Anonymous Public-Key Quantum Money and Quantum Voting

Alper Çakan* Vipul Goyal† Takashi Yamakawa‡

Abstract

Quantum information allows us to build *quantum money* schemes, where a bank can issue banknotes in the form of authenticatable quantum states that cannot be cloned or counterfeited: a user in possession of k banknotes cannot produce $k+1$ banknotes. Similar to paper banknotes, in existing quantum money schemes, a banknote consists of an unclonable quantum state and a classical serial number, signed by bank. Thus, they lack one of the most fundamental properties cryptographers look for in a currency scheme: privacy. In this work, we first further develop the formal definitions of privacy for quantum money schemes. Then, we construct the first public-key quantum money schemes that satisfy these security notions. Namely,

- Assuming existence of indistinguishability obfuscation and hardness of Learning with Errors, we construct a public-key quantum money scheme with anonymity against users *and* traceability by authorities.

Since it is a policy choice whether authorities should be able to track banknotes or not, we also construct an *untraceable* money scheme, where no one (not even the authorities) can track banknotes.

- Assuming existence of indistinguishability obfuscation and hardness of Learning with Errors, we construct a public-key quantum money scheme with untraceability.

Further, we show that the *no-cloning principle*, a result of quantum mechanics, allows us to construct schemes, with security guarantees that are classically impossible, for a seemingly unrelated application: *voting*!

- Assuming existence of indistinguishability obfuscation and hardness of Learning with Errors, we construct a universally verifiable quantum voting scheme with classical votes.

Finally, as a technical tool, we introduce the notion of *publicly rerandomizable encryption with strong correctness*, where no adversary is able to produce a malicious ciphertext and a malicious random tape such that the ciphertext before and after rerandomization (with the malicious tape) decrypts to different values! We believe this might be of independent interest.

- Assuming the (quantum) hardness of Learning with Errors, we construct a (post-quantum) classical *publicly rerandomizable encryption scheme with strong correctness*.

*Carnegie Mellon University. acakan@andrew.cmu.edu.

†NTT Research & Carnegie Mellon University. vipul@vipulgoyal.org

‡NTT Social Informatics Laboratories, Tokyo, Japan. takashi.yamakawa@ntt.com.

Contents

1 Introduction

The exotic nature of quantum mechanics allows us to build cryptographic primitives that were once unimaginable, or are outright impossible with classical information alone. For example, one of the most fundamental results of quantum mechanics, called *the no-cloning principle*, shows that arbitrary unknown quantum states cannot be cloned. This simple principle, which provably has no counterpart in classical world since classical information can always be copied, allowed cryptographers to build applications that are impossible in a classical world. Starting with the seminal work of Wiesner [Wie83] which introduced *quantum money*, a plethora of work built exciting primitives based on the no-cloning principle. The examples include more realistic version of quantum money called *public-key quantum money* [AC12, Zha19], where any user can verify a banknote on their own without going to the central bank, or even more advanced notions such as copy-protecting software/functionalities where a user that is given some number of copies of a software cannot create more copies of it[1] [Aar09, CLLZ21, ÇG23].

While quantum money is one of the most important notions in quantum cryptography, unfortunately existing schemes [AC12, Zha19] lack some of the most basic privacy and security guarantees that cryptographers look for in a currency scheme. In fact, in all known public-key quantum money schemes, a banknote consists of an unclonable quantum state and a classical serial number which is signed by the bank. However, this means that any party can track any banknote and learn when and where it was used simply by recording its serial number, meaning there is no privacy at all. For example, imagine a scenario where you pay a large sum of money to a merchant. If the merchant pools the data with other (adversarial) sources, it maybe feasible[2] for them to recover a history on many of these banknotes, potentially revealing your employer, clients and business partners, and even family members! Similarly an employer who pays you salary can potentially track where you travel to by collaborating with other sources and tracking, say, your spending at gas stations or restaurants. Indeed it is hard to imagine privacy in any aspect of your life, if all your spending can be traced. In fact, privacy and anonymity is the central focus in many cryptocurrency projects [mon, Zca].

This state of affairs leaves open the following natural question:

> *Is it possible to construct a publicly verifiable quantum money scheme with privacy guarantees?*

Let us emphasize that the above question is highly non-trivial for the following reason. To satisfy privacy guarantees, one needs to build a quantum money scheme where the users can create a new banknote so that the adversarial parties who have seen the banknote before will not be able to recognize it. However, such a task, while still challenging but doable for classical information (e.g. rerandomizable signatures [CL04]), seems to be at odds with the main point of quantum money: *unclonability*! While any user should be enabled to create new banknotes, somehow we also need to make sure that they cannot create $k+1$ valid banknotes if they started with k banknotes because otherwise they can increase the amount of money they have at wish!

Going beyond privacy concerns, one useful property of the existing quantum money schemes is that since any party can track a banknote, in particular *law enforcement* can also track a banknote. This brings us to our next natural question?

> *Is it possible to construct a publicly verifiable quantum money scheme with privacy guarantees against users, while still providing traceability for authorities?*

[1] Again, a classical software can always be copied, so this is impossible classically.

[2] In fact, even innocent amateur efforts that were built for fun, such as wheresgeorge.com, have been able to track millions of paper dollar banknotes all over the world.

We note that it is a political choice whether authorities should be able to track banknotes or not. Therefore, we also ask

> *Is it possible to construct a publicly verifiable quantum money scheme with privacy guarantees* against everyone*, including the bank/authorities?*

Finally, we observe an interesting connection between quantum money with privacy (which is impossible classically) and *voting*. In both cases, we care about privacy and a security notion relating to *non-increasibility*: in quantum money, users should not be able to increase their amount of money (in particular, given a single banknote, one should not be able to create two banknotes), and in voting, a single user should be able to vote once. Thus, we ask the following question:

> *Using quantum information, is it possible to construct voting schemes with advanced security guarantees that are not possible classically?*

1.1 Our Results

In this work, we answer all of these open questions affirmatively.

We construct the first public-key quantum money scheme with anonymity (against users) and traceability (by the authorities). For anonymity, we require that a malicious user will not be able distinguish a banknote it has seen before from a freshly minted banknote. For tracing, we require that no malicious user can produce a banknote with a particular tag (which is hidden inside the serial numbers, except to authorities) without being given a banknote with that tag to begin with - meaning that law enforcement can perfectly track banknotes.

Theorem 1 (Informal)**.** *Assuming the existence of indistinguishability obfuscation ($i\mathcal{O}$) and a* publicly rerandomizable encryption scheme with strong correctness and publicly testable ciphertexts*, there exists a public-key quantum money scheme with anonymity and traceability.*

In fact, in our model, we separate *authorities* into two completely independent entities: *the bank* (who mints the banknotes) and *the tracing authority*. Our scheme satisfies anonymity even against the bank, and satisfies unclonability (also called counterfeiting security) even against the tracing authority.

We note that previously, anonymous quantum money had been constructed only[3] in the private key setting [MS10, BS21, AMR20]. However, aside from the impracticality of the private key setting, the anonymity notion in the private-key setting is also less meaningful: Once we are at the central bank to verify a banknote, we might as well ask them to replace our banknote with a fresh one. The previous solutions are based on *Haar random states* or their computational version, *pseudorandom states*; however, in the private-key setting, a trivial solution based on quantum fully homomorphic encryption also exists, where we can just encrypt banknotes and use the homomorphic encryption scheme to rerandomize them.

Going further, we construct the first public-key quantum money scheme with anonymity against all parties (including the bank and the authorities). We call this notion *untraceability*.

Theorem 2 (Informal)**.** *Assuming the existence of $i\mathcal{O}$, a* publicly rerandomizable encryption scheme with strong correctness and publicly testable ciphertexts *and a non-interactive zero-knowledge (NIZK) argument system for NP, there exists a public-key quantum money scheme with* untraceability *in the common* random *string model.*

[3]Strictly speaking, [BS21] calls their model *almost-public*. They simply use an existing private-key quantum money scheme with pure states (more precisely, pseudorandom quantum states [JLS18]), and an alleged banknote is compared to user's existing banknotes to verify it. This requires that the user always has more money than she can receive. From our point of view, this is not a public-key scheme.

Finally, we construct the *first* voting scheme with universal verifiability (anyone in the world can verify any vote), privacy against all parties (including voting authority!) and uniqueness (i.e. no double-voting). We note that a voting scheme satisfying these three properties at the same time provably cannot exist in a classical voting scheme. Thus, the voting tokens of our scheme are quantum, but a cast vote is classical.

Theorem 3 (Informal)**.** *Assuming the existence of $i\mathcal{O}$, a* publicly rerandomizable encryption scheme with strong correctness and publicly testable ciphertexts *and a non-interactive zero-knowledge (NIZK) argument system for NP, there exists a quantum voting scheme with* universal verifiability *and classical votes, in the common* random *string model.*

As discussed, such a scheme cannot exist classically. Even using quantum voting tokens, ours is the first to achieve these guarantees: there is no previous work achieving universal verifiability (or classical votes).

We note that while $i\mathcal{O}$ is a strong assumption, all of our constructions above imply standard public-key quantum money, whose all existing constructions in the plain model also use $i\mathcal{O}$[4]. In fact, it is one of the key open questions in quantum cryptography to construct public-key quantum money without $i\mathcal{O}$. Thus, unless a major breakthrough is achieved, our $i\mathcal{O}$ assumption is necessary.

To achieve our results, we also introduce the notion of *publicly rerandomizable encryption with strong correctness*, where no adversary is able to produce a (malicious) ciphertext whose decryption result differs between before and after rerandomization (even with a maliciously chosen rerandomization randomness tape); and we construct such a scheme that is secure against quantum adversaries. Our schemes also satisfy public testability, where anyone can test whether a ciphertext is *bad* (in the above sense, where rerandomization can lead to decrypting to a different message), such that there *does not exist* a ciphertext that passes this test but decrypts to different values before/after rerandomization. We note that this notion is useful in constructions/proofs that use indistinguishability obfuscation, where merely computational hardness of finding bad ciphertexts would not be sufficient. Our results/constructions here are classical and we believe they might be of independent interest.

Theorem 4 (informal)**.** *Assuming hardness of Learning with Errors (LWE) [Reg09], there exits a* publicly rerandomizable encryption with publicly testable ciphertexts and strong correctness*.*

Thus, instantiating our constructions with our *publicly rerandomizable encryption scheme with strong correctness*, and the NIZK argument system with the LWE-based (post-quantum) construction of Peikert-Shiehian [PS19] (which is in the common random string model), all of our quantum money and quantum voting schemes can be based on $i\mathcal{O}$ and LWE.

2 Technical Overview

2.1 Definitional Work

We first review our model. In a quantum money scheme, we consider a bank that produces quantum banknotes that can be publicly verified, and the first security requirement is counterfeiting security (also called *unclonability*). For counterfeiting security, we require that any efficient adversary that has the public verification key and k banknotes produced by the bank, is not to able to produce $k+1$ valid banknotes, for any k (not a-priori bounded).

[4]We note that there are some candidate constructions based on non-standard ad-hoc assumptions

Prior work ([MS10, AMR20, BS21]) has introduced the notion of anonymity for quantum money schemes (albeit in the privately verifiable money setting). In their anonymity security game ([AMR20, BS21]), an adversary observes some banknotes, and later he is (depending on a challenge bit) either given those banknotes in the same order or in a permuted order; and the security requirement is that he cannot tell which case it is. We note that one downfall of private-key setting (apart from impracticality) is that achieving anonymity is not as interesting: once at the central bank to verify a banknote, we might as well also ask the central bank to replace our banknote with a fresh one. We first move onto the publicly verifiable banknote setting (dubbed *public-key quantum money* or PKQM for short), which is the ultimate goal in quantum money literature. Further, we introduce a new security notion called *fresh banknote indistinguishability security*. In this game, the adversary is given either the banknote it has seen before (in fact created itself in our model) or a freshly minted banknote. We require that the adversary cannot tell which case it is with probability better than $1/2 + \mathsf{negl}(\lambda)$. By a simple hybrid argument, one can see that this security notion implies the previous anonymity notions.

Going beyond anonymity, we also introduce the notion of *traceability* for quantum money. While the notion of *traceability* has a long history in cryptography, our work is the first to formally define it for quantum money. In this setting, we allow the bank/mint to include a *tag* value in each banknote. In the security game, we require that an adversary that obtains some banknotes from the bank with various tag values is not able to produce a valid banknote with a new tag value. In fact, we require that even the number of banknotes with each tag value cannot be increased. Thus, our security notion implies that all that an adversary can do, while still outputting valid banknotes, is to permute the banknotes given to it, and drop some of them! This allows the authorities to correctly track banknotes. We will always consider traceability (by the authorities) alongside anonymity (against users), which makes the problem highly non-trivial.

Before moving onto our final security model, *untraceability*, a couple of further remarks about our models is in order. First, we note that in our models, we consider two separate independent entities: the central bank/mint (who creates the banknotes) and a tracing authority. In real life, these entities can be two independent governmental organizations. We require that the bank can produce banknotes on its own (with no help from the tracing authority), while the tracing authority can trace banknotes on its own (with no help from the bank). In fact, we require that the anonymity holds against even the bank itself, who produced the banknotes in the first place! Further, we require that unclonability and tracing security also holds against the tracing authority.

Finally, we introduce the notion of *untraceability* for the first time. In this setting, we require that there are no entities, including the bank or the government, that is able to track banknotes. Here, we require that fresh banknote security applies to even a malicious bank that is allowed to choose the banknotes and even the verification key maliciously. Due to our strong anonymity model, this security notion follows easily in the trusted setup model[5]. However, ad-hoc/application-based trusted structured setup models are highly undesirable. Thus, we consider *untraceability* in the common *random* string model, which is a much weaker and more realistic assumption. We refer the reader to Section 6 for formal definitions.

2.2 Anonymous and Traceable Construction

Background on Subspace State Quantum Money [AC12, Zha19] The starting point of our construction is the subspace-state based public-key quantum money scheme of Aaronson -

[5]We can imagine a trusted setup that honestly creates the quantum money scheme, but deliberately forgets the tracing key and only outputs the minting key secretly/directly to the central bank. Since our anonymity security applies even against the bank, the security follows.

Christiano [AC12], which originally used classical ideal oracles but was also proven secure using $i\mathcal{O}$ by Zhandry [Zha19]. We note that, to date, subspace states (and their close relatives, such as subspace-coset states) are the only known public-key quantum money schemes with provable security based on standard assumptions.

A subspace-state is a state consisting of equal superposition over all elements of a subspace, that is, it is $\sum_{v\in A}\frac{1}{2^{n/4}}|v\rangle =: |A\rangle$, where A is a linear subspace of the vector space $\mathbb{F}_2^n$. The subspace-state quantum money construction works as follows. The bank's secret key is simply a (post-quantum) classical signature scheme. To mint a banknote, the bank simply samples a random subspace A of dimension $n/2$, and signs the obfuscated membership checking programs $i\mathcal{O}(A), i\mathcal{O}(A^\perp)$ for the subspace A and its orthogonal complement $A^\perp$. Then, the banknote is the state $|A\rangle$, together with the progras and the signature on them. Here, we can consider the string $i\mathcal{O}(A)||i\mathcal{O}(A^\perp)$ as the serial number of the banknote. To verify an alleged banknote $(|\psi\rangle, \mathsf{sn} = i\mathcal{O}(A)||i\mathcal{O}(A^\perp), \mathsf{sig})$, one verifies the signature sig on the serial number sn, then coherently runs the first program $i\mathcal{O}(A)$ on the state $|A\rangle$, verifying that the output is 1. After rewinding, we apply quantum Fourier transform (QFT) to the state $|\psi\rangle$, and this time coherently evaluate the second program $i\mathcal{O}(A^\perp)$ to verify that it outputs 1. [AC12] shows that this primal/orthogonal basis (more accurately, called *computational/Hadamard basis*) verification implements a projection onto $|A\rangle$, thus correctness follows. The counterfeiting security follows in two steps. First, they show that for a random subspace A of dimension $n/2$; no adversary can produce $|A\rangle \otimes |A\rangle$ given $|A\rangle$ and the membership checking programs. In the general setting where any number of banknotes is in circulation, they show that the security of the bank's classical signature scheme is sufficient to reduce to the single banknote setting. After all, by signature security, any supposed banknote produced by the adversary will have to use one of the original serial numbers produced by the bank, thus the adversary cannot create its own banknotes by sampling subspace states: it has to try and clone one of the subspace-states produced by the bank, which is only produced as a single copy. Thus the reduction to $1 \to 2$ setting follows.

Challenges for Anonymity Observe that the subspace-based quantum money scheme is trivially trackable. Each valid banknote contains a serial number and the bank's signature on the serial number. Thus, much like paper banknotes, simply recording serial numbers is sufficient to track banknotes: whenever we receive and validate a banknote, we can simply search for the serial number in our database.

An initial attempt to provide anonymity might be to draw from the classical literature, and for example to use *re-randomizable signatures* [CL04]. We first note that, most of the existing constructions depend on number-theoretic assumptions and to the best of our knowledge, there are no known *post-quantum* re-randomizable digital signature constructions in the plain model. Setting this issue aside, this initial attempt is still trivially broken, since while the signature is rerandomizable, the message (the serial number) will stay the same, which uniquely identifies the banknote. Going further, one might imagine a solution where the signature is rerandomized along with the message (serial number) inside, and we can imagine a scheme such that *somehow* the subspace state can still be verified with this updated serial number, to preserve correctness of the scheme. While this would be challenging to achieve, it is still not sufficient. Observe that only information needed to perfectly recognize a banknote is its *old* classical serial number. Imagine the following scenario. We (the adversary) have two banknotes $(|A_1\rangle, \mathsf{sn}_1, \mathsf{sig}_1), (|A_2\rangle, \mathsf{sn}_2, \mathsf{sg}_2)$. Later on, the challenger rerandomizes these banknotes, and gives us back one of the banknotes ($b \in \{1, 2\}$) as $(|\psi\rangle, \mathsf{sn}', \mathsf{sig}')$. We can simply perform the computational basis/Hadarmard basis verification on $|\psi\rangle$ using the old serial number sn_1 that we had recorded. Since this dual basis test implements a

projection onto the subspace state $|A_1\rangle$, our test will accept $|\psi\rangle$ with probability $\langle A_1|\psi\rangle$. In the case $b = 1$, this value equals $\langle A_1|A_1\rangle = 1$. In the case $b = 2$, this value will be equal to $\langle A_1|A_2\rangle$, which is exponentially small in expectation. Thus, using this simple test, we can recognize which banknote we got back almost perfectly! In fact, this test can performed *coherently* by the Gentle Measurement Lemma [Aar16], so that we do not even damage the banknote at all while doing the test!

Thus, we can see that the only way to provide anonymity is to actually change (rerandomize) the quantum state itself too! Here, a viable candidate could be using fully homomorphic encryption for quantum messages to rerandomize our quantum states, since homomorphism is known to be closely related to rerandomization in classical cryptography. However, this would completely forego the most important property, public-verifiability, since the users cannot verify encrypted banknotes. Another idea could be to try and sign the quantum parts of the banknotes, and try to construct a signature and message rerandomizable signature scheme for quantum messages. However, it is known that signing quantum messages is impossible [AGM21].

This brings us to the fundamental challenge in this task. We need to build a scheme where the users can create essentially a new quantum state, so that the adversarial parties who have seen the banknote before will not be able recognize it. However, at the same time we also need to make sure that these states are unclonable and that users cannot create more quantum states than what they already have!

