Searching for ELFs in the Cryptographic Forest

Marc Fischlin Felix Rohrbach

Cryptoplexity, Technische Universität Darmstadt, Germany
www.cryptoplexity.de
{marc.fischlin, felix.rohrbach}@cryptoplexity.de

Abstract. Extremely Lossy Functions (ELFs) are families of functions that, depending on the choice during key generation, either operate in injective mode or instead have only a polynomial image size. The choice of the mode is indistinguishable to an outsider. ELFs were introduced by Zhandry (Crypto 2016) and have been shown to be very useful in replacing random oracles in a number of applications.

One open question is to determine the minimal assumption needed to instantiate ELFs. While all constructions of ELFs depend on some form of exponentially-secure public-key primitive, it was conjectured that exponentially-secure secret-key primitives, such as one-way functions, hash functions or one-way product functions, might be sufficient to build ELFs. In this work we answer this conjecture mostly negative: We show that no primitive, which can be derived from a random oracle (which includes all secret-key primitives mentioned above), is enough to construct even moderately lossy functions in a black-box manner. However, we also show that (extremely) lossy functions themselves do not imply public-key cryptography, leaving open the option to build ELFs from some intermediate primitive between the classical categories of secret-key and public-key cryptography.

1 Introduction

Extremely lossy functions, or short ELFs, are collections of functions that support two modes: the injective mode, in which each image has exactly one preimage, and the lossy mode, in which the function merely has a polynomial image size. The mode is defined by a seed or public key pk which parameterizes the function. The key pk itself should not reveal whether it describes the injective mode or the lossy mode. In case the lossy mode does not result in a polynomially-sized image, but the function compresses by at least a factor of 2, we will speak of a (moderately) lossy function (LF).

Extremely lossy functions were introduced by Zhandry [Zha16, Zha19] to replace the use of the random oracle model in some cases. The random oracle model (ROM) [BR93] introduces a truly random function to which all parties have access to. This random function turned out to be useful in modeling hash functions for security proofs of real-world protocols. However, such a truly random function clearly does not exist in reality and it has been shown that no hash function can replace such an oracle without some protocols becoming insecure [CGH98]. Therefore, a long line of research aims to replace the random oracle by different modeling of hash functions, e.g., by the notion of correlation intractability [CGH98] or by Universal Computational Extractors (UCEs) [BHK13]. However, all these attempts seem to have their own problems: Current constructions of correlation intractability require extremely strong assumptions [CCR16], while for UCEs, it is not quite clear which versions are instantiable [BFM14, BST16]. Extremely lossy functions, in turn, can be built from relatively standard assumptions.

Indeed, it turns out that extremely lossy functions are useful in removing the need for a random oracle in many applications: Zhandry shows it can be used to generically boost selective security to adaptive security in signatures and identity-based encryption, construct a hash function which is output intractable, point obfuscation in the presence of auxiliary information and many more [Zha16, Zha19]. Agrikola, Couteau and Hofheinz [ACH20] show that ELFs can be used to construct probabilistic indistinguishability obfuscation from only polynomially-secure indistinguishability obfuscation. In 2022, Murphy, O'Neill and Zaheri [MOZ22] used ELFs to give full instantiations of the OAEP and Fujisaki-Okamoto transforms. Recently, Brzuska et al. [BCE$^+$23] improve on the instantiation of the Fujisaki-Okamoto transform and instantiate the hash-then-evaluate paradigm for pseudorandom functions using ELFs.

While maybe not as popular as their extreme counterpart, moderately lossy functions have their own applications as well: Braverman, Hassidim and Kalai [BHK11] build leakage-resistant pseudo-entropy functions from lossy functions, and Dodis, Vaikuntanathan and Wichs [DVW20] use lossy functions to construct extractor-dependent extractors with auxiliary information.

1.1 Our Contributions

One important open question for extremely lossy functions, as well as for moderately lossy functions, is the minimal assumption to build them. The constructions presented by Zhandry are based on the exponential security of the decisional Diffie-Hellman problem, but he conjectures that public-key cryptography should not be necessary and suggests for future work to try to construct ELFs from exponentially-secure symmetric primitives (As Zhandry shows as well in his work, polynomial security assumptions are unlikely to be enough for ELFs[1]). Holmgren and Lombardi [HL18] wondered whether their definition of one-way product functions might suffice to construct ELFs.

For moderately lossy functions, the picture is quite similar: While all current constructions require (polynomially-secure) public-key cryptography, it is generally assumed that public-key cryptography should not be necessary for them and that private-key assumptions should suffice (see, e.g., [QWW21]).

In this work, we answer the questions about building (extremely) lossy functions from symmetric-key primitive mostly negative: There exists no fully-black box construction of extremely lossy functions, or even moderately lossy functions, from a large number of primitives, including exponentially-secure one-way functions, exponentially-secure collision resistant hash functions or one-way product functions. Indeed, any primitive that exists unconditionally relative to a random oracle is not enough. We will call this family of primitives *Oraclecrypt*, in reference to the famous naming convention by Impagliazzo [Imp95], in which Minicrypt refers to the family of primitives that can be built from one-way functions in a black-box way.

Note that most of the previous reductions and impossibility results, such as the renowned result about the impossibility of building key exchange protocols from black-box one-wayness [IR89], are in fact already cast in the Oraclecrypt world. We only use this term to emphasize that we also rule out primitives that are usually not included in Minicrypt, like collision resistant hash functions [Sim98].

On the other hand, we show that public-key primitives might not strictly be needed to construct ELFs or moderately lossy functions. Specifically, we show that no fully black-box construction of key agreement is possible from (moderately) lossy functions, and extend this result to prevent any fully black-box construction even from extremely lossy functions (for a slightly weaker setting, though). This puts the primitives lossy functions and extremely lossy functions into the intermediate area between the two classes Oraclecrypt and Public-Key Cryptography.

Finally, we discuss the relationship of lossy functions to hard-on-average problems in SZK, the class of problems that have a statistical zero-knowledge proof. We see hard-on-average SZK as a promising minimal assumption to build lossy functions from – indeed, it is already known that hard-on-average SZK problems follow from lossy functions with sufficient lossiness. While we leave open the question of building such a

[1]ELFs can be distinguished efficiently using a super-logarithmic amount of non-determinism. It is consistent with our knowledge, however, that NP with an super-logarithmic amount of non-determinism is solvable in polynomial time while polynomially-secure cryptographic primitives exist. Any construction of ELFs from polynomially-secure cryptographic primitives would therefore change our understanding of NP-hardness.

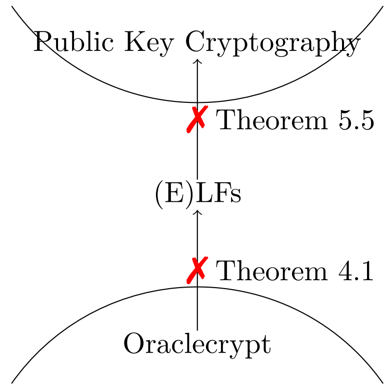

Figure 1: We show both an oracle separation between Oraclecrypt and (E)LFs as well as (E)LFs and key agreement.

construction for future work, we give a lower bound for hard-on-average SZK problems that might be of independent interest, showing that hard-on-average SZK problems cannot be built from any Oraclecrypt primitive in a fully black-box way. While this is already known for some primitives in Oraclecrypt [BD19], these results do not generalize to all Oraclecrypt primitives as our proof does.

Note that all our impossibility results only rule out black-box constructions, leaving the possibility of future non-black-box constructions. However, while there is a growing number of non-black-box constructions in the area of cryptography, the overwhelming majority of constructions are still black-box constructions. Further, as all mentioned primitives like exponentially-secure one-way functions, extremely lossy functions or key agreement might exist unconditionally, ruling out black-box constructions is the best we can hope for to show that a construction probably does not exist.

1.2 Our Techniques

Our separation of Oraclecrypt primitives and extremely/moderately lossy functions is based on the famous oracle separation by Impagliazzo and Rudich [IR89]: We first introduce a strong oracle that makes sure no complexity-based cryptography exists unconditionally, and then add an independent random oracle that allows for specific cryptographic primitives (specifically, all Oraclecrypt primitives) to exist again. We then show that relative to these oracles, (extremely) lossy functions do not exist by constructing a distinguisher between the injective and lossy mode for any candidate construction. A key ingredient here is that we can identify the *heavy queries* in a lossy function with high probability with just polynomially many queries to the random oracle, a common technique used for example in the work by Bitansky and Degwekar [BD19]. Finally, we use the two-oracle technique by Hsiao and Reyzin [HR04] to fix a set of oracles. We note that our proof technique is similar to a technique in the work by Pietrzak, Rosen and Segev to show that the lossiness of lossy functions cannot be increased well in a black-box way [PRS12]. Our separation result for SZK, showing that primitives in Oraclecrypt may not suffice to derive hard problems in SZK, follows a similar line of reasoning.

Our separation between lossy functions and key agreement is once more based on the work by Impagliazzo and Rudich [IR89], but this time using their specific result for key agreement protocols. Similar to the techniques in [GHMM18], we try to compile out the lossy function to be then able to apply the Impagliazzo-Rudich adversary: We first show that one can build (extremely) lossy function oracles relative to a random oracle (where the lossy function itself is efficiently computable via oracle calls, but internally makes an exponentially number of random oracle evaluations). The heart of our separation is then a simulation lemma showing that any efficient game relative to our (extremely) lossy function oracle can be simulated efficiently and sufficiently close given only access to a random oracle. Here, sufficiently close means an inverse polynomial gap between the two cases but where the polynomial can be set arbitrarily. Given this we can apply the key agreement separation result of Impagliazzo and Rudich [IR89], with a careful argument that the simulation gap does not infringe with their separation.

1.3 Related Work

Lossy Trapdoor Functions. Lossy trapdoor functions were defined by Peikert and Waters in [PW08, PW11] who exclusively considered such functions to have a trapdoor in injective mode. Whenever we talk about lossy functions in this work, we refer to the moderate version of extremely lossy functions which does not necessarily have a trapdoor. The term extremely lossy function (ELFs) is used as before to capture strongly compressing lossy functions, once more without requiring a trapdoor for the injective case.

Targeted Lossy Functions. Targeted lossy functions were introduced by Quach, Waters and Wichs [QWW21] and are a relaxed version of lossy functions in which the lossiness only applies to a small set of specified inputs. The motivation of the authors is the lack of progress in creating lossy functions from other assumptions than public-key cryptography. Targeted lossy functions, however, can be built from Minicrypt assumptions, and, as the authors show, already suffices for many applications, such as construct extractor-dependent extractors with auxiliary information and pseudo-entropy functions. Our work very much supports this line of research, as it shows that any further progress in creating lossy functions from Minicrypt/Oraclecrypt assumptions is unlikely (barring some construction using non-black-box techniques) and underlines the need of such a relaxation for lossy functions, if one wants to build them from Minicrypt assumptions.

Amplification of Lossy Functions. Pietrzak, Rosen and Segev [PRS12] show that it is impossible to improve the relative lossiness of a lossy function in a black-box way by more than a logarithmic amount. This translates into another obstacle in building ELFs, even when having access to a moderately lossy function. Note that this result strengthens our result, as we show that even moderately lossy functions cannot be built from anything in Oraclecrypt.

2 Technical Overview

In this chapter, we will give an overview about our two main theorems of this paper and techniques used to prove them. The (extremely) lossy functions consists of two algorithms, $\mathsf{Gen}(1^\lambda, \mathsf{mode})$ for generating a public key in the input mode $\mathsf{mode} = \mathsf{loss}$ or $\mathsf{mode} = \mathsf{inj}$, and $\mathsf{Eval}(\mathsf{pk}, \cdot)$ taking a public key and some input of size $\mathrm{in}(\lambda)$ and outputting a value. Here, the function Eval is (extremely) lossy if the key pk has been generated in lossy mode, and injective if the key has been generated in injective mode. In most of our results we give both algorithms also access to one or more oracles.

2.1 No (E)LFs in Oraclecrypt

Our first results says that one cannot build lossy functions, and thus neither extremely lossy functions, from any primitive that exists (unconditionally) relative to a random oracle:

Theorem 4.1 (informal). *There exists no fully black-box construction of lossy functions from any Oraclecrypt primitive.*

Our proof for this Theorem follows a proof idea by Pietrzak, Rosen and Segev [PRS12], which they used to show that lossy functions cannot be amplified well, i.e., one cannot build a lossy function which is very compressing in the lossy mode from a lossy function that is only slightly compressing in the lossy mode. We adapt the idea to show that lossy functions cannot be built unconditionally from a random oracle. Note that some technical details of our construction differ from [PRS12], though. For example, we show the result relative to a modified PSPACE oracle instead of an EXPTIME oracle.

We will now explain how the proof for our result works. First note that our result only holds for fully black-box constructions. The reason for this is that we use the two-oracle technique by Hsiao and Reyzin [HR04]. This approach considers a "constructive" oracle which allows to build the primitive in question, and a "breaking" oracle which allows to break any construction of the other primitive which is only based on the first oracle. Presenting such oracles provides a fully black-box separation of the two primitives.

In our case, the "constructive" oracle $\mathcal{O}$ in the two-oracle technique is a random oracle, supporting the implementation of any Oraclecrypt primitive. For the "breaking" oracle, we introduce a PSPACE-oracle with a twist: The PSPACE-oracle will itself have oracle access to another, independent random oracle $\mathcal{O}'$. The reason for this extra random oracle will become apparent soon. For now we remark that this independent random oracle $\mathcal{O}'$ does not invalidate the security of the Oraclecrypt primitives relative to $\mathcal{O}$.

The main part of our proof consists of showing that we can use the breaking oracle $\mathrm{PSPACE}^{\mathcal{O}'}$ to distinguish $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$ for injective public key pk from $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$ for lossy public key pk . The approach is now to approximate the image size of $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$ by the image size of $\mathsf{Eval}^{\mathcal{O}'}(\mathsf{pk},\cdot)$. If we can indeed approximate this sufficiently close, then we can use the $\mathrm{PSPACE}^{\mathcal{O}'}$-oracle to give an estimate for the image size $\mathsf{Eval}^{\mathcal{O}'}(\mathsf{pk},\cdot)$ for the same oracle $\mathcal{O}'$ to get the answer for the original function $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$. However, the evaluation algorithm for oracle $\mathcal{O}$ and for oracle $\mathcal{O}'$ may be quite far apart. To get a sufficiently close approximation we need to "modify" $\mathcal{O}'$ to at least coincide on the more likely queries of the evaluation algorithm to $\mathcal{O}$. This is accomplished based on the the *heavy queries* technique of Bitansky et al. [BDV17, BD19]:

Definition 4.3 (heavy queries, informal). *We call a query q to $\mathcal{O}$* heavy *for given key* pk *and oracle $\mathcal{O}$ if, for a large fraction of $x \in \{0,1\}^{in(\lambda)}$, the evaluation* $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},x)$ *queries $\mathcal{O}$ about q at some point. We denote by Q_H the set of all heavy queries (for $\mathsf{pk},\mathcal{O}$).*

We note that we actually need to also take into account the queries of the key generating algorithm but omit this here in the overview. Remarkably, determining a superset of all heavy queries is easy:

Lemma 4.4 (informal). *We can compute in probabilistic polynomial-time (in λ) a set $\hat{Q}_H$ which contains all heavy queries of* $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$ *for $\mathsf{pk},\mathcal{O}$ with overwhelming probability.*

We simply run $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$ for a sufficiently high, but still polynomial number of random inputs $x \in \{0,1\}^{\mathrm{in}(\lambda)}$, recording all random oracle queries to get all heavy queries with probability at least $1-2^{-\mathrm{in}(\lambda)}$. The next step is to switch from oracle $\mathcal{O}$ to oracle $\mathcal{O}'$ which is available to the attacker via the $\mathrm{PSPACE}^{\mathcal{O}'}$ oracle. That is, we consider the evaluation function $\mathsf{Eval}^{\mathcal{O}'}(\mathsf{pk},\cdot)$ for oracle $\mathcal{O}'$ instead of $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$. In order to be still close to the original evaluation function for oracle $\mathcal{O}$, we let $\mathcal{O}'$ agree on the (computable set of) heavy queries with $\mathcal{O}$. Hence, we formally consider the oracle $\mathcal{O}'_H$ that is identical to $\mathcal{O}$ on all heavy queries.

The next lemma now states that ignoring the non-heavy queries in oracle $\mathcal{R} := \mathcal{O}'_H$ is inconsequential. More precisely, the lemma says that if we would also enforce consistency with $\mathcal{O}$ on non-heavy queries, denoted as oracle $\mathcal{R}'$, this would not change the size of the evaluation function noticeably:

Lemma 4.5 (informal). *For any oracle $\mathcal{R}'$ that is identical to $\mathcal{R}$ everywhere except for the non-heavy queries Q_G^{nonh} by key generation, i.e., $\mathcal{R}(q) = \mathcal{R}'(q)$ for any $q \notin Q_G^{\mathsf{nonh}}$, the image sizes of* $\mathsf{Eval}^{\mathcal{R}}(\mathsf{pk},\cdot)$ *and* $\mathsf{Eval}^{\mathcal{R}'}(\mathsf{pk},\cdot)$ *differ by at most $\frac{2^{in(\lambda)}}{10}$.*

We next show that we can use an oracle $\mathcal{O}'_H$ to distinguish lossy from injective keys: We know that for lossy keys, the image size of $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},x)$ over all inputs x should be at most $\frac{1}{2}\cdot 2^{\mathrm{in}(\lambda)}$, while for injective keys, the image size should be $2^{\mathrm{in}(\lambda)}$. As switching the oracle from $\mathcal{O}$ to $\mathcal{O}'_H$ only changes the image size slightly, there is still a large gap between the image size for injective keys and the image size for lossy keys. Therefore, if we can calculate the image size of $\mathsf{Eval}^{\mathcal{O}'_H}(\mathsf{pk}, \cdot)$ relative to $\mathcal{O}'_H$, we can decide between lossy keys and injective keys.

