1\rangle$$

for some $\alpha, \beta$ such that $|\beta|^2 \geq g(\lambda)$, and for some $|\phi_0\rangle, |\phi_1\rangle \in S_{\text{comp}} \cap S_{k, \text{no-claw}}$. To see that $|\phi_0\rangle, |\phi_1\rangle \in S_{\text{comp}}$, notice that $H_Q \otimes S_{\text{comp}}$ is invariant under the action of $\Pi_{y,i}$, and is invariant under the action of $M \otimes O$, where $M$ is any linear operator on $H_Q$, and $O = \text{StdDecomp}^{-1} \circ O \circ \text{StdDecomp}$. Moreover, (52) must be true for some $|\phi_0\rangle, |\phi_1\rangle \in S_{k, \text{no-claw}}$, otherwise the following would be an efficient algorithm to obtain a claw: run $P C^{\text{PHO}}(k)$; measure the $Y$ register to obtain $y$; perform the measurement $M_{y,I}$ to obtain $b_I$ followed by a standard basis measurement of the database register. With non-negligible distance the database would contain a claw for $y_i$ for some $i$.

For the same reason as above, since $|\psi_k\rangle \in S_{\text{comp}}$, it is also the case that $(\sum_y |y\rangle \langle y| \otimes \Pi_{y,i}^{b_i}) |\psi_k\rangle \in S_{\text{comp}}$. Hence, we can apply Lemma 8 to deduce that

$$||1 \langle 1 | Q \otimes (I - \Pi_{\text{valid}}) |\psi_k\rangle\|^2 \leq 2^{-\lambda'}.$$

for some $c > 0$.

Plugging (52) into the above, gives

$$|\beta|^2 \cdot ||(I - \Pi_{\text{valid}}) |\phi_1\rangle\|^2 \leq 2^{-\lambda'}.$$

Hence,

$$|\beta|^2 \cdot ||\Pi_{\text{valid}} |\phi_1\rangle\|^2 \geq |\beta|^2 - 2^{-\lambda'}.$$

Since $\beta$ is non-negligible, then there exists a polynomial $p$, such that $\beta(\lambda) > \frac{1}{p(\lambda)}$ for infinitely many $\lambda$. Let $\Lambda$ denote this set. Then, we have that, for $\lambda \in \Lambda$,

$$||\Pi_{\text{valid}} |\phi_1\rangle\|^2 \geq 1 - O(2^{-\lambda'})$$

for some $c' > 0$.

We can apply Corollary 2 to deduce that, for each $i$, $|\phi_i\rangle$ is $O(2^{-\lambda'})$-close to a state of the form

$$\sum_{z,d,y,x,e,w,D} \alpha_{z,d,y,x,e,w,D} |z, d, y, x, e, w, D \cup \{x_0^{y_i}\} + (-1)^d (x_0^{y_i} + x_1^{y_i}) |D \cup \{x_1^{y_i}\})\left(|D\cup\{x_0^{y_i}\}\right)$$

$$+ \sum_{z,d,y,x,e,w,D} \beta_{z,d,y,x,e,w,D} |z, d, y, x, e, w, D \cup \{x_0^{y_i}, x_1^{y_i}\}\left(|D\cup\{x_0^{y_i}, x_1^{y_i}\}\right),$$

(53)
for some $\alpha_{z,d,y,x,e,w,D}$ and $\beta_{z,d,y,x,e,w,D}$.

Now, notice that, for all but a negligible fraction of $k$, the weight on the second branch must be negligible in $\lambda$, otherwise this would yield an algorithm to recover a claw. Thus, for any polynomial $q$, for sufficiently large $\lambda \in \Lambda$ (and for all but negligible fraction of $k \in \mathcal{J}$), we have that $|\phi_1|$ is $\frac{1}{q(\lambda)}$-close to a state of the form

$$\sum_{z,d,y,x,e,w,D} \alpha_{z,d,y,x,e,w} |d, y, x, e, w\rangle \left( |D \cup \{x_0^y\} \rangle + (-1)^{m+d} |x_0^w + x_1^y\rangle |D \cup \{x_1^y\} \rangle \right). \quad (54)$$

Then, for any polynomial $q$, and for sufficiently large $\lambda \in \Lambda$, conditioned on Ext having sampled a pair $(I, s_I)$ satisfying condition (c) (which happens with probability $\frac{1}{\text{poly}(\lambda)}$), the algorithm Ext obtains a different pre-image at step (iii) (for the same $y_i$) than the one obtained at step (ii) with probability $1/2 - \frac{1}{q(\lambda)}$.