Solution Step 1: Rerandomizing Quantum States First step of our solution is to build a new quantum money scheme (without anonymity), though the quantum state part of our scheme will still be essentially subspace states. In our scheme, the banknote will consist of a serial number id, and the quantum state will be a superposition over the elements of the subspace obtained by pseudorandomly applying a *rotation-reflection-scaling* to the canonical $n/2$-dimensional subspace $\mathsf{A}_{\mathsf{Can}} = \mathsf{Span}(e_1, \ldots, e_{n/2}) \subseteq \mathbb{F}_2^n$ (i.e. it is the subspace consisting of all vectors whose last $n/2$ components are 0). More formally, the quantum part of the banknote will be the state $|T \cdot \mathsf{A}_{\mathsf{Can}}\rangle = \sum_{v \in \mathsf{A}_{\mathsf{Can}}} \frac{1}{2^{\lambda/4}} |T(v)\rangle$, where T is the full rank linear mapping $\mathbb{F}_2^\lambda \to \mathbb{F}_2^\lambda$ sampled using the randomness $F(K, id)$ where F is a pseudorandom function (PRF) and id is the serial number of the banknote. The bank's secret key will simply be the PRF key K. Finally, the public key will be an obfuscated program P that takes as input a vector $v \in \mathbb{F}_2^\lambda$, a bit $b \in \{0, 1\}$ and a serial number id. The obfuscated program first recovers the hidden full rank linear map T_{id} using the hardcoded PRF key K and the input id. For $b = 0$, the program computes $w = T^{-1}(v)$, and verifies if its last $n/2$ entries are 0 (that is, it checks $w \in \mathsf{A}_{\mathsf{Can}}$). For $b = 1$, the program computes $w' = T^{\mathsf{T}}(v)$ and verifies if $w' \in \mathsf{A}_{\mathsf{Can}}^{\perp}$. We show that testing $P(\cdot, b = 0, id)$ in superposition, then applying QFT and testing $P(\cdot, b = 1, id)$ in superposition implements a projection onto the state $|T_{id} \cdot \mathsf{A}_{\mathsf{Can}}\rangle$, thus correctness is satisfied.

Now, we move onto adding anonymity to our construction. First observation is that the set of full rank linear mappings on $\mathbb{F}_2^n$ has a *nice* structure (under composition/multiplication): it is the general linear group $\mathsf{GL}(n, \mathbb{F}_2)$. This means that we can re-randomize a (full-rank) map T by simply multiplying with a fresh random (full-rank) map T', due to group rerandomization property. However, we cannot allow the users to rerandomize using any T', because the banknote would lose its consistency with the serial number-quantum state mapping induced by $F(K, \cdot)$, since the user will not be able to come up with the new serial number. In fact, in general, given T, T', id, it might not be possible to come up with a new serial number id' such that the randomness $F(K, id')$ gives us the full rank map T'' satisfying $T'' = T' \cdot T$. To solve this, we include an obfuscated program $\mathsf{PReRand}$ as part of the public key that allows the user to renrandomize the banknote consistently.

On input id, the program will first re-randomize id using some $\mathsf{IdReRandomizeAlgorithm}$ to obtain id', and then generate $T_{id'}$ using the randomness $F(K, id')$. Finally, it will output a canonical representation of the map $T^* = T' \cdot T^{-1}$, along with id'. Given T^*, the user can *convert* its banknote $|T \cdot \mathsf{A}_{\mathsf{Can}}\rangle$ into $|T' \cdot \mathsf{A}_{\mathsf{Can}}\rangle$ by applying the mapping T^* coherently/in-superposition. Note applying T^* coherently is possible since (i) T^* is a bijection and (ii) it is efficiently implementable in both directions $T^*, (T^*)^{-1}$ given the description T^*. Thus, we achieve a meaningful rerandomization algorithm while preserving correctness after rerandomization.

Solution Step 2: Non-intersecting/Traceable Rerandomization Cones Since the quantum part of our scheme are subspace states, to prove the unclonability of our scheme, we will need to reduce to the unclonability of subspace states, which are only 1-$\mathsf{copy} \to$ 2-copy unclonable. Now, consider the *chain* (or more accurately, the tree/cone) of rerandomizations for a serial number id that consists of the initial serial number id, and then (each possible) rerandomization of id, and then rerandomizations of those values and so on. Let us denote this as C_{id}. Now consider these cones for each of the k initial banknotes that the adversary obtains, $\mathsf{C}_{id_1}, \ldots, \mathsf{C}_{id_k}$. Now, observe that if any of these cones intersect at any point, say C_{id_i} and C_{id_j} at id^*, in a way that an efficient adversary can find, then the adversary can simply emulate the path (i.e. the malicious rerandomization random tape choices) that leads to this intersection on the banknotes id_i, id_j to obtain *two* copies of the exact same state $|T_{id^*} \cdot \mathsf{A}_{\mathsf{Can}}\rangle$! However, we need to reduce to the 1-$\mathsf{copy} \to$ 2-copy unclonability of our quantum states. Thus, we need to design a serial number re-randomization algorithm where the adversaries cannot find such intersections of the rerandomization cones.

In fact going further, we observe that while *non-intersecting* (to efficient adversaries) rerandomization cones is sufficient to prevent identical-copy attacks, it is not sufficient to actually give a reduction to $1 \to 2$ unclonability and prove security. Observe that to be able to achieve a reduction to $1 \to 2$ unclonability, the reduction algorithm will need to place the subspace state $|A\rangle$ it receives from its challenger at some index $\in [k]$, and once the adversary produces $k+1$ banknotes, the reduction will need to somehow extract two identical copies of $|A\rangle$ using these states. This means the reduction itself actually needs to mount a partial tracing attack, so that it can track back the forged banknotes produced by the adversary to find two that are *rooted* at the same initial subspace state, and undo the rerandomizations and convert these forged banknotes to two copies of $|A\rangle$.

To resolve this issue, we show that a *rerandomizable encryption* scheme with *strong rerandomization correctness* gives us exactly what we want. In our money scheme, the serial numbers will be rerandomizable public-key encryption (RPKE) ciphertexts (that encrypt *tags*) and $\mathsf{IdReRandomizeAlgorithm}$ will simply be the rerandomization algorithm of the RPKE scheme (and the rest of the money scheme is as described above). By the rerandomization security of the RPKE scheme, we will be able to prove anonymity easily. Further, the notion of *strong rerandomization correctness* gives us the *tracable* non-intersecting cone property we needed to achieve unclonability. First, let us define *strong rerandomization correctnes*, which is a notion we introduce for the first time and we believe it might be of independent interest. For this notion, we require that for any (malicious) ciphertext string ct and (malicious) random tape r chosen by an efficient adversary, we have $\mathsf{Dec}(sk, ct) = \mathsf{Dec}(sk, \mathsf{ReRand}(pk, ct; r))$. Applying this transitively, we can see that (efficient) cones of any two serial numbers will not intersect. Finally, the decryption algorithm of the RPKE scheme gives us exactly the traceability we wanted from our rerandomization cones to do our unclonability reduction! Thus, we are able to prove unclonability and anonymity security using a *rerandomizable encryption* scheme with *strong rerandomization correctness*. For technical reasons relating to use of $i\mathcal{O}$, we actually require *strong correctness with public testing* - see Section 2.5.

2.3 Untraceable Construction

Our starting point is our anonymous and traceable quantum money scheme. The main challenging point in this setting is the fact that all parties (users and the bank) are mutually distrustful. The first issue is that, in the untraceability game, the adversary gets to choose all the values including the public key of the scheme (which includes $\mathsf{PReRand}$). Thus, the adversary can simply use the corresponding decryption key to decrypt the serial numbers and obtain the hidden tags it placed inside them, and hence track the banknotes. Further, even if we hypothetically trusted the adversary to not use the decryption key of the RPKE scheme to track banknotes, it can still create a malicious obfuscated program $\mathsf{PReRand}'$ so that the new banknote state we end up with after rerandomization includes a hidden signal that depends on the old serial number! Note that we cannot trust the user to rerandomize their serial numbers themselves with no checks either, since that would break down our unclonability argument.

Solution Step 1: Proofs of Projectiveness Our solution is to include, as part of the verification key, a non-interactive zero-knowledge (NIZK) argument showing that the money scheme (with its maliciously chosen keys) is still *projective*. More formally, the NIZK argument will prove that there exists a (hidden) serial number - quantum state mapping M such that computational/Hadamard basis verification using $\mathsf{PMem}(\cdot, b = 0, id)/\mathsf{PMem}(\cdot, b = 1, id)$ implements a projection onto the state $|M(id)\rangle$. Thus, any user (as part of banknote verification) can rerandomize a banknote and test again using the new id', and once the test passes, the user is assured that he has the state $|M(id')\rangle$. If the test does not pass, either the initial banknote was not valid at all (i.e. a counterfeiting attack) or it was designed to be tracked (which also makes it invalid)! With the proof of projective, we are now assured that if the adversary really does not have the decryption key for RPKE, then $|M(id')\rangle$ is indistinguishable from $|M(id^*)\rangle$ where id^* is a fresh ciphertext, since the indistinguishability would follow from indistinguishability of id' versus id^*, so it still applies even though the adversary knows M. Finally, the zero knowledge property is needed since revealing M to the users would allow them to clone states trivially. We note that post-quantum NIZKs based on LWE can be constructed in the common random string model [PS19].

Solution Step 2: Truly Random Public Keys However, there is still a major issue: obviously we cannot trust the adversary to not create a decryption key that corresponds to the public key. To solve this, we will simply use a truly random string as the public rerandomization key. However, we seem to be at a dead-end: we need to prove, at the same time, that no one can track banknotes, while the unclonability proof (as we discussed) needs to mount a partial tracking attack. Our solution is to construct a rerandomizable encryption scheme with *strong rerandomization correctness with testing* that also satisfies (i) *pseudorandom public (encryption) keys*, (ii) *simulatable testing keys* (Section 2.5) and (iii) *statistical rerandomization security for truly uniformly random public keys*. This allows us to do the following: In the actual construction, the users use (part of) the common random string (CRS) as the public key of the RPKE scheme. For the unclonability proof, we rely on the pseudorandom public keys property of the RPKE scheme: The reduction simulates the cloning adversary while planting an actual public key of our RPKE scheme (which the reduction samples together with the corresponding decryption key sk) as the last part of the CRS, instead of a truly random string. Due to the pseudorandom public keys property of RPKE, this means that the adversary still succeeds, while the reduction can also mount the partial tracking attack using sk as discussed before. Finally, we use the property (ii) because we cannot embed the ciphertext testing key inside the common uniformly random string, since such a key obviously cannot be pseudorandom. Thus, in the actual construction we use the simulated ciphertext testing key.

For anonymity, we will be able to rely on property (iii) since we are rerandomizing the ciphertexts using a truly random string. More precisely, as discussed above, we first rely on the soundness of the NIZK argument system to show that if the adversary succeeds, there exists a mapping M as defined above. However, for the challenger to start outputting $|M(id')\rangle$ directly instead of running adversary's malicious rerandomization program, we need to recover M. At this point, we switch to an unbounded challenger that does recover M by trying all possibilities. The final step is to replace $|M(id')\rangle$ with $|M(id^*)\rangle$ relying on the rerandomization security of RPKE. Fortunately, by property (iii), we can indeed do so even with an unbounded challenger/reduction. Note that in the RPKE we end up constructing (Section 5), we will only have computational rerandomization for honest public keys (which comes with a secret key), but we do have statistical rerandomization when a truly random string is used as the public key.

2.4 Quantum Voting

In a voting scheme, we would like to allow users to vote for various candidates, while ensuring that their choices are private (*privacy)*, and also ensuring that they cannot vote twice (*uniqueness*). Finally, we would also like to ensure that the final tallies are calculated correctly. The best possible notion for the latter is the so-called *universal verifiability* setting, where voters post their votes on a public bulletin board, and anyone in the world is able to verify all the posted votes (and hence the final tallies). We note that universal verifiability cannot be achieved by voting schemes such as the ones based on homomorphic encryption, since the votes will need to be decrypted by the owner of the secret key, and revealing this secret key would trivially defeat privacy of votes. In fact, the combination of all three of these properties, (i) universal verifiability, (ii) privacy, (iii) uniqueness, cannot be achieved by any fully classical voting scheme. If the casting of a particular voting token is indistinguishable (to everyone) from casting of a fresh voting token, then an adversary can simply vote twice as follows. It creates copies of its initial voting token, rerandomizes these copies, and votes twice (or more) using all these tokens. The bulletin board cannot reject the second vote, since due to privacy, from its viewpoint, these votes are indistinguishable from someone else's (who has not voted yet) vote. Note that the impossibility persists even in trusted setup and ideal oracle models - it only (and crucially) relies on cloning the voting tokens (i.e. rewinding an adversary). However, this means that the attack does not work if the voting tokens are (unclonable) quantum states. This brings us to our construction. Our construction will be similar to our untraceable quantum money construction. A voting token will be the tuple of states $|T_{ct,1} \cdot \mathsf{A}_{\mathsf{Can}}\rangle \otimes \cdots \otimes |T_{ct,2\cdot\lambda} \cdot \mathsf{A}_{\mathsf{Can}}\rangle$ (together with the serial number ct) where $(T_{ct,1}, \ldots, T_{ct,2\cdot\lambda})$ is a tuple of full rank linear maps sampled using the randomness $F(K, ct)$. After rerandomizing these states (same as in our money scheme, generalized to $2\cdot\lambda$ states), to vote for a particular candidate $c \in [\lambda]$, the user samples a random tag[6] $r \leftarrow \{0,1\}^\lambda$ and measures the i-th state in the computational basis if the i-th bit $(c||r)_i = 0$, and in the Hadamard basis (i.e. QFT-and-measure) if $(c||r)_i = 1$. The measurement results $v_1, \ldots, v_{2\cdot\lambda}$ along with the rerandomized serial number ct' and c, r will be the user's (classical) vote for the candidate c. Anyone can verify a vote by simply checking $\mathsf{PMem}(vk, c||r, v_1|| \ldots ||v_{2\cdot\lambda}) = 1$. Thus, our scheme satisfies universal verifiability.

Our scheme also satisfies privacy, against all parties, including the government/voting authority that creates the voting tokens (and it is allowed to create the tokens and keys maliciously). The privacy proof follows in the same way as the untraceablity proof of our quantum money scheme. Finally, uniqueness. For this property, we rely on the so-called *direct product hardness* of subspace states [BDS23], which says that no efficient adversary, given $|A\rangle$, can produce vectors v, w such

[6] A serial number for the cast votes that the voter creates

that $v \in A$ (computational basis) and $w \in A^{\perp}$ (Hadamard basis). Through the same arguments as in the unclonability proof of our untraceable money scheme (where reduced unclonability of our scheme to $1 \to 2$ unclonability of subspace states), we will be able to reduce the uniqueness security of our scheme to the direct product hardness of subspace states. In particular, we will show that no efficient adversary, given k voting tokens (modeling a group of malicious voters cooperating) can output $k+1$ valid cast votes, each with different tags r. We note that this in particular (with $k=0$) means that non-eligible parties (i.e. parties who did not receive a token) will not be able vote at all. Thus, during tallying/verification, all one needs to do is (i) verify each submitted cast vote V using the public key, (ii) check that the exact same tag r has not appeared before[7].

Finally, we re-emphasize that the cast votes in our scheme are classical, meaning that the voters will not need to transmit quantum states to vote.

We refer the reader to Section 9 for the full construction and the proofs.

2.5 Rerandomizable Encryption with Strong Correctness

In this section, we introduce the notion of rerandomizable public-key encryption with *strong rerandomization correctness* and discuss our solution. For rerandomizable encryption, the most common security notion is the following. In the security game, the challenger samples a random a bit $b \in \{0,1\}$, and if $b=0$, the adversary (who has pk) is given an honest encryption $ct \leftarrow \mathsf{RPKE.Enc}(pk, m)$ of a message m of its choice, and also an honest rerandomization $ct' \leftarrow \mathsf{RPKE.ReRand}(pk, ct)$ of ct. If $b=1$, the adversary is given the honest encryption $ct \leftarrow \mathsf{RPKE.Enc}(pk, m)$, and an independent fresh encryption $ct' \leftarrow \mathsf{RPKE.Enc}(pk, m)$. We require that no adversary can predict the challenge case b with probability better than $1/2 + \mathsf{negl}(\lambda)$. A strengthening of this notion requires the same for any string ct (rather than for honestly randomly sampled ciphertexts).

Definitions However, as discussed previously, this rerandomization guarantee alone (while sufficient for anonymity) is not sufficient for our unclonability proofs. On top of this guarantee, we also require that an efficient adversary cannot come up with a malicious ciphertext and a malicious random tape for $\mathsf{RPKE.ReRand}$ that leads to different decryptions before and after rerandomization. We say that such a scheme satisfies (I) *strong (rerandomization) correctness*. While this notion is interesting in its own right, it will not be sufficient in $i\mathcal{O}$-based constructions. While efficient adversaries cannot come up with *bad* ciphertexts, note that such ciphertexts might still exist, whereas in $i\mathcal{O}$-based security proofs one needs to prove that two circuits (e.g., the initial circuit and the one that rejects bad ciphertexts) are completely equivalent - as opposed to just proving the hardness of finding an input where the circuits have different output. Thus, we also introduce the notion of (II) *strong correctness with public testing*, where the public key includes a testing key. We require that there *does not exist* a malicious ciphertext ct and a malicious random tape s such that

- $\mathsf{RPKE.Test}(pk, ct) = \mathsf{GOOD}$.
- $\mathsf{RPKE.Dec}(sk, ct) \neq \mathsf{RPKE.Dec}(sk, \mathsf{RPKE.ReRand}(pk, ct); s)$

Since the testing key is public, an encryption scheme satisfying this notion can be utilized even with $i\mathcal{O}$-based proofs, since the original obfuscated circuit in the actual construction can already test and reject bad ciphertexts.

We introduce two more stronger notions. First, we consider (III) *strong correctness with simulatable (public) testing keys*. This notion is the same as our previous notion of public testing, with the addition that we split public key into two parts (encryption/rerandomization key and testing

[7] If it has, only the first counts.

key), and we require that the testing key can be simulated (indistinguishable from real testing keys) using only the public encryption key (in fact, our constructions will need only the security parameter 1^λ to simulate). Such a notion is useful in our untraceable quantum money and voting schemes, since we will not be able to include the testing key in the truly uniformly random common string (CRS). We finally introduce (IV) *strong correctness with simulatable all-accept testing keys.* In this case, we require that the scheme has a simulatable testing key such that it accepts *all* strings ct as good. This notion allows us to construct schemes (even with $i\mathcal{O}$-based proofs) that do not use any ciphertext testing (or testing key) at all - at least in the original construction. Since the simulated keys always accept, the original constructions do not need to test, whereas in the proof (in an indistinguishable manner) we can start testing ciphertexts and reject bad ones, thus rely on the strong correctness guarantees discussed above.

We note that there are no existing post-quantum rerandomizable encryption schemes that satisfies any of these four security notions (even the simplest one, plain strong correctness (I)). We also note that approaches like fully homomorphic encryption or bootstrapping does not work, since they use honest randomness to *refresh* cipertexts that have been honestly sampled and have been honestly computed on - no maliciousness is allowed in any part. Bad ciphertexts for such schemes can easily be produced efficiently.

In this work, we construct various schemes that satisfy all of the notions defined above, based solely on hardness of Learning with Errors [Reg09].

Background on Regev's PKE Our starting point is the well-known Regev's public-key encryption scheme [Reg09], based on the Learning with Errors (LWE) problem. Let $n(\lambda), m(\lambda)$ denote the vector space dimension and the number of samples respectively, $\chi(\lambda)$ denote the error distribution, $B(\lambda)$ denote the bounds for the values in the support of $\chi(\lambda)$ (i.e. $\chi(\lambda) \subseteq [-B, B]$) and $q(\lambda)$ denote the modulus. The scheme is as follows, where all operations are performed modulo q (i.e., in $\mathbb{Z}_q$). The public key is (A, y) and the secret key is s, where $A \leftarrow \mathbb{Z}_q^{n\times m}$, $s \leftarrow \mathbb{Z}_q^n$, $e \leftarrow \chi^m$ and $y^\mathsf{T} = s^\mathsf{T} \cdot A + e^\mathsf{T}$. To encrypt a bit $b \in \{0,1\}$, we sample a random *binary* vector $r \leftarrow \{0,1\}^m$ and output $(A \cdot r, y^\mathsf{T} \cdot r + b \cdot \lfloor \frac{q}{2} \rfloor)$. To rerandomize a ciphertext $ct = (\vec{a}, c)$, we simply add a fresh encryption of 0 to it, which gives us $(\vec{a} + A \cdot r', c + y^\mathsf{T} \cdot r')$. Using LWE and leftover hash lemma [ILL89], one can show that rerandomization security is satisfied for *any* ct. Finally, to decrypt $ct = (\vec{a}, c)$, we compute $|c - s^\mathsf{T} \cdot \vec{a}|$ and decrypt to 0 if the value is less than $q/4$, and decrypt to 1 otherwise.