Now the reason why we need an augmented PSPACE oracle also becomes apparent. In general, we cannot compute the image size of a function relative to a randomly sampled function $\mathcal{O}'$ in PSPACE. As we have to compute the function for every input x to calculate the image size, a lazily sampled $\mathcal{O}'$ might be asked on exponentially many input queries, which we would have to save to answer consistently, which clearly is not in PSPACE anymore. To work around this problem, we add a randomly sampled $\mathcal{O}'$ to which the PSPACE oracle has full access. Now we only have to modify $\mathcal{O}'$ on the polynomially many heavy queries and then can calculate the image size of any function relative to $\mathcal{O}'_H$.

Lemma 4.6 (informal). *Given a superset of the heavy queries $\hat{Q}_H \supseteq Q_H$, we can decide correctly whether $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk}, \cdot)$ is lossy or injective with overwhelming probability.*

This lemma suffices to prove the theorem's statement.

2.2 No Key Agreement from (E)LFs

Our first result shows that lossy functions (and therefore also ELFs) are not part of Oraclecrypt, a set which contains symmetric primitives and also, for example, collision-resistant hash functions. However, this does not mean that ELFs inherently require public-key cryptography to build them. To show this, our second main theorem shows that lossy functions do not imply key agreement in a fully black-box way. In Chapter 5, we also extend this result to ELFs, but in this overview, we will only focus on lossy functions.

Theorem 5.5 (informal). *There exists no fully black-box construction of a secure key agreement protocol from lossy functions.*

Our result relies on the famous result by Impagliazzo and Rudich [IR89], showing that no key agreement can be built in a black-box way from one-way functions. More concretely, they construct an adversary that, relative to a random oracle and, e.g., a PSPACE oracle, any candidate key agreement protocol can be broken. We would like to deploy this adversary, but we cannot use it directly: We would need to show that the adversary also works relative to an oracle that allows for lossy functions to exist, which a random oracle does not (as we have seen in the previous result). To explain why this makes the problem more challenging than in the case of separating key agreement from a random oracle, consider the following possibility to use lossy functions in interactive protocols: Assume that Alice picks a secret bit b randomly and, depending on the value of b, either generates an injective or a lossy key pk, and sends this public key over to Bob. For Bob, as well as for an outsider attacker, it is infeasible to determine if pk is lossy or not. However, Bob is now able to perform computations *implicitly depending on Alice's bit* b via the $\mathsf{Eval}(\mathsf{pk}, \cdot)$ algorithm. Potentially, Bob also returns information about the outcomes of these evaluations back to Alice. In other words, a lossy function already implements a very skewed form of secret bit transfer in which the recipient only has implicit access to the transfered bit. We have to show that this still does not facilitate the task of designing a full-fledged key exchange protocol.

Therefore, we start by defining an oracle that implements a lossy function. The oracle consists of two functions $\mathsf{Gen}^{\Gamma,\Pi}(1^\lambda,)$ and $\mathsf{Eval}^{\Gamma,\Pi}(\mathsf{pk}, x)$, where the first function produces a public key either in injective or in lossy mode, and the second function lets us evaluate the lossy function for public key pk on input x. The oracle makes use of two random permutations Γ and Π, where Γ is considered an integral part of the oracle and cannot be accessed directly, while Π can be accessed directly. Note that our lossy function oracle is, by definition, efficient when queried as an oracle, but is internally inefficient , i.e., it makes exponentially many queries to Γ internally. Further, we define the oracle in such a way that it has another useful property for us: Every random public key, i.e., a key not generated by $\mathsf{Gen}^{\Gamma,\Pi}$, is a valid public key in injective mode with overwhelming probability.

We could now try to modify the adversary of Impagliazzo-Rudich to our new oracle. Instead, the heart of our result is the Simulation Lemma which shows, for any security game Game, access to the lossy function oracle can be simulated sufficiently close by an efficient algorithm (we call Wrap) that only has access to the random permutation Π. The quality of the simulation is not negligibly close, but instead determined by an inverse polynomial $\alpha(\lambda)$ and yields a statistical distance of at most $\alpha(\lambda)$.

Lemma 5.2 (Simulation Lemma, informal). *For any polynomial $p(\lambda)$ and any inverse polynomial $\alpha(\lambda)$ there exists an efficient algorithm* Wrap *such that for any efficient algorithm $\mathcal{A}$, any efficient experiment* Game *making at most $p(\lambda)$ calls to the oracle, the statistical distance between* $\mathsf{Game}^{\mathcal{A},(\mathsf{Gen}^{\Gamma,\Pi},\mathsf{Eval}^{\Gamma,\Pi},\Pi)}(1^\lambda)$ *and* $\mathsf{Game}^{\mathcal{A},\mathsf{Wrap}^{\Pi}}$ *is at most $\alpha(\lambda)$.*

We prove Lemma 5.2 with a series of game hops, starting with the original game and ending with the finished wrapper algorithm. The main idea is to, at some point, replace all keys by injective keys. Indeed, as it should be hard to distinguish between lossy keys and injective keys, this switch cannot change the underlying game significantly. The proof is the main technical challenge in this part.

Now, we have everything together to show that key agreement is not possible relative to our lossy function oracle. Let us consider a key agreement protocol between Alice and Bob. We show that we can wrap Alice and Bob using the algorithm defined in the Simulation Lemma, and still have a valid key agreement protocol. Now, as the wrapped parties in the protocol only rely on Π, we show that the Impagliazzo-Rudich adversary is successful here. However, by the Simulation Lemma, this means that the Impagliazzo-Rudich adversary is also successful for the original game (as we would have an efficient distinguisher otherwise).

3 Preliminaries

We say that two random variables X and Y, both indexed by security parameter λ, are computationally indistinguishable if for any probabilistic polynomial-time algorithm A

$$\left|\Pr\left[A(1^\lambda, X(1^\lambda) = 1\right] - \Pr\left[A(1^\lambda, Y(1^\lambda)) = 1\right]\right|$$

is negligible. We denote by $[X(1^\lambda)]$ the support of a random variable (or algorithm), i.e., the set of output values which are hit with positive probability.

We use the common notion for capturing the statistical distance between two random variables X and Y, both indexed by a security parameter λ. That is,

$$\mathsf{SD}\left(X,Y\right) := \frac{1}{2}\sum_{z}\left|\Pr\left[X(1^\lambda) = z\right] - \Pr\left[Y(1^\lambda) = z\right]\right|,$$

such that the statistical distance is a function of the parameter λ. We sometimes use the fact that for any sequences of random variables $X_1, X_2, \ldots, X_n$, all indexed by λ, it holds

$$\mathsf{SD}\left(X_1, X_n\right) \leq \sum_{i=0}^{n-1} \mathsf{SD}\left(X_i, X_{i+1}\right).$$

The latter allows us to perform game-hopping type of arguments and bound the statistical distance by the sum of the differences due to the individual hops.

3.1 Lossy Functions

A lossy function can be either injective or compressing, depending on the mode the public key pk has been generated with. The desired mode (inj or loss) is passed as argument to a (randomized) key generating algorithm Gen, together with the security parameter 1^λ. We sometimes write $\mathsf{pk}_{\mathsf{inj}}$ or $\mathsf{pk}_{\mathsf{loss}}$ to emphasize that the public key has been generated in either mode, and also $\mathsf{Gen}_{\mathsf{inj}}(\cdot) = \mathsf{Gen}(\cdot, \mathsf{inj})$ as well as $\mathsf{Gen}_{\mathsf{loss}}(\cdot) = \mathsf{Gen}(\cdot, \mathsf{loss})$ to explicitly refer to key generation in injective and lossy mode, respectively. The type of key is indistinguishable to outsiders. This holds even though the adversary can evaluate the function via deterministic algorithm Eval under this key, taking 1^λ, a key pk and a value x of input length $\mathrm{in}(\lambda)$ as input, and returning an image $f_{\mathsf{pk}}(x)$ of an implicitly defined function f. We usually assume that 1^λ is included in pk and thus omit 1^λ for Eval's input.

In the literature, one can find two slightly different definitions of lossy function. One, which we call the strict variant, requires that for any key generated in injective or lossy mode, the corresponding function is perfectly injective or lossy. In the non-strict variant this only has to hold with overwhelming probability over the choice of the key pk. We define both variants together:

Definition 3.1 (Lossy Functions) *An ω-lossy function consists of two efficient algorithms* $(\mathsf{Gen}, \mathsf{Eval})$ *of which* Gen *is probabilistic and* Eval *is deterministic and it holds that:*

(a) For $\mathsf{pk}_{\mathsf{inj}} \leftarrow\!\$\, \mathsf{Gen}(1^\lambda, \mathsf{inj})$ *the function* $\mathsf{Eval}(\mathsf{pk}_{\mathsf{inj}}, \cdot) : \{0,1\}^{in(\lambda)} \to \{0,1\}^*$ *is injective with overwhelming probability over the choice of* $\mathsf{pk}_{\mathsf{inj}}$.

(b) For $\mathsf{pk}_{\mathsf{loss}} \leftarrow\!\$\, \mathsf{Gen}(1^\lambda, \mathsf{loss})$*, the function* $\mathsf{Eval}(\mathsf{pk}_{\mathsf{loss}}, \cdot) : \{0,1\}^{in(\lambda)} \to \{0,1\}^*$ *is ω-compressing i.e.,* $\left|\{\mathsf{Eval}(\mathsf{pk}_{\mathsf{loss}}, \{0,1\}^{in(\lambda)})\}\right| \leq 2^{in(\lambda)-\omega}$*, with overwhelming probability over the choice of* $\mathsf{pk}_{\mathsf{loss}}$.

(c) The random variables $\mathsf{Gen}_{\mathsf{inj}}$ *and* $\mathsf{Gen}_{\mathsf{loss}}$ *are computationally indistinguishable.*

We call the function strict *if properties (a) and (b) hold with probability* 1.

Extremely lossy functions need a more fine-grained approach where the key generation algorithm takes an integer r between 1 and $2^{\mathrm{in}(\lambda)}$ instead of inj or loss. This integer determines the image size, with $r = 2^{\mathrm{in}(\lambda)}$ asking for an injective function. As we want to have functions with a sufficiently high lossiness that the image size is polynomial, say, $p(\lambda)$, we cannot allow for any polynomial adversary. This is so because an adversary making $p(\lambda) + 1$ many random (but distinct) queries to the evaluating function will find a collision in case that pk was lossy, while no collision will be found for an injective key. Instead, we define the minimal r such that $\mathsf{Gen}(1^\lambda, 2^\lambda)$ and $\mathsf{Gen}(1^\lambda, r)$ are indistinguishable based on the runtime and desired advantage of the adversary:

Definition 3.2 (Extremely Lossy Function) *An extremely lossy function consists of two efficient algorithms* $(\mathsf{Gen}, \mathsf{Eval})$ *of which* Gen *is probabilistic and* Eval *is deterministic and it holds that:*

(a) For $r = 2^{in(\lambda)}$ and $\mathsf{pk} \leftarrow\!\$\, \mathsf{Gen}(1^\lambda, r)$ *the function* $\mathsf{Eval}(\mathsf{pk}, \cdot) : \{0,1\}^{in(\lambda)} \to \{0,1\}^*$ *is injective with overwhelming probability.*

(b) For $r < 2^{in(\lambda)}$ and $\mathsf{pk} \leftarrow\!\$\, \mathsf{Gen}(1^\lambda, r)$ *the function* $\mathsf{Eval}(\mathsf{pk}, \cdot) : \{0,1\}^{in(\lambda)} \to \{0,1\}^*$ *has an image size of at most r with overwhelming probability.*

(c) For any polynomials p and d there exists a polynomial q such that for any adversary $\mathcal{A}$ with a runtime bounded by $p(\lambda)$ and any $r \in [q(\lambda), 2^{in(\lambda)}]$, algorithm $\mathcal{A}$ distinguishes $\mathsf{Gen}(1^\lambda, 2^{in(\lambda)})$ *from* $\mathsf{Gen}(1^\lambda, r)$ *with advantage at most $\frac{1}{d(\lambda)}$.*

Note that extremely lossy functions do indeed imply the definition of (moderately) lossy functions (as long as the lossiness-parameter ω still leaves an exponential-sized image size in the lossy mode):

Lemma 3.3 *Let* $(\mathsf{Gen}, \mathsf{Eval})$ *be an extremely lossy function. Then* $(\mathsf{Gen}, \mathsf{Eval})$ *is also a (moderately) lossy function with lossiness parameter* $\omega = 0.9\lambda$.

Proof. Assume $(\mathsf{Gen}, \mathsf{Eval})$ is not such a lossy function, i.e., for $r = 2^{\frac{\lambda}{10}}$ there exists an adversary $\mathcal{A}$ running in time $p(\lambda)$ which is able to distinguish the lossy mode from the injective mode with advantage $d(\lambda)$. Now, as any polynomial $q(\lambda)$ will be eventually smaller than $2^{\frac{\lambda}{10}}$, this directly violates property (c) of the extremely lossy function. □

3.2 Notions of Black-box Constructions and Oracle Separations

Most constructions in Cryptography are *black-box*, i.e., they are built in a way that they do not depend on the specifics of the instance of the underlying problem or construction (i.e., a one-way function), but only access it as an abstract primitive. Reingold, Trevisan and Vadhan [RTV04] as well as Baecher, Brzuska and Fischlin [BBF13] have given an extensive overview over the different classes of black-box constructions. In this paper, we will be mostly concerned with *fully black-box* (in the notion of [RTV04]) or *BBB black-box* (in the notion of [BBF13]) construction.

Oracle separations, introduced in the seminal work by Impagliazzo and Rudich [IR89], are very useful in proving black-box impossibility results, i.e., that a primitive P cannot be built from another primitive Q in a black-box way. To show such an impossibility result, one comes up with an oracle A (or a collection of such) such that, while a secure implementation of Q exists unconditionally relative to A, every implementation of P can be broken. As black-box constructions are relativizing, i.e., the result does not change if we provide access to some oracle, this shows that no black-box construction can exist.

A simplification of oracle separations, known as the two-oracle technique, was introduced by Hsiao and Reyzin [HR04]. Here, we have two oracles A and B, where the secure construction of Q only relies on A, any adversary gets access to both A and B, though. The two-oracles technique only rules out *fully* black-box constructions.

Lemma 3.4 (Two-Oracle Technique [HR04]) *To show that no fully black-box construction of P from Q exists, it suffices to show that*

1. *there is an efficient implementation N^A of Q,*
2. *for every efficient implementation M^A of P, there exists an adversary $\mathcal{A}^B$ such that $\mathcal{A}^B$ breaks the security of M^A, and*
3. *there exists no adversary $\mathcal{B}$ such that $\mathcal{B}^{A,B}$ breaks the security of N^A.*

Note that according to this lemma, adversary $\mathcal{A}$ does not have access to the oracle A. However, we can always redefine our oracles as $A' = A$ and $B' = (A, B)$, which will yield the same result, but allows $\mathcal{A}$ to access oracle A as well.

Another helpful lemma in fixing an oracle as part of an oracle separation is the Borel-Cantelli Lemma:

Lemma 3.5 (Borel-Cantelli) *Let $E_1, E_2, \ldots$ be a sequence of events on the same probability space. Then, if the sum of probabilities of events converges, the probability that infinitely many of the events occur is* 0*:*

$$\sum_{\lambda=1}^{\infty} \Pr[E_\lambda] < \infty \Rightarrow \Pr\left[\bigwedge_{k=1}^{\infty} \bigvee_{\lambda \geq k} E_\lambda\right] = 0$$

3.3 Oraclecrypt

In his seminal work [Imp95], Impagliazzo introduced five possible worlds we might be living in, including two in which computational cryptography exists: Minicrypt, in which one-way functions exist, but public-key cryptography does not, and Cryptomania, in which public-key cryptography exists as well. In reference to this classification, cryptographic primitives that can be built from one-way functions in a black-box way are often called Minicrypt primitives.

In this work, we are interested in the set of all primitives that exist relative to a truly random function. This of course includes all Minicrypt primitives, as one-way functions exist relative to a truly random function (with high probability), but it also includes a number of other primitives, like collision-resistant hash functions and exponentially-secure one-way functions, for which we don't know that they exist relative to a one-way function, or even have a black-box impossibility result. In reference to the set of Minicrypt primitives, we will call all primitives existing relative to a truly random function *Oraclecrypt* primitives.

Definition 3.6 (Oraclecrypt) *We say that a cryptographic primitive is an Oraclecrypt primitive, if there exists an implementation relative to truly random function oracle (except for a measure zero of random oracles).*

We will now show that by this definition, indeed, many symmetric primitives are Oraclecrypt primitives:

Lemma 3.7 *The following primitives are Oraclecrypt primitives:*

- *Exponentially-secure one-way functions,*
- *Exponentially-secure collision resistant hash functions,*
- *One-way product functions.*

We moved the proof for this lemma to Appendix C.1.

4 On the Impossibility of Building (E)LFs in Oraclecrypt

In this chapter, we will show that we cannot build lossy functions from a number of symmetric primitives, including (exponentially-secure) one-way functions, collision-resistant hash functions and one-way product functions, in a black-box way. Indeed, we will show that any primitive in Oraclecrypt is not enough to build lossy functions. As extremely lossy functions imply (moderately) lossy functions, this result applies to them as well.