Thus, all in all, Ext outputs a claw with non-negligible probability.

To conclude the proof of Theorem 5, we are left to prove Lemma 9.

Proof of Lemma 9. For $i \in \{1, \ldots, L/2\}$, define $I_i := [i]$, and for $s \in \{0, 1\}$, define $b_{i,s} := (\perp i^{-1}, s)$.

By hypothesis, $P$ returns a valid equation and proof with non-negligible probability. This implies that there exists a polynomial $p$, and an infinite sized set $\Lambda$ such that the following holds for all $\lambda \in \Lambda$: there exists a set $\mathcal{J}_\lambda \subseteq \mathcal{K}$ with $\frac{|\mathcal{J}_\lambda|}{|\Lambda|} \geq 1/p(\lambda)$ such that, for $k \in \mathcal{J}_\lambda$, $\Pr[\text{Verifier accepts } k] \geq \frac{1}{p(\lambda)}$. For the rest of the proof, we restrict our attention to $\lambda \in \Lambda$ and $k \in \mathcal{J}_\lambda$.

Denote by $|\psi_k\rangle$ the state of $P_{\text{CPhO}}(k)$ just before measurement of the output. We argue that it must be the case that for any polynomial $q$, for all large enough $\lambda$, for all but a negligible fraction of $k \in \mathcal{J}_\lambda$,

$$\left\| |1\rangle \langle 1| Q_k^{\text{CPhO}} \left( \sum_y |y\rangle \langle y| \otimes \Pi_{\psi,\Pi_{y,I_{L/2}}}^{(1)} \right) |\psi_k\rangle \right\|^2 \leq \frac{1}{q(\lambda)}.$$

Suppose not, then there is some polynomial $q$, and a polynomial $r$ such that for infinitely many $\lambda \in \Lambda$, there exists some subset $\mathcal{J}_\lambda' \subseteq \mathcal{J}_\lambda$, with $|\mathcal{J}_\lambda'| \geq \frac{1}{p(\lambda)}|\mathcal{J}_\lambda| \geq \frac{1}{q(\lambda)}|\Lambda|$, such that

$$\left\| |1\rangle \langle 1| Q_k^{\text{CPhO}} \circ \text{DecompAll} \left( \sum_y |y\rangle \langle y| \otimes \Pi_{\psi,\Pi_{y,I_{L/2}}}^{(1)} \right) |\psi_k\rangle \right\|^2$$

$$> \frac{1}{q(\lambda)}$$

(55)

where the first equality follows from the fact that, for $\phi \in \mathcal{S}_{\text{comp}}$, $Q_k^{\text{CPhO}} \circ \text{DecompAll} = \text{DecompAll} \circ Q_k^{\text{CPhO}}$, and that $|1\rangle \langle 1|_Q$ commutes with $\text{DecompAll}$. Let $\Lambda' \subseteq \Lambda$ be this infinitely sized set of $\lambda$.

For convenience, denote $|\psi_k'\rangle := \left( \sum_y |y\rangle \langle y| \otimes \Pi_{\psi,\Pi_{y,I_{L/2}}}^{(1)} \right) |\psi_k\rangle$. Let $\Pi_{k,y,\text{claw}}$ denote a projection onto databases that contain a claw for $y_i$ for some $i$.

Then, for all $\lambda \in \Lambda'$, for all but a negligible fraction of $k \in \mathcal{J}_\lambda'$,

$$\left\| \left( \sum_y |y\rangle \langle y| \otimes \Pi_{k,y,\text{claw}} \right) |\psi_k'\rangle \right\|^2 = \text{negl}(\lambda).$$

Otherwise, we immediately obtain a strategy to recover a claw with non-negligible probability.

Thus, for all $\lambda \in \Lambda'$, for all but a negligible fraction of $k \in \mathcal{J}_\lambda'$, we have that the (unnormalized) state $|\psi_k'\rangle$ is negligibly close to a state $C : |\psi_k''\rangle$, for some $0 \leq C \leq 1$ and $|\psi_k''\rangle \in \mathcal{S}_{\text{claw}}$.