Thwarting Malicious Rerandomization Attacks with Hidden Shifts While Regev's PKE satisfies rerandomization security, it does not satisfy even the plain strong correctness property (I). Note that even honestly rerandomizing an honest ciphertext roughly $O(q/B)$ times is likely to produce a *bad* ciphertext (decrypting to a different value after one more rerandomization). Thus, as our first step, we move to the *subexponential modulus-to-noise ratio* LWE regime where $q = \mathsf{subexp}(\lambda)$, and $B = \mathsf{poly}(\lambda)$. Then, no polynomial number of honest rerandomizations on honest ciphertexts will produce bad ciphertexts. However, the adversary can still easily produce a malicious bad ciphertext: for example, consider $(\vec{a}, q/4 - 2 - k)$ where $\vec{a}$ is any vector and k is a random value in $[-m \cdot B, m \cdot B]$. This ciphertext will decrypt to 0 initially, whereas after rerandomizing it will decrypt to 1 with probability roughly $1/k = 1/\mathsf{poly}(\lambda)$.

Our solution is to use *hidden shifts*. We update our scheme so that the protected decryption key includes a hidden shift value L, selected uniformly at random from $[0, q/16]$. To decrypt $ct = (\vec{a}, c)$,

we now test $|c - s^\mathsf{T} \cdot \vec{a} - L| < q/4$. Now, we prove that a ciphertext $(\vec{a}, c)$ can be bad only if

$$\text{Turns } 0 \to 1 : c - s^\mathsf{T} \cdot \vec{a} \in [q/4 - m \cdot B + L, q/4 - 1 + L] \cup [3q/4 + 1 + L, 3q/4 + m \cdot B + L]$$
$$\text{Turns } 1 \to 0 : c - s^\mathsf{T} \cdot \vec{a} \in [q/4 + L, q/4 + m \cdot B - 1 + L] \cup [3q/4 - m \cdot B + 1 + L, 3q/4 + L]$$

Since L is unpredictable, no adversary (even given the secret s) will be able to produce a bad ciphertext except with subexponentially small probability.

Testing Ciphertexts and Obfuscating Evasive Programs As discussed above, in most applications, we will also need to be able to publicly test if a given ciphertext is bad, that is, if it can decrypt to different values before and after (malicious) rerandomization. By above, (once s and $\vec{a}$ are fixed) the set of bad ciphertexts $(\vec{a}, c)$ is actually *evasive* - the program that checks for bad ciphertexts would output 0 (meaning, *not bad*) almost everywhere, except for a set of size $O(m \cdot B) = \mathsf{poly}(\lambda)$. This means that if we can obfuscate this program, we would be done. Unfortunately, obfuscating evasive programs in general is impossible [BBC+14]. However, observe that in our case, the set of accepting points actually has a nice structure: they are consecutive! As a result, we actually only need point function obfuscation. We can simply use the point function obfuscation for the point $q/4 + L$. To test a ciphertext $(\vec{a}, c)$, we can simply run the point function with values $c - s^\mathsf{T} \cdot \vec{a} + m \cdot B, c - s^\mathsf{T} \cdot \vec{a} + m \cdot B + 1, \ldots, c - s^\mathsf{T} \cdot \vec{a} - m \cdot B + 1$ for the lower two ranges, and then similarly with $c - s^\mathsf{T} \cdot \vec{a} + m \cdot B - 1 - q/2, \ldots, c - s^\mathsf{T} \cdot \vec{a} - m \cdot B - q/2$ for the higher two ranges. Note that this is only $\leq 4m \cdot B = \mathsf{poly}(\lambda)$ tests. The only remaining issue is that we are actually using the secret decryption key s for testing. Our solution is that, instead of point function obfuscation, above we use *compute-and-compare (CC) obfuscation* [WZ17] (which can be constructed from LWE) and obfuscate the compute-and-compare function $f_s(x) =^? \mathsf{TARGET}$ where $f_s((\vec{a}, c)) = c - s^\mathsf{T} a$ and the target point is $\mathsf{TARGET} = q/4 + L$. Since $q/4 + L$ is subexponentially unpredictable, by CC obfuscation guarantee, our public testing program will hide both L and s!

Pseudorandom Public Keys and Simulatable Testing Keys For our untraceable schemes, we need that the public keys of our scheme be pseudorandom. Our scheme described above can already be shown to have pseudorandom encryption keys (A, y^T) by LWE. However, the ciphertext testing key is an obfuscated evasive program which outputs 0 almost everywhere. Thus, it obviously is not a pseudorandom string. However, due to CC obfuscation security, we can actually simulate our testing key by obfuscating the program that always outputs 0! This means that we can simply use the encryption key (which is pseudorandom) as our public encryption key, while users can simulate the ciphertext testing key on their own.

3 Related Work

Aaroson-Christiano [AC12] constructs public-key quantum money from subspace states in the ideal classical oracle model. Zhandry [Zha19] later showed that the same construction is still secure when instantiated with indistinguishability obfuscation.

In the private-key setting; Ji et al [JLS18] shows that pseudorandom quantum states can be constructed from one-way functions. Since pseudorandom quantum states are indistinguishable from Haar random states, which gives identical/pure state quantum money in the private-key setting, they automatically satisfy anonymity. Alagic et al [AMR20] constructs anonymous quantum money in private key setting with a stateful (updated each time a new banknote is issued) bank. Behera and Sattath [BS21] gave a construction where banknotes are pseudorandom states and a received banknote is *compared* to existing banknotes to verify.

Okamoto et al [OST08] propose a quantum voting scheme (based conjugate coding) with quantum votes that satisfies privacy and uniqueness, but not universal verifiability (only the voting token creator can verify votes). There are also various classical voting schemes based on various assumptions [Adi08, FOO93], however, as discussed, they cannot satisfy all the desired properties at the same time.

4 Preliminaries

4.1 Notation

Unless otherwise specified, adversaries are stateful quantum polynomial time (QPT) and our cryptographic assumptions are implicitly post-quantum. We write $\leftarrow$ to denote a random sampling from some distribution or uniform sampling from a set.

4.2 Digital Signature Schemes

In this section we introduce the basic definitions of signatures schemes.

Definition 1. *A digital signature scheme with message space $\mathcal{M}$ consists of the following algorithms that satisfy the correctness and security guarantees below.*

- $\mathsf{Setup}(1^\lambda)$: *Outputs a signing key sk and a verification key vk.*
- $\mathsf{Sign}(sk, m)$: *Takes the signing key sk, returns a signature for m.*
- $\mathsf{Verify}(vk, m, s)$: *Takes the public verification key vk, a message m and supposed signature s for m, outputs 1 if s is a valid signature for m.*

Correctness *We require the following for all messages $m \in \mathcal{M}$.*

$$\Pr\left[\mathsf{Verify}(vk, m, s) = 1 : \begin{array}{c} sk, vk \leftarrow \mathsf{Setup}(1^\lambda) \\ s \leftarrow \mathsf{Sign}(sk, m) \end{array}\right] = 1.$$

Adaptive existential-unforgability security under chosen message attack (EUF-CMA) *Any QPT adversary $\mathcal{A}$ with* classical *access to the signing oracle has negligible advantage in the following game.*

1. *Challenger samples the keys $sk, vk \leftarrow \mathsf{Setup}(1)$.*
2. *$\mathcal{A}$ receives vk, interacts with the signing oracle by sending classical messages and receiving the corresponding signatures.*
3. *$\mathcal{A}$ outputs a message m that it has not queried the oracle with and a forged signature s for m.*
4. *The challenger outputs 1 if and only if $\mathsf{Ver}(vk, m, s) = 1$.*

If $\mathcal{A}$ outputs the message m before the challenger samples the keys, we call it selective EUF-CMA *security.*

4.3 Puncturable Pseudorandom Functions

In this section, we recall puncturable pseudorandom functions.

Definition 2 ([SW14])**.** *A puncturable pseudorandom function (PRF) is a family of functions* $\{F : \{0,1\}^{c(\lambda)} \times \{0,1\}^{m(\lambda)} \to \{0,1\}^{n(\lambda)}\}_{\lambda \in \mathbb{N}^+}$ *with the following efficient algorithms.*

- $F.\mathsf{Setup}(1^\lambda)$: *Takes in a security parameter and outputs a key in* $\{0,1\}^{c(\lambda)}$.
- $F(K, x)$:[8] *Takes in a key and an input, outputs an evaluation of the PRF.*
- $F.\mathsf{Puncture}(K, S)$: *Takes as input a key and a set* $S \subseteq \{0,1\}^{m(\lambda)}$*, outputs a punctured key.*

We require the following.

Correctness. *For all efficient distributions* $\mathcal{D}(1^\lambda)$ *over the power set* $2^{\{0,1\}^{m(\lambda)}}$*, we require*

$$\Pr\left[\forall x \notin S \quad F(K_S, x) = F(K, x) : \begin{array}{c} S \leftarrow \mathcal{D}(1^\lambda) \\ K \leftarrow \mathsf{KeyGen}(1^\lambda) \\ K_S \leftarrow \mathsf{Puncture}(K, S) \end{array}\right] = 1.$$

Puncturing Security *We require that any stateful QPT adversary* $\mathcal{A}$ *wins the following game with probability at most* $1/2 + \mathsf{negl}(\lambda)$.

1. $\mathcal{A}$ *outputs a set* S.
2. *The challenger samples* $K \leftarrow \mathsf{KeyGen}(1^\lambda)$ *and* $K_S \leftarrow \mathsf{Puncture}(K, S)$
3. *The challenger samples* $b \leftarrow \{0,1\}$*. If* $b = 0$*, the challenger submits* $K_S, \{F(K, x)\}_{x \in S}$ *to the adversary. Otherwise, it submits* $K_S, \{y_s\}_{s \in S}$ *to the adversary where* $y_s \leftarrow \{0,1\}^{n(\lambda)}$ *for all* $s \in S$.
4. *The adversary outputs a guess* b' *and we say that the adversary has won if* $b' = b$.

Theorem 5 ([SW14, GGM86, Zha12])**.** *Let* $n(\cdot), m(\cdot)$ *be efficiently computable functions.*

- *If (post-quantum) one-way functions exist, then there exists a (post-quantum) puncturable PRF with input space* $\{0,1\}^{m(\lambda)}$ *and output space* $\{0,1\}^{n(\lambda)}$.
- *If we assume subexponentially-secure (post-quantum) one-way functions exist, then for any* $c > 0$*, there exists a (post-quantum)* $2^{-\lambda^c}$*-secure*[9] *puncturable PRF against subexponential time adversaries, with PRF input space* $\{0,1\}^{m(\lambda)}$ *and output space* $\{0,1\}^{n(\lambda)}$.

4.4 Indistinguishability Obfuscation

In this section, we recall indistinguishability obfuscation.

Definition 3. *An indistinguishability obfuscation scheme* $i\mathcal{O}$ *for a class of circuits* $\mathcal{C} = \{\mathcal{C}_\lambda\}_\lambda$ *satisfies the following.*

[8] We overload the notation and write F to both denote the function itself and the evaluation algorithm.

[9] While the original results are for negligible security against polynomial time adversaries, it is easy to see that they carry over to subexponential security. Further, by scaling the security parameter by a polynomial and simple input/output conversions, subexponentially secure (for any exponent c') one-way functions is sufficient to construct for any c a puncturable PRF that is $2^{-\lambda^c}$-secure.

Correctness. *For all* $\lambda, C \in \mathcal{C}_\lambda$ *and inputs* x, $\Pr\left[\tilde{C}(x) = C(x) : \tilde{C} \leftarrow i\mathcal{O}(1^\lambda, C)\right] = 1$.

Security. Let $\mathcal{B}$ be any QPT algorithm that outputs two circuits $C_0, C_1 \in \mathcal{C}$ of the same size, along with auxiliary information, such that $\Pr\left[\forall x\ C_0(x) = C_1(x) : (C_0, C_1, \mathsf{R}_{\mathsf{aux}}) \leftarrow \mathcal{B}(1^\lambda)\right] \geq 1 - \mathsf{negl}(\lambda)$. Then, for any QPT adversary $\mathcal{A}$,

$$\left| \Pr\left[\mathcal{A}(i\mathcal{O}(1^\lambda, C_0), \mathsf{R}_{\mathsf{aux}}) = 1 : (C_0, C_1, \mathsf{R}_{\mathsf{aux}}) \leftarrow \mathcal{B}(1^\lambda)\right] - \Pr\left[\mathcal{A}(i\mathcal{O}(1^\lambda, C_1), \mathsf{R}_{\mathsf{aux}}) = 1 : (C_0, C_1, \mathsf{R}_{\mathsf{aux}}) \leftarrow \mathcal{B}(1^\lambda)\right] \right| \leq \mathsf{negl}(\lambda).$$

4.5 Compute-and-Compare Obfuscation

In this section, we recall the notion of compute-and-compare obfuscation [WZ17].

Definition 4 (Compute-and-compare program)**.** *Let* $f : \{0,1\}^{a(\lambda)} \to \{0,1\}^{b(\lambda)}$ *be a function and* $y \in \{0,1\}^{b(\lambda)}$ *be a target value. The following program* P*, described by* (f, y, z)*, is called a* compute-and-compare program.

$P(x)$: *Compute* $f(x)$ *and compare it to* y*. If they are equal, output* 1*. Otherwise, output* 0*.*

A distribution $\mathcal{D}$ of such programs (along with quantum auxiliary information $\mathsf{R}_{\mathsf{aux}}$) is called sub-exponentially unpredictable if for any QPT adversary, given the auxiliary information $\mathsf{R}_{\mathsf{aux}}$ and the description of f, the adversary can predict the target value y with at most subexponential probability.

Definition 5. *A compute-and-compare obfuscation scheme for a class of distributions consists of efficient algorithms* $\mathsf{CCObf.Obf}$ *and* $\mathsf{CCObf.Sim}$ *that satisfy the following. Consider any distribution* $\mathcal{D}$ *over compute-and-compare programs, along with quantum auxiliary input, in this class.*

Correctness. *For any function* (f, y, z) *in the support of* $\mathcal{D}$, $\Pr[\forall x\ D'(x) = D(x) : D' \leftarrow \mathsf{CCObf.Obf}(f, y)] \geq 1 - \mathsf{negl}(\lambda)$.

All-Zero Simulation *Program output by* $\mathsf{CCObf.Sim}$ *satisfy* $P(x) = 0$ *for all* x.

Security $(\mathsf{CCObf.Obf}(f, y), \mathsf{R}_{\mathsf{aux}}) \approx (\mathsf{CCObf.Sim}(1^\lambda, |f|, |y|), \mathsf{R}_{\mathsf{aux}})$ *where* $(f, y), \mathsf{R}_{\mathsf{aux}} \leftarrow \mathcal{D}(1^\lambda)$.

Theorem 6 ([WZ17, CLLZ21])**.** *Assuming the hardness of LWE, there exists compute-and-compare obfuscation for any class of sub-exponentially unpredictable distributions.*

4.6 Learning with Errors

Definition 6. *Let* $m(\lambda), n(\lambda), q(\lambda)$ *be integers and* $\chi(\lambda)$ *be a probability distribution over* $\mathbb{Z}_q$*. The* (m, n, q, χ)*-LWE assumption says that the following distributions are indistinguishable to any QPT adversary*

$$(A, s^\mathsf{T} \cdot A + e) \approx_c (A, u)$$

where $A \leftarrow \mathbb{Z}_q^{n \times m}, s \leftarrow \mathbb{Z}_q^n$*,* $e \leftarrow \chi^m$ *and* $u \leftarrow \mathbb{Z}_q^m$*.*

4.7 Subspace States

A subspace state is $|A\rangle = \sum_{v\in A}|v\rangle$ where A is a subspace of the vector space $\mathbb{F}_2^n$. We will overload the notation and usually write $A, A^\perp$ to also denote the membership checking programs for the subspace A and its orthogonal complement $A^\perp$.

Theorem 7 ($1 \to 2$-Unclonability [Zha19])**.** *Consider the following game between a challenger and an adversary $\mathcal{A}$.*

$\underline{\mathsf{Exp}_{\mathcal{A}}(1^\lambda)}$

1. *The challenger samples a subspace $A \le \mathbb{F}_2^\lambda$ of dimension $\lambda/2$.*
2. *The challenger submits $|A\rangle, i\mathcal{O}(A), i\mathcal{O}(A^\perp)$ to $\mathcal{A}$.*
3. *The adversary outputs a (entangled) bipartite register $\mathsf{R}_1, \mathsf{R}_2$.*
4. *The challenger applies the projective measurement $\{|A\rangle\langle A|, I - |A\rangle\langle A|\}$, and outputs 1 if the measurement succeeds. Otherwise, it outputs 0.*

Then, for any QPT adversary $\mathcal{A}$, we have $\Pr\left[\mathsf{Exp}_{\mathcal{A}}(1^\lambda) = 1\right] \le \mathsf{negl}(\lambda)$.

Theorem 8 (Direct Product Hardness [SW22])**.** *Consider the following game between a challenger and an adversary $\mathcal{A}$.*

$\underline{\mathsf{Exp}_{\mathcal{A}}(1^\lambda)}$

1. *The challenger samples a subspace $A \le \mathbb{F}_2^\lambda$ of dimension $\lambda/2$.*
2. *The challenger submits $|A\rangle, i\mathcal{O}(A), i\mathcal{O}(A^\perp)$ to $\mathcal{A}$.*
3. *The adversary outputs two vectors $v, w \in \mathbb{F}_2^\lambda$.*
4. *The challenger checks if $v \in A$ and $w \in A^\perp$, and outputs 1 if so. Otherwise, it outputs 0.*

Then, for any QPT adversary $\mathcal{A}$, we have $\Pr\left[\mathsf{Exp}_{\mathcal{A}}(1^\lambda) = 1\right] \le \mathsf{negl}(\lambda)$.

5 Rerandomizable Encryption

In this section, we first recall rerandomizable encryption, then introduce some new security notions and give secure constructions.

Recall that a rerandomizable encryption (RPKE) scheme is a public-key encryption with an additional efficient randomized algorithm $\mathsf{ReRandomize}(pk, ct)$. We require rerandomization security.

Definition 7 (Rerandomization Security)**.** *A rerandomizable encryption scheme* RPKE *is said to satisfy rerandomization security*[10] *if for any efficient adversary $\mathcal{A}$, we have*

$$\Pr\left[b' = b : \begin{array}{c} pk, sk \leftarrow \mathsf{RPKE.Setup}(1^\lambda) \\ s \leftarrow \mathcal{A}(pk) \\ s_0 \leftarrow \mathsf{RPKE.ReRand}(pk, s) \\ s_1 \leftarrow \mathsf{RPKE.Enc}(pk, 0^{p(\lambda)}) \\ b' \leftarrow \mathcal{A}(s_b) \end{array}\right] \le \frac{1}{2} + \mathsf{negl}(\lambda).$$

[10] Some work consider the weaker notion where an honest ciphertext is rerandomized rather than adversarial s.

We also introduce a related security notion called *statistical rerandomization security for truly random public keys*. For this, we require that if a truly random pk is used instead of $\mathsf{RPKE.Setup}$, then the above security holds for unbounded $\mathcal{A}$.

Now we introduce the notion of *strong correctness*, which says no efficient adversary can find a malicious ciphertext and randomness tape such that decryption of the ciphertext before and after rerandomization differ.

Definition 8 (Strong Correctness Security)**.** *A rerandomizable encryption scheme* RPKE *is said to satisfy strong correctness if for any efficient adversary* $\mathcal{A}$*, we have*

$$\Pr\left[m \neq m' : \begin{array}{c} pk, sk \leftarrow \mathsf{RPKE.Setup}(1^\lambda) \\ ct, r \leftarrow \mathcal{A}(pk) \\ ct' = \mathsf{RPKE.ReRand}(pk, ct; r) \\ m = \mathsf{RPKE.Dec}(sk, ct) \\ m' = \mathsf{RPKE.Dec}(sk, ct') \end{array}\right] \leq \mathsf{negl}(\lambda).$$

We now introduce a stronger notion, called *strong correctness with public testing*. This requires that there exists a public ciphertext testing procedure such that there simply does not exist a malicious ciphertext (i) that passes the verification and (ii) whose decryption result before and rerandomization can differ. This can be considered a *statistical* version, and it will be a useful property in $i\mathcal{O}$-based proofs.