Note that for exponentially-secure one-way functions, this was already known for lossy functions that are sufficiently lossy: Lossy functions with sufficient lossiness imply collision-resistant hash functions, and Simon's result [Sim98] separates these from (exponentially-secure) one-way functions. However, this does not apply for lossy functions with e.g. only a constant number of bits of lossiness.

Theorem 4.1 *There exists no fully black-box construction of lossy functions from any Oraclecrypt primitive, including exponentially-secure one-way functions, collision resistant hash functions, and one-way product functions.*

We will use an oracle separation to show this theorem. For this, we will start by introducing two oracles, a random oracle and a modified PSPACE oracle. We will then, for a candidate construction of a lossy function based on the random oracle and a public key pk, approximate the heavy queries asked by $\mathsf{Eval}(\mathsf{pk}, \cdot)$ to the random oracle. Next, we show that this approximation of the set of heavy queries is actually enough for us approximating the image size of $\mathsf{Eval}(\mathsf{pk},\cdot)$ (using our modified PSPACE oracle) and therefore gives an efficient way to distinguish lossy keys from injective keys. Finally, we have to fix a set of oracles (instead of arguing with a distribution of oracles) and then use the two-oracle technique to show the theorem.

4.1 Introducing the Oracles

A common oracle to use in an oracle separation in cryptography is the PSPACE oracle, as relative to this oracle, all non-information theoretic cryptography is broken. As we do not know which (or whether any) cryptographic primitives exist unconditionally, this is a good way to level the playing field. However, in our case, PSPACE is not quite enough. In our proof, we want to calculate the image size of a function relative to a (newly chosen) random oracle. It is not possible to simulate this oracle by lazy-sampling, though, as to calculate the image size of a function, we might have to save an exponentially large set of queries, which is not possible in PSPACE. Therefore, we give the PSPACE oracle access to its own random oracle $\mathcal{O}' : \{0,1\}^\lambda \to \{0,1\}^\lambda$ and will give every adversary access to $\mathrm{PSPACE}^{\mathcal{O}'}$.

The second oracle is a random oracle $\mathcal{O} : \{0,1\}^\lambda \to \{0,1\}^\lambda$. Now, we know that a number of primitives exist relative to a random function, including exponentially-secure one-way functions, collision-resistant hash functions and even more complicated primitives like one-way product functions. Further, they still exist if we give the adversary access to $\mathrm{PSPACE}^{\mathcal{O}'}$, too, as $\mathcal{O}'$ is independent from $\mathcal{O}$ and $\mathrm{PSPACE}^{\mathcal{O}'}$ does not have direct access to $\mathcal{O}$.

We will now show that every candidate construction of a lossy function with access to $\mathcal{O}$ can be broken by an adversary $\mathcal{A}^{\mathcal{O},\mathrm{PSPACE}^{\mathcal{O}'}}$. Note that we do not give the construction access to $\mathrm{PSPACE}^{\mathcal{O}'}$ — this is necessary, as $\mathcal{O}'$ should look like a randomly sampled oracle to the construction. However, giving the construction access to $\mathrm{PSPACE}^{\mathcal{O}'}$ would enable the construction to behave differently for this specific oracle $\mathcal{O}'$. Not giving the construction access to the oracle is fine, however, as we are using the two-oracle technique.

Our proof for Theorem 4.1 will now work in two steps. First, we will show that with overwhelming probability over independently sampled $\mathcal{O}$ and $\mathcal{O}'$, no lossy functions exist relative to $\mathcal{O}$ and $\mathrm{PSPACE}^{\mathcal{O}'}$. However, for an oracle separation, we need one fixed oracle. Therefore, as a second step (Section 4.4), we will use standard techniques to select one set of oracles relative to which any of our Oraclecrypt primitives exist, but lossy functions do not.

For the first step, we will now define how our definition of lossy functions with access to both oracles looks like:

Definition 4.2 (Lossy functions with Oracle Access) *A family of functions* $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot) : \{0,1\}^{in(\lambda)} \to \{0,1\}^*$ *with public key* pk *and access to the oracles* $\mathcal{O}$ *is called* ω*-lossy if there exist two PPT algorithms* $\mathsf{Gen}_{\mathsf{inj}}$ *and* $\mathsf{Gen}_{\mathsf{loss}}$ *such that for all* $\lambda \in \mathbb{N}$*,*

(a) For all pk *in* $[\mathsf{Gen}^{\mathcal{O}}_{\mathsf{inj}}(1^\lambda)] \cup [\mathsf{Gen}^{\mathcal{O}}_{\mathsf{loss}}(1^\lambda)]$*,* $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$ *is computable in polynomial time in* λ*,*

(b) For $\mathsf{pk} \leftarrow\$\, \mathsf{Gen}^{\mathcal{O}}_{\mathsf{inj}}(1^\lambda)$*,* $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$ *is injective with overwhelming probability (over the choice of* pk *as well as the random oracle* $\mathcal{O}$*),*

(c) For $\mathsf{pk} \leftarrow\$\, \mathsf{Gen}^{\mathcal{O}}_{\mathsf{loss}}(1^\lambda)$*,* $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$ *is* ω*-compressing with overwhelming probability (over the choice of* pk *as well as the random oracle* $\mathcal{O}$*)*

(d) The random variables $\mathsf{Gen}^{\mathcal{O}}_{\mathsf{inj}}$ *and* $\mathsf{Gen}^{\mathcal{O}}_{\mathsf{loss}}$ *are computationally indistinguishable for any polynomial-time adversary* $\mathcal{A}^{\mathcal{O},\mathrm{PSPACE}^{\mathcal{O}'}}$ *with access to both* $\mathcal{O}$ *and* $\mathrm{PSPACE}^{\mathcal{O}'}$*.*

4.2 Approximating the Set of Heavy Queries

In the next two subsections, we will construct an adversary $\mathcal{A}^{\mathcal{O},\mathrm{PSPACE}^{\mathcal{O}'}}$ against lossy functions with access to the random oracle $\mathcal{O}$ as described in Definition 4.2.

Let $(\mathsf{Gen}^{\mathcal{O}}, \mathsf{Eval}^{\mathcal{O}})$ be some candidate implementation of a lossy function relative to the oracle $\mathcal{O}$. Further, let $\mathsf{pk} \leftarrow \mathsf{Gen}^{\mathcal{O}}_{?}$ be some public key generated by either $\mathsf{Gen}_{\mathsf{inj}}$ or $\mathsf{Gen}_{\mathsf{loss}}$. Looking at the queries asked by the lossy function to $\mathcal{O}$, we can divide them into two parts: The queries asked during the generation of the key pk, and the queries asked during the execution of $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk}, \cdot)$. We will denote the queries asked during the generation of pk by the set Q_G. As the generation algorithm has to be efficient, Q_G has polynomial size. Let k_G be the maximal number of queries asked by any of the two generators. Further, denote by k_f the maximum number of queries of $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk}, x)$ for any pk and x — again, k_f is polynomial. Finally, let $k = \max\{k_G, k_f\}$.

The set of all queries done by $\mathsf{Eval}(\mathsf{pk}, \dot{)}$ for a fixed key pk might be of exponential size, as the function might ask different queries for each input x. However, we are able to shrink the size of the relevant subset significantly, if we concentrate on *heavy* queries — queries that appear for a significant fraction of all inputs x:

Definition 4.3 (Heavy Queries) *Let k be the maximum number of $\mathcal{O}$-queries made by the generator $\mathsf{Gen}^{\mathcal{O}}_{?}$, or the maximum number of queries of $\mathsf{Eval}(\mathsf{pk}, \cdot)$ over all inputs $x \in \{0,1\}^{in(\lambda)}$, whichever is higher. Fix some key pk and a random oracle $\mathcal{O}$. We call a query q to $\mathcal{O}$* heavy *if, for at least a $\frac{1}{10k}$-fraction of $x \in \{0,1\}^{in(\lambda)}$, the evaluation $\mathsf{Eval}(\mathsf{pk}, x)$ queries $\mathcal{O}$ about q at some point. We denote by Q_H the set of all heavy queries (for $\mathsf{pk}, \mathcal{O}$).*

The set of heavy queries is polynomial, as $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk}, \cdot)$ only queries the oracle a polynomial number of times and each heavy query has to appear in a polynomial fraction of all x. Further, we will show that the adversary $\mathcal{A}^{\mathcal{O},\mathrm{PSPACE}^{\mathcal{O}'}}$ is able to approximate the set of heavy queries, and that this approximation is actually enough to decide whether pk was generated in injective or in lossy mode. We will start with a few key observations that help us prove this statement.

The first one is that the generator, as it is an efficiently-computable function, will only query $\mathcal{O}$ at polynomially-many positions, and these polynomially-many queries already define whether the function is injective or lossy:

Observation 1 *Let Q_G denote the queries by the generator. For a random $\mathsf{pk} \leftarrow \mathsf{Gen}^{\mathcal{O}}_{\mathsf{inj}}$ generated in injective mode and a random $\mathcal{O}'$ that is consistent with Q_G, the image size of $\mathsf{Eval}^{\mathcal{O}'}(\mathsf{pk}, \cdot)$ is 2^{λ} (except with a negligible probability over the choice of pk and $\mathcal{O}'$). Similarly, for a random $\mathsf{pk} \leftarrow \mathsf{Gen}^{\mathcal{O}}_{\mathsf{loss}}$ generated in lossy mode and a random $\mathcal{O}'$ that is consistent with Q_G, the image size of $\mathsf{Eval}^{\mathcal{O}'}(\mathsf{pk}, \cdot)$ is at most $2^{\lambda-1}$ (except with a negligible probability over the choice of pk and $\mathcal{O}'$).*

This follows directly from the definition: As $\mathsf{Gen}^{\mathcal{O}}_{?}$ has no information about $\mathcal{O}$ except the queries Q_G, properties (2) and (3) of Definition 3.1 have to hold for every random oracle that is consistent with $\mathcal{O}$ on Q_G. We will use this multiple times in the proof to argue that queries to $\mathcal{O}$ that are not in Q_G are, essentially, useless randomness for the construction, as the construction has to work with almost any possible answer returned by these queries.

An adversary is probably very much interested in learning the queries Q_G. There is no way to capture them in general, though. Here, we need our second key observation. Lossiness is very much a global property: to switch a function from lossy to injective, at least half of all inputs x to $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk}, x)$ must produce a different result, and vice versa. However, as we learned from the first observation, whether $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk}, \cdot)$ is lossy or injective, depends just on Q_G. Therefore, some queries in Q_G must be used over and over again for different inputs x — and will therefore appear in the heavy set Q_H. Further, due to the heaviness of these queries, the adversary is indeed able to learn them!

Our proof works alongside these two observations: First, we show in Lemma 4.4 that for any candidate lossy function, an adversary is able to compute a set $\hat{Q}_H$ of the interesting heavy queries. Afterwards, we show in Lemma 4.6 that we can use $\hat{Q}_H$ to decide whether $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$ is lossy or injective, breaking the indistinguishability property of the lossy function.

Lemma 4.4 *Let $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$ be a (non-strict) lossy function and $\mathsf{pk} \leftarrow \mathsf{Gen}_?^{\mathcal{O}}(1^\lambda)$ for oracle $\mathcal{O}$. Then we can compute in probabilistic polynomial-time (in λ) a set $\hat{Q}_H$ which contains all heavy queries of $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$ for $\mathsf{pk},\mathcal{O}$ with overwhelming probability.*

Proof. To find the heavy queries we will execute $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},x)$ for t random inputs x and record all queries to $\mathcal{O}$ in $\hat{Q}_H$. We will now argue that, with high probability, $\hat{Q}_H$ contains all heavy queries.

First, recall that a query is heavy if it appears for at least an ε-fraction of inputs to $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$ for $\varepsilon = \frac{1}{10k}$. Therefore, the probability for any specific heavy query q_{heavy} to not appear in $\hat{Q}_H$ after the t evaluations can be bounded by

$$\Pr\Big[q_{\mathsf{heavy}} \notin \hat{Q}_H\Big] = (1-\varepsilon)^t \leq 2^{-\varepsilon t}.$$

Furthermore, there exist at most $\frac{k}{\varepsilon}$ heavy queries, because each heavy query accounts for at least $\varepsilon \cdot 2^{\mathrm{in}(\lambda)}$ of the at most $k \cdot 2^{\mathrm{in}(\lambda)}$ possible queries of $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},x)$ when iterating over all x. Therefore, the probability that any heavy query q_{heavy} is not included in $\hat{Q}_H$ is given by

$$\Pr\Big[\exists q_{\mathsf{heavy}} \notin \hat{Q}_H\Big] \leq \frac{k}{\varepsilon} \cdot 2^{-\varepsilon t}$$

Choosing $t = 10k\lambda$ we get

$$\Pr\Big[\exists q_{\mathsf{heavy}} \notin \hat{Q}_H\Big] \leq 10k^2 \cdot 2^{-\lambda}$$

which is negligible. Therefore, with all but negligible probability, all heavy queries are included in $\hat{Q}_H$. □

4.3 Distinguishing Lossiness from Injectivity

We next make the transition from oracle $\mathcal{O}$ to our PSPACE-augmenting oracle $\mathcal{O}'$. According to the previous subsection, we can compute (a superset $\hat{Q}_H$ of) the heavy queries efficiently. Then we can fix the answers of oracle $\mathcal{O}$ on such frequently asked queries in $\hat{Q}_H$, but otherwise use the independent oracle $\mathcal{O}'$ instead. Denote this partly-set oracle by $\mathcal{O}'_{|\hat{Q}_H}$. Then the distinguisher for injective and lossy keys, given some pk, can approximate the image size of $\#im(\mathsf{Eval}^{\mathcal{O}'_{|\hat{Q}_H}}(\mathsf{pk},\cdot))$ with the help of its $\mathrm{PSPACE}^{\mathcal{O}'}$ oracle and thus also derives a good approximiation for the actual oracle $\mathcal{O}$. This will be done in Lemma 4.6.

We still have to show that the non-heavy queries do not violate the above approach. According to the proof of Lemma 4.5 it suffices to look at the case that the image sizes of oracles $\mathcal{R} := \mathcal{O}'_{|\hat{Q}_H}$ and for oracle $\mathcal{R}' := \mathcal{O}'_{|\hat{Q}_H \cup Q_G}$, where we als fix on the key generator's non-heavy queries to values from $\mathcal{O}$, cannot differ significantly. Put differently, missing out the generator's non-heavy queries Q_G in $\hat{Q}_H$ only slightly affects the image size of $\mathsf{Eval}^{\mathcal{O}'_{|\hat{Q}_H}}(\mathsf{pk},\cdot)$, and we can proceed with our approach to consider only heavy queries.

Lemma 4.5 *Let $\mathsf{pk} \leftarrow \mathsf{Gen}_?^{\mathcal{R}}(1^\lambda)$ and $Q_G^{\mathsf{nonh}} = \{q_1,\ldots,q_{k'}\}$ be the k' generator's queries to $\mathcal{R}$ in Q_G when computing pk that are not heavy for $\mathsf{pk},\mathcal{R}$. Then, for any oracle $\mathcal{R}'$ that is identical to $\mathcal{R}$ everywhere except for the queries in Q_G^{nonh}, i.e., $\mathcal{R}(q) = \mathcal{R}'(q)$ for any $q \notin Q_G^{\mathsf{nonh}}$, the image sizes of $\mathsf{Eval}^{\mathcal{R}}(\mathsf{pk},\cdot)$ and $\mathsf{Eval}^{\mathcal{R}'}(\mathsf{pk},\cdot)$ differ by at most $\frac{2^{in(\lambda)}}{10}$.*

Proof. As the queries in Q_G^{nonh} are non-heavy, every $q_i \in Q_G^{\mathsf{nonh}}$ is queried for at most $\frac{2^{\mathrm{in}(\lambda)}}{10k}$ inputs x to $\mathsf{Eval}^{\mathcal{R}}(\mathsf{pk}, \cdot)$ when evaluating the function. Therefore, any change in the oracle $\mathcal{R}$ at $q_i \in Q_G^{\mathsf{nonh}}$ affects the output of $\mathsf{Eval}^{\mathcal{R}}(\mathsf{pk}, \cdot)$ for at most $\frac{2^{\mathrm{in}(\lambda)}}{10k}$ inputs. Hence, when considering the oracle $\mathcal{R}'$, which differs from $\mathcal{R}$ only on the k' queries from Q_G^{nonh}, moving from $\mathcal{R}$ to $\mathcal{R}'$ for evaluating $\mathsf{Eval}^{\mathcal{R}}(\mathsf{pk}, \cdot)$ changes the output for at most $\frac{k' 2^{\mathrm{in}(\lambda)}}{10k}$ inputs x. In other words, letting Δ_f denote the set of all x such that $\mathsf{Eval}^{\mathcal{R}}(\mathsf{pk}, x)$ queries some $q \in Q_G^{\mathsf{nonh}}$ during the evaluation, we know that

$$|\Delta_f| \leq \frac{k' 2^{\mathrm{in}(\lambda)}}{10k}$$

and

$$\mathsf{Eval}^{\mathcal{R}}(\mathsf{pk}, x) = \mathsf{Eval}^{\mathcal{R}'}(\mathsf{pk}, x) \text{ for all } x \notin \Delta_f.$$

We are interested in the difference of the image sizes of $\mathsf{Eval}^{\mathcal{R}}(\mathsf{pk}, \cdot)$ and $\mathsf{Eval}^{\mathcal{R}'}(\mathsf{pk}, \cdot)$. Each $x \in \Delta_f$ may add or subtract an image in the difference, depending on whether the modified output $\mathsf{Eval}^{\mathcal{R}'}(\mathsf{pk}, x)$ introduces a new image or redirects the only image $\mathsf{Eval}^{\mathcal{R}}(\mathsf{pk}, x)$ to an already existing one. Therefore, the difference between the image sizes is at most

$$\left| \#im(\mathsf{Eval}^{\mathcal{R}}(\mathsf{pk}, \cdot)) - \#im(\mathsf{Eval}^{\mathcal{R}'}(\mathsf{pk}, \cdot)) \right| \leq \frac{k' 2^{\mathrm{in}(\lambda)}}{10k} \leq \frac{2^{\mathrm{in}(\lambda)}}{10},$$

where the last inequality is due to $k' \leq k$. □

Lemma 4.6 *Given $\hat{Q}_H \supseteq Q_H$, we can decide correctly whether $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk}, \cdot)$ is lossy or injective with overwhelming probability.*

Proof. As described in Section 4.1, we give the adversary, who has to distinguish a lossy key from a injective key, access to $\mathrm{PSPACE}^{\mathcal{O}'}$, where $\mathcal{O}'$ is another random oracle sampled independently of $\mathcal{O}$. This is necessary for the adversary, as we want to calculate the image size of $\mathsf{Eval}^{\mathcal{O}'}(\mathsf{pk}, \cdot)$ relative to a random oracle $\mathcal{O}'$, and we cannot do this in PSPACE with lazy sampling.