Applying Corollary 1, we have that, for all $\lambda \in \Lambda'$, for all but a negligible fraction of $k \in \mathcal{J}_\lambda'$,

$$\left\| \Pi_{\text{valid DecompAll}} |\psi_k'\rangle \right\|^2 \leq 2^{-L/2} + \text{negl}(\lambda). \quad (56)$$
Combining (55) and (56), we get that, for all $\lambda \in \Lambda'$, for all but a negligible fraction of $k \in J_\lambda'$,

$$\| \langle 1 \rangle_\mathbb{Q} (Q_k^{\text{PhO}} \circ (I - \Pi_{\text{valid}}) \circ \text{DecompAll}) \psi' \| \geq \frac{1}{q(\lambda)} (1 - 2^{L/2}) - \text{negl}(\lambda). \quad (57)$$

Now, we can write

$$\text{DecompAll} |\psi_k'\rangle = \frac{1}{\sqrt{2^{n+1}}} \sum_H |\psi_{k,H}\rangle |H\rangle,$$

for some $|\psi_{k,H}\rangle$, where here we are abusing notation slightly and equating a full database to the function it defines (also note that $2^{n+1}$ is the number of functions from $n$ bits to 1 bit).

Then, for all $\lambda \in \Lambda'$, for all but a negligible fraction of $k \in J_\lambda'$, we have

$$\frac{1}{2^{n+1}} \sum_H \| \langle 1 \rangle_\mathbb{Q} (Q_k^{\text{PhO}} \circ (I - \Pi_{\text{valid}}) |\psi_{k,H}\rangle |H\rangle \| \geq \frac{1}{q(\lambda)} (1 - 2^{L/2}) - \text{negl}(\lambda). \quad (58)$$

Then (58) implies that there exists $H$ such that:

$$\| \langle 1 \rangle_\mathbb{Q} (Q_k^{\text{PhO}} \circ (I - \Pi_{\text{valid}}) |\psi_{k,H}\rangle |H\rangle \| \geq \frac{1}{2q(\lambda)} (1 - 2^{L/2}). \quad (59)$$

We can further write $|\psi_{k,H}\rangle = \sum_{z,d,y} |z,d,y\rangle |\psi_{k,z,d,y,H}\rangle$. Then, all in all, we get that there exists $k$, $H$ and $z, d, y$ such that

$$\Pr[\text{acc} = 1 \wedge z_i \neq d_i, (x_0^i \oplus x_1^i) \oplus H(x_0^i) \oplus H(x_1^i) \text{ for some } i : \text{acc} \leftarrow Q_k^H |z, d, y\rangle \langle k |\psi_{k,z,d,y,H}\rangle] \geq \frac{1}{2q(\lambda)} (1 - 2^{L/2}).$$

This contradicts the hypothesis that $Q$ accepts invalid $z, d, y$'s only with exponentially small probability. Hence, we have shown that, for any polynomial $q$, for all large enough $\lambda \in \Lambda$, for all but a negligible fraction of $k \in J_\lambda$,

$$\| \langle 1 \rangle_\mathbb{Q} Q_k^{\text{PhO}} (\sum_y |y\rangle \langle y| \otimes \Pi_{y,L/2}^{\perp}) |\psi_k\rangle \| \leq \frac{1}{q(\lambda)}. \quad (60)$$

Let us restrict our attention to $\lambda, k$ as above. Recall that, for $i \in \{1, \ldots, L/2\}$, we have defined $I_i := [i]$, and for $s \in \{0, 1\}$, $b_{i,s} := (\perp^{i-1}, s)$.