Definition 9 (Strong Correctness with Public Testing)**.** *A rerandomizable encryption scheme* RPKE *is said to satisfy strong correctness with public testing if*

$$\Pr_{pk,sk \leftarrow \mathsf{RPKE.Setup}(1^\lambda)}[\exists ct, r \ \ s.t. \ \ \begin{array}{c} \mathsf{RPKE.Test}(pk, ct) = 1 \ \ \wedge \\ \mathsf{RPKE.Dec}(sk, ct) \neq \mathsf{RPKE.Dec}(sk, \mathsf{RPKE.ReRand}(pk, ct; r)) \end{array}] \leq \mathsf{negl}(\lambda).$$

Finally, we introduce two more notions that are useful in our constructions. The first is called *simulatable testing key* property. In this setting, we consider the public testing key ptk separately from the public encryption key pk. We require that there is a test key simulation algorithm such that $(\mathsf{RPKE.SimulateTestKey}(pk), pk)$ is (computationally) indistinguishable from (ptk, pk) where $(pk, ptk, sk) \leftarrow \mathsf{RPKE.Setup}(1^\lambda)$. Finally, if the simulated test key output by $\mathsf{SimulateTestKey}$ satisfies $\mathsf{RPKE.Test}(ptk, ct) = 1$ for all strings ct, then we call it *simulatable all-accept testing key*. This property allows us to not use any ciphertext testing in actual constructions, and use $\mathsf{RPKE.Test}$ only in security proofs.

5.1 Construction

In this section, we give a rerandomizable encryption scheme based on LWE (Section 4.6) that satisfies all of the security properties discussed above: (i) rerandomization security, (ii) strong correctness, (iii) strong correctness with public testing, (iv) simulatable all-accept testing keys, (v) pseudorandom public encryption keys and (vi) statistical rerandomization with truly random public keys. Also honest ciphertexts can be honestly rerandomized any polynomial number of times while staying *good*.

Let $m(\lambda), n(\lambda)$ denote LWE dimension and number of samples parameters, $\chi(\lambda)$ denote the error distribution, $B(\lambda)$ denote the bounds for the values in the support of $\chi(\lambda)$ (i.e. $\chi(\lambda) \subseteq [-B, B]$) and $q(\lambda)$ denote the modulus; with $q(\lambda) = \mathsf{subexp}(\lambda)$ and $B = \mathsf{poly}(\lambda)$. All algebra in the scheme will implicitly be in $\mathbb{Z}_q$. Let $\ell(\lambda)$ denote the length of plaintexts. Finally, let CCObf be a compute-and-compare obfuscation scheme for subexponentially unpredictable distributions (Definition 5), which exists assuming LWE (Theorem 6).

$\underline{\mathsf{RPKE.Setup}(1^\lambda)}$

1. Sample $A \leftarrow \mathbb{Z}_q^{n \times m}$, $s \leftarrow \mathbb{Z}_q^n$ and $e \leftarrow \chi^m$.
2. Compute $y^\mathsf{T} = s^\mathsf{T} \cdot A + e^\mathsf{T}$.
3. For $i \in [\ell(\lambda)]$
    1. Sample $L_i \leftarrow [0, \lfloor \frac{q}{16} \rfloor]$.
    2. Let P_i be the compute-and-compare program that on input $(\vec{a}, c)$ computes $f_s(\vec{a}, c)$ and compares it to $\lfloor \frac{q}{4} \rfloor + L_i$ and outputs 1 on equality and 0 otherwise; where $f_s(\vec{a}, c) := (c - s^\mathsf{T} \cdot a)$. Sample $\mathsf{OP}_i \leftarrow \mathsf{CCObf}(P_i)$.
4. Set $pk = (A, y)$, $ptk = (\mathsf{OP}_i)_{i \in [\ell(\lambda)]}$ and $sk = (s, (L_i)_{i \in [\ell(\lambda)]})$. Output pk, ptk, sk.

$\underline{\mathsf{RPKE.Enc}(pk, \mu)}$

1. Parse $(A, y) = pk$.
2. For $i \in [\ell(\lambda)]$
    1. Sample $r_i \leftarrow \{0,1\}^m$
    2. Compute $ct_i = (A \cdot r, y^\mathsf{T} \cdot r + \mu_i \cdot \lfloor \frac{q}{2} \rfloor)$.
3. Output $(ct_i)_{i \in [\ell(\lambda)]}$.

$\underline{\mathsf{RPKE.ReRand}(pk, ct)}$

1. Sample $(ct'_i)_{i \in [\ell(\lambda)]} \leftarrow \mathsf{RPKE.Enc}(pk, 0^{\ell(\lambda)})$.
2. Output $(ct_i + ct'_i)_{i \in [\ell(\lambda)]}$.

$\underline{\mathsf{RPKE.Test}(ptk, ct)}$

1. Parse $(\mathsf{OP}_i)_{i \in [\ell(\lambda)]} = ptk$.
2. Parse $((\vec{a}_i, c_i)_{i \in [\ell(\lambda)]} = ct$.
3. For $i \in [\ell(\lambda)]$
    1. For $sh \in [-m \cdot B + 1, m \cdot B] \bigcup [2 \cdot \lfloor \frac{q}{4} \rfloor - m \cdot B, 2 \cdot \lfloor \frac{q}{4} \rfloor - 1 + m \cdot B] \subset \mathbb{Z}_q$, check if $\mathsf{OP}_i((\vec{a}_i, c_i + sh)) = 1$, if not, output 0 and terminate.
4. Output 1.

$\underline{\mathsf{RPKE.SimulateTestKey}(1^\lambda)}$

1. Output $\mathsf{CCObf.Sim}(1^\lambda, 1^{m(\lambda)}, 1^{n(\lambda)}, 1^{q(\lambda)})$

$\underline{\mathsf{RPKE.Dec}(sk, ct)}$

1. Parse $(s, (L_i)_{i\in[\ell(\lambda)]}) = sk$.
2. Parse $((\vec{a}_i, c_i)_{i\in[\ell(\lambda)]} = ct$.
3. For $i \in [\ell(\lambda)]$, check if $|c_i - s^{\mathsf{T}}\vec{a}_i - L_i| < \lfloor \frac{q}{4} \rfloor$. If so, set $\mu_i = 0$, otherwise set $\mu_i = 1$.
4. Output μ.

Theorem 9. RPKE *satisfies CPA security.*

Proof. Observe that the ciphertexts and the public key are the same as Regev's PKE, thus the result follows by [Reg09]. □

Theorem 10. RPKE *satisfies rerandomization security, statistical rerandomization security for truly random public keys and pseudorandom public key property.*

Proof. The first prove the statistical rerandomization security. For simplicity we will consider single bit messages. Observe that a ciphertext $(\vec{a}, c)$ is rerandomized by adding $(A \cdot r, y^{\mathsf{T}} \cdot r)$ where $r \leftarrow \{0,1\}^m$ and (A, y) is the public key. However, when the public key (consider it as a matrix by adding y^{T} as a row to A) is truly random, then by a simple application of leftover hash lemma [ILL89], we get that $(A \cdot r, y^{\mathsf{T}} \cdot r)$ is also truly random. Similarly, a fresh encryption of 0 will also be a truly (independent) random string by leftover hash lemma. Hence, the statistical reradomization follows.

For pseudorandomness, observe that the public key is simply LWE samples. Thus by LWE security (Section 4.6) it is indistinguishable from a truly random matrix.

To see rerandomization security, we simply combine the above two properties. □

Theorem 11. RPKE *satisfies strong rerandomization correctness, strong rerandomization correctness with testing and simulatable all-accept testing keys property.*

Proof. We will first prove strong rerandomization correctness with testing. For simplicity we will prove the single bit plaintext case $\ell = 1$, the general case follows by the same argument. Let $(\vec{a}, c)$ be a (malicious) ciphertext. Since we decrypt it by checking $|c - s^{\mathsf{T}} \cdot \vec{a} - L| < \lfloor \frac{q}{4} \rfloor$, observe that it decrypts to 0 if and only if $c - s^{\mathsf{T}} \cdot \vec{a} \in [L, \lfloor \frac{q}{4} \rfloor + L - 1] \bigcup [q - \lfloor \frac{q}{4} \rfloor + 1 + L, q - 1 + L]$. Similarly, it decrypts to 1 if and only if $c - s^{\mathsf{T}} \cdot \vec{a} \in [\lfloor \frac{q}{4} \rfloor + L, q - \lfloor \frac{q}{4} \rfloor + L]$. To rerandomize, we add $(A \cdot r, y^{\mathsf{T}} \cdot r)$ and get $(\vec{a} + A \cdot r, c + y^{\mathsf{T}} \cdot r)$. To decrypt this, we check $|c + y^{\mathsf{T}} \cdot r - s^{\mathsf{T}} \cdot (\vec{a} + A \cdot r) - L| < \lfloor \frac{q}{4} \rfloor$, but we have $c + y^{\mathsf{T}} \cdot r - s^{\mathsf{T}} \cdot (\vec{a} + A \cdot r) - L = c - s^{\mathsf{T}} \cdot \vec{a} + e^{\mathsf{T}} \cdot r$ where e is the error vector and $r \leftarrow \{0,1\}^{m(\lambda)}$, since $y^{\mathsf{T}} = s^{\mathsf{T}} \cdot A + e^{\mathsf{T}}$. However, observe that $e^{\mathsf{T}} \cdot r \in [-m \cdot B, m \cdot B]$ since the error distribution satisfies $\chi \subseteq [-B, B]$. Thus, for decryption of the original ciphertext and the rerandomized ciphertext to potentially differ, the original value $c - s^{\mathsf{T}} \cdot \vec{a}$ needs to be near rounding thresholds. More formally, we need to have $c - s^{\mathsf{T}}\vec{a} \in [\lfloor \frac{q}{4} \rfloor + L - m \cdot B, \lfloor \frac{q}{4} \rfloor + L + m \cdot B - 1]$ or $c - s^{\mathsf{T}}\vec{a} \in [q - \lfloor \frac{q}{4} \rfloor + L - m \cdot B + 1, q - \lfloor \frac{q}{4} \rfloor + L + m \cdot B]$. By correctness of CCObf, this is exactly what our test algorithm Test is checking, thus strong rerandomization correctness with testing follows.

Now, observe that the target values $\lfloor \frac{q}{4} \rfloor + L_i$ of the compute-and-compare programs P_i are subexponentially unpredictable. Thus, by the security of CCObf, the obfuscated programs OP_i are indistinguishable from obfuscations of all zero programs. Hence, simulatable all-accept testing property follows. Further, we can argue strong rerandomization correctness as follows. Consider the strong rerandomization correctness game and call it the first hybrid, Hyb_0. In Hyb_1, we modify the challenger so that the challenger first tests adversary's malicious ciphertext choice using

$\mathsf{RPKE.Test}(ptk, s)$ where $ptk \leftarrow \mathsf{RPKE.SimulateTestKey}(1^\lambda)$, and if it does not pass, the adversary automatically loses. However, by all-accepting simulated key property, $\mathsf{Hyb}_0 \equiv \mathsf{Hyb}_1$. In Hyb_2, we instead use the actual testing key rather than the simulated one. By simulated key security, we have $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$. Finally, we have $\Pr[\mathsf{Hyb}_2 = 1] \leq 1/2 + \mathsf{negl}(\lambda)$ by strong rerandomization correctness with testing security. □

6 Definitions

In this section, we give the formal definitions for quantum money schemes that support privacy and tracing.

Definition 10 (Quantum Money with Privacy and Tracing)**.** *A quantum money scheme* Bank *consists of the following efficient algorithms.*

- $\mathsf{Setup}(1^\lambda)$*: Takes in a security parameter, outputs a minting key* mk*, a tracing key* tk *and a public verification key* vk*.*
- $\mathsf{GenBanknote}(mk, t)$*: Takes in the minting key* mk *and a tag* t*, outputs a quantum banknote register.*
- $\mathsf{Verify}(vk, \mathsf{R})$*: Takes in the public key and a quantum register* R*, outputs* 0 *or* 1*.*
- $\mathsf{ReRandomize}(vk, \mathsf{R})$*: Takes in the public key and a quantum register* R*, outputs the updated register.*
- $\mathsf{Trace}(tk, \mathsf{R})$*: Takes in the tracing key and a quantum register* R*, outputs a tag value or* $\perp$*.*

We will require the following properties.

Definition 11 (Correctness)**.** *For any tag* $t \in \{0,1\}^{t(\lambda)}$*,*

$$\Pr\left[b = 1 : \begin{array}{c} vk, mk, tk \leftarrow \mathsf{Bank.Setup}(1^\lambda) \\ \mathsf{R} \leftarrow \mathsf{Bank.GenBanknote}(mk, t) \\ b \leftarrow \mathsf{Bank.Verify}(vk, \mathsf{R}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

We will also consider *projectiveness*, which requires that $\mathsf{Verify}(vk, (sn, \cdot))$ implements a rank-1 projector.

We now define correctness after rerandomization, which says that a valid banknote will stay valid after rerandomization.

Definition 12 (Correctness After Rerandomization)**.** *For any efficient algorithm* $\mathcal{B}$*,*

$$\Pr\left[b = 0 \vee b' = 1 : \begin{array}{c} vk, mk, tk \leftarrow \mathsf{Bank.Setup}(1^\lambda) \\ \mathsf{R} \leftarrow \mathcal{B}(vk, mk, tk) \\ b \leftarrow \mathsf{Bank.Verify}(vk, \mathsf{R}) \\ \mathsf{R} \leftarrow \mathsf{Bank.ReRandomize}(vk, \mathsf{R}) \\ b' \leftarrow \mathsf{Bank.Verify}(vk, \mathsf{R}) \end{array}\right] \geq 1 - \mathsf{negl}(\lambda).$$

Finally, we will require that banknotes do not grow in size with rerandomization, and call this property *compactness*[11].

[11] We note that, for compact schemes, we can also simply make re-randomization the last step of the verification algorithm

We now define counterfeiting (i.e unclonability) security for quantum money schemes. The security requires that any (QPT) adversary that obtains k banknotes will note be able to produce $k+1$ banknotes. While our security notion is quite similar to previous counterfeiting definitions (such as [AC12]), we will require unclonability security even against an adversary that has the tracing key. We give the formal game-based definition in Appendix A.

6.1 Fresh Banknote Security

Previous work has defined a privacy notion, called *anonymity*, where an adversary either gets back their banknotes in the original order or in permuted order (see Appendix A.1). We introduce a new, stronger security notion called *indistinguishability from fresh banknotes*.

Definition 13 (Indistinguishability from Fresh Banknotes)**.** *Consider the following game between a challenger and an adversary $\mathcal{A}$.*

$\underline{\mathsf{QM}-\mathsf{FRESH}-\mathsf{IND}_{\mathcal{A}}(1^\lambda)}$

1. *Sample* $vk, mk, tk \leftarrow \mathsf{Bank.Setup}(1^\lambda)$.
2. *Submit* vk, mk *to* $\mathcal{A}$.
3. *Adversary* $\mathcal{A}$ *outputs a register* R_0.
4. *Run* $\mathsf{Bank.Verify}(vk, \mathsf{R}_0)$. *If it fails, output* 0 *and terminate.*
5. *Run* $\mathsf{Bank.ReRandomize}(vk, \mathsf{R}_0)$.
6. *Sample* $\mathsf{R}_1 \leftarrow \mathsf{Bank.GenBanknote}(mk, 0^{t(\lambda)})$.
7. *Submit* R_b *to the adversary* $\mathcal{A}$.
8. *Adversary* $\mathcal{A}$ *outputs a bit* b'.
9. *Output* 1 *if and only if* $b' = b$.

We say that the quantum money scheme Bank *satisfies* fresh banknote indistinguishability *if for any QPT adversary $\mathcal{A}$, we have*

$$\Pr\left[\mathsf{QM}-\mathsf{FRESH}-\mathsf{IND}_{\mathcal{A}}(1^\lambda)=1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Theorem 12. *Any quantum money scheme that satisfies* basic fresh banknote indistinguishability *also satisfies* anonymity*.*

The proof follows by a simple hybrid argument.

6.2 Traceability

In this section, we introduce tracing security for public-key quantum money.

Definition 14 (Tracing Security)**.** *Consider the following game between the challenger and an adversary $\mathcal{A}$.*

$\underline{\mathsf{PKQM} - \mathsf{TRACING}_{\mathcal{A}}(1^\lambda)}$

1. *Initialize the list*[12] $\mathsf{TAGS} = []$.
2. *Sample* $vk, mk, tk \leftarrow \mathsf{Bank}, \mathsf{Setup}(1^\lambda)$.
3. *Submit* vk, tk *to* $\mathcal{A}$.
4. ***Banknote Query Phase:*** *For multiple rounds,* $\mathcal{A}$ *queries for a banknote by sending a tag* $t \in \{0,1\}^{t(\lambda)}$. *For each query, the challenger executes* $\mathsf{R}_{\mathsf{bn}} \leftarrow \mathsf{Bank.GenBanknote}(mk, t)$ *and submits* R_{bn} *to the adversary. The challenger also adds* t *to the list* TAGS.
5. $\mathcal{A}$ *outputs a value* k *and a* k*-partite register* $(\mathsf{R}_i)_{i\in[k]}$.
6. *For* $i \in [k]$, *the challenger tests* $\mathsf{Bank.Verify}(vk, \mathsf{R}_i) = 1$ *and adds the output of* $\mathsf{Bank.Dec}(tk, \mathsf{R}_i)$ *to the list* TAGS'. *If any of the tests output* 0, *the challenger outputs* 0 *and terminates.*
7. *The challenger checks if the list* $\mathsf{SORT}(\mathsf{TAGS}')$ *is a sublist of* $\mathsf{SORT}(\mathsf{TAGS})$. *If so, it outputs* 0. *Otherwise, it outputs* 1.

6.3 Untraceability

In this section, we introduce the notion of *untraceability* for quantum money for the first time. We will require that the banknotes are anonymous to everyone, including the malicious bank. We will consider this model in the common *random* string model.

Definition 15 (Untraceability)**.** *Consider the following game between a challenger and an adversary* $\mathcal{A}$.

$\underline{\mathsf{PKQM} - \mathsf{UNTRACE}_{\mathcal{A}}(1^\lambda)}$

1. *Sample* $crs \leftarrow \{0,1\}^{q(\lambda)}$.
2. *Submit* crs *to the adversary* $\mathcal{A}$.
3. *Adversary* $\mathcal{A}$ *outputs keys* vk, mk *and a register* R_0.
4. *Run* $\mathsf{Bank.Verify}(vk, \mathsf{R}_0)$. *If it fails, output* 0 *and terminate.*
5. *Sample* $\mathsf{R}_1 \leftarrow \mathsf{Bank.GenBanknote}(mk)$.
6. *Run* $\mathsf{Bank.Verify}(vk, \mathsf{R}_1)$. *If it fails, output* 0 *and terminate.*
7. *Sample* $b \leftarrow \{0,1\}$.
8. *Submit* R_b *to the adversary* $\mathcal{A}$.
9. *Adversary* $\mathcal{A}$ *outputs a bit* b'.
10. *Output* 1 *if and only if* $b' = b$.

We say that the quantum money scheme Bank *satisfies* untraceability *if for any QPT adversary* $\mathcal{A}$, *we have*

$$\Pr\left[\mathsf{PKQM} - \mathsf{UNTRACE}_{\mathcal{A}}(1^\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

[12] In particular, it can contain the same element multiple times.

7 Construction with Anonymity and Traceability

In this section, we give our public-key quantum money construction and prove that it satisfies unclonability, fresh banknote security, and tracing security.

We assume the existence of the following primitives that we use in our construction: (i) $i\mathcal{O}$, subexponentially secure indistinguishability obfuscation, (ii) RPKE, a rerandomizable public key encryption scheme with strong correctness and public testing (Definition 9) and (iii) F, a subexponentially secure puncturable PRF with input length $p_1(\lambda)$ and output length $p_2(\lambda)$.

We also utilize the following primitives that we only use in our security proofs: (i) SKE, a private-key encryption scheme with pseudorandom ciphertexts, and (ii) DS, a signature scheme.