We will consider the following adversary $\mathcal{A}$: It defines an oracle $\mathcal{O}'_{|\hat{Q}_H}$ that is identical to $\mathcal{O}'$ for all queries $q \notin \hat{Q}_H$ and identical to $\mathcal{O}$ for all queries $q \in \hat{Q}_H$. Then, it calculates the image size

$$\#im(\mathsf{Eval}^{\mathcal{O}'_{|\hat{Q}_H}}(\mathsf{pk}, \cdot)) = \left| \{\mathsf{Eval}^{\mathcal{O}'_{|\hat{Q}_H}}(\mathsf{pk}, \{0,1\}^{\mathrm{in}(\lambda)})\} \right|.$$

Note that this can be done efficiently using $\mathrm{PSPACE}^{\mathcal{O}'}$ as well as polynomially many queries to $\mathcal{O}$. If $\#im(\mathsf{Eval}^{\mathcal{O}'_{|\hat{Q}_H}}(\mathsf{pk}, \cdot))$ is bigger than $\frac{3}{4} 2^{\mathrm{in}(\lambda)}$, $\mathcal{A}$ will guess that $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk}, \cdot)$ is injective, and lossy otherwise. For simplicity reasons, we will assume from now on that pk was generated by $\mathsf{Gen}_{\mathsf{inj}}$ — the case where pk was generated by $\mathsf{Gen}_{\mathsf{loss}}$ follows by a symmetric argument.

First, assume that all queries Q_G of the generator are included in $\hat{Q}_H$. In this case, any $\mathcal{O}'$ that is consistent with Q_H is also consistent with all the information $\mathsf{Gen}_{\mathsf{inj}}$ have about $\mathcal{O}$. However, this means that by definition, $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk}, \cdot)$ has to be injective with overwhelming probability, and therefore, an adversary can easily check whether pk was created by $\mathsf{Gen}_{\mathsf{inj}}$.

Otherwise, let $q_1, \ldots, q_{k'}$ be a set of queries in Q_G which are not included in $\hat{Q}_H$. With overwhelming probability, this means that $q_1, \ldots, q_{k'}$ are all non-heavy. We now apply Lemma 4.5 for oracles $\mathcal{R} := \mathcal{O}'_{|\hat{Q}_H}$ and $\mathcal{R}' := \mathcal{O}'_{|\hat{Q}_H \cup Q_G}$. These two oracles may only differ on the non-heavy queries in Q_G, where $\mathcal{R}$ coincides with $\mathcal{O}'$ and $\mathcal{R}'$ coincides with $\mathcal{O}$; otherwise the oracles are identical. Lemma 4.5 tells us that this will change the image size by at most $\frac{2^{\mathrm{in}(\lambda)}}{10}$. Therefore, with overwhelming probability, the image size calculated by the distinguisher is bounded from below by

$$\#im(\mathsf{Eval}^{\mathcal{O}'_{|\hat{Q}_H}}(\mathsf{pk},\cdot)) \geq 2^{\mathrm{in}(\lambda)} - \frac{2^{\mathrm{in}(\lambda)}}{10} \geq \frac{3}{4}2^{\mathrm{in}(\lambda)}$$

and the distinguisher will therefore correctly decide that $\mathsf{Eval}^{\mathcal{O}}(\mathsf{pk},\cdot)$ is in injective mode. □

Theorem 4.7 *Let $\mathcal{O}$ and $\mathcal{O}'$ be two independent random oracles. Then, with overwhelming probability over the choice of the two random oracles, lossy functions do not exist relative the oracles $\mathcal{O}$ and* $\mathrm{PSPACE}^{\mathcal{O}'}$.

Proof. Given the key pk, our distinguisher (with oracle access to random oracle $\mathcal{O}$) against the injective and lossy mode first runs the algorithm of Lemma 4.4 to efficiently construct a super set $\hat{Q}_H$ of the heavy queries Q_H for $\mathsf{pk}, \mathcal{O}$. This succeeds with overwhelming probability, and from now on we assume that indeed $Q_H \subseteq \hat{Q}_H$. Then our algorithm continues by running the decision procedure of Lemma 4.6 to distinguish the cases. Using the $\mathrm{PSPACE}^{\mathcal{O}'}$ oracle, the latter can also be carried out efficiently. □

4.4 Fixing an Oracle

We have shown now (in Theorem 4.7) that no lossy function exists relative to a random oracle with overwhelming probability. However, to prove our main theorem, we have to show that there exists one fixed oracle relative to which one-way functions (or collision-resistant hash functions, or one-way product functions) exist, but lossy functions do not.

In Lemma 3.7, we have already shown that (exponentially-secure) one-way functions, collision-resistant hash functions and one-way product functions exist relative to a random oracle with high probability. In the next lemma, we will show that there exists a fixed oracle relative to which exponentially-secure one-way functions exist, but lossy functions do not. The proofs for existence of oracles relative to which exponentially-secure collision-resistant hash functions or one-way product functions, but no lossy functions exist follow similarly.

Lemma 4.8 *There exists a fixed set of oracles $\mathcal{O}$,* $\mathrm{PSPACE}^{\mathcal{O}'}$ *such that relative to these oracles, one-way functions using $\mathcal{O}$ exist, but no construction of lossy functions from $\mathcal{O}$ exists.*

Now, our main theorem of this section directly follows from this lemma (and its variants for the other primitives):

Theorem 4.1 (restated) *There exists no fully black-box construction of lossy functions from any Oraclecrypt primitive, including exponentially-secure one-way functions, collision resistant hash functions, and one-way product functions.*

Proof. By Lemma 4.8, there exist two oracles $\mathcal{O}$ and $\mathrm{PSPACE}^{\mathcal{O}'}$ such that exponentially-secure one-way functions (or any of the other Oraclecrypt primitives) exist relative to $\mathcal{O}$, even if the adversary against the one-wayness has additional access to $\mathrm{PSPACE}^{\mathcal{O}'}$. However, there exists an adversary with access to $\mathcal{O}$ and $\mathrm{PSPACE}^{\mathcal{O}'}$ that breaks any construction of a lossy function relative to $\mathcal{O}$. The two-oracle technique then shows that this means no fully black-box construction of lossy functions from exponentially-secure one-way functions (or, from any other primitive in Oraclecrypt) can exist. □

Let us now focus on Lemma 4.8 again. Up to this point, we have argued over distributions of oracles (i.e., we have required the oracles $\mathcal{O}$ and $\mathcal{O}'$ to be chosen at random from any possible oracle). For a proper oracle separation, however, we have to show that our results hold for a set of fixed oracles.

We use the following Borel-Cantelli-style theorem from Mahmoody, Mohammed, Nematihaji, Pass and Shelat [MMN+16]:

Lemma 4.9 ([MMN+16], Lemma 2.9) *Let $E_1, E_2, \dots$ be a sequence of events such that $\exists c \forall n \in \mathbb{N}$: $\Pr[E_n] \geq c$, where c is a constant, $0 < c < 1$. Then,*

$$\Pr\left[\bigwedge_{k=1}^{\infty} \bigvee_{n>k} E_n\right] \geq c$$

Further, we also need the so-called splitting lemma [PS00] which allows to relate the probability of events over a product space $X \times Y$ to the ones when the X-part is fixed:

Lemma 4.10 (Splitting Lemma [PS00]) *Let $\mathcal{D} = \mathcal{D}_X \times \mathcal{D}_Y$ be some product distributions over $X \times Y$. Let $Z \subseteq X \times Y$ be such that $\Pr_{\mathcal{D}}[(x, y) \in Z] > \varepsilon$. For any $\alpha < \varepsilon$ call $x \in X$ to be α-good if*

$$\Pr_{y \leftarrow\$ \mathcal{D}_Y}[(x, y) \in Z] > \varepsilon - \alpha.$$

Then we have $\Pr_{x \leftarrow\$ \mathcal{D}_X}[x \text{ is } \alpha\text{-good}] \geq \alpha$.

Proof (of Lemma 4.8). We will show that for each primitive out of exponentially-secure one-way functions, collision-resistant hash functions and one-way product functions, there exist two fixed oracles $\mathcal{O}$ and $\mathcal{O}'$ such that, relative to $\mathcal{O}$ and $\text{PSPACE}^{\mathcal{O}'}$, this primitive exists, but lossy functions do not. We will now prove this for exponentially-secure one-way functions – the proof for the other primitives works analogously.

Let $\mathcal{A}^{\mathcal{O}, \text{PSPACE}^{\mathcal{O}'}}$ be the adversary against lossy functions as described in the last sections. Then, by Theorem 4.7, we know that the adversary wins with overwhelming probability over the choice of the two random oracles as well as any internal randomness of $\mathcal{A}$, i.e., there exists a negligible function $\varepsilon(\lambda)$ such that

$$\forall \lambda \in \mathbb{N} : \Pr_{\mathcal{O}, \mathcal{O}', \mathcal{A}}\left[\mathcal{A}^{\mathcal{O}, \text{PSPACE}^{\mathcal{O}'}} \text{ wins}\right] \geq 1 - \varepsilon(\lambda).$$

We will now first fix an oracle $\mathcal{O}$. To do this, we have to split out oracle $\mathcal{O}$ using the Splitting Lemma (4.10) with $\alpha = \frac{1}{2}(1 - \varepsilon(\lambda))$:

$$\forall \lambda \in \mathbb{N} : \Pr_{\mathcal{O}}\left[\Pr_{\mathcal{O}', \mathcal{A}}\left[\mathcal{A}^{\mathcal{O}, \text{PSPACE}^{\mathcal{O}'}} \text{ wins}\right] \geq \frac{1}{2}(1 - \varepsilon(\lambda))\right] \geq \frac{1}{2}(1 - \varepsilon(\lambda))$$

Next, we want to use the constant version of the Borel-Cantelli Lemma (4.9), for which we need a constant bound in the outer probability. For large security parameters, we can just bound the outer probability by $\frac{1}{3}$, as a negligible function will eventually be smaller than any constant. For small security parameters, this might not work directly, but we can modify the negligible function ε' to be 1 in these cases (ε' stays negligible, as the modification only happens for small security parameters).

$$\forall \lambda \in \mathbb{N} : \Pr_{\mathcal{O}}\left[\Pr_{\mathcal{O}', \mathcal{A}}\left[\mathcal{A}^{\mathcal{O}, \text{PSPACE}^{\mathcal{O}'}} \text{ wins}\right] \geq \frac{1}{2}(1 - \varepsilon'(\lambda))\right] \geq \frac{1}{3}$$

Now, the constant version of Borel-Cantelli gives us

$$\Pr_{\mathcal{O}}\left[\bigwedge_{k=1}^{\infty} \bigvee_{\lambda>k} \Pr_{\mathcal{O}', \mathcal{A}}\left[\mathcal{A}^{\mathcal{O}, \text{PSPACE}^{\mathcal{O}'}} \text{ wins}\right] \geq \frac{1}{2}(1 - \varepsilon'(\lambda))\right] \geq \frac{1}{3}.$$

In other words, for a $\frac{1}{3}$ fraction of all oracles $\mathcal{O}$, adversary $\mathcal{A}$ wins for infinitely many security parameters with probability $\frac{1}{2}(1 - \varepsilon'(\lambda))$ (over the choice of $\mathcal{O}'$ and the randomness of $\mathcal{A}$).

Now, we want to have a fixed oracle $\mathcal{O}$ relative to which not only lossy functions do not exist, but also exponentially-secure one-way functions do exist. However, by Lemma 3.7 we know that exponentially-secure one-way functions exist relative to a 1-measure of random oracles $\mathcal{O}$. Therefore, it is clear we will find a fixed oracle $\mathcal{O}$ relative to which exponentially-secure one-way functions exist, but lossy functions do not. Let us now fix such an oracle $\mathcal{O}$.

Now, it remains for us to fix the other random oracle, $\mathcal{O}'$. We can apply the splitting lemma again to get

$$\forall\lambda \in \mathbb{N} : \Pr_{\mathcal{O}'}\left[\Pr_{\mathcal{A}}\left[\mathcal{A}^{\mathcal{O},\mathrm{PSPACE}^{\mathcal{O}'}} \text{ wins}\right] \geq \frac{1}{4}(1-\varepsilon'(\lambda))\right] \geq \frac{1}{4}(1-\varepsilon')$$

By a similar argument as above, we can fix the outer probability by modifying the negligible function to ε'', which lets us apply Borell-Cantelli again:

$$\Pr_{\mathcal{O}'}\left[\bigwedge_{k=1}^{\infty}\bigvee_{\lambda>k}\Pr_{\mathcal{A}}\left[\mathcal{A}^{\mathcal{O},\mathrm{PSPACE}^{\mathcal{O}'}} \text{ wins}\right] \geq \frac{1}{4}(1-\varepsilon''(\lambda))\right] \geq \frac{1}{5}.$$

Fixing an oracle $\mathcal{O}'$ out of the $\frac{1}{5}$ fraction gives us the desired result. $\square$

5 On the Impossibility of Building Key Agreement Protocols from (Extremely) Lossy Functions

In the previous section we showed that lossy functions cannot be built from many symmetric primitives in a black-box way. This raises the question if lossy functions and extremely lossy functions might be inherent asymmetric primitives. In this section we provide evidence to the contrary, showing that key agreement cannot be built from lossy functions in a black-box way. For this, we adapt the proof by Impagliazzo and Rudich [IR89] showing that key agreement cannot be built from one-way functions to our setting. We extend this result to also hold for extremely lossy functions, but in a slightly weaker setting.

5.1 Lossy Function Oracle

We specify our lossy function oracle relative to a (random) permutation oracle Π, and further sample (independently of Π) a second random permutation Γ as integral part of our lossy function oracle. The core idea of the oracle is to evaluate $\mathsf{Eval}^{\Gamma,\Pi}(\mathsf{pk}_{\mathsf{inj}}, x) = \Pi(\mathsf{pk}_{\mathsf{inj}}\|ax+b)$ for the injective mode, but set $\mathsf{Eval}^{\Gamma,\Pi}(\mathsf{pk}_{\mathsf{loss}}, x) = \Pi(\mathsf{pk}_{\mathsf{loss}}\|\mathsf{setlsb}(ax+b))$ for the lossy mode, where a, b describe a pairwise independent hash permutation $ax+b$ over the field $\mathrm{GF}(2^\mu)$ with $a \neq 0$ and setlsb sets the least significant bit to 0. Then the lossy function is clearly two to one. The values a, b will be chosen during key generation and placed into the public key, but we need to hide them from the adversary in order to make the keys of the two modes indistinguishable. Else a distinguisher, given pk, could check if $\mathsf{Eval}^{\Gamma,\Pi}(\mathsf{pk}, x) = \mathsf{Eval}^{\Gamma,\Pi}(\mathsf{pk}, x')$ for appropriately computed $x \neq x'$ with $\mathsf{setlsb}(ax+b) = \mathsf{setlsb}(ax'+b)$. Therefore, we will use the secret permutation Γ to hide the values in the public key. We will denote the preimage of pk under Γ as pre-key.

Another feature of our construction is to ensure that the adversary cannot generate a lossy key $\mathsf{pk}_{\mathsf{loss}}$ without calling $\mathsf{Gen}^{\Gamma,\Pi}$ in lossy mode, while allowing it to generate keys in injective mode. We accomplish this by having a value k in our public pre-key that is zero for lossy keys and may take any non-zero value for an injective public key. Therefore, with overwhelming probability, any key generated by the adversary without a call to the $\mathsf{Gen}^{\Gamma,\Pi}$ oracle will be an injective key.

We finally put both ideas together. For key generation we hide a, b and also the string k by creating pk as a commitment to the values, $\mathsf{pk} \leftarrow \Gamma(k\|a\|b\|z)$ for random z. To unify calls to Γ in regard of the security parameter λ, we will choose all entries in the range of $\lambda/5$.[2] When receiving pk the evaluation

[2]For moderately lossy function we could actually use $\lambda/4$ but for compatibility to the extremely lossy case it is convenient to use $\lambda/5$ already here.

algorithm $\mathsf{Eval}^{\Gamma,\Pi}$ first recovers the preimage $k\|a\|b\|z$ under Π, then checks if k signals injective or lossy mode, and then computes $\Pi(a\|b\|ax+b)$ resp. $\Pi(a\|b\|\mathsf{setlsb}(ax+b))$ as the output.