We argue that there exists $i \in \left[\frac{L}{2}\right]$ and $s \in \{0, 1\}$ such that

$$\| \langle 1 \rangle_\mathbb{Q} Q_k^{\text{PhO}} (\sum_y |y\rangle \langle y| \otimes \Pi_{y,I_i}^{b_{i,s}}) |\psi\rangle \| \geq \frac{1}{p(\lambda)L^2}.$$

Suppose not, then a straightforward argument (which uses the definition of $p$ at the start of this proof) shows that it must be

$$\| \langle 1 \rangle_\mathbb{Q} Q_k^{\text{PhO}} (\sum_y |y\rangle \langle y| \otimes \Pi_{y,I_i/2}^{b_{i,s}}) |\psi\rangle \| \geq \frac{1}{p(\lambda)} \left(1 - \frac{1}{L^2}\right)^{\frac{L}{2}}, \quad (61)$$

which is greater than $\frac{1}{2p(\lambda)}$ for large enough $L$. This contradicts (60) for the choice $q = 2 \cdot p$.

\[ \square \]

References

[Aar09] Scott Aaronson. Quantum copy-protection and quantum money. In 2009 24th Annual IEEE Conference on Computational Complexity, pages 229–242. IEEE, 2009.
[AC02] Mark Adcock and Richard Cleve. A quantum goldreich-levin theorem with cryptographic applications. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 323–334. Springer, 2002.

[AGM21] Shweta Agrawal, Shafi Goldwasser, and Sailect Mossel. Deniable fully homomorphic encryption from learning with errors. In *Annual International Cryptology Conference*, pages 641–670. Springer, 2021.

[BB84] Charles H Bennett and Gilles Brassard. Quantum cryptography. In *Proc. IEEE Int. Conf. on Computers, Systems and Signal Processing, Bangalore, India*, pages 175–179, 1984.

[BCKM21] James Bartusek, Andrea Coladangelo, Dakshita Khurana, and Fermi Ma. One-way functions imply secure computation in a quantum world. In *Annual International Cryptology Conference*, pages 467–496. Springer, 2021.

[BCM+18] Zvika Brakerski, Paul Christiano, Urmila Mahadev, Umesh Vazirani, and Thomas Vidick. A cryptographic test of quantumness and certifiable randomness from a single quantum device. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 320–331. IEEE, 2018.

[BDGM20] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate io from homomorphic encryption schemes. 2020.

[BDS16] Shalev Ben-David and Or Sattath. Quantum tokens for digital signatures. *arXiv preprint arXiv:1609.09047*, 2016.

[BKVV20] Zvika Brakerski, Venkata Koppula, Umesh Vazirani, and Thomas Vidick. Simpler proofs of quantumness. In *15th Conference on the Theory of Quantum Computation, Communication and Cryptography*, 2020.

[BL20] Anne Broadbent and Sébastien Lord. Uncloneable quantum encryption via oracles. In *15th Conference on the Theory of Quantum Computation, Communication and Cryptography*, 2020.

[CDNO97] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In *Annual International Cryptology Conference*, pages 90–104. Springer, 1997.

[CK88] Claude Crépeau and Joe Kilian. Achieving oblivious transfer using weakened security assumptions. In *Proceedings 1988 29th Annual Symposium on Foundations of Computer Science*, pages 42–52. IEEE Computer Society, 1988.

[CLLZ21] Andrea Coladangelo, Jiahui Liu, Qipeng Liu, and Mark Zhandry. Hidden cosets and applications to uncloneable cryptography. In *Annual International Cryptology Conference*, pages 556–584. Springer, 2021.

[CPP20] Ran Canetti, Sunoo Park, and Oxana Poburinnaya. Fully deniable interactive encryption. In *Annual International Cryptology Conference*, pages 807–835. Springer, 2020.

[DQV+21] Lalita Devadas, Willy Quach, Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Succinct lwe sampling, random polynomials, and obfuscation. *Cryptology ePrint Archive*, 2021.

[Eke92] Artur K Ekert. Quantum cryptography and bell’s theorem. In *Quantum Measurements in Optics*, pages 413–418. Springer, 1992.

[GP21] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 736–749, 2021.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.
[Mah18] Urmila Mahadev. Classical verification of quantum computations. In 2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS), pages 259–267. IEEE, 2018.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In Proceedings of the forty-sixth annual ACM symposium on Theory of computing, pages 475–484, 2014.

[Wie83] Stephen Wiesner. Conjugate coding. SIGACT News, 15(1):78–88, January 1983.

[WW21] Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious lwe sampling. In Annual International Conference on the Theory and Applications of Cryptographic Techniques, pages 127–156. Springer, 2021.

[Zha19] Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, Advances in Cryptology – CRYPTO 2019, pages 239–268, Cham, 2019. Springer International Publishing.