We also set the following parameters: $t(\lambda)$ to be the desired tag length, $sg(\lambda)$ to be the signature length of DS for messages of length $\lambda + t(\lambda)$, $c(\lambda)$ to be the ciphertext size of SKE for messages of length $\lambda + sg(\lambda)$, $p_1(\lambda)$ to be the ciphertext size of RPKE for messages of length $t(\lambda) + c(\lambda)$, $p_2(\lambda)$ to be the randomness size of the algorithm $\mathsf{SampleFullRank}$, and $p_3(\lambda)$ to be the randomness size of the algorithm $\mathsf{PKE.ReRandomize}$.

We now move onto our construction. Let $\mathsf{A}_{\mathsf{Can}}$ denote the *canonical* $\lambda/2$-dimensional subspace of $\mathbb{F}_2^\lambda$ defined as $\mathsf{Span}(e_1, \ldots, e_{\lambda/2})$ where $e_i \in \mathbb{F}_2^\lambda$ is the vector that has 1 at the i-th index and 0 at all the others.

$\underline{\mathsf{Bank.Setup}(1^\lambda)}$

1. Sample $K \leftarrow F.\mathsf{Setup}(1^\lambda)$.
2. Sample $pk, sk \leftarrow \mathsf{RPKE.Setup}(1^\lambda)$.
3. Sample $\mathsf{OPMem} \leftarrow i\mathcal{O}(\mathsf{PMem})$ where PMem is the following program.

> $\underline{\mathsf{PMem}_K(id, v, b)}$
>
> **Hardcoded:** K
>
> 1. $T = \mathsf{SampleFullRank}(1^\lambda; F(K, id))$.
> 2. Compute $w = T^{-1}(v)$ if $b = 0$; otherwise, compute $w = T^{\mathsf{T}}(v)$.
> 3. Output 1 if $w \in \mathsf{A}_{\mathsf{Can}}$ if $b = 0$ and if $w \in \mathsf{A}_{\mathsf{Can}}^{\perp}$ if $b = 1$. Otherwise, output 0.

4. Sample $\mathsf{OPReRand} \leftarrow i\mathcal{O}(\mathsf{PReRand})$ where $\mathsf{PReRand}$ is the following program[13].

> $\underline{\mathsf{PReRand}_K(id, s)}$
>
> **Hardcoded:** K, pk
>
> 1. Check if $\mathsf{RPKE.Test}(pk, id) = 1$. If not, output $\perp$ and terminate.
> 2. $id' = \mathsf{RPKE.ReRand}(pk, id; s)$.
> 3. $T_1 = \mathsf{SampleFullRank}(1^\lambda; F(K, id))$.
> 4. $T_2 = \mathsf{SampleFullRank}(1^\lambda; F(K, id'))$.

[13] We note that if we make the stronger assumption of RPKE with *all-accepting simulatable testing keys*, we can actually remove the $\mathsf{RPKE.Test}$ line from the construction.

5. Output $id', T_2 \cdot T_1^{-1}$.

5. Set $vk = (\mathsf{OPMem}, \mathsf{OPReRand})$.
6. Set $mk = (K, pk)$.
7. Set $tk = sk$.
8. Output vk, mk, tk.

$\underline{\mathsf{Bank.GenBanknote}(mk, tag)}$

1. Parse $(K, pk) = mk$.
2. Sample $ict \leftarrow \{0,1\}^{c(\lambda)}$.
3. Sample $ct \leftarrow \mathsf{RPKE.Enc}(pk, tag||ict)$.
4. $T = \mathsf{SampleFullRank}(1^\lambda; F(K, ct))$.
5. Set $|\$\rangle = \sum_{v \in A} |T(v)\rangle$.
6. Output $ct, |\$\rangle$.

$\underline{\mathsf{Bank.Verify}(vk, \mathsf{R})}$

1. Parse $(\mathsf{OPMem}, \mathsf{OPReRand}) = vk$.
2. Parse $(id, \mathsf{R}') = \mathsf{R}$.
3. Run OPMem coherently on $id, \mathsf{R}', 0$. Check if the output is 1, and then rewind (as in Gentle Measurement Lemma [Aar16]).
4. Apply QFT to R'.
5. Run OPMem coherently on $id, \mathsf{R}', 1$. Check if the output is 1, and then rewind.
6. Output 1 if both verifications passed above. Otherwise, output 0.

$\underline{\mathsf{Bank.ReRandomize}(vk, \mathsf{R})}$

1. Parse $(\mathsf{OPMem}, \mathsf{OPReRand}) = vk$.
2. Parse $(ct, \mathsf{R}') = \mathsf{R}$.
3. Sample $s \leftarrow \{0,1\}^{p_3(\lambda)}$.
4. $ct', T = \mathsf{OPReRand}(ct, s)$.
5. Apply the linear map $T : \mathbb{F}_2^\lambda \to \mathbb{F}_2^\lambda$ coherently[14] to R'.
6. Output ct', R'.

[14] That is, in superposition. Note that since T is an efficient (in both directions) bijection, we can indeed apply it in superposition with no garbage left.

$\underline{\mathsf{Bank.Trace}(tk, \mathsf{R})}$

1. Parse $sk = tk$.
2. Parse $(ct, \mathsf{R}') = \mathsf{R}$.
3. Compute $pl = \mathsf{RPKE.Dec}(sk, ct)$.
4. If $pl = \bot$, output $\bot$. Otherwise, output first $t(\lambda)$ bits of pl.

Theorem 13. Bank *satisfies correctness after rerandomization and projectiveness.*

See Section 7.1 and Section 7.2 for the proofs.

Theorem 14. Bank *satisfies fresh banknote indistinguishability (Definition 13).*

See Section 7.4 for the proof.

Theorem 15. Bank *satisfies tracing security (Definition 14).*

See Section 7.5 for the proof.

Theorem 16. Bank *satisfies counterfeiting security (Definition 19).*

See Section 7.3 for the proof.

7.1 Projectiveness

Let T be a full rank linear map. Observe that a vector v satisfies $T^{-1}(v) \in \mathsf{A_{Can}}$ if and only if $v \in A^*$ where A^* is the set $\{T(w) : w \in \mathsf{A_{Can}}\}$, which is a subspace of dimension $\lambda/2$ since T is a full rank linear map. Similarly, we can show that a vector v satisfies $T^{\mathsf{T}}(v) \in \mathsf{A}_{\mathsf{Can}}^{\perp}$ if and only if $v \in (A^*)^{\perp}$. Note that $T^{\mathsf{T}}(v) \in \mathsf{A}_{\mathsf{Can}}^{\perp}$ if and only if $\langle T^{\mathsf{T}} \cdot v, u\rangle = 0$ for all $u \in \mathsf{A_{Can}}$. However, this inner product is equal to $\langle v, T \cdot u\rangle$, and as u ranges over all $\mathsf{A_{Can}}$, $T \cdot u$ ranges over all A^*. Thus, we get that the above is equivalent to $\langle v, u'\rangle = 0$ for all $u' \in A^*$, which is equivalent to $v \in (A^*)^{\perp}$.

The above shows that our verification algorithm is equivalent to the subspace state verification algorithm (for the subspace A^*) of Aaronson-Christiano [AC12], which they prove implements a projection onto the subspace. Thus, projectiveness of our scheme follows.

7.2 Correctness

We prove correctness after rerandomization. For a banknote $ct, |\psi\rangle$ that has been verified, we know by projectiveness that $|\psi\rangle = \sum_{v \in \mathsf{A_{Can}}} |T(v)\rangle$ where $T = \mathsf{SampleFullRank}(1^\lambda; F(K, ct))$. During rerandomization, $\mathsf{OPReRand}$ outputs ct' and $T'' = T' \cdot T^{-1}$ where $T' = \mathsf{SampleFullRank}(1^\lambda; F(K, ct'))$. Applying T'' in superposition, we get the state $\sum_{v \in \mathsf{A_{Can}}} |T''(v)\rangle$, which is perfectly the state for the serial number ct''.

7.3 Proof of Unclonability (Counterfeiting) Security

We prove counterfeiting security (i.e. unclonability) through a sequence of hybrids, each of which is constructed by modifying the previous one. We implicitly pad all the $i\mathcal{O}$ obfuscated programs to an appropriate size.

$\underline{\mathsf{Hyb}_0}$: The original game $\mathsf{PKQM} - \mathsf{CF}_{\mathcal{A}}(1^\lambda)$.

Hyb_1 : First, at the beginning of the game, the challenger samples $isk \leftarrow \mathsf{SKE.Setup}(1^\lambda)$. It also initializes a stateful counter $cnt = 0$ and a list $\mathsf{CT} = []$. Further, we modify the way the challenger mints banknotes. Instead of calling $\mathsf{Bank.GenBanknote}(mk, tag)$, it now executes the following subroutine for each query.

Minting Subroutine(tag)

1. Parse $(K, pk) = mk$.
2. Add 1 to cnt.
3. Sample $ict \leftarrow \mathsf{SKE.Enc}(isk, cnt||0^{sg(\lambda)})$.
4. Sample $ct \leftarrow \mathsf{RPKE.Enc}(pk, tag||ict)$.
5. Add ct to the list CT.
6. $T = \mathsf{SampleFullRank}(1^\lambda; F(K, ct))$.
7. Set $|\$\rangle = \sum_{v \in \mathsf{A_{Can}}} |T(v)\rangle$.
8. Output $ct, |\$\rangle$.

Hyb_2 : We change the way the challenger verifies the banknotes output by the adversary. Instead of executing Verify, it instead executes the following subroutine for each banknote.

Verify Subroutine(R)

1. Parse $(\mathsf{OPMem}, \mathsf{OPReRand}) = vk$.
2. Parse $(id, \mathsf{R}') = \mathsf{R}$.
3. Parse $pl_1||pl_2 = \mathsf{RPKE.Dec}(sk, id)$ with $|pl_1| = t(\lambda), |pl_2| = c(\lambda)$.
4. Compute $ipl_1||ipl_2 = \mathsf{SKE.Dec}(isk, pl_2)$ with $|ipl_1| = \lambda$ and $|ipl_2| = sg(\lambda)$.
5. Check if $ipl_1 \in [k]$. Output 0 and terminate the subroutine if not.
6. Run OPMem coherently on $id, \mathsf{R}', 0$. Check if the output is 1, and then rewind.
7. Apply QFT to R'.
8. Run OPMem coherently on $id, \mathsf{R}', 1$. Check if the output is 1, and then rewind.
9. Output 1 if both verifications passed above. Otherwise, output 0.

Hyb_3 : We now sample $i^* \leftarrow [k]$ and also initialize the list $\mathsf{INDICES} = []$ at the beginning of the game. We also modify the verification subroutine so that each ipl_1 value is added to INDICES. At the end of the game, the challenger (in addition to the previous checks) also checks if i^* appears at least twice in INDICES, and outputs 0 if not.

$\underline{\mathsf{Hyb}_4}$: We sample $K' \leftarrow F.\mathsf{Setup}(1^\lambda)$ and a random full rank linear map $T^* : \mathbb{F}_2^\lambda \to \mathbb{F}_2^\lambda$ at the beginning of the game. We also compute $A^* = T^*(\mathsf{A}_{\mathsf{Can}})$ and create the following function/program.

$$M_{K',\mathsf{CT}_{i^*}}(ct) = \begin{cases} \mathsf{SampleFullRank}(1^\lambda; F(K', id)), \text{ if } id \neq \mathsf{CT}_{i^*} \\ I, \text{ if } id = \mathsf{CT}_{i^*} \end{cases}$$

Further, we now sample OPMem and $\mathsf{OPReRand}$ as $\mathsf{OPMem} \leftarrow i\mathcal{O}(\mathsf{PMem}')$ and $\mathsf{OPReRand} \leftarrow i\mathcal{O}(\mathsf{PReRand}')$.

$\underline{\mathsf{PMem}'_K(id, v, b)}$

Hardcoded: $K, sk, isk, i^*, T^*, M_{K',\mathsf{CT}_{i^*}}$

1. Parse $pl_1||pl_2 = \mathsf{RPKE.Dec}(sk, id)$ with $|pl_1| = t(\lambda), |pl_2| = c(\lambda)$.
2. Parse $ipl_1||ipl_2 = \mathsf{SKE.Dec}(isk, pl_2)$ with $|ipl_1| = \lambda, |ipl_2| = sg(\lambda)$.
3. If $ipl_1 = i^*$, set $T = M_{K',\mathsf{CT}_{i^*}}(id) \cdot T^*$. Otherwise $T = \mathsf{SampleFullRank}(1^\lambda; F(K, id))$.
4. Compute $w = T^{-1}(v)$ if $b = 0$; otherwise, compute $w = T^{\mathsf{T}}(v)$.
5. Output 1 if $w \in \mathsf{A}_{\mathsf{Can}}$ if $b = 0$ and if $w \in \mathsf{A}_{\mathsf{Can}}^{\perp}$ if $b = 1$. Otherwise, output 0.

$\underline{\mathsf{PReRand}'_K(id, s)}$

Hardcoded: $K, pk, sk, isk, i^*, T^*, M_{K',\mathsf{CT}_{i^*}}$

1. Check if $\mathsf{RPKE.Test}(pk, id) = 1$. Otherwise, output $\perp$ and terminate.
2. Parse $pl_1||pl_2 = \mathsf{RPKE.Dec}(sk, id)$ with $|pl_1| = t(\lambda), |pl_2| = c(\lambda)$.
3. Parse $ipl_1||ipl_2 = \mathsf{SKE.Dec}(isk, pl_2)$ with $|ipl_1| = \lambda, |ipl_2| = sg(\lambda)$.
4. $id' = \mathsf{RPKE.ReRand}(pk, id; s)$.
5. If $ipl_1 = i^*$, set $T_1 = M_{K',\mathsf{CT}_{i^*}}(id) \cdot T^*$. Otherwise, $T_1 = \mathsf{SampleFullRank}(1^\lambda; F(K, id))$.
6. Parse $pl'_1||pl'_2 = \mathsf{RPKE.Dec}(sk, id')$ with $|pl'_1| = t(\lambda), |pl'_2| = c(\lambda)$.
7. Parse $ipl'_1||ipl'_2 = \mathsf{SKE.Dec}(isk, pl'_2)$ with $|ipl'_1| = \lambda, |ipl'_2| = sg(\lambda)$.
8. If $ipl'_1 = i^*$, set $T_2 = M_{K',\mathsf{CT}_{i^*}}(id') \cdot T^*$. Otherwise, $T_2 = \mathsf{SampleFullRank}(1^\lambda; F(K, id'))$.
9. Output $id', T_2 \cdot T_1^{-1}$.

Finally, we modify the minting subroutine as follows.

$\underline{\text{Minting Subroutine}(\mathsf{R})}$

1. Parse $(K, pk) = mk$.
2. Add 1 to cnt.
3. Sample $ict \leftarrow \mathsf{SKE.Enc}(isk, cnt||0^{sg(\lambda)})$.

4. Sample $ct \leftarrow \mathsf{RPKE.Enc}(pk, tag||ict)$.
5. If $cnt = i^*$, set $|\$\rangle = \sum_{v \in A^*} |v\rangle$ and jump to the final step.
6. $T = \mathsf{SampleFullRank}(1^\lambda; F(K, ct))$.
7. $A = T(\mathsf{A_{Can}})$.
8. Set $|\$\rangle = \sum_{v \in A} |v\rangle$.
9. Output $ct, |\$\rangle$.

$\underline{\mathsf{Hyb}_5}$: At the beginning of the game, after we sample T^*, we also sample $\mathsf{P}_0 \leftarrow i\mathcal{O}(A^*)$ and $\mathsf{P}_1 \leftarrow i\mathcal{O}((A^*)^\perp)$. Further, we now sample OPMem as $\mathsf{OPMem} \leftarrow i\mathcal{O}(\mathsf{PMem}'')$.

$\underline{\mathsf{PMem}''_K(id, v, b)}$
Hardcoded: $K, sk, isk, i^*, \mathsf{P}_0, \mathsf{P}_1$

1. Parse $pl_1||pl_2 = \mathsf{RPKE.Dec}(sk, id)$ with $|pl_2| = c(\lambda)$.
2. Parse $ipl_1||ipl_2 = \mathsf{SKE.Dec}(isk, pl_2)$ with $|ipl_1| = \lambda$ and $|ipl_3| = sg(\lambda)$.
3. If $ipl_1 = i^*$,
    1. Set $T = M_{K', \mathsf{CT}_{i^*}}(id)$.
    2. Compute $w = T^{-1}(v)$ if $b = 0$; otherwise, compute $w = (T^{-1})^\mathsf{T}(v)$.
    3. Output the output $\mathsf{P}_b(v)$ and terminate.
4. If $ipl_1 \neq i^*$,
    1. Set $T = \mathsf{SampleFullRank}(1^\lambda; F(K, id))$.
    2. Compute $w = T^{-1}(v)$ if $b = 0$; otherwise, compute $w = (T)^\mathsf{T}(v)$.
    3. Output 1 if $w \in \mathsf{A_{Can}}$ if $b = 0$ and if $w \in \mathsf{A}^\perp_{\mathsf{Can}}$ if $b = 1$. Otherwise, output 0.

$\underline{\mathsf{Hyb}_6}$: We now sample $\mathsf{OPReRand}$ as $\mathsf{OPReRand} \leftarrow i\mathcal{O}(\mathsf{PReRand}'')$.

$\underline{\mathsf{PReRand}''_K(id, s)}$
Hardcoded: $K, pk, sk, isk, i^*, M_{K', \mathsf{CT}_{i^*}}$

1. Check if $\mathsf{RPKE.Test}(pk, id) = 1$. Otherwise, output $\perp$ and terminate.
2. Parse $pl_1||pl_2 = \mathsf{RPKE.Dec}(sk, id)$ with $|pl_2| = c(\lambda)$.
3. Parse $ipl_1||ipl_2 = \mathsf{SKE.Dec}(isk, pl_2)$ with $|ipl_1| = \lambda$ and $|ipl_2| = sg(\lambda)$.
4. $id' = \mathsf{RPKE.ReRand}(pk, id; s)$.
5. If $ipl_1 = i^*$, set $T_1 = M_{K', \mathsf{CT}_{i^*}}(id)$. Otherwise, $T_1 = \mathsf{SampleFullRank}(1^\lambda; F(K, id))$.

6. If $ipl_1 = i^*$, set $T_2 = M_{K',\mathsf{CT}_{i^*}}(id')$. Otherwise, $T_2 = \mathsf{SampleFullRank}(1^\lambda; F(K, id'))$.

7. Output $id', T_2 \cdot T_1^{-1}$.

Lemma 1. $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$.

Proof. These hybrids differ in two places. First, the challenger executes the minting procedure directly instead of calling $\mathsf{Bank.GenBanknote}$. This is only a semantic change and makes no difference. Second, we replace the random strings $ict \leftarrow \{0,1\}^{c(\lambda)}$ with ciphertexts of SKE. Thus, the security follows by the pseudorandom ciphertext security of SKE, since the experiments do not use sk and can be simulated only using the ciphertexts. □

Lemma 2. $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$.

Proof. We will show that by strong rerandomization correctness property, any banknote whose serial number decrypts to a value outside $[k]$ is *rooted* (i.e is a rerandomization of) at a subspace state that was not even given to the adversary. Thus, the result then will follow by unlearnability of subspaces (we can also think of this as $0 \to 1$ unclonability).

We give a formal proof in Appendix B.2. □

Lemma 3. $\Pr[\mathsf{Hyb}_3 = 1] \geq \frac{\Pr[\mathsf{Hyb}_2 = 1]}{k}$

Proof. Observe that all ipl_1 added to $\mathsf{INDICES}$ are required to be in $[k]$, whereas the list at the end will have size $k+1$ (assuming the challenger has not terminated with output 0 already). Thus, by pigeonhole principle, there is a value $i^{**} \in [k]$ such that it appears in $\mathsf{INDICES}$ more than once. Our random guess i^* will satisfy $i^* = i^{**}$ with probability $1/k$. □

Lemma 4. $\mathsf{Hyb}_3 \approx \mathsf{Hyb}_4$.

Proof. This follows through a hybrid argument using the puncturing security of the PRF and the security $i\mathcal{O}$, where we create hybrids over all strings id. Through sufficient padding, the result follows by subexponential security of the PRF scheme and $i\mathcal{O}$.

We give a formal proof in Appendix B.1. □

Lemma 5. $\mathsf{Hyb}_4 \approx \mathsf{Hyb}_5$.