Definition 5.1 (Lossy Function Oracle) *Let Π, Γ be permutation oracles with $\Pi, \Gamma : \{0,1\}^\lambda \to \{0,1\}^\lambda$ for all λ. Let $\mu = \mu(\lambda) = \lfloor(\lambda-2)/5\rfloor$ and $\mathsf{pad} = \mathsf{pad}(\lambda) = \lambda - 2 - 5\mu$ define the length that the rounding-off loses to $\lambda - 2$ in total (such that $\mathsf{pad} \in \{0,1,2,3,4\}$). Define the lossy function $(\mathsf{Gen}^{\Gamma,\Pi}, \mathsf{Eval}^{\Gamma,\Pi})$ with input length $in(\lambda) = \mu(\lambda)$ relative to Π and Γ now as follows:*

Key Generation: *Oracle $\mathsf{Gen}^{\Gamma,\Pi}$ on input 1^λ and either mode inj or loss picks random $b \leftarrow\!\$\, \{0,1\}^\mu$, $z \leftarrow\!\$\, \{0,1\}^{2\mu+\mathsf{pad}}$ and random $a, k \leftarrow\!\$\, \{0,1\}^\mu \setminus \{0^\mu\}$. For mode inj the algorithm returns $\Gamma(k\|a\|b\|z)$. For mode loss the algorithm returns $\Gamma(0^\mu\|a\|b\|z)$ instead.*

Evaluation: *On input $\mathsf{pk} \in \{0,1\}^\lambda$ and $x \in \{0,1\}^\mu$ algorithm $\mathsf{Eval}^{\Gamma,\Pi}$ first recovers (via exhaustive search) the preimage $k\|a\|b\|z$ of pk under Γ for $k, a, b \in \{0,1\}^\mu$, $z \in \{0,1\}^{2\mu+\mathsf{pad}}$. Check that $a \neq 0$ in the field $GF(2^\mu)$. If any check fails then return $\perp$. Else, next check if $k = 0^\mu$. If so, return $\Pi(a\|b\|\mathsf{setlsb}(ax+b))$, else return $\Pi(a\|b\|ax+b)$.*

We now show that there exist permutations Π and Γ such that relative to Π and the lossy function oracle $(\mathsf{Gen}^{\Gamma,\Pi}, \mathsf{Eval}^{\Gamma,\Pi})$, lossy functions exist, but key agreement does not. We will rely on the seminal result by Impagliazzo and Rudich [IR89] showing that no key agreement exists relative to a random permutation. Note that we do not give direct access to Γ — it will only be accessed by the lossy functions oracle and is considered an integral part of it.

The following lemma is the technical core of our results. It says that the partly exponential steps of the lossy-function oracles $\mathsf{Gen}^{\Gamma,\Pi}$ and $\mathsf{Eval}^{\Gamma,\Pi}$ in our construction can be simulated sufficiently close and efficiently through a stateful algorithm Wrap, given only oracle access to Π, even if we filter out the mode for key generation calls. For this we define security experiments as efficient algorithms Game with oracle access to an adversary $\mathcal{A}$ and lossy function oracles $\mathsf{Gen}^{\Gamma,\Pi}, \mathsf{Eval}^{\Gamma,\Pi}, \Pi$ and which produces some output, usually indicating if the adversary has won or not. We note that we can assume for simplicity that $\mathcal{A}$ makes oracle queries to the lossy function oracles and Π via the game only. Algorithm Wrap will be black-box with respect to $\mathcal{A}$ and Game but needs to know the total number $p(\lambda)$ of queries the adversary and the game make to the primitive and the quality level $\alpha(\lambda)$ of the simulation upfront.

Lemma 5.2 (Simulation Lemma) *Let Filter be a deterministic algorithm which for calls $(1^\lambda, \mathsf{mode})$ to $\mathsf{Gen}^{\Gamma,\Pi}$ only outputs 1^λ and leaves any input to calls to $\mathsf{Eval}^{\Gamma,\Pi}$ and to Π unchanged. For any polynomial $p(\lambda)$ and any inverse polynomial $\alpha(\lambda)$ there exists an efficient algorithm Wrap such that for any efficient algorithm $\mathcal{A}$, any efficient experiment Game making at most $p(\lambda)$ calls to the oracle,the statistical distance between $\mathsf{Game}^{\mathcal{A},(\mathsf{Gen}^{\Gamma,\Pi},\mathsf{Eval}^{\Gamma,\Pi},\Pi)}(1^\lambda)$ and $\mathsf{Game}^{\mathcal{A},\mathsf{Wrap}^{\mathsf{Gen}^{\Gamma,\Pi},\Pi}\circ\mathsf{Filter}}$ is at most $\alpha(\lambda)$. Furthermore Wrap initially makes a polynomial number of oracle calls to $\mathsf{Gen}^{\Gamma,\Pi}$, but then makes at most two calls to Π for each query.*

In fact, since $\mathsf{Gen}^{\Gamma,\Pi}$ is efficient relative to Γ, and Wrap only makes calls to $\mathsf{Gen}^{\Gamma,\Pi}$ for all values up to a logarithmic length L_0, we can also write $\mathsf{Wrap}^{\Gamma|_{L_0},\Pi}$ to denote the limited access to the Γ-oracle. We also note that the (local) state of Wrap only consists of such small preimage-image pairs of Γ and Π for such small values (but Wrap later calls Π also about longer inputs).

Proof. The proof strategy is to process queries of Game and $\mathcal{A}$ efficiently given only access to Π, making changes to the oracle gradually, depending on the type of query. The changes will be actually implemented by our stateful algorithm Wrap, and eventually we will add Filter at the end. To do so, we will perform a series of games hops where we change the behavior of the key generation and evaluation oracles. For each game $\mathsf{Game}_1, \mathsf{Game}_2, \ldots$ let $\mathsf{Game}_i(\lambda)$ be the randomized output of the game with access to $\mathcal{A}$. Let

Game	$\mathsf{Gen}_{\mathsf{loss}}$	$\mathsf{Gen}_{\mathsf{inj}}$	$\mathsf{Eval}(\mathsf{pk}, x)$	$\Pi(x)$
Game_0	$\mathsf{pk} \leftarrow \mathsf{Gen}_{\mathsf{loss}}^{\Gamma,\Pi}(1^\lambda)$ **return** pk	$\mathsf{pk} \leftarrow \mathsf{Gen}_{\mathsf{inj}}^{\Gamma,\Pi}(1^\lambda)$ **return** pk	$y \leftarrow \mathsf{Eval}^{\Gamma,\Pi}(\mathsf{pk}, x)$ **return** y	$\Pi(x)$

Game_2	$(\mathsf{pk}, b) \leftarrow\$ \{0,1\}^{6\mu}$
	$a \leftarrow\$ \{0,1\}^{\mu}_{\neq 0^\mu}$
	$k \leftarrow\$ \{0,1\}^{\mu}_{\neq 0^\mu}$
	$\mathsf{st}_{\mathsf{pk}} \leftarrow (k, a, b)$
	return pk
		$(\mathsf{pk}, b) \leftarrow\$ \{0,1\}^{6\mu}$
		$a \leftarrow\$ \{0,1\}^{\mu}_{\neq 0^\mu}$
		$\mathsf{st}_{\mathsf{pk}} \leftarrow (0^\mu, a, b)$
		return pk
			if $\mathsf{st}_{\mathsf{pk}} = \perp$
			  $k, b \leftarrow\$ \{0,1\}^{2\mu}$
			  $a \leftarrow\$ \{0,1\}^{\mu}_{\neq 0^\mu}$
			  $\mathsf{st}_{\mathsf{pk}} \leftarrow (k, a, b)$
			$(k, a, b) \leftarrow \mathsf{st}_{\mathsf{pk}}$
			if $k = 0^\mu$
			  return $\Pi(\mathsf{pk} \Vert \mathsf{setlsb}(ax + b))$
			else
			  return $\Pi(\mathsf{pk} \Vert ax + b)$
				$\Pi(x)$

Game_3	$[\ldots]$
	$\mathsf{st}_{\mathsf{pk}} \leftarrow (\mathsf{loss}, a, b)$
	$[\ldots]$
		$[\ldots]$
		$\mathsf{st}_{\mathsf{pk}} \leftarrow (\mathsf{inj}, a, b)$
		$[\ldots]$
			if $\mathsf{st}_{\mathsf{pk}} = \emptyset$
			  $b \leftarrow\$ \{0,1\}^{\mu}$
			  $a \leftarrow\$ \{0,1\}^{\mu}_{\neq 0^\mu}$
			  $\mathsf{st}_{\mathsf{pk}} \leftarrow (\mathsf{inj}, a, b)$
			$(\mathsf{mode}, a, b) \leftarrow \mathsf{st}_{\mathsf{pk}}$
			if $\mathsf{mode} = \mathsf{loss}$
			  return $\Pi(\mathsf{pk} \Vert \mathsf{setlsb}(ax + b))$
			else
			  return $\Pi(\mathsf{pk} \Vert ax + b)$
				$\Pi(x)$

Game_4	$[\ldots]$ $\mathsf{st}_{\mathsf{pk}} \leftarrow (a, b)$ $[\ldots]$	$[\ldots]$ $\mathsf{st}_{\mathsf{pk}} \leftarrow (a, b)$ $[\ldots]$	$[\ldots]$ $\mathsf{st}_{\mathsf{pk}} \leftarrow (a, b)$ $a, b \leftarrow \mathsf{st}_{\mathsf{pk}}$ **return** $\Pi(\mathsf{pk} \Vert ax + b)$	$\Pi(x)$
Game_5	$[\ldots]$	$[\ldots]$	$[\ldots]$ **return** $\Pi_1(\mathsf{pk} \Vert ax + b)$	$\Pi_1(x)$
Game_6	$\mathsf{pk} \leftarrow\$ \{0,1\}^{5\mu}$ **return** pk	$\mathsf{pk} \leftarrow\$ \{0,1\}^{5\mu}$ **return** pk	$a \Vert b \Vert \cdots \leftarrow \Pi_0(\mathsf{pk})$ **return** $\Pi_1(\mathsf{pk} \Vert ax + b)$	$\Pi_1(x)$
Game_7	$[\ldots]$	$[\ldots]$	**return** $\Pi_0(\mathsf{pk} \Vert x)$	$\Pi_1(x)$

Figure 2: An overview of all the game hops. Note that for simplicity we ignored the modifications related to inputs of length L_0 here, in particular the game hop to Game_1.

$p(\lambda)$ denote the total number of oracle queries the game itself and $\mathcal{A}$ make through the game, and let $\mathsf{Game}_0(\lambda)$ be the original attack of $\mathcal{A}$ with the defined oracles. The final game will then immediately give our algorithm Wrap with the upstream Filter. We give an overview over all the game hops in Figure 2.

Game_1. In the first game hops we let Wrap collect all information about very short queries (of length related to L_0) in a list and use this list to answer subsequent queries. Change the oracles as follows. Let

$$L_0 := L_0(\lambda) := \left\lceil \log_2(80\alpha^{-1}(\lambda) \cdot p(\lambda)^2 + p(\lambda)) \right\rceil.$$

Then our current version of algorithm Wrap, upon initialization, queries Π about all inputs of size at most $2L_0$ and stores the list of queries and answers. The reason for using $2L_0$ is that the evaluation algorithm takes as input a key of security parameter λ and some input of size $\mu \approx \lambda/5$, such that we safely cover all evaluations for keys of security size $\lambda \leq L_0$.

Further, for any security parameter less than $2L_0$, our algorithm queries $\mathsf{Gen}^{\Gamma,\Pi}$ for $\lambda 2^{2L_0}$ times; recall that we do not assume that parties have direct access to Γ but only via $\mathsf{Gen}^{\Gamma,\Pi}$. This way, for any valid key, we know that it was created at some point except with probability $(1-2^{-2L_0})^{\lambda 2^{2L_0}} \leq 2^{-\lambda}$ and therefore the probability that any key was not generated is at most $2^{L_0}2^{-\lambda}$, which is negligible. Further, for every public key, it evaluates $\mathsf{Eval}^{\Gamma,\Pi}$ at $x = 0$ and uses the precomputed list for Π to invert, revealing the corresponding a and b. Note that all of this can be done in polynomial time.

Any subsequent query to $\mathsf{Gen}^{\Gamma,\Pi}$ for security parameter at most L_0, as well as to $\mathsf{Eval}^{\Gamma,\Pi}$ for a public keys of size at most L_0 (which corresponds to a key for security parameter at most L_0), as well as to Π for inputs of size at most $2L_0$, are answered by looking up all necessary data in the list. If any data is missing, we will return $\perp$. Note that as long as we do not return $\perp$, this is only a syntactical change. As returning $\perp$ happens at most with negligible probability over the randomness of Wrap,

$$\mathsf{SD}\left(\mathsf{Game}_0, \mathsf{Game}_1\right) \leq 2^{2L_0}2^{-\lambda}.$$

From now one we will implicitly assume that queries of short security length up to L_0 are answered genuinely with the help of tables and do not mention this explicitly anymore.

Game_2. In this game, we will stop using the lossy function oracles altogether, and instead introduce a global state for the Wrap algorithm. Note that this state will be shared between all parties having access to the oracles (via Wrap). Now, for every call to $\mathsf{Gen}^{\Gamma,\Pi}$, we do the following: If the key is created in injective mode, Wrap will sample $b \leftarrow\$ \{0,1\}^\mu$ and $a, k \leftarrow\$ \{0,1\}^\mu \setminus \{0^\mu\}$, if the key is created in lossy mode, it sets $k = 0^\mu$. Further, it samples a public key $\mathsf{pk} \leftarrow\$ \{0,1\}^{5\mu+\mathsf{pad}}$, and sets the state $\mathsf{st}_{\mathsf{pk}} \leftarrow (k, a, b)$. Finally it returns pk. Any call to $\mathsf{Eval}^{\Gamma,\Pi}(\mathsf{pk}, x)$ will be handled as follows: First, Wrap checks whether a state for pk exists. If this is not the case, we generate $k, a, b \leftarrow\$ \{0,1\}^\mu$ (with checking that $a \neq 0$) and save $\mathsf{st}_{\mathsf{pk}} \leftarrow (k, a, b)$. Then, we read $(k, a, b) \leftarrow \mathsf{st}_{\mathsf{pk}}$ from the (possibly just initialized) state and return $\Pi(a\|b\|ax + b)$.

What algorithm Wrap does here can be seen as emulating Γ. However, there are two differences: We do not sample z, and we allow for collisions. The collisions can be of either of two types: Either we sample the same (random) public key $\mathsf{pk} = \mathsf{pk}'$ but for different state values $(k, a, b) \neq (k', a', b')$, or we sample the same values $(k, a, b) = (k', a', b')$ but end up with different public keys $\mathsf{pk} \neq \mathsf{pk}'$. In this case, an algorithm that finds such a collision of size at least μ for $\mu \geq L_0/5$ —smaller values are precomputed and still answered as before— could be able to distinguish the two games. Still, the two games are statistically close since such collisions happen with probability at most $2^{-2L_0/5+1}$ for each pair of generated keys:

$$\mathsf{SD}\left(\mathsf{Game}_2, \mathsf{Game}_1\right) \leq 2p(\lambda)^2 \cdot 2^{-2L_0/5+1} \leq \frac{\alpha(\lambda)}{8}$$

Game_3. Next, instead of generating and saving a value k depending on the lossy or injective mode, we just save a label inj or loss for the mode the key was created for. Further, whenever $\mathsf{Eval}^{\Gamma,\Pi}(\mathsf{pk}, x)$ is called on a public key without saved state, i.e., if it has not been created via key generation, then we always label this key as injective.

The only way the adversary is able to recognize the game hop change is because a self-chosen public key, not determined by key generation, will now never be lossy (or will be invalid because $a = 0$). However, any adversarially chosen string of size at least $5\mu \geq L_0$ would only describe a lossy key with probability at most $\frac{1}{2^\mu - p(\lambda)}$ and yield an invalid $a = 0$ with the same probability. Hence, taking into account that the adversary learns at most $p(\lambda)$ values about Γ though genuinely generated keys, and the adversary makes at most $p(\lambda)$ queries, the statistical difference between the two games is small:

$$\mathsf{SD}\left(\mathsf{Game}_2, \mathsf{Game}_3\right) \leq 2p(\lambda) \cdot \frac{1}{2^{-L_0/5+1} - p(\lambda)} \leq \frac{\alpha(\lambda)}{8}.$$

Game_4. Now, we remove the label inj or loss again. Wrap will now, for any call to Eval, calculate everything in injective mode.

There are two ways an adversary can distinguish between the two games: Either by inverting Π, e.g., noting that the last bit in the preimage is not as expected, or by finding a pair $x \neq x'$ for a lossy key $\mathsf{pk}_{\mathsf{loss}}$ such that $\mathsf{Eval}(\mathsf{pk}_{\mathsf{loss}}, x) = \mathsf{Eval}(\mathsf{pk}_{\mathsf{loss}}, x')$ in Game_3. Inverting Π (or guessing a and b) only succeeds with probability $\frac{2(p(\lambda)+1)}{2^\mu}$. For the probability of finding a collision, note that viewing the random permutation Π as being lazy sampled (see Appendix A) shows that the answers are chosen independently of the input (except for repeating previous answers), and especially of a, b for any lossy public key of the type considered here. Hence, we can imagine to choose a, b for any possible pairs of inputs only after x, x' have been determined. But then the probability of creating a collision among the $p(\lambda)^2$ many pairs for the same key is at most $\frac{2p(\lambda)^2}{2^\mu}$ for $\mu > L_0/5$. Therefore, the distance between these two games is bounded by

$$\mathsf{SD}\left(\mathsf{Game}_3, \mathsf{Game}_4\right) \leq 3(p(\lambda) + p(\lambda)^2) \cdot 2^{-L_0/5+1} \leq \frac{\alpha(\lambda)}{8}.$$

Game_5. We split the random permutation Π to have two oracles. For $\beta \in \{0,1\}$ and $x \in \{0,1\}^{5\mu}$, we now define

$$\Pi_\beta(x) = \Pi(\beta \| x)_{1 \ldots 5\mu-1},$$

i.e., we add a prefix β and drop the last bit. We now replace any use of Π in Wrap, including direct queries to Π, by Π_1.

Would Π_1 be a permutation, this would be a perfect simulation. However, Π_1 is not even injective anymore, but finding a collision is still very unlikely (as random functions are collision resistant). In particular, using once more that we only look at sufficiently large values, the statistical distance of the games is still small:

$$\mathsf{SD}\left(\mathsf{Game}_4, \mathsf{Game}_5\right) \leq \frac{2p(\lambda)^2}{2^{5\mu}} \leq \frac{\alpha(\lambda)}{8}.$$

Game_6. Next, we stop using the global state st for information about the values related to a public key (except for keys of security parameter at most L_0). The wrapper for Gen now only generates a uniformly random pk and returns it. For Eval calls, Wrap instead calculates $a \| b \leftarrow \Pi_0(\mathsf{pk})$ on the fly. Note that there is a small probability of $2^{-L_0/5+1}$ of $a = 0$, yielding an invalid key. Except for this, since the adversary does not have access to Π_0, this game otherwise looks completely identical to the adversary:

$$\mathsf{SD}\left(\mathsf{Game}_5, \mathsf{Game}_6\right) \leq p(\lambda) \cdot 2^{-L_0/5+1} \leq \frac{\alpha(\lambda)}{8}.$$

Game_7. For our final game, we use Π_0 to evaluate the lossy function:

$$\mathsf{Eval}^{\Pi}(\mathsf{pk}, x) = \Pi_0(\mathsf{pk}\|x).$$

Note that, as $\mathcal{A}$ has no access to Π_0, calls to Eval in Game_7 are random for $\mathcal{A}$. For Game_6, calls to Eval looks random as long as $\mathcal{A}$ does not invert Π_1, which happens at most with probability $\frac{2(p(\lambda)+1)}{2^{\mu}}$. Therefore, the statistical distance between the two games is bound by

$$\mathsf{SD}\left(\mathsf{Game}_6, \mathsf{Game}_7\right) \leq 3p(\lambda) \cdot 2^{-2L_0/5+1} \leq \frac{\alpha(\lambda)}{8}.$$

In the final game the algorithm Wrap now does not need to save any state related to large public keys, and it behaves identically for the lossy and injective generators. We can therefore safely add our algorithm Filter, stripping off the mode before passing key generation requests to Wrap. Summing up the statistical distances we obtain a maximal statistical of $\frac{7}{8}\alpha(\lambda) \leq \alpha(\lambda)$ between the original game and the one with our algorithms Wrap and Filter. □

We next argue that the simulation lemma allows us to conclude immediately that the function oracle in Definition 5.1 is indeed a lossy function:

Theorem 5.3 *The function in Definition 5.1 is a lossy function for lossiness parameter* 2.

Proof. We first prove the easier structural properties. Clearly, the evaluation can be done efficiently given $\mathsf{Eval}^{\Gamma,\Pi}$ as an oracle (despite $\mathsf{Eval}^{\Gamma,\Pi}$ itself requiring exponential time). The same holds for key generation. We next argue that key generation in mode inj leads to an injective function. The reason is that, as Γ is a permutation, k will not be zero by construction when being reconstructed by $\mathsf{Eval}^{\Gamma,\Pi}$, such that subsequent calls to $\mathsf{Eval}^{\Gamma,\Pi}$ will transform different inputs $x \neq x' \in \{0,1\}^{\mu}$ to different inputs values $a\|b\|ax+b \neq a\|b\|ax'+b$ for Π. Similarly, $\mathsf{Eval}^{\Gamma,\Pi}$, for a key in mode loss, will recover a value $k = 0^{\mu}$, causing evaluation to set the least significant bit after the hashing step. Then there are exactly two inputs inputs $x \neq x'$ such that $\mathsf{setlsb}(ax+b) = \mathsf{setlsb}(ax'+b)$. Both inputs thus yield the same output. In conclusion, the function is then two-to-one.

The indistinguishability of injective and lossy keys holds by the simulation lemma as follows. Let Game be the security experiment which on input 1^{λ} first picks a challenge bit $\beta \leftarrow\$ \{0,1\}$ at random, and then queries oracle Gen^{Π} about $(1^{\lambda}, \mathsf{inj})$ if $\beta = 0$ resp. $(1^{\lambda}, \mathsf{loss})$ is $\beta = 1$, obtaining a public key pk. It then initializes the adversary $\mathcal{A}$ on input $(1^{\lambda}, \mathsf{pk})$, from then on relaying all queries of $\mathcal{A}$ to the oracles $\mathsf{Gen}^{\Gamma,\Pi}, \mathsf{Eval}^{\Gamma,\Pi}, \Pi$ and the responses. When $\mathcal{A}$ eventually outputs a guess $\beta' \in \{0,1\}$ the game checks if $\beta = \beta'$ and outputs 1 if and only if this is the case.

Now assume that there was an adversary successfully attacking our lossy function. In particular, the game outputs 1 with probability at least $\frac{1}{2} + \alpha(\lambda)$ for some inverse polynomial α and infinitely many λ. Then we can simulate the game and this adversary with statistical distance at most $\alpha(\lambda)/2$ for any λ via our algorithms Wrap and Filter as in the Simulation Lemma 5.2, such that the game still outputs 1 with probability at least $\frac{1}{2} + \alpha(\lambda)/2$ infinitely often. This, however, contradicts the fact that Wrap with upstream Filter operate completely independent of the key mode, such that $\mathcal{A}$ cannot do better than guessing β with probability $\frac{1}{2}$ in this simulation. □

5.2 Key Exchange

We next argue that given our oracle-based lossy function in the previous section one cannot build a secure key agreement protocol based only this lossy function (and having also access to Π). The line of reasoning follows the one in the renowned work by Impagliazzo and Rudich [IR89]. They show that one cannot build a secure key agreement protocol between Alice and Bob, given only a random permutation oracle Π. To this end they argue that, if we can find NP-witnesses efficiently, say, if we have access to a PSPACE oracle, then the adversary with oracle access to Π can efficiently compute Alice's key given only a transcript of a protocol run between Alice and Bob (both having access to Π).

We use the same argument as in [IR89] here, noting that according to our Simulation Lemma 5.2 we could replace the lossy function oracle relative to Π by our algorithm Wrap^{Π}. This, however, requires some care, especially as Wrap does not provide access to the original Π.

We first define (weakly) secure key exchange protocols relative to some oracle (or a set of oracles) $\mathcal{O}$. We assume that we have an interactive protocol $\left\langle \mathrm{Alice}^{\mathcal{O}}, \mathrm{Bob}^{\mathcal{O}} \right\rangle$ between two efficient parties, both having access to the oracle $\mathcal{O}$. The interactive protocol execution for security parameter 1^λ runs the interactive protocol between $\mathrm{Alice}^{\mathcal{O}}(1^\lambda; z_A)$ for randomness z_A and $\mathrm{Bob}^{\mathcal{O}}(1^\lambda, z_B)$ with randomness z_B, and we define the output to be a triple $(k_A, T, k_B) \leftarrow \left\langle \mathrm{Alice}^{\mathcal{O}}(1^\lambda; z_A), \mathrm{Bob}^{\mathcal{O}}(1^\lambda; z_B) \right\rangle$, where k_A is the local key output by Alice, T is the transcript of communication between the two parties, and k_B is the local key output by Bob. When talking about probabilities over this output we refer to the random choice of randomness z_A and z_B.

Note that we define completeness in a slightly non-standard way by allowing the protocol to create non-matching keys with a polynomial (but non-constant) probability, compared to the negligible probability the standard definition would allow. The main motivation for this definition is that it makes our proof easier, but as we will prove a negative result, this relaxed definition makes our result even stronger.

Definition 5.4 *A key agreement protocol $\langle Alice, Bob \rangle$ relative to an oracle $\mathcal{O}$ is*

complete *if there exists an at least linear polynomial $p(\lambda)$ such that for all large enough security parameters λ:*

$$\Pr\Big[k_A \neq k_B : (k_A, T, k_B) \leftarrow\!\$ \left\langle Alice^{\Pi}(1^\lambda), Bob^{\mathcal{O}}(1^\lambda) \right\rangle\Big] \leq \frac{1}{p(\lambda)}.$$

secure *if for any efficient adversary $\mathcal{A}$ the probability that*

$$\Pr\Big[k^* = k_A : (k_A, T, k_B) \leftarrow\!\$ \left\langle Alice^{\mathcal{O}}(1^\lambda), Bob^{\mathcal{O}}(1^\lambda) \right\rangle, k^* \leftarrow\!\$ \mathcal{A}^{\mathcal{O}}(1^\lambda, T)\Big]$$

is negligible.

Theorem 5.5 *There exist random oracles Π and Γ such that relative to $\mathsf{Gen}^{\Gamma,\Pi}$, $\mathsf{Eval}^{\Gamma,\Pi}$, Π and PSPACE, the function oracle $(\mathsf{Gen}^{\Gamma,\Pi}, \mathsf{Eval}^{\Gamma,\Pi})$ from Definition 5.1 is a lossy function, but no construction of secure key agreement from $\mathsf{Gen}^{\Gamma,\Pi}$, $\mathsf{Eval}^{\Gamma,\Pi}$ and Π exists.*

From this theorem and using the two-oracle technique, the following corollary follows directly:

Corollary 5.6 *There exists no fully black-box construction of a secure key agreement protocol from lossy functions.*

Proof (Theorem 5.5). Assume, to the contrary, that a secure key agreement exists relative to these oracles. We first note that it suffices to consider adversaries in the Wrap-based scenario. That is, $\mathcal{A}$ obtains a transcript T generated by the execution of $\mathrm{Alice}^{\mathsf{Wrap}^{\Gamma,\Pi} \circ \mathsf{Filter}}(1^\lambda; z_A)$ with $\mathrm{Bob}^{\mathsf{Wrap}^{\Gamma,\Pi} \circ \mathsf{Filter}}(1^\lambda; z_A)$ where Wrap is initialized with randomness z_W and itself interacts with Π. Note that $\mathsf{Wrap}^{\Pi} \circ \mathsf{Filter}$ is efficiently computable and only requires local state (holding the oracle tables for small values), so we can interpret the wrapper as part of Alice and Bob without needing any additional communication between the two parties—see Figure 3.

We now prove the following two statements about the key agreement protocol in the wrapped mode:

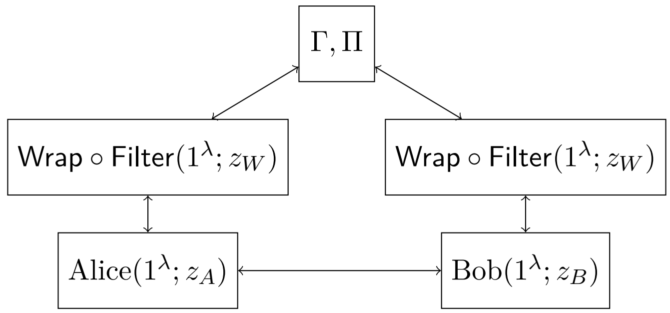

Figure 3

1. For non-constant $\alpha(\lambda)$, the protocol $\langle \text{Alice}^{\mathsf{Wrap}^{\Gamma,\Pi} \circ \mathsf{Filter}}, \text{Bob}^{\mathsf{Wrap}^{\Gamma,\Pi} \circ \mathsf{Filter}} \rangle$ still fulfills the completeness property of the key agreement, i.e., at most with polynomial probability, the keys generated by Alice and Bob differ; and
2. there exists a successful adversary $\mathcal{E}^{\mathsf{Wrap}^{\Gamma,\Pi} \circ \mathsf{Filter}, \text{PSPACE}}$ with additional PSPACE access, that, with at least polynomial probability, recovers the key from the transcript of Alice and Bob.

If we show these two properties, we have derived a contradiction: If there exists a successful adversary against the wrapped version of the protocol, then this adversary must also be successful against the protocol with the original oracles with at most a negligible difference in the success probability – otherwise, this adversary could be used as a distinguisher between the original and the wrapped oracles, contradicting the Simulation Lemma 5.2.

Completeness. The first property holds by the Simulation Lemma: Assume there exists a protocol between Alice and Bob such that in the original game, the keys generated differ for at most a polynomial probability $\frac{1}{p(\lambda)}$, while in the case where we replace the access to the oracles by $\mathsf{Wrap}^{\Gamma,\Pi} \circ \mathsf{Filter}$ for some $\alpha(\lambda)$, the keys differ with constant probability $\frac{1}{c_\alpha}$. In such a case, we could—in a thought experiment—modify Alice and Bob to end their protocol by revealing their keys. A distinguisher could now tell from the transcripts whether the keys of the parties differ or match. Such a distinguisher would however now be able to distinguish between the oracles and the wrapper with probability $\frac{1}{c_\alpha} - \frac{1}{p(\lambda)}$, which is larger than $\alpha(\lambda)$ for large enough security parameters, which is a contradiction to the Simulation Lemma.

Attack. For the second property, we will argue that the adversary by Impagliazzo and Rudich from their seminal work on key agreement from one-way functions [IR89] works in our case as well. For this, first note that the adversary has access to both Π_1 (by Π-calls to Wrap) and Π_0 (by Eval-calls to Wrap) and Wrap also makes the initial calls to Γ. Combining Γ, Π_0 and Π_1 into a single function we can apply the Impagliazzo-Rudich adversary. Specifically, [IR89, Theorem 6.4] relates the agreement error, denoted ϵ here, to the success probability approximately $1 - 2\epsilon$ of breaking the key agreement protocol. Hence, let $\epsilon(\lambda)$ be the at most polynomial error rate of the original key exchange protocol. We choose now $\alpha(\lambda)$ sufficiently small such that $\epsilon(\lambda) + \alpha(\lambda)$ is an acceptable error rate for a key exchange, i.e., at most $1/4$. Then this key exchange using the wrapped oracles is a valid key exchange using only our combined random oracle, and therefore, we can use the Impagliazzo-Rudich adversary to recover the key with non-negligible probability.

Fixing the oracles. Finally, we have to fix the random permutations Π and Γ such that the Simulation Lemma holds and the Impagliazzo-Rudich attack works. The Impagliazzo-Rudich attack is known to work for all but a zero-measure of random oracles. For the Simulation Lemma, we can use Borel-Cantelli to show that for a one-measure of oracle choices of Γ and Π, no successful adversary can distinguish between the original oracles and the wrapped ones.

To show this, let us fix some game, an adversary $\mathcal{A}$ and an $\alpha(\lambda)$. We now know that there exists a wrapper Wrap^{Π} such that the statistical distance between the original game and the wrapped game is smaller than $\frac{\alpha(\lambda)}{\lambda^2}$:

$$\mathsf{SD}\left(\mathsf{Game}_{\mathcal{A}}^{\mathsf{Gen}^{\Gamma,\Pi},\mathsf{Eval}^{\Gamma,\Pi},\Pi}, \mathsf{Game}_{\mathcal{A}}^{\mathsf{Wrap}^{\Pi}\circ\mathsf{Filter}}\right) \leq \frac{\alpha(\lambda)}{\lambda^2}.$$

We can now define $sd_{\mathcal{A},\Gamma,\Pi}$ as the statistical distance for fixed Γ and Π. Then, we know that the expected value of $sd_{\Gamma,\Pi}$ is bound by

$$\mathbb{E}_{\Gamma,\Pi}[sd_{\mathcal{A},\Gamma,\Pi}] \leq \frac{\alpha(\lambda)}{\lambda^2}.$$

Using the Markov inequality, we get

$$\Pr[sd_{\mathcal{A},\Gamma,\Pi} \geq \alpha(\lambda)] \leq \frac{1}{\lambda^2}.$$

Let us now denote by E_λ the event that for security parameter λ, the statistical distance $sd_{\mathcal{A},\Gamma,\Pi}$ is at least $\alpha(\lambda)$. Since the hyperharmonic series converges, we have that

$$\sum_{\lambda=1}^{\infty} \Pr[E_\lambda] < \infty$$

and we can therefore apply Borel-Cantelli to get that the statistical distance is not bound by $\alpha(\lambda)$ for infinitely many security parameters only happens for a zero measure of oracles Γ and Π:

$$\Pr_{\Gamma,\Pi}\left[\bigwedge_{k=1}^{\infty} \bigvee_{\lambda \geq k} E_\lambda\right] = 0.$$

As there exist only countable many (uniform) adversaries, there exists a one-measure of oracles Γ, Π such that the Simulation Lemma holds for any adversary. Therefore, there clearly exist some oracles Γ, Π such that the Simulation Lemma holds, while the Impagliazzo-Rudich attack is successful. $\square$

5.3 ELFs

We will show next that our result can also be extended to show that no fully black-box construction of key agreement from *extremely* lossy functions is possible. However, we are only able to show a slightly weaker result: In our separation, we only consider constructions that access the extremely lossy function on the same security parameter as used in the key agreement protocol. We call such constructions *security-level-preserving*. This leaves the theoretic possibility of building key agreement from extremely lossy functions of (significantly) smaller security parameters. At the same time it simplifies the proof of the Simulation Lemma for this case significantly since we can omit the step where Wrap samples Γ for all small inputs, and we can immediately work with the common negligible terms.

We start by defining an ELF oracle. In general, the oracle is quite similar to our lossy function oracle. Especially, we still distinguish between an injective and a lossy mode, and make sure that any key sampled without a call to the $\mathsf{Gen}_{\mathsf{ELF}}^{\Gamma,\Pi}$ oracle will be injective with overwhelming probability. For the lossy mode, we now of course have to save the parameter r in the public key. Instead of using setlsb to lose one bit of information, we take the result of $ax + b$ (calculated in $GF(2^\mu)$) modulo r (calculated on the integers) to allow for the more fine-grained lossiness that is required by ELFs.