Proof. We will show that the programs PMem' and PMem'' have the same functionality. Then, the result follows by the security of $i\mathcal{O}$.

The behaviour of the two programs can possibly differ only on inputs such that $ipl = i^*$. However, by the same argument as in the proof of projectiveness of our scheme, we know that PMem' implements membership checking programs for the subspaces $A^*, (A^*)^\perp$ when $ipl = i^*$. By correctness of the obfuscation used to create $\mathsf{P}_0, \mathsf{P}_1$; PMem'' does the same thing. Thus, we get that the programs PMem' and PMem'' have the same functionality, and the result follows by the security of $i\mathcal{O}$. □

Lemma 6. $\mathsf{Hyb}_5 \approx \mathsf{Hyb}_6$.

Proof. Observe that by the strong rerandomization correctness of RPKE, there does not exist id and s such that $\mathsf{RPKE.Test}(pk, id) = 1$, but id and $id' = \mathsf{ReRand}(pk, ct; s)$ decrypt to different values. Thus, in $\mathsf{PReRand}'$, we know that id and id' will decrypt to the same value pl_2, which will decrypt to the same value ipl_1 due to deterministic decryption of SKE. Thus, separately decrypting id' to obtain ipl_1' versus directly using ipl_1 instead makes no difference. Further, this means that for both

id and id', we will be in the same case with respect to the test $ipl_1 =^? i^*$. Therefore, the factor T^* cancels outs when we compute $T_2 \cdot T_1^{-1} = (M_{K',\mathsf{CT}_{i^*}}(id') \cdot T^*) \cdot (M_{K',\mathsf{CT}_{i^*}}(id) \cdot T^*)^{-1}$. Therefore, the programs $\mathsf{PReRand}', \mathsf{PReRand}''$ have exactly the same functionality. The result follows by the security of $i\mathcal{O}$. □

Lemma 7. $\Pr[\mathsf{Hyb}_6 = 1] = \mathsf{negl}(\lambda)$.

Proof. Suppose otherwise for a contradiction. This result follows by the $1 \to 2$ unclonability of the subspace state $|A^*\rangle$. Observe that this hybrid can be simulated using only a single copy of $|A^*\rangle$ along with $i\mathcal{O}(A^*), i\mathcal{O}((A^*)^\perp)$ (by sampling the other keys e.g. K, K' ourselves). At the end, we know that at least two of the forged banknotes are such that their serial numbers decrypt to i^*. Let ct_j and ct_ℓ denote the serial numbers of these banknotes. We also know that by projectiveness of the verification, these two banknote states (when they pass the verification) will be exactly $\left|M_{K',\mathsf{CT}_{i^*}}(ct_j) \cdot A^*\right\rangle$ and $\left|M_{K',\mathsf{CT}_{i^*}}(ct_\ell) \cdot A^*\right\rangle$. However, since we (the reduction) sample K', we can actually take both of these back to $|A^*\rangle$ by applying $(M_{K',\mathsf{CT}_{i^*}}(ct_j))^{-1}, (M_{K',\mathsf{CT}_{i^*}}(ct_\ell))^{-1}$ in superposition, and obtain two copies of $|A^*\rangle$. However, this means that we cloned $|A^*\rangle$ with non-negligible probability, which is a contradiction by Theorem 7. □

Now, suppose for a contradiction that there exists a QPT $\mathcal{A}$ such that $\Pr\left[\mathsf{PKQM} - \mathsf{CF}_{\mathcal{A}}(1^\lambda) = 1\right]$ is non-negligible. Then, we get that $\Pr[\mathsf{Hyb}_6 = 1]$ is also non-negligible, which is a contradiction by Lemma 7.

7.4 Proof of Fresh Banknote Indistinguishability

We will prove security through a sequence of hybrids, each of which is obtained by modifying the previous one. Let $\mathcal{A}$ be a QPT adversary.

$\underline{\mathsf{Hyb}_0}$: The original game $\mathsf{QM} - \mathsf{FRESH} - \mathsf{IND}_{\mathcal{A}}(1^\lambda)$.

$\underline{\mathsf{Hyb}_1}$: In the re-randomization step (in Step 5), instead of performing $\mathsf{ReRandomize}(vk, \mathsf{R}_0)$, we instead execute the following subroutine.

Rerandomize Subroutine

1. Parse $(ct^*, \mathsf{R}'_0) = \mathsf{R}_0$.
2. Sample $s^* \leftarrow \{0,1\}^{p_3(\lambda)}$.
3. $ct_0^{**} = \mathsf{RPKE.ReRand}(pk, ct^*; s^*)$.
4. $T_1 = \mathsf{SampleFullRank}(1^\lambda; F(K, ct^*))$.
5. $T_0^* = \mathsf{SampleFullRank}(1^\lambda; F(K, ct_0^{**}))$.
6. Set $M^* = (T_0^*) \cdot T_1^{-1}$.
7. Apply the linear map $M^* : \mathbb{F}_2^\lambda \to \mathbb{F}_2^\lambda$ coherently to R'_0.
8. Set $\mathsf{R}_0 = (ct_0^{**}, \mathsf{R}'_0)$.

$\underline{\mathsf{Hyb}_2}$: In the rerandomization subroutine, we replace the line

$$ct_0^{**} = \mathsf{RPKE.ReRand}(pk, ct^*; s^*)$$

with

$$ct_0^{**} \leftarrow \mathsf{RPKE.Enc}(pk, 0^{t(\lambda)+c(\lambda)}).$$

$\underline{\mathsf{Hyb}_3}$: We further modify the rerandomization subroutine as follows.

Rerandomize Subroutine

1. Parse $(ct^*, \mathsf{R}'_0) = \mathsf{R}_0$.
2. Sample $ct_0^{**} \leftarrow \mathsf{RPKE.Enc}(pk, 0^{t(\lambda)+c(\lambda)})$.
3. $T_0^* = \mathsf{SampleFullRank}(1^\lambda; F(K, ct_0^{**}))$.
4. $A_0^* = T_0^*(\mathsf{A_{Can}})$.
5. Set $\mathsf{R}'_0 = \sum_{v \in A^*} |v\rangle$.
6. Set $\mathsf{R}_0 = (ct_0^{**}, \mathsf{R}'_0)$.

$\underline{\mathsf{Hyb}_4}$: In the challenge phase, for the case $b = 1$, instead of executing

$$\mathsf{R}_1 \leftarrow \mathsf{GenBanknote}(mk, 0^{t(\lambda)})$$

we execute

$$\begin{aligned}
ct_1^{**} &\leftarrow \mathsf{RPKE.Enc}(pk, 0^{t(\lambda)+c(\lambda)}) \\
T_1^* &= \mathsf{SampleFullRank}(1^\lambda; F(K, ct_1^{**})) \\
\mathsf{R}'_1 &= \sum_{v \in \mathsf{A_{Can}}} |T_1^*(v)\rangle \\
\mathsf{R}_1 &= (ct_1^{**}, \mathsf{R}'_1)
\end{aligned}$$

Lemma 8. $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$.

Proof. Hyb_1 simply unwraps $\mathsf{ReRandomize}(vk, \mathsf{R}_0)$, including having the challenger execute the code of the program $\mathsf{PReRand}$ rather than using $\mathsf{OPReRand}$. By correctness of $i\mathcal{O}$, the result follows. □

Lemma 9. $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$.

Proof. The result follows by rerandomization security of RPKE (Definition 7). □

Lemma 10. $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_3$.

Proof. As proven in the proof of correctness after rerandomization, once we rerandomize a banknote to the new serial number ct_0^{**}, our banknote state becomes the perfect banknote state associated with ct^{**}. Thus the result follows. □

Lemma 11. $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_4$.

Proof. In this hybrid the challenger simply executes the minting itself, rather than calling $\mathsf{Bank.GenBanknote}$. This is only a semantic change. □

Lemma 12. $\Pr[\mathsf{Hyb}_4 = 1] \leq 1/2$.

Proof. The results follows due to the fact that two challenge cases $b = 0$ and $b = 1$ are completely symmetrical. □

Thus, we get $\Pr\left[\mathsf{QM-FRESH-IND}_{\mathcal{A}}(1^\lambda) = 1\right] \leq 1/2 + \mathsf{negl}(\lambda)$, completing the proof.

7.5 Proof of Tracing Security

We will prove security through a sequence of hybrids, each of which is constructed by modifying the previous one. Let $\mathcal{A}$ be a QPT adversary.

$\underline{\mathsf{Hyb}_0}$**:** The original game $\mathsf{PKQM-TRACING}_{\mathcal{A}}(1^\lambda)$.

$\underline{\mathsf{Hyb}_1}$**:** At the beginning of the game, the challenger initializes a stateful counter for tags, $\mathsf{cnt}(t)$, that is initially set to 0 for each tag. It also initializes the lists $\mathsf{PL} = []$, $\mathsf{PL}' = []$. It also samples

$$isk \leftarrow \mathsf{SKE.Setup}(1^\lambda)$$
$$ivk, isgk \leftarrow \mathsf{DS.Setup}(1^\lambda).$$

We also modify the way the challenger answers banknote queries. Instead of executing

$$\mathsf{R_{bn}} \leftarrow \mathsf{GenBanknote}(mk, tag)$$

on a query for tag, it instead executes the following subroutine.

Minting Subroutine(tag)

1. Parse $(K, pk) = mk$.
2. Increase $\mathsf{cnt}(tag)$ by 1.
3. Sample $sig \leftarrow \mathsf{DS.Sign}(sk, tag||\mathsf{cnt}(tag))$.
4. Sample $ict \leftarrow \mathsf{SKE.Enc}(isk, \mathsf{cnt}(tag)||sig)$.
5. Add $tag||\mathsf{cnt}(tag)$ to the list PL.
6. Sample $ct \leftarrow \mathsf{RPKE.Enc}(pk, tag||ict)$.
7. $T = \mathsf{SampleFullRank}(1^\lambda; F(K, ct))$.
8. Set $|\$\rangle = \sum_{v \in \mathsf{A_{Can}}} |T(v)\rangle$.
9. Output $ct, |\$\rangle$.

$\underline{\mathsf{Hyb}_2}$**:** We change the way the challenger verifies the banknotes output by the adversary. Instead of executing Verify, it instead executes the following subroutine for each banknote.

Verify Subroutine(R)

1. Parse $(\mathsf{OPMem}, \mathsf{OPReRand}) = vk$.
2. Parse $(id, \mathsf{R}') = \mathsf{R}$.
3. Parse $pl_1||pl_2 = \mathsf{RPKE.Dec}(sk, id)$ with $|pl_1| = t(\lambda), |pl_2| = c(\lambda)$.
4. Parse $ipl_1||ipl_2 = \mathsf{SKE.Dec}(isk, pl_2)$ with $|ipl_1| = \lambda$, $|ipl_2| = sg(\lambda)$.
5. Check if $\mathsf{DS.Verify}(ivk, pl_1||ipl_1, ipl_2) = 1$. Output 0 and terminate the subroutine if the checks fail.
6. Add $ipl_1||ipl_2$ to the list PL'.
7. Run OPMem coherently on $id, \mathsf{R}', 0$. Check if the output is 1, and then rewind.
8. Apply QFT to R'.
9. Run OPMem coherently on $id, \mathsf{R}', 1$. Check if the output is 1, and then rewind.
10. Output 1 if both verifications passed above. Otherwise, output 0.

Hyb_3**:** In the verification subroutine, after checking $\mathsf{DS.Verify}(ivk, pl_1||ipl_1, ipl_2) = 1$, the challenger also checks if $ipl_1||ipl_2 \in \mathsf{PL}$, and outputs 0 and terminates if not.

Hyb_4**:** We add an additional winning condition at the end of the game: If the list PL' contains any duplicate elements, the challenger outputs 0 and hence the adversary loses.

Lemma 13. $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$.

Proof. Since the adversary does not have the secret key isk, the result follows by the semantic security of SKE. □

Lemma 14. $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$.

Proof. As in Lemma 2, this follows due to $0 \to 1$ unclonability (i.e. unlearnability) of the subspace states. The proof follows similarly to Lemma 2 (proven in Appendix B.2). The main difference in this case is that we will need to perform the *rooted at original* banknote (banknotes whose serial numbers decrypt to one of the original tags) versus *outside banknote* test for tag values chosen by the adversary, whereas in Lemma 2 the original set was just $[k]$, which is efficiently. In this case, thanks to the hidden signatures $isig$, we can still verify that a tag is one of the original tags efficiently without having to hardcode all the tag values inside the $i\mathcal{O}$ program. □

Lemma 15. $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_3$.

Lemma 16. *Observe that the adversary only obtains signatures for the elements of* PL*. Thus, the result follows by the unforgeability security of the signature scheme* DS*.*

Lemma 17. $\mathsf{Hyb}_3 \approx \mathsf{Hyb}_4$.

Proof. Observe that this is essentially unclonability of each indiviual banknote: Any duplicate value in PL' means that a banknote with that payload value was cloned. Thus, the proof follows similarly to the proof of unclonability of our scheme (especially hybrids Hyb_3 through Hyb_6 in Section 7.3). □

Lemma 18. $\Pr[\mathsf{Hyb}_4 = 1] = 0$.

Proof. Observe that at the end of the game, if the challenger already has not terminated with 0, we will have that PL' will be a subset of PL. Further, for the adversary to win the game, each element in PL' can appear once. Finally, note that $\mathsf{PL}, \mathsf{PL}'$ actually contains the tags (with a multiplicity counter attached to each tag) of the banknotes given to the adversary and the banknotes output by the adversary, respectively. Thus, combining these means that TAGS' will contain the same tags as TAGS, but fewer or equal multiplicity for each tag. This implies that $\mathsf{SORT}(\mathsf{TAGS}')$ is a sublist of $\mathsf{SORT}(\mathsf{TAGS})$. Hence, the adversary loses surely in Hyb_4. □

By above, we conclude that $\Pr\left[\mathsf{PKQM} - \mathsf{TRACING}_{\mathcal{A}}(1^\lambda) = 1\right] = \mathsf{negl}(\lambda)$ for any QPT adversary $\mathcal{A}$.

8 Construction with Untraceability

In this section, we give our public-key quantum money construction with untraceability, in the common (uniformly) random string model. In our scheme, verification automatically rerandomizes a banknote.

We assume the existence of the following primitives that we use in our construction. (i) $i\mathcal{O}$, subexponentially secure indistinguishability obfuscation, (ii) RPKE, a rerandomizable public key encryption scheme with strong correctness, public testing, pseudorandom public encryption keys, simulatable testing keys and statistical rerandomization security for truly random public keys (Definition 7), (iii) F, a subexponentially secure puncturable PRF with input length $p_1(\lambda)$ and output length $p_2(\lambda)$, and (iv) NIZK, a non-interactive zero knowledge argument system in the common random string model (CRS) for the language L defined below.

We also set the following parameters: $p_1(\lambda)$ to be the ciphertext size of RPKE for messages of length λ, $p_2(\lambda)$ to be the randomness size of the algorithm $\mathsf{SampleFullRank}$, $p_3(\lambda)$ to be the randomness size of the algorithm $\mathsf{PKE.ReRandomize}$, $q_1(\lambda)$ to be the CRS length of NIZK, $q_2(\lambda)$ to be the public key size of RPKE, $q(\lambda) = q_1(\lambda) + q_2(\lambda)$ to be the CRS length of our scheme.

Finally, we define the NP language L as consisting of x such that

$$\exists K, r \quad x = i\mathcal{O}(\mathsf{PMem}_K; r)$$

$\underline{\mathsf{Setup}(crs)}$

1. Parse[15] $crs_{nizk}||pk = crs$ with $|crs_{nizk}| = q_1(\lambda)$ and $|pk| = q_2(\lambda)$.
2. Sample $K \leftarrow F.\mathsf{Setup}(1^\lambda)$.
3. Sample random tape r_{iO} and compute $\mathsf{OPMem} = i\mathcal{O}(\mathsf{PMem}; r_{iO})$ where PMem is the following program.

   $\underline{\mathsf{PMem}_K(id, v, b)}$
   Hardcoded: K
   1. $T = \mathsf{SampleFullRank}(1^\lambda; F(K, id))$.

[15] See Appendix C for a remark on representing the public key as a binary string.

2. Compute $w = T^{-1}(v)$ if $b = 0$; otherwise, compute $w = T^{\mathsf{T}}(v)$.
3. Output 1 if $w \in \mathsf{A}_{\mathsf{Can}}$ if $b = 0$ and if $w \in \mathsf{A}^{\perp}_{\mathsf{Can}}$ if $b = 1$. Otherwise, output 0.

4. Sample $ptk \leftarrow \mathsf{RPKE.SimulateTestKey}(pk)$.
5. Sample $\mathsf{OPReRand} \leftarrow i\mathcal{O}(\mathsf{PReRand})$ where $\mathsf{PReRand}$ is the following program[16].

$\underline{\mathsf{PReRand}_K(id, s)}$

Hardcoded: K, pk, ptk

1. Check if $\mathsf{RPKE.Test}(ptk, id) = 1$. If not, output $\perp$ and terminate.
2. $id' = \mathsf{RPKE.ReRand}(pk, id; s)$.
3. $T_1 = \mathsf{SampleFullRank}(1^\lambda; F(K, id))$.
4. $T_2 = \mathsf{SampleFullRank}(1^\lambda; F(K, id'))$.
5. Output $T_2 \cdot T_1^{-1}$.

6. Sample the proof $\pi \leftarrow \mathsf{NIZK.Prove}(crs_{nizk}, x = \mathsf{OPMem}, w = (K, r_{iO}))$.
7. Set $vk = (\mathsf{OPMem}, \mathsf{OPReRand}, \pi)$.
8. Set $mk = (K, pk)$.
9. Output vk, mk.

$\underline{\mathsf{GenBanknote}(mk)}$

1. Parse $(K, pk) = mk$.
2. Sample $ct \leftarrow \mathsf{RPKE.Enc}(pk, 0^\lambda)$.
3. $T = \mathsf{SampleFullRank}(1^\lambda; F(K, ct))$.
4. Set $|\$\rangle = \sum_{v \in \mathsf{A}_{\mathsf{Can}}} |T(v)\rangle$.
5. Output $ct, |\$\rangle$.

$\underline{\mathsf{Verify}(crs, vk, \mathsf{R})}$

1. Parse $crs_{nizk}||pk = crs$ with $|pk| = q_2(\lambda)$.
2. Parse $(\mathsf{OPMem}, \mathsf{OPReRand}, \pi) = vk$.
3. Check[17] if $\mathsf{NIZK.Verify}(crs_{nizk}, x = \mathsf{OPMem}, \pi) = 1$. If not, output 0 and terminate.
4. Parse $(id, \mathsf{R}') = \mathsf{R}$.
5. Run OPMem coherently on $id, \mathsf{R}', 0$. Check if the output is 1, and then rewind.

[16] We note that if we make the stronger assumption of RPKE with *all-accepting simulatable testing keys*, we can actually remove the $\mathsf{RPKE.Test}$ line from the construction.

[17] We note that in reality, a user can just perform this verification only once on vk and later on keep using vk.

6. Apply QFT to R'.
7. Run OPMem coherently on $id, \mathsf{R}', 1$. Check if the output is 1, and then rewind.
8. If any verification failed, output 0 and terminate.
9. Sample $s \leftarrow \{0,1\}^{p_3(\lambda)}$.
10. $T = \mathsf{OPReRand}(id, s)$.
11. Compute $id' = \mathsf{RPKE.ReRand}(pk, id; s)$.
12. Apply the linear map $T : \mathbb{F}_2^\lambda \to \mathbb{F}_2^\lambda$ coherently to R'.
13. Run OPMem coherently on $id', \mathsf{R}', 0$. Check if the output is 1, and then rewind.
14. Apply QFT to R'.
15. Run OPMem coherently on $id', \mathsf{R}', 1$. Check if the output is 1, and then rewind.
16. If any verification failed, output 0. Otherwise, output 1.

Theorem 17. Bank *satisfies correctness and projectiveness.*

The proof of this theorem follows similarly to Section 7.1 and Section 7.2.

Theorem 18. Bank *satisfies untraceability (Definition 15).*

See Section 8.1 for the proof.

Theorem 19. Bank *satisfies counterfeiting security (Definition 19).*

See Section 8.2 for the proof.

8.1 Proof of Untraceability

We will prove security through a sequence of hybrids, each of which is obtained by modifying the previous one. Let $\mathcal{A}$ be a QPT adversary.

$\underline{\mathsf{Hyb}_0}$: The original game $\mathsf{PKQM} - \mathsf{UNTRACE}_{\mathcal{A}}(1^\lambda)$.

$\underline{\mathsf{Hyb}_1}$: In the verification steps (Step 4 and 6 of the experiment), instead of performing $\mathsf{Bank.Verify}(vk, \mathsf{R})$, the challenger executes the code of $\mathsf{Bank.Verify}$ directly on vk, R. Further, instead of verifying the proof π inside vk twice (in Step 4 and 6), it performs it only once when $\mathcal{A}$ outputs vk at Step 3.

$\underline{\mathsf{Hyb}_2}$: After verifying the NIZK argument π at Step 3, the challenger executes the following. It iterates over all strings K, r and checks if OPMem (parsed from vk as in $\mathsf{Bank.Verify}$) satisfies $\mathsf{OPMem} = i\mathcal{O}(\mathsf{PMem}_K; r)$. If the challenger finds such a value, it sets K^*, r^* to these value. If it cannot find such a value, it outputs 0 and terminates.

Note that from this hybrid onwards, the challenger (and the experiments) will be exponential time.

$\underline{\mathsf{Hyb}_3}$: We change the way the challenger performs the banknote verification.

Verify Subroutine(R)

1. Parse $crs_{nizk}||pk = crs$ with $|pk| = q_2(\lambda)$.
2. Parse $(\mathsf{OPMem}, \mathsf{OPReRand}, \pi) = vk$.
3. Parse $(id, \mathsf{R}') = \mathsf{R}$.
4. Compute $T_{id} = \mathsf{SampleFullRank}(1^\lambda; F(K^*, id))$.
5. Coherently run the following on R': On input v, check if $T_{id}^{-1}(v) \in \mathsf{A}_{\mathsf{Can}}$. Check if the output is 1, and then rewind.
6. Apply QFT to R'.
7. Coherently run the following on R': On input v, check if $T_{id}^{\mathsf{T}}(v) \in \mathsf{A}_{\mathsf{Can}}^{\perp}$. Check if the output is 1, and then rewind.
8. If any verification failed, output 0 and terminate.
9. Sample $s \leftarrow \{0,1\}^{p_3(\lambda)}$.
10. $T = \mathsf{OPReRand}(id, s)$.
11. Compute $id' = \mathsf{RPKE.ReRand}(pk, id; s)$.
12. Apply the linear map $T : \mathbb{F}_2^\lambda \to \mathbb{F}_2^\lambda$ coherently to R'.
13. Compute $T_{id'} = \mathsf{SampleFullRank}(1^\lambda; F(K^*, id'))$.
14. Coherently run the following on R': On input v, check if $T_{id'}^{-1}(v) \in \mathsf{A}_{\mathsf{Can}}$. Check if the output is 1, and then rewind.
15. Apply QFT to R'.
16. Coherently run the following on R': On input v, check if $T_{id'}^{\mathsf{T}}(v) \in \mathsf{A}_{\mathsf{Can}}^{\perp}$. Check if the output is 1, and then rewind.
17. If any verification failed, output 0. Otherwise, output 1.

Hyb_4 : Let id_b the initial serial number and id'_b be the rerandomized serial number ($id'_b = \mathsf{RPKE.ReRand}(pk, id_b; s)$) obtained during verification subroutine for R_b. We change the challenge output behaviour of the challenger. Instead of sending R_b to the adversary $\mathcal{A}$ in the challenge case b, it instead sends $(id'_b, |\psi_b\rangle)$ to the adversary where $|\psi_b\rangle = \sum_{v \in \mathsf{A}_{\mathsf{Can}}} \left| T_{id'_b}(v) \right\rangle$ and $T_{id'_b} = \mathsf{SampleFullRank}(1^\lambda; F(K^*, id'_b))$.

Lemma 19. $\mathsf{Hyb}_0 \equiv \mathsf{Hyb}_1$.

Proof. This is only a semantic change. □

Lemma 20. $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$.

Proof. By the computational soundness of NIZK, if it was the case that $\mathsf{OPMem} \notin L$, then the proof π produced by QPT adversary $\mathcal{A}$ would pass verification with only negligible probability. Thus, if the challenger has not already terminated, we know that the proof π passed and (except with negligibly small probability), $\mathsf{OPMem} \in L$. Hence, by definition of the language L, there exists K, r such that $\mathsf{OPMem} = i\mathcal{O}(\mathsf{PMem}_K; r)$, which the challenger will be able to find through exhaustive search. □

Lemma 21. $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_3$.

Proof. The change here is that the challenger executes the code PMem_{K^*} directly instead of using OPMem. However, we have at this point that $\mathsf{OPMem} = i\mathcal{O}(\mathsf{PMem}_{K^*}; r^*)$. By perfect correctness of $i\mathcal{O}$, the result follows. □

Lemma 22. $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_4$.

Proof. As shown in Section 7.1, the verification procedure implemented here is projective. Thus, in Hyb_3 the challenger is already outputting $(id'_b, |\psi_b\rangle)$ in the challenge phase, and there is no change between Hyb_3 and Hyb_4. □

Lemma 23. $\Pr[\mathsf{Hyb}_4 = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$.

Proof. First note that the public key pk is a truly random key, since it is taken from the CRS. Thus, by the *statistical rerandomization security using truly random public key* property (Definition 7), the ciphertexts id'_0, id'_1 (which are rerandomizations of id_0, id_1) are indistinguishable to the adversary, thus the challenge banknotes $(id'_0, |\psi_0\rangle), (id'_1, |\psi_1\rangle)$ are also indistinguishable. Note that this still applies even though the adversary knows K^* - since the security comes from indistinguishability of id'_0, id'_1. Finally, we note that while the challenger/experiments are exponential time, we are relying on *statistical rerandomization security*, thus the security indeed applies. □

Combining the above shows $\Pr[\mathsf{PKQM} - \mathsf{UNTRACE}_{\mathcal{A}} = 1] \leq \frac{1}{2} + \mathsf{negl}(\lambda)$, completing the proof.

8.2 Proof of Unclonability (Counterfeiting) Security

We will prove security through a sequence of hybrids, each of which is obtained by modifying the previous one. The proof will follow similarly to Section 7.3 after Hyb_3. Let $\mathcal{A}$ be a QPT adversary.

$\underline{\mathsf{Hyb}_0}$: The original game $\mathsf{PKQM} - \mathsf{CF}_{\mathcal{A}}(1^\lambda)$.

$\underline{\mathsf{Hyb}_1}$: At the beginning, the challenger samples $pk, ptk, sk \leftarrow \mathsf{RPKE.Setup}(1^\lambda)$. Then, instead of sampling the CRS as $crs \leftarrow \{0,1\}^{q(\lambda)}$, it samples $crs_{nizk} \leftarrow \{0,1\}^{q_1(\lambda)}$ and sets $crs = crs_{nizk}||pk$.

$\underline{\mathsf{Hyb}_2}$: During $\mathsf{Bank.Setup}$, instead of sampling $ptk \leftarrow \mathsf{RPKE.SimulateTestKey}(pk)$, the challenger instead uses ptk it obtained from $\mathsf{RPKE.Setup}(1^\lambda)$.

$\underline{\mathsf{Hyb}_3}$: During $\mathsf{Bank.Setup}$, instead of sampling $crs_{nizk} \leftarrow \{0,1\}^{q_1(\lambda)}$, the challenger instead simulates it as $(crs_{nizk}, st) \leftarrow \mathsf{NIZK.SimulateCRS}(1^\lambda)$. Then, instead of honestly proving $\pi \leftarrow \mathsf{NIZK.Prove}(crs_{nizk}, x = \mathsf{OPMem}, w = (K, r_{iO}))$, the challenger instead simulates the NIZK proof π as $\pi \leftarrow \mathsf{NIZK.SimulateProof}(st, \mathsf{OPMem})$.

$\underline{\mathsf{Hyb}_4}$: First, the challenger initializes a stateful counter $cnt = 1$ at the beginning. Further, we change the way the challenger samples banknotes. During minting, instead of sampling $ct \leftarrow \mathsf{RPKE.Enc}(pk, 0^\lambda)$, it now samples $ct \leftarrow \mathsf{RPKE.Enc}(pk, cnt)$ and adds 1 to cnt afterwards.

$\underline{\mathsf{Hyb}_5}$: First, the challenger initializes a list $\mathsf{PL} = []$ at the beginning. Further, we change the verification subroutine of the challenger applied to adversary's forged banknotes. For a banknote (id, R), after verifying the banknote, the challenger additionally performs the following check. It computes $pl = \mathsf{RPKE.Dec}(sk, id)$, adds pl to the list $\mathsf{INDICES}$ and checks if $pl \in [k]$. If not, it outputs 0 and terminates.

$\underline{\mathsf{Hyb}_6}$: At the beginning of the game, the challenger samples a random index i^*. Also, we will set id^* to the serial number of the i^*-th banknote produced by the challenger. Further, we require an additional condition for the adversary to win. At the end of the game, the challenger checks if i^* appears at least twice in $\mathsf{INDICES}$. If not, the challenger outputs 0 and the adversary loses.

$\underline{\mathsf{Hyb}_7}$: The challenger samples $K' \leftarrow F.\mathsf{Setup}(1^\lambda)$ and a random full rank linear map $T^* : \mathbb{F}_2^\lambda \to \mathbb{F}_2^\lambda$ at the beginning of the game. We also compute $A^* = T^*(\mathsf{A_{Can}})$ and create the following function/program.

$$M_{K',\mathsf{CT}_{i^*}}(ct) = \begin{cases} \mathsf{SampleFullRank}(1^\lambda; F(K', id)), & \text{if } id \neq id^* \\ I, & \text{if } id = id^* \end{cases}$$

Further, we now sample OPMem as $\mathsf{OPMem} \leftarrow i\mathcal{O}(\mathsf{PMem}')$ instead of sampling a random tape r_{iO} and setting $\mathsf{OPMem} = i\mathcal{O}(\mathsf{PMem}; r_{iO})$. We also change sampling of $\mathsf{OPReRand}$ to $\mathsf{OPReRand} \leftarrow i\mathcal{O}(\mathsf{PReRand}')$.

$\underline{\mathsf{PMem}'_K(id, v, b)}$

Hardcoded: $K, sk, isk, i^*, T^*, M_{K',\mathsf{CT}_{i^*}}$

1. Compute $pl = \mathsf{RPKE.Dec}(sk, id)$.
2. If $pl = i^*$, set $T = M_{K',\mathsf{CT}_{i^*}}(id) \cdot T^*$. Otherwise $T = \mathsf{SampleFullRank}(1^\lambda; F(K, id))$.
3. Compute $w = T^{-1}(v)$ if $b = 0$; otherwise, compute $w = T^{\mathsf{T}}(v)$.
4. Output 1 if $w \in \mathsf{A_{Can}}$ if $b = 0$ and if $w \in \mathsf{A}^\perp_{\mathsf{Can}}$ if $b = 1$. Otherwise, output 0.

$\underline{\mathsf{PReRand}'_K(id, s)}$

Hardcoded: $K, ptk, sk, isk, i^*, T^*, M_{K',\mathsf{CT}_{i^*}}$

1. Check if $\mathsf{RPKE.Test}(ptk, id) = 1$. Otherwise, output $\perp$ and terminate.
2. Compute $pl = \mathsf{RPKE.Dec}(sk, id)$.
3. $id' = \mathsf{RPKE.ReRand}(pk, id; s)$.
4. If $pl = i^*$, set $T_1 = M_{K',\mathsf{CT}_{i^*}}(id) \cdot T^*$. Otherwise, $T_1 = \mathsf{SampleFullRank}(1^\lambda; F(K, id))$.
5. Compute $pl' = \mathsf{RPKE.Dec}(sk, id')$.

6. If $pl' = i^*$, set $T_2 = M_{K',\mathsf{CT}_{i^*}}(id') \cdot T^*$. Otherwise, $T_2 = \mathsf{SampleFullRank}(1^\lambda; F(K, id'))$.
7. Output $T_2 \cdot T_1^{-1}$.

Finally, we modify the minting subroutine as follows.

Minting Subroutine(R)

1. Parse $(K, pk) = mk$.
2. Add 1 to cnt.
3. Sample $ct \leftarrow \mathsf{RPKE.Enc}(pk, 0^\lambda)$.
4. If $cnt = i^*$, set $id^* = ct$ and $|\$\rangle = \sum_{v \in A^*} |v\rangle$, and jump to the final step.
5. $T = \mathsf{SampleFullRank}(1^\lambda; F(K, ct))$.
6. Set $|\$\rangle = \sum_{v \in \mathsf{A}_{\mathsf{Can}}} |T(v)\rangle$.
7. Output $ct, |\$\rangle$.

Hyb_8 : At the beginning of the game, after we sample T^*, we also sample $\mathsf{P}_0 \leftarrow i\mathcal{O}(A^*)$ and $\mathsf{P}_1 \leftarrow i\mathcal{O}((A^*)^\perp)$. Further, we now sample OPMem as $\mathsf{OPMem} \leftarrow i\mathcal{O}(\mathsf{PMem}'')$.

$\mathsf{PMem}''_K(id, v, b)$
Hardcoded: $K, sk, isk, i^*, \mathsf{P}_0, \mathsf{P}_1$

1. Compute $pl = \mathsf{RPKE.Dec}(sk, id)$.
2. If $pl = i^*$,
   1. Set $T = M_{K',\mathsf{CT}_{i^*}}(id)$.
   2. Compute $w = T^{-1}(v)$ if $b = 0$; otherwise, compute $w = (T^{-1})^{\mathsf{T}}(v)$.
   3. Output the output $\mathsf{P}_b(v)$ and terminate.
3. If $ipl_1 \neq i^*$,
   1. Set $T = \mathsf{SampleFullRank}(1^\lambda; F(K, id))$.
   2. Compute $w = T^{-1}(v)$ if $b = 0$; otherwise, compute $w = (T)^{\mathsf{T}}(v)$.
   3. Output 1 if $w \in \mathsf{A}_{\mathsf{Can}}$ if $b = 0$ and if $w \in \mathsf{A}^\perp_{\mathsf{Can}}$ if $b = 1$. Otherwise, output 0.

Hyb_9 : We now sample $\mathsf{OPReRand}$ as $\mathsf{OPReRand} \leftarrow i\mathcal{O}(\mathsf{PReRand}'')$.

$\mathsf{PReRand}''_K(id, s)$
Hardcoded: $K, ptk, sk, isk, i^*, M_{K',\mathsf{CT}_{i^*}}$

1. Check if $\mathsf{RPKE.Test}(ptk, id) = 1$. Otherwise, output $\perp$ and terminate.
2. Compute $pl = \mathsf{RPKE.Dec}(sk, id)$.
3. $id' = \mathsf{RPKE.ReRand}(pk, id; s)$.

4. If $pl = i^*$, set $T_1 = M_{K',\mathsf{CT}_{i^*}}(id)$. Otherwise, $T_1 = \mathsf{SampleFullRank}(1^\lambda; F(K, id))$.
5. If $pl = i^*$, set $T_2 = M_{K',\mathsf{CT}_{i^*}}(id')$. Otherwise, $T_2 = \mathsf{SampleFullRank}(1^\lambda; F(K, id'))$.
6. Output $T_2 \cdot T_1^{-1}$.

Lemma 24. $\mathsf{Hyb}_0 \approx \mathsf{Hyb}_1$

Proof. The result follows from the pseudorandom public encryption key property of RPKE. □

Lemma 25. $\mathsf{Hyb}_1 \approx \mathsf{Hyb}_2$

Proof. The result follows from the indistinguishability of simulated testing keys and honest testing keys of RPKE. □

Lemma 26. $\mathsf{Hyb}_2 \approx \mathsf{Hyb}_3$.

Proof. The result follows by the computational zero knowledge security of NIZK. □

Lemma 27. $\mathsf{Hyb}_3 \approx \mathsf{Hyb}_4$.

Proof. The result follows from the CPA-security of RPKE. □

Lemma 28. $\mathsf{Hyb}_4 \approx \mathsf{Hyb}_5$.

Proof. This result follows essentially from $0 \to 1$ unclonability of subspace states and by strong rerandomization correctness of RPKE. This proof follows similarly to Lemma 2. □

Lemma 29. $\Pr[\mathsf{Hyb}_6 = 1] \geq \frac{\Pr[\mathsf{Hyb}_5 = 1]}{k}$.

Proof. The adversary outputs $k+1$ forged banknotes, thus $|\mathsf{INDICES}| = k+1$. However, all the values in INDICES are between 1 and k. Thus, there exists i^{**} such that i^{**} appears at least twice in INDICES, and the random i^* satisfies $i^* = i^{**}$ with probability $1/k$. □

Lemma 30. $\mathsf{Hyb}_6 \approx \mathsf{Hyb}_7$.

Proof. Follows similarly to Lemma 4. □

Lemma 31. $\mathsf{Hyb}_7 \approx \mathsf{Hyb}_8$.

Proof. Observe that by correctness of the $i\mathcal{O}$ used to sample $\mathsf{P}_1, \mathsf{P}_2$, the programs $\mathsf{PMem}', \mathsf{PMem}''$. Thus, the result follows by the security of $i\mathcal{O}$ used to sample OPMem. □

Lemma 32. $\mathsf{Hyb}_8 \approx \mathsf{Hyb}_9$.

Proof. This follows similarly to Lemma 6: Note that due to strong rerandomization correctness of RPKE, there does not exist id and s such that $\mathsf{RPKE.Test}(tpk, id) = 1$ but $\mathsf{RPKE.Dec}(sk, id) \neq \mathsf{RPKE.Dec}(sk, \mathsf{RPKE.ReRand}(pk, id; s))$. Thus, in $\mathsf{PReRand}'$, we have that pl and pl' always have the same value. Thus, the tests $pl =^? i^*$ and $pl' =^? i^*$ will always go to the same case, and as a result the factor T^* will be cancelled out when $T_2 \cdot T_1^{-1}$ is computed. Hence, $\mathsf{PReRand}'$ and $\mathsf{PReRand}''$ have exactly the same functionality, and the result follows by security of $i\mathcal{O}$. □

Lemma 33. $\Pr[\mathsf{Hyb}_9 = 1] \leq \mathsf{negl}(\lambda)$.

Proof. The result follows by the $1 \to 2$ unclonability of subspace states. The proof follows similarly to Lemma 7: we can show that from the forged banknotes of the adversary, we can convert two of them to $|A^*\rangle$, whereas the experiment can be simulated given a single copy $|A^*\rangle$. This means cloning $|A^*\rangle$, which is a contradiction by Theorem 7. □

9 Quantum Voting Schemes

9.1 Definitions

In this section, we recall security notions for voting schemes. We will be working in the *universal verifiability* setting: The users will cast their votes by posting them on a public bulletin board, and anyone can verify the validity of a cast vote using the public-key. Further, we will be working in the common (uniformly) random string model and will allow voting tokens to be quantum states (however, voting is done by simply posting a classical string on the bulletin board).

The first requirement we have is *correctness*. In the quantum voting tokens setting, we extend the correctness notion to also apply against malicious voting tokens and verification keys. A voter will be assured that their vote will be correctly verified by the public.

Definition 16 (Correctness against Malicious Keys and Tokens)**.** *For any efficient algorithm $\mathcal{B}$ and candidate choice c,*

$$\Pr\left[b = 0 \vee b' = 1 : \begin{array}{c} \mathsf{R}, vk \leftarrow \mathcal{B}(vk, mk, tk) \\ b \leftarrow \mathsf{QV.VerifyVotingToken}(vk, \mathsf{R}) \\ vo \leftarrow \mathsf{QV.Vote}(\mathsf{R}, c) \\ b' \leftarrow \mathsf{QV.VerifyVote}(vk, vo, c) \end{array}\right] = 1.$$

We also introduce the notion of privacy against authorities, where privacy of votes holds against everyone, including the voting authority that creates the voting tokens. For our privacy requirement, we require that a vote from a token that the adversary has seen is indistinguishable from a vote from a fresh voting token. By a simple hybrid argument, this easily implies indistinguishability of votes of any two voters. In fact, we require privacy even for tokens created by the adversary.