Definition 5.7 (Extremely Lossy Function Oracle) *Let Π, Γ be permutation oracles with $\Pi, \Gamma : \{0,1\}^\lambda \to \{0,1\}^\lambda$ for all λ. Let $\mu = \mu(\lambda) = \lfloor(\lambda-2)/5\rfloor$ and $\mathsf{pad} = \mathsf{pad}(\lambda) = \lambda - 2 - 5\mu$ defines the length that the rounding-off loses to $\lambda - 2$ in total (such that $\mathsf{pad} \in \{0,1,2,3,4\}$. Define the extremely lossy function $(\mathsf{Gen}_{\mathsf{ELF}}^{\Gamma,\Pi}, \mathsf{Eval}_{\mathsf{ELF}}^{\Gamma,\Pi})$ with input length $in(\lambda) = \mu(\lambda)$ relative to Γ and Π now as follows:*

Key Generation: *Oracle $\mathsf{Gen}_{\mathsf{ELF}}^{\Gamma,\Pi}$ on input 1^λ and mode r picks random $b \leftarrow\$ \{0,1\}^\mu$, $z \leftarrow\$ \{0,1\}^{\mu+\mathsf{pad}}$ and random $a, k \leftarrow\$ \{0,1\}^\mu \setminus \{0^\mu\}$. For mode $r = 2^{in(\lambda)}$ the algorithm returns $\Gamma(k\|a\|b\|r\|z)$. For mode $r < 2^{in(\lambda)}$ the algorithm returns $\Gamma(0^\mu\|a\|b\|r\|z)$ instead.*

Evaluation: *On input $\mathsf{pk} \in \{0,1\}^\lambda$ and $x \in \{0,1\}^\mu$ algorithm $\mathsf{Eval}_{\mathsf{ELF}}^{\Gamma,\Pi}$ first recovers (via exhaustive search) the preimage $k\|a\|b\|r\|z$ of pk under Γ for $k, a, b, r \in \{0,1\}^\mu$, $z \in \{0,1\}^{\mu+\mathsf{pad}}$. Check that $a \neq 0$ in the field $GF(2^\mu)$. If any check fails then return $\perp$. Else, next check if $k = 0^m$. If so, return $\Pi(a\|b\|(ax+b \mod r))$, else return $\Pi(a\|b\|ax+b)$.*

We can now formulate versions of Theorem 5.5 and Corollary 5.6 for the extremely lossy case.

Theorem 5.8 *There exist random oracles Π and Γ such that relative to $\mathsf{Gen}_{\mathsf{ELF}}^{\Gamma,\Pi}$, $\mathsf{Eval}_{\mathsf{ELF}}^{\Gamma,\Pi}$, Π and PSPACE, the extremely lossy function oracle $(\mathsf{Gen}_{\mathsf{ELF}}^{\Gamma,\Pi}, \mathsf{Eval}_{\mathsf{ELF}}^{\Gamma,\Pi})$ from Definition 5.7 is indeed an ELF, but no security-level-preserving construction of secure key agreement from $\mathsf{Gen}_{\mathsf{ELF}}^{\Gamma,\Pi}$,$\mathsf{Eval}_{\mathsf{ELF}}^{\Gamma,\Pi}$ and Π exists.*

Corollary 5.9 *There exists no fully black-box security-level-preserving construction of a secure key agreement protocol from extremely lossy functions.*

Proving Theorem 5.8 only needs minor modifications of the proof of Theorem 5.5 to go through. Indeed, the only real difference lies in a modified Simulation Lemma for ELFs, which we will formulate next, together with a proof sketch that explains where differences arrive in the proof compared to the original Simulation Lemma. To stay as close to the previous proof as possible, we will continue to distinguish between an injective generator $\mathsf{Gen}_{\mathsf{inj}}(1^\lambda)$ and a lossy generator $\mathsf{Gen}_{\mathsf{loss}}(1^\lambda, r)$, where the latter also receives the parameter r. Figure 4 provides an overview of the modified game hops.

Lemma 5.10 (Simulation Lemma (ELFs)) *Let Filter be a deterministic algorithm which for calls $(1^\lambda, \mathsf{mode})$ to $\mathsf{Gen}_{\mathsf{ELF}}^{\Gamma,\Pi}$ only outputs 1^λ and leaves any input to calls to $\mathsf{Eval}_{\mathsf{ELF}}^{\Gamma,\Pi}$ and to Π unchanged. There exists an efficient algorithm Wrap such that for any polynomials p and d' there exists a polynomial q such that for any adversary $\mathcal{A}$ which makes at most $p(\lambda)$ queries to the oracles, any efficient experiment Game making calls to the $\mathsf{Gen}_{\mathsf{ELF}}^{\Gamma,\Pi}$ oracle with $r > q(\lambda)$ the distinguishing advantage between $\mathsf{Game}^{\mathcal{A},(\mathsf{Gen}_{\mathsf{ELF}}^{\Gamma,\Pi},\mathsf{Eval}_{\mathsf{ELF}}^{\Gamma,\Pi},\Pi)}(1^\lambda)$ and $\mathsf{Game}^{\mathcal{A},\mathsf{Wrap}^\Pi\circ\mathsf{Filter}}$ is at most $\frac{1}{d'(\lambda)}$ for sufficiently large λ. Furthermore Wrap makes at most two calls to Π for each query.*

Proof (Sketch). We will now describe how the game hops differ from the proof of Lemma 5.2, and how these changes affect the advantage of the distinguisher. Note that allowing only access to the ELF oracle at the current security parameter allows us to argue that differences between game hops are negligible, instead of having to give a concrete bound.

Game_1**.** stays identical to Game_0 – as we only allow access to the ELF oracle at the current security level, precomputing all values smaller than some L_0 is not necessary here.

Game	$\mathsf{Gen}_{\mathsf{loss}}(r)$	$\mathsf{Gen}_{\mathsf{inj}}$	$\mathsf{Eval}(\mathsf{pk}, x)$	$\Pi(x)$
Game_0	$\mathsf{pk} \leftarrow \mathsf{Gen}_{\mathsf{loss}}^{\Gamma,\Pi}(1^\lambda, r)$ **return** pk	$\mathsf{pk} \leftarrow \mathsf{Gen}_{\mathsf{inj}}^{\Gamma,\Pi}(1^\lambda)$ **return** pk	$y \leftarrow \mathsf{Eval}_{\mathsf{ELF}}^{\Gamma,\Pi}(\mathsf{pk}, x)$ **return** y	$\Pi(x)$

Game_2	$(\mathsf{pk}, b) \leftarrow\$ \{0,1\}^{6\mu}$
	$a \leftarrow\$ \{0,1\}^{\mu}_{\neq 0^\mu}$
	$k \leftarrow\$ \{0,1\}^{\mu}_{\neq 0^\mu}$
	$\mathsf{st}_{\mathsf{pk}} \leftarrow (k, a, b, r)$
	return pk
		$(\mathsf{pk}, b) \leftarrow\$ \{0,1\}^{6\mu}$
		$a \leftarrow\$ \{0,1\}^{\mu}_{\neq 0^\mu}$
		$\mathsf{st}_{\mathsf{pk}} \leftarrow (0^\mu, a, b, 0^\mu)$
		return pk
			if $\mathsf{st}_{\mathsf{pk}} = \emptyset$
			  $k, b, r \leftarrow\$ \{0,1\}^{3\mu}$
			  $a \leftarrow\$ \{0,1\}^{\mu}_{\neq 0^\mu}$
			  $\mathsf{st}_{\mathsf{pk}} \leftarrow (k, a, b, r)$
			$k, a, b, r \leftarrow \mathsf{st}_{\mathsf{pk}}$
			if $k = 0^\mu$
			  return $\Pi(\mathsf{pk} \\| (ax + b) \bmod r)$
			else
			  return $\Pi(\mathsf{pk} \\| ax + b)$
				$\Pi(x)$

Game_3	$[\dots]$
	$\mathsf{st}_{\mathsf{pk}} \leftarrow (\mathsf{loss}, a, b, r)$
	$[\dots]$
		$[\dots]$
		$\mathsf{st}_{\mathsf{pk}} \leftarrow (\mathsf{inj}, a, b, 0^\mu)$
		$[\dots]$
			if $\mathsf{st}_{\mathsf{pk}} = \emptyset$
			  $b \leftarrow\$ \{0,1\}^{\mu}$
			  $a \leftarrow\$ \{0,1\}^{\mu}_{\neq 0^\mu}$
			  $\mathsf{st}_{\mathsf{pk}} \leftarrow (\mathsf{inj}, a, b, 0^\mu)$
			$\mathsf{mode}, a, b, r \leftarrow \mathsf{st}_{\mathsf{pk}}$
			if $\mathsf{mode} = \mathsf{loss}$
			  return $\Pi(\mathsf{pk} \\| (ax + b) \bmod r)$
			else
			  return $\Pi(\mathsf{pk} \\| ax + b)$
				$\Pi(x)$

Game_4	$[\dots]$ $\mathsf{st}_{\mathsf{pk}} \leftarrow (a, b)$ $[\dots]$	$[\dots]$ $\mathsf{st}_{\mathsf{pk}} \leftarrow (a, b)$ $[\dots]$	$[\dots]$ $\mathsf{st}_{\mathsf{pk}} \leftarrow (a, b)$ $a, b \leftarrow \mathsf{st}_{\mathsf{pk}}$ **return** $\Pi(\mathsf{pk} \\| ax + b)$	$\Pi(x)$
Game_5	$[\dots]$	$[\dots]$	$[\dots]$ **return** $\Pi_1(\mathsf{pk} \\| ax + b)$	$\Pi_1(x)$
Game_6	$\mathsf{pk} \leftarrow\$ \{0,1\}^{5\mu}$ **return** pk	$\mathsf{pk} \leftarrow\$ \{0,1\}^{5\mu}$ **return** pk	$a \\| b \\| \cdots \leftarrow \Pi_0(\mathsf{pk})$ **return** $\Pi_1(\mathsf{pk} \\| ax + b)$	$\Pi_1(x)$
Game_7	$[\dots]$	$[\dots]$	**return** $\Pi_0(\mathsf{pk} \\| x)$	$\Pi_1(x)$

Figure 4: An overview of all the game hops for the Simulation Lemma, ELF version.

Game_2. introduces changes similar to Game_2 in Lemma 5.2 – however, we now of course also have to save the parameter r in the state. Again, the only notable difference to the distinguisher is that we sample pk independently of the public key parameters and therefore, collisions might happen more often. However, the probability for this is clearly negligible:

$$\mathsf{SD}\left(\mathsf{Game}_1, \mathsf{Game}_2\right) \leq \mathsf{negl}(\lambda)$$

Game_3. replaces k with a label inj or loss. Again, the only noticeable difference is that keys sampled without calling $\mathsf{Gen}_{\mathsf{inj}}$ or $\mathsf{Gen}_{\mathsf{loss}}$ will now always be injective, while they are lossy with probability $2^{-\mu}$ in Game_2, yielding only a negligible difference between the two games however.

$$\mathsf{SD}\left(\mathsf{Game}_2, \mathsf{Game}_3\right) \leq \mathsf{negl}(\lambda)$$

Game_4. is the game where we start to always evaluate in injective mode. There are two options a distinguisher might distinguish between the two games: Either by inverting Π, or by finding a collision for a lossy key. Inverting Π only happens with probability $\frac{2(p(\lambda)+1)}{2^{\mu}}$, while finding a collision happens with probability $\frac{2p(\lambda)^2}{r}$. Let $d(\lambda) = \frac{d'(\lambda)}{2}$ be the advantage we want to allow for the distinguisher in this game hop. Choosing

$$q(\lambda) = 4p(\lambda)^2 d(\lambda)$$

for the bound on r of the ELF, we get

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Game}_3,\mathsf{Game}_4}(\lambda) \leq \frac{1}{d(\lambda)}$$

Game_4 is now identical to Game_4 in the proof of Lemma 5.2 (except for the different handling of calls to security parameters smaller than L_0). Therefore, all game hops up to Game_7 are identical to the ones in the proof of Lemma 5.2, with the statistical difference being negligible for all of them. Therefore, the overall advantage of an distinguisher is bounded by $\frac{1}{d(\lambda)} + \mathsf{negl}(\lambda) \leq \frac{1}{d'(\lambda)}$ for large enough security parameters λ. $\square$

Let $\langle \text{Alice}^{\mathsf{Gen}_{\mathsf{ELF}}^{\Gamma,\Pi},\mathsf{Eval}_{\mathsf{ELF}}^{\Gamma,\Pi},\Pi}, \text{Bob}^{\mathsf{Gen}_{\mathsf{ELF}}^{\Gamma,\Pi},\mathsf{Eval}_{\mathsf{ELF}}^{\Gamma,\Pi},\Pi}\rangle$ be some candidate key agreement protocol with completeness error $\frac{1}{\epsilon(\lambda)} < \frac{1}{8}$ that makes at most $p(\lambda)$ queries in sum, and let $\frac{1}{d'(\lambda)} < \frac{1}{8}$ be the advantage bound for any adversary against the key agreement we are trying to reach.

To determine the correct parameters for the ELF oracle, we need to know how many queries the Impagliazzo-Rudich adversary makes against the transcript of the wrapped version of the protocol $\langle \text{Alice}^{\mathsf{Wrap}^{\Pi}\circ\mathsf{Filter}}, \text{Bob}^{\mathsf{Wrap}^{\Pi}\circ\mathsf{Filter}}\rangle$, which depends on the number of queries of the protocol. Note that we know that Wrap^{Π} makes at most two queries to Π for each internal query of Alice or Bob, so we know that the wrapped version makes at most $2p(\lambda)$ queries to Π. Let $p'(\lambda)$ be the number of queries needed by the Impagliazzo-Rudich protocol.

First, we have to show that completeness still holds for the wrapped version of the protocol. The wrapped protocol has an error rate of at most $\frac{1}{\epsilon'} < \frac{1}{\epsilon} + \frac{1}{d'} \leq \frac{1}{4}$, as otherwise, we would have a successful distinguisher for the Simulation Lemma. Further, as the error rate $\frac{1}{\epsilon'}$ is smaller than $\frac{1}{4}$, we know that Impagliazzo-Rudich will have a success probability of at least $\frac{1}{2}$.

Further, we know from the Simulation Lemma that we need $d(\lambda) = \frac{d'(\lambda)}{2}$ for it to hold. Therefore, we set the bound for r in the ELF oracle to

$$q(\lambda) = 4p'(\lambda)^2 d(\lambda).$$

Now, the Impagliazzo-Rudich attack has to be successful for the original protocol with polynomial probability $\frac{1}{d''}$, as otherwise, there would be an distinguisher for the Simulation Lemma with advantage $\frac{1}{2} - \mathsf{negl}(\lambda) > \frac{1}{d'(\lambda)}$. Fixing oracles Π, Γ such that $(\mathsf{Gen}_{\mathsf{ELF}}^{\Gamma,\Pi}, \mathsf{Eval}_{\mathsf{ELF}}^{\Gamma,\Pi})$ is an ELF, while the Impagliazzo-Rudich attack is successful yields the Theorem.

6 Relationship of Lossy Functions to Statistical Zero-Knowledge

The complexity class (average-case) SZK, introduced by Goldwasser, Micali and Rackoff [GMR85], contains all languages that can be proven by a statistical zero-knowledge proof, and is often characterized by its complete promise problem (average-case) Statistical Distance [SV03]. Hardness of Statistical Zero-Knowledge follows from a number of algebraic assumptions like Discrete Logarithm [GK93] and lattice problems [MV03] and the existence of some Minicrypt primitives like one-way functions [Ost91] and distributional collision resistant hash functions [KY18] follow from hard problems in SZK – it is not known to follow from any Minicrypt assumptions, however, and for some, e.g., collision-resistant hash functions, there exist black-box separations [BD19].

Therefore, average-case hard problems in SZK seem to be a natural candidate for a non-public key assumption to build lossy functions from. Intuitively, one can see similarities between lossy functions and statistical distance: Both are, in a sense, promise problems, if one looks at the image size of a lossy function with a large gap between the injective mode and the lossy mode. Further, it is known that hard problems in SZK follow from lossy functions (this seems to be folklore knowledge – we give a proof for this fact in Appendix B).

Note that a construction of lossy functions would also be interesting from a different perspective: As collision-resistant hash functions can be build from sufficiently lossy functions, a construction of (sufficiently) lossy functions from average-case SZK hardness would mean that collision resistance follows from average-case SZK hardness. However, right now, this is only known for *distributional* collision resistance, a weaker primitive [KY18].

Alas, we are unable to either give a construction of a lossy function from a hard-on-average statistical zero-knowledge problem or to prove an black-box impossibility result between the two, leaving this as an interesting open question for future work. Instead, we give a lower bound on the needed assumptions for hard-on-average problems in SZK by showing that no Oraclecrypt primitive can be used in a black-box way to construct a hard-on-average problem in SZK – this serves as hint that indeed SZK is an interesting class of problems to look at for building lossy functions, but the result might also be interesting independently.

Note some Oraclecrypt primitives, such a separation already exists: For example, Bitansky and Degwekar give an oracle separation between collision-resistant hash functions and (even worst-case) hard problems in SZK. However, this result uses a Simon-style oracle separation (using a *break*-oracle that depends on the random oracle), which means that the result is specific to the primitive and does not easily generalize to all Oraclecrypt primitives.