Definition 17 (Privacy)**.** *Consider the following game between a challenger and a stateful adversary $\mathcal{A}$.*

$\underline{\mathsf{QVS} - \mathsf{PRIVACY}_{\mathcal{A}}(1^\lambda)}$

1. *Sample $crs \leftarrow \{0,1\}^{q(\lambda)}$.*
2. *Submit crs to the adversary $\mathcal{A}$.*
3. *Adversary $\mathcal{A}$ outputs keys vk, mk, a register R_0 and candidate choice c.*
4. *Run $\mathsf{QV.VerifyVotingToken}(vk, \mathsf{R}_0)$. If it fails, output 0 and terminate.*
5. *Sample $\mathsf{R}_1 \leftarrow \mathsf{QV.GenVotingToken}(mk)$.*
6. *Run $\mathsf{QV.VerifyVotingToken}(vk, \mathsf{R}_1)$. If it fails, output 0 and terminate.*
7. *Sample $b \leftarrow \{0,1\}$.*
8. *Sample $vo \leftarrow \mathsf{QV.Vote}(\mathsf{R}_b, c)$.*
9. *Submit vo to the adversary $\mathcal{A}$.*
10. *Adversary $\mathcal{A}$ outputs a bit b'.*
11. *Output 1 if and only if $b' = b$.*

We say that the quantum voting scheme QV *satisfies* privacy *if for any QPT adversary* $\mathcal{A}$*, we have*

$$\Pr\left[\mathsf{QVS}-\mathsf{PRIVACY}_{\mathcal{A}}(1^\lambda)=1\right]\leq\frac{1}{2}+\mathsf{negl}(\lambda).$$

We remark that our scheme will actually satisfy the stronger notion where the voting token (or a fresh token) is returned to the adversary (without being cast, as a quantum state), and the adversary still will not be able to win the privacy game. This trivially implies the definition above.

The final essential property we require is uniqueness: adversaries should not be able to more than once.

Definition 18 (Uniqueness)**.** *Consider the following game between a challenger and an adversary* $\mathcal{A}$*.*

$\underline{\mathsf{QVS}-\mathsf{UNIQUE}_{\mathcal{A}}(1^\lambda)}$

1. *Sample* $vk, mk \leftarrow \mathsf{QV.Setup}(1^\lambda)$ *and submit* vk *to* $\mathcal{A}$*.*
2. ***Token Query Phase:*** *For multiple rounds,* $\mathcal{A}$ *queries for a voting token and the challenger executes* $\mathsf{R}_{\mathsf{token}} \leftarrow \mathsf{QV.GenVotingToken}(mk)$ *and submits* $\mathsf{R}_{\mathsf{token}}$ *to the adversary. Let* k *be the number of queries made by the adversary.*
3. $\mathcal{A}$ *outputs* $k+1$ *cast-votes* $(vo_i = (c_i, s_i, \ell_i))_{i\in[k+1]}$ *where* c_i *denotes the candidate choice and* ℓ_i *denotes the tag*[18]*.*
4. *The challenger tests* $\mathsf{QV.Verify}(vk, (c_i, s_i, \ell_i)) = 1$ *for* $i \in [k+1]$*. It also checks that all tags* ℓ_i *are unique. It outputs* 1 *if all the tests pass; otherwise, it outputs* 0*.*

We say that the quantum voting scheme QV *satisfies* uniqueness security *if for any QPT adversary* $\mathcal{A}$*,*

$$\Pr\left[\mathsf{PKQM}-\mathsf{CF}_{\mathcal{A}}(1^\lambda)=1\right]\leq\mathsf{negl}(\lambda).$$

9.2 Construction

In this section, we give our universally verifiable quantum voting scheme construction, in the common random string model. In our scheme, verifying a voting token automatically rerandomizes it. Our scheme will be similar to our untraceable quantum money scheme (Section 8), with the main difference being that it will contain $2\cdot\lambda$ money states in voting tokens.

We assume the existence of the same primitives as in Section 8.

$\underline{\mathsf{QV.Setup}(crs)}$

1. Parse $crs_{nizk}||pk = crs$ with $|crs_{nizk}| = q_1(\lambda)$ and $|pk| = q_2(\lambda)$.
2. Sample $K \leftarrow F.\mathsf{Setup}(1^\lambda)$.
3. Sample random tape r_{iO} and compute $\mathsf{OPMem} = i\mathcal{O}(\mathsf{PMem}; r_{iO})$ where PMem is the following program.

[18] A voter assigned serial number that is part of the cast-vote.

$\mathsf{PMem}_K(id, (v_i)_{i\in[2\cdot\lambda]}, b)$

Hardcoded: K

1. Parse $r_1||\ldots||r_{2\lambda} = F(K, id)$ into $2\cdot\lambda$ equal size strings.
2. For $i \in [2\cdot\lambda]$,
   1. Set $T_i = \mathsf{SampleFullRank}(1^\lambda; r_i)$.
   2. Compute $w_i = T^{-1}(v)$ if $b_i = 0$; otherwise, compute $w_i = T^{\mathsf{T}}(v_i)$.
   3. Check if $w_i \in \mathsf{A}_{\mathsf{Can}}$ if $b_i = 0$ and if $w_i \in \mathsf{A}^{\perp}_{\mathsf{Can}}$ if $b_i = 1$.
3. If all the checks pass, output 1. Otherwise, output 0.

4. Sample $ptk \leftarrow \mathsf{RPKE.SimulateTestKey}(pk)$.
5. Sample $\mathsf{OPReRand} \leftarrow i\mathcal{O}(\mathsf{PReRand})$ where $\mathsf{PReRand}$ is the following program[19].

$\mathsf{PReRand}_K(id, s)$

Hardcoded: K, pk, ptk

1. Check if $\mathsf{RPKE.Test}(ptk, id) = 1$. If not, output $\perp$ and terminate.
2. $id' = \mathsf{RPKE.ReRand}(pk, id; s)$.
3. Parse $r_1||\ldots||r_{2\lambda} = F(K, id)$ into $2\cdot\lambda$ equal size strings.
4. Parse $r'_1||\ldots||r'_{2\lambda} = F(K, id')$ into $2\cdot\lambda$ equal size strings.
5. For $i \in [2\cdot\lambda]$,
   1. Set $T_{i,1} = \mathsf{SampleFullRank}(1^\lambda; r_i)$.
   2. Set $T_{i,2} = \mathsf{SampleFullRank}(1^\lambda; r'_i)$.
   3. Set $T_{i,\mathsf{out}} = T_{i,2} \cdot T^{-1}_{i,1}$.
6. Output $(T_{i,\mathsf{out}})_{i\in[2\cdot\lambda]}$

6. Sample the proof $\pi \leftarrow \mathsf{NIZK.Prove}(crs_{nizk}, x = \mathsf{OPMem}, w = (K, r_{iO}))$.
7. Set $vk = (\mathsf{OPMem}, \mathsf{OPReRand}, \pi)$.
8. Set $mk = (K, pk)$.
9. Output vk, mk.

$\mathsf{QV.GenVotingToken}(mk)$

1. Parse $(K, pk) = mk$.
2. Sample $ct \leftarrow \mathsf{RPKE.Enc}(pk, 0^\lambda)$.
3. Parse $r_1||\ldots||r_{2\lambda} = F(K, ct)$ into $2\cdot\lambda$ equal size strings.
4. For $i \in [2\cdot\lambda]$,

[19] We note that if we make the stronger assumption of RPKE with *all-accepting simulatable testing keys*, we can actually remove the $\mathsf{RPKE.Test}$ line from the construction.

1. $T_i = \mathsf{SampleFullRank}(1^\lambda; r_i)$.
2. Set $|\$_i\rangle = \sum_{v \in \mathsf{A}_{\mathsf{Can}}} |T_i(v)\rangle$.

5. Output $ct, \bigotimes_{i\in[2\cdot\lambda]} |\$_i\rangle$.

$\underline{\mathsf{QV.VerifyVotingToken}(crs, vk, \mathsf{R})}$

1. Parse $crs_{nizk}||pk = crs$ with $|pk| = q_2(\lambda)$.
2. Parse $(\mathsf{OPMem}, \mathsf{OPReRand}, \pi) = vk$.
3. Check if $\mathsf{NIZK.Verify}(crs_{nizk}, x = \mathsf{OPMem}, \pi) = 1$. If not, output 0 and terminate.
4. Parse $(id, \mathsf{R}') = \mathsf{R}$.
5. Run OPMem coherently on $id, \mathsf{R}', 0^{2\cdot\lambda}$. Check if the output is 1, and then rewind.
6. Apply QFT to R'.
7. Run OPMem coherently on $id, \mathsf{R}', 1^{2\cdot\lambda}$. Check if the output is 1, and then rewind.
8. If any verification failed, output 0 and terminate.
9. Sample $s \leftarrow \{0,1\}^{p_3(\lambda)}$.
10. $(T_1, \ldots, T_{2\cdot\lambda}) = \mathsf{OPReRand}(id, s)$.
11. Compute $id' = \mathsf{RPKE.ReRand}(pk, id; s)$.
12. For $i \in [2\cdot\lambda]$, apply the linear map $T_i : \mathbb{F}_2^\lambda \to \mathbb{F}_2^\lambda$ coherently to the i-th part of the register R'.
13. Run OPMem coherently on $id, \mathsf{R}', 0^{2\cdot\lambda}$. Check if the output is 1, and then rewind.
14. Apply QFT to R'.
15. Run OPMem coherently on $id', \mathsf{R}', 1^{2\cdot\lambda}$. Check if the output is 1, and then rewind.
16. If any verification failed, output 0. Otherwise, output 1.

$\underline{\mathsf{QV.Vote}(\mathsf{R}, c)}$

1. Sample $r \in \{0,1\}^\lambda$.
2. Parse $(id, (\mathsf{R}'_i)_{i\in[2\cdot\lambda]}) = \mathsf{R}$.
3. For $i \in [2\cdot\lambda]$: If $(c||r)_i = 0$, measure R'_i in the computational basis to obtain v_i. If $(c||r)_i = 1$, measure R'_i in the Hadamard basis (i.e. QFT-and-measure) to obtain v_i.
4. Output $c, (id, (v_i)_{i\in[2\cdot\lambda]}), r$.

$\underline{\mathsf{QV.VerifyCastVote}(\mathsf{R}, c, vo)}$

1. Parse $(\mathsf{OPMem}, \mathsf{OPReRand}, \pi) = vk$.
2. Parse $(id, (v_i)_{i\in[2\cdot\lambda]}, r) = vo$.
3. Check if $\mathsf{OPMem}(id, (v_i)_{i\in[2\cdot\lambda]}, c||r)$.

Theorem 20. QV *satisfies correctness against malicious tokens and keys.*

Proof. By the same argument as in the projectiveness proof of our quantum money schemes Section 7.1, we can show that the token verification algorithm of our scheme projects onto a perfect token. Thus, the result follows. □

Theorem 21. QV *satisfies privacy.*

Proof. This follows similarly to the untraceability of our quantum money scheme (Section 8.1). □

Theorem 22. QV *satisfies uniqueness security.*

Proof. Through the same proof (generalized to multiple subspace states) as the unclonability proof of our quantum money scheme (Section 8.2), we can reduce the uniqueness security of our scheme to the direct product hardness of subspace states (Theorem 8), since creating $k+1$ valid votes with all different tags will amount to adversary finding v, w with $v \in A$ and $w \in A^{\perp}$. □

10 Acknowledgements

Alper Çakan was supported by the following grants of Vipul Goyal: NSF award 1916939, DARPA SIEVE program, a gift from Ripple, a DoE NETL award, a JP Morgan Faculty Fellowship, a PNC center for financial services innovation award, and a Cylab seed funding award.

References

[Aar09] Scott Aaronson. Quantum copy-protection and quantum money. In *2009 24th Annual IEEE Conference on Computational Complexity*, pages 229–242, 2009.

[Aar16] Scott Aaronson. The complexity of quantum states and transformations: From quantum money to black holes, 2016.

[AC12] Scott Aaronson and Paul Christiano. Quantum money from hidden subspaces. In *Proceedings of the Forty-Fourth Annual ACM Symposium on Theory of Computing*, STOC '12, page 41–60, New York, NY, USA, 2012. Association for Computing Machinery.

[Adi08] Ben Adida. Helios: Web-based open-audit voting. In *USENIX security symposium*, volume 17, pages 335–348, 2008.

[AGM21] Gorjan Alagic, Tommaso Gagliardoni, and Christian Majenz. Can you sign a quantum state? *Quantum*, 5:603, December 2021.

[AMR20] Gorjan Alagic, Christian Majenz, and Alexander Russell. Efficient simulation of random states and random unitaries. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part III 39*, pages 759–787. Springer, 2020.

[BBC+14] Boaz Barak, Nir Bitansky, Ran Canetti, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Obfuscation for evasive functions. In *Theory of Cryptography Conference*, pages 26–51. Springer, 2014.

[BDS23] Shalev Ben-David and Or Sattath. Quantum Tokens for Digital Signatures. *Quantum*, 7:901, January 2023.

[BS21] Amit Behera and Or Sattath. Almost public quantum coins. *QIP*, 2021.

[ÇG23] Alper Çakan and Vipul Goyal. Unclonable cryptography with unbounded collusions. Cryptology ePrint Archive, Paper 2023/1841, 2023.

[CL04] Jan Camenisch and Anna Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In *Annual international cryptology conference*, pages 56–72. Springer, 2004.

[CLLZ21] Andrea Coladangelo, Jiahui Liu, Qipeng Liu, and Mark Zhandry. Hidden cosets and applications to unclonable cryptography. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021*, pages 556–584, Cham, 2021. Springer International Publishing.

[FOO93] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In *Advances in Cryptology—AUSCRYPT'92: Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia, December 13–16, 1992 Proceedings 3*, pages 244–251. Springer, 1993.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, aug 1986.

[ILL89] Russell Impagliazzo, Leonid A Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 12–24, 1989.

[JLS18] Zhengfeng Ji, Yi-Kai Liu, and Fang Song. Pseudorandom quantum states. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part III 38*, pages 126–152. Springer, 2018.

[mon] The monero project. https://www.getmonero.org/. Accessed: 2024-11-06.

[MS10] Michele Mosca and Douglas Stebila. Quantum coins. *Error-correcting codes, finite geometries and cryptography*, 523:35–47, 2010.

[OST08] Tatsuaki Okamoto, Koutarou Suzuki, and Yuuki Tokunaga. Quantum voting scheme based on conjugate coding. *NTT Technical Review*, 6(1):1–8, 2008.

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for np from (plain) learning with errors. In *Annual International Cryptology Conference*, pages 89–114. Springer, 2019.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[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*, STOC '14, page 475–484, New York, NY, USA, 2014. Association for Computing Machinery.

[SW22] Or Sattath and Shai Wyborski. Uncloneable decryptors from quantum copy-protection, 2022.

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

[WZ17] Daniel Wichs and Giorgos Zirdelis. Obfuscating compute-and-compare programs under lwe. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 600–611, 2017.

[Zca] Zcash. https://z.cash. Accessed: 2024-11-06.

[Zha12] Mark Zhandry. How to construct quantum random functions. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pages 679–687, 2012.

[Zha19] Mark Zhandry. Quantum lightning never strikes the same state twice. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 408–438, Cham, 2019. Springer International Publishing.

A Additional Definitions

Definition 19 (Counterfeiting Security)**.** *Consider the following game between the challenger and an adversary $\mathcal{A}$.*

$\underline{\mathsf{PKQM}-\mathsf{CF}_{\mathcal{A}}(1^\lambda)}$

1. *Sample $vk, mk, tk \leftarrow \mathsf{Bank.Setup}(1^\lambda)$ and submit vk, tk to $\mathcal{A}$.*
2. ***Banknote Query Phase:*** *For multiple rounds, $\mathcal{A}$ queries for a banknote by sending a tag t. For each query, the challenger executes $\mathsf{R_{bn}} \leftarrow \mathsf{Bank.GenBanknote}(mk, t)$ and submits $\mathsf{R_{bn}}$ to the adversary. Let k be the number of queries made by the adversary.*
3. *$\mathcal{A}$ outputs a $(k+1)$-partite register $(\mathsf{R}_i)_{i\in[k+1]}$.*
4. *The challenger tests $\mathsf{Bank.Verify}(vk, \mathsf{R}_i) = 1$ for $i \in [k+1]$. It outputs 1 if all the tests pass; otherwise, it outputs 0.*

We say that the quantum money scheme Bank *satisfies* counterfeiting security *if for any QPT adversary $\mathcal{A}$,*

$$\Pr\Big[\mathsf{PKQM}-\mathsf{CF}_{\mathcal{A}}(1^\lambda) = 1\Big] \leq \mathsf{negl}(\lambda).$$

A.1 Anonymity

In this section, we introduce various anonymity notions for public-key quantum money. Similar to previous work [AMR20, BS21], our anonymity definition either randomly permutes or does not touch the banknote registers, according to a random bit; and the adversary needs to predict which case it is. However, our definition is significantly stronger than previous work: The adversary receives the minting key at the beginning whereas in previous definitions it received it at the final step of the security game. This also means that unlike previous definitions, we do not need to have a banknote query phase: the adversary can mint its own challenge banknotes.

Definition 20 (Anonymity)**.** *Consider the following game between a challenger and an adversary $\mathcal{A}$.*

$\underline{\mathsf{PKQM}-\mathsf{ANON}_{\mathcal{A}}(1^\lambda)}$

1. *Sample $vk, mk, tk \leftarrow \mathsf{Bank.Setup}(1^\lambda)$.*
2. *Submit vk, mk to $\mathcal{A}$.*
3. *Adversary $\mathcal{A}$ outputs a value k and a (possibly entangled) k-partite register $(\mathsf{R}_i)_{i\in[k]}$.*
4. *Run $\mathsf{Bank.Verify}(vk, \mathsf{R}_i)$ for $i \in [k]$. If any of them fails, output 0 and terminate.*
5. *Run $\mathsf{Bank.ReRandomize}(vk, \mathsf{R}_i)$ for $i \in [k]$.*
6. *Sample a permutation $\pi : [k] \to [k]$ and a bit b.*
7. *If $b = 0$, submit $(\mathsf{R}_i)_{i\in[k]}$; otherwise, submit $(\mathsf{R}_{\pi(i)})_{i\in[k]}$ to the adversary $\mathcal{A}$.*
8. *Adversary $\mathcal{A}$ outputs a bit b'.*
9. *Output 1 if and only if $b' = b$.*

We say that the quantum money scheme Bank *satisfies* anonymity *if for any QPT adversary $\mathcal{A}$, we have*

$$\Pr\left[\mathsf{PKQM}-\mathsf{ANON}_{\mathcal{A}}(1^\lambda) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

B Omitted Proofs

B.1 Proof of Lemma 4

This lemma follows by a (standard) PRF puncturing argument where we create hybrids for all strings id. In each hybrid j, we will switch to the new serial number - linear map association for all id satisfying $id < j$. To go from j to $j + 1$, we do a small number of subhybrids where we first puncture the PRF key at j, replace the associated T with a truly random sample. We also puncture K'. This allows us to switch over to the new mapping, and then depuncture K' and K.

B.2 Proof of Lemma 2

To prove this, first we do the same argument as in Appendix B.1 to switch to a completely new association between serial numbers and linear maps T for any serial number id that decrypts to a value not in $[k]$. Then, by the same argument as in the rest of the unclonability proof and as in Lemma 7, we can show that if the adversary forges any valid banknote whose serial number decrypts to a value $\notin [k]$, we can extract a subspace state $|A^*\rangle$, with using only $i\mathcal{O}(A^*), i\mathcal{O}((A^*)^\perp)$. This is a contradiction to the unlearnability (or $0 \to 1$ unclonability) of subspaces.

C Remark on Representing the Public Key

A remark here is in order. In our RPKE constructions, we will have that the public key is a matrix in $\mathbb{Z}_q^{m\times n}$, not a binary string. However, we can always represent such a matrix entrywise, where each number $\mathbb{Z}_q$ can be publicly and canonically represented as a binary string of length $\lceil \log_2(q) \rceil$ (any extra binary strings can be rolled over and assigned to unique values in $\mathbb{Z}_q$), and we can efficiently go back and forth between any value in $\mathbb{Z}_q$ and any binary string in $\{0,1\}^{\lceil \log_2(q) \rceil}$. Thus, without loss of generality we can assume that the public key is a binary string.