Theorem 6.1 *There exists no black-box construction of an hard-on-average problem in SZK from any Oraclecrypt primitive.*

Our proof techniques will be quite similar to Chapter 4: First, we will reuse the oracles $\mathcal{O}$ and $\text{PSPACE}^{\mathcal{O}'}$. As average-case statistical distance is complete for average-case SZK, we will assume there exists an hard-on-average statistical distance problem relative to these random oracles. We will then calculate the heavy queries of the circuits produced by the statistical distance problem and show that the heavy queries are sufficient to decide whether the circuits are statistically far from each other or not, yielding a contradiction to the assumed hardness-on-average of statistical distance. Fixing $\mathcal{O}$ and $\text{PSPACE}^{\mathcal{O}'}$ to specific oracles then yields the theorem.

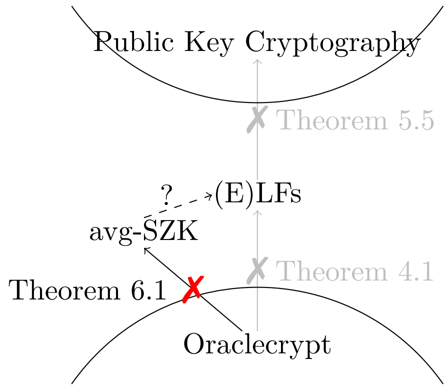

Figure 5: We show an oracle separation between Oraclecrypt and average-case SZK as well. The question whether lossy functions can be build from average-case SZK is still open.

We will start by defining average-case statistical distance:

Definition 6.2 (Average-case Statistical Distance) *An average-case statistical distance problem is characterized by a pair of probabilistic polynomial-time algorithms $D_Y^{\mathcal{O}}(1^\lambda)$ and $D_N^{\mathcal{O}}(1^\lambda)$ each producing pairs of circuits $C_0^{\mathcal{O}}, C_1^{\mathcal{O}} : \{0,1\}^\lambda \to \{0,1\}^\lambda$ such that:*

$$\forall (C_0^{\mathcal{O}}, C_1^{\mathcal{O}}) \leftarrow D_Y^{\mathcal{O}}(1^\lambda) : \mathsf{SD}\left(C_0^{\mathcal{O}}, C_1^{\mathcal{O}}\right) \geq \frac{2}{3},$$
$$\forall (C_0^{\mathcal{O}}, C_1^{\mathcal{O}}) \leftarrow D_N^{\mathcal{O}}(1^\lambda) : \mathsf{SD}\left(C_0^{\mathcal{O}}, C_1^{\mathcal{O}}\right) \leq \frac{1}{3}.$$

Statistical distance is considered hard-on-average if there exists such a pair (D_Y, D_N) such that no polynomial-time adversary $\mathcal{A}$ has more than negligible advantage in distinguishing Yes-instances from No-instances:

$$\forall \mathcal{A} \forall \lambda \in \mathbb{N} : \Pr_{b \leftarrow\$ \{Y,N\};\, (C_0,C_1) \leftarrow D_b(1^\lambda)}\left[\mathcal{A}(1^\lambda, C_0, C_1) = b\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda).$$

Lemma 6.3 *Let $D_?^{\mathcal{O}}(1^\lambda)$ be a polynomial-time sampler of instances of the statistical distance problem for either Yes- or No-instances, and let $(C_0^{\mathcal{O}}, C_1^{\mathcal{O}}) \leftarrow D_?^{\mathcal{O}}(1^\lambda)$. Then we can compute in probabilistic polynomial-time (in λ) a set $\hat{Q}_H$ which contains all $\frac{1}{10k}$-heavy queries of both $C_0^{\mathcal{O}}$ and $C_1^{\mathcal{O}}$ to $\mathcal{O}$ with overwhelming probability.*

Again, k denotes the maximum number of queries of either $D_?^{\mathcal{O}}$ or $C_0^{\mathcal{O}}$ or $C_1^{\mathcal{O}}$, whichever is highest. The proof is identical to the one of Lemma 4.4, except that we now have to find the heavy queries for two circuits instead of one function.

Next, we want to show that we can approximate the statistical distance of $C_0^{\mathcal{O}}$ and $C_1^{\mathcal{O}}$ with the heavy queries and by using the $\mathrm{PSPACE}^{\mathcal{O}'}$ oracle. First, note that we can calculate the statistical difference of $C_0^{\mathcal{O}'}$ and $C_1^{\mathcal{O}'}$ efficiently using the $\mathrm{PSPACE}^{\mathcal{O}'}$ oracle – see Figure 6 for an algorithm. Similar to Chapter 4, we observe that the circuit sampler $D_Y^{\mathcal{O}}$ (or $D_N^{\mathcal{O}}$) only makes polynomially many queries to the oracle $\mathcal{O}$ and, by correctness, has therefore to produce circuits $C_0^{\tilde{\mathcal{O}}}, C_1^{\tilde{\mathcal{O}}}$ for any extension $\tilde{\mathcal{O}}$ of the polynomially-many queries.

Lemma 6.4 *Let $D_?^{\mathcal{O}}(1^\lambda)$ be a polynomial-time sampler of instances of the statistical distance problem for either Yes- or No-instances, and let $(C_0^{\mathcal{O}}, C_1^{\mathcal{O}}) \leftarrow D_?^{\mathcal{O}}(1^\lambda)$. Let $\hat{Q}_H$ be the heavy queries of both $C_0^{\mathcal{O}}$ and $C_1^{\mathcal{O}}$. Then, we can distinguish whether $(C_0^{\mathcal{O}}, C_1^{\mathcal{O}})$ are far or close with high probability using* $\mathrm{PSPACE}^{\mathcal{O}'}$.

$$
\begin{array}{l}
\underline{\mathsf{SD}\left(C_0^{\mathcal{O}'}, C_1^{\mathcal{O}'}\right)} \\
d \leftarrow 0 \\
\forall y \in im(C_0^{\mathcal{O}'} \cup C_1^{\mathcal{O}'}) : \\
\quad s_0 \leftarrow \left|\{x : C_0^{\mathcal{O}'}(x) = y\}\right| \\
\quad s_1 \leftarrow \left|\{x : C_1^{\mathcal{O}'}(x) = y\}\right| \\
\quad d \leftarrow d + |s_0 - s_1| \\
\textbf{return } \frac{1}{2} \cdot \frac{d}{2^n}
\end{array}
$$

Figure 6: An algorithm to calculate the statistical distance of two oracle-aided circuits using the $\mathrm{PSPACE}^{\mathcal{O}'}$ oracle. We go through all possible images of the two circuits, then check for each image how many preimages it has under either circuit. We then add up the difference of the number of preimages between the circuits, and finally return the normalized sum. As we do not use more than polynomial space at any point in time, this algorithm is indeed in $\mathrm{PSPACE}^{\mathcal{O}'}$.

Proof. We distinguish the two cases as follows: We define $\mathcal{O}'_{\hat{Q}_H}$ as the random oracle $\mathcal{O}'$ modified to match $\mathcal{O}$ for all queries in $\hat{Q}_H$. Now, we use the $\mathrm{PSPACE}^{\mathcal{O}'}$ oracle to calculate the statistical distance of $C_0^{\mathcal{O}'_{\hat{Q}_H}}$ and $C_1^{\mathcal{O}'_{\hat{Q}_H}}$ using the $\mathrm{PSPACE}^{\mathcal{O}'}$ oracle (note that by definition, $\hat{Q}_H$ is of polynomial size and therefore can be passed to the $\mathrm{PSPACE}^{\mathcal{O}'}$ oracle). If the statistical distance is at least $\frac{1}{2}$, we claim it was generated by $D_Y^{\mathcal{O}}$, otherwise, we claim it was created by $D_N^{\mathcal{O}}$.

Now, of course, we still need to show that if the two circuits were far for the original oracle, they are still far for our new oracle – and that if they were close for the original oracle, they are still close for our new oracle. We know this must true for all changes in the oracle that were not queried by the distribution sampler $D_?$, and we made sure that no heavy queries are changed between the two oracles, so the only thing that might influence the closeness or farness of the circuits are the polynomially many queries done by $D_?$ that are not heavy queries. We will now show that while these queries might push the statistical difference beyond the $\frac{1}{3}$ or $\frac{2}{3}$ bound, they will not change the statistical difference enough to push the statistical difference beyond the $\frac{1}{2}$ bound, meaning that our distinguisher is still able to tell them apart.

Let k' denote the number of these non-heavy queries asked by the distribution sampler. By definition of k, we know that $k' \leq k$. Each of the queries is non-heavy, which means that only a $\frac{1}{10k}$-fraction of inputs generates different outputs if one of these queries is changed between the oracles. Each changed output changes the statistical difference by at most $\frac{1}{2^\lambda}$. Therefore, each changed non-heavy query changes the statistical distance by at most $\frac{1}{10k}$, and all non-heavy queries asked by the distribution sampler change the statistical distance by at most $\frac{k'}{10k} \leq \frac{1}{10}$. However, as $\frac{1}{3} + \frac{1}{10} < \frac{1}{2}$ (and $\frac{2}{3} - \frac{1}{10} > \frac{1}{2}$), the changes do not push the statistical distance beyond the $\frac{1}{2}$ bound, and our distinguisher is therefore successful. ☐

Proof (of Theorem 6.1). The proof works again in the same vein as the proof of Theorem 4.1: Lemma 6.3 and Lemma 6.4 show that with overwhelming probability, hard-on-average SZK problems do not exist relative to $\mathcal{O}$ and $\mathrm{PSPACE}^{\mathcal{O}'}$, over the probability of choosing $\mathcal{O}$ and $\mathcal{O}'$. Now, we just have to show that there exists a fixed pair of oracles such that our Oraclecrypt primitive exists, but hard-on-average SZK problems do not – which we can easily do using the methods described in Section 4.4. This separation then proves that no fully-black-box construction of a hard-on-average SZK problem from any Oraclecrypt primitive may exist. ☐

Acknowledgments

We thank the anonymous reviewers for valuable comments.

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1119 – 236615297 and by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References

[ACH20] Thomas Agrikola, Geoffroy Couteau, and Dennis Hofheinz. The usefulness of sparsifiable inputs: How to avoid subexponential iO. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12110 of *Lecture Notes in Computer Science*, pages 187–219, Edinburgh, UK, May 4–7, 2020. Springer, Heidelberg, Germany. (Cited on page 2.)

[BBF13] Paul Baecher, Christina Brzuska, and Marc Fischlin. Notions of black-box reductions, revisited. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 296–315, Bengalore, India, December 1–5, 2013. Springer, Heidelberg, Germany. (Cited on page 9.)

[BCE+23] Chris Brzuska, Geoffroy Couteau, Christoph Egger, Pihla Karanko, and Pierre Meyer. New random oracle instantiations from extremely lossy functions. Cryptology ePrint Archive, Report 2023/1145, 2023. `https://eprint.iacr.org/2023/1145`. (Cited on page 2.)

[BD19] Nir Bitansky and Akshay Degwekar. On the complexity of collision resistant hash functions: New and old black-box separations. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 422–450, Nuremberg, Germany, December 1–5, 2019. Springer, Heidelberg, Germany. (Cited on pages 3, 5, and 29.)

[BDV17] Nir Bitansky, Akshay Degwekar, and Vinod Vaikuntanathan. Structure vs. hardness through the obfuscation lens. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 696–723, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany. (Cited on page 5.)

[BFM14] Christina Brzuska, Pooya Farshim, and Arno Mittelbach. Indistinguishability obfuscation and UCEs: The case of computationally unpredictable sources. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 188–205, Santa Barbara, CA, USA, August 17–21, 2014. Springer, Heidelberg, Germany. (Cited on page 1.)

[BHK11] Mark Braverman, Avinatan Hassidim, and Yael Tauman Kalai. Leaky pseudo-entropy functions. In Bernard Chazelle, editor, *Innovations in Computer Science - ICS 2011, Tsinghua University, Beijing, China, January 7-9, 2011. Proceedings*, pages 353–366. Tsinghua University Press, 2011. (Cited on page 2.)

[BHK13] Mihir Bellare, Viet Tung Hoang, and Sriram Keelveedhi. Instantiating random oracles via UCEs. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 398–415, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany. (Cited on page 1.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93: 1st Conference on Computer and Communications Security*, pages 62–73, Fairfax, Virginia, USA, November 3–5, 1993. ACM Press. (Cited on page 1.)

[BST16] Mihir Bellare, Igors Stepanovs, and Stefano Tessaro. Contention in cryptoland: Obfuscation, leakage and UCE. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 542–564, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany. (Cited on page 1.)

[CCR16] Ran Canetti, Yilei Chen, and Leonid Reyzin. On the correlation intractability of obfuscated pseudorandom functions. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A: 13th Theory of Cryptography Conference, Part I*, volume 9562 of *Lecture Notes in Computer Science*, pages 389–415, Tel Aviv, Israel, January 10–13, 2016. Springer, Heidelberg, Germany. (Cited on page 1.)

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th Annual ACM Symposium on Theory of Computing*, pages 209–218, Dallas, TX, USA, May 23–26, 1998. ACM Press. (Cited on page 1.)

[Dur64] Richard Durstenfeld. Algorithm 235: Random permutation. *Commun. ACM*, 7(7):420, 1964. (Cited on page 35.)

[DVW20] Yevgeniy Dodis, Vinod Vaikuntanathan, and Daniel Wichs. Extracting randomness from extractor-dependent sources. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 313–342, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany. (Cited on page 2.)

[GHMM18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ameer Mohammed. Limits on the power of garbling techniques for public-key encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 335–364, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. (Cited on page 3.)

[GK93] Oded Goldreich and Eyal Kushilevitz. A perfect zero-knowledge proof system for a problem equivalent to the discrete logarithm. *Journal of Cryptology*, 6(2):97–116, June 1993. (Cited on page 29.)

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In *17th Annual ACM Symposium on Theory of Computing*, pages 291–304, Providence, RI, USA, May 6–8, 1985. ACM Press. (Cited on page 29.)

[HL18] Justin Holmgren and Alex Lombardi. Cryptographic hashing from strong one-way functions (or: One-way product functions and their applications). In Mikkel Thorup, editor, *59th Annual Symposium on Foundations of Computer Science*, pages 850–858, Paris, France, October 7–9, 2018. IEEE Computer Society Press. (Cited on pages 2 and 37.)

[HR04] Chun-Yuan Hsiao and Leonid Reyzin. Finding collisions on a public road, or do secure hash functions need secret coins? In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 92–105, Santa

Barbara, CA, USA, August 15–19, 2004. Springer, Heidelberg, Germany. (Cited on pages 3, 5, and 9.)

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Proceedings of the Tenth Annual Structure in Complexity Theory Conference, Minneapolis, Minnesota, USA, June 19-22, 1995*, pages 134–147. IEEE Computer Society, 1995. (Cited on pages 2 and 10.)

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st Annual ACM Symposium on Theory of Computing*, pages 44–61, Seattle, WA, USA, May 15–17, 1989. ACM Press. (Cited on pages 2, 3, 4, 6, 9, 17, 18, 22, 23, and 24.)

[Knu98] Donald Ervin Knuth. *The art of computer programming, Volume II: Seminumerical Algorithms, 3rd Edition*. Addison-Wesley, 1998. (Cited on page 35.)

[KY18] Ilan Komargodski and Eylon Yogev. On distributional collision resistant hashing. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 303–327, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany. (Cited on page 29.)

[MF21] Arno Mittelbach and Marc Fischlin. *The Theory of Hash Functions and Random Oracles*. Springer, 2021. (Cited on page 37.)

[MMN+16] Mohammad Mahmoody, Ameer Mohammed, Soheil Nematihaji, Rafael Pass, and abhi shelat. A note on black-box separations for indistinguishability obfuscation. Cryptology ePrint Archive, Report 2016/316, 2016. `https://eprint.iacr.org/2016/316`. (Cited on pages 15 and 16.)

[MOZ22] Alice Murphy, Adam O'Neill, and Mohammad Zaheri. Instantiability of classical random-oracle-model encryption transforms. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part IV*, volume 13794 of *Lecture Notes in Computer Science*, pages 323–352, Taipei, Taiwan, December 5–9, 2022. Springer, Heidelberg, Germany. (Cited on page 2.)

[MV03] Daniele Micciancio and Salil P. Vadhan. Statistical zero-knowledge proofs with efficient provers: Lattice problems and more. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 282–298, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany. (Cited on page 29.)

[Ost91] Rafail Ostrovsky. One-way functions, hard on average problems, and statistical zero-knowledge proofs. In *Proceedings of the Sixth Annual Structure in Complexity Theory Conference, Chicago, Illinois, USA, June 30 - July 3, 1991*, pages 133–138. IEEE Computer Society, 1991. (Cited on page 29.)

[PRS12] Krzysztof Pietrzak, Alon Rosen, and Gil Segev. Lossy functions do not amplify well. In Ronald Cramer, editor, *TCC 2012: 9th Theory of Cryptography Conference*, volume 7194 of *Lecture Notes in Computer Science*, pages 458–475, Taormina, Sicily, Italy, March 19–21, 2012. Springer, Heidelberg, Germany. (Cited on pages 3, 4, and 5.)

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, June 2000. (Cited on page 16.)

[PW08] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 187–196, Victoria, BC, Canada, May 17–20, 2008. ACM Press. (Cited on page 4.)

[PW11] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. *SIAM J. Comput.*, 40(6):1803–1844, 2011. (Cited on page 4.)

[QWW21] Willy Quach, Brent Waters, and Daniel Wichs. Targeted lossy functions and applications. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 424–453, Virtual Event, August 16–20, 2021. Springer, Heidelberg, Germany. (Cited on pages 2 and 4.)

[RTV04] Omer Reingold, Luca Trevisan, and Salil P. Vadhan. Notions of reducibility between cryptographic primitives. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 1–20, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany. (Cited on page 9.)

[Sim98] Daniel R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In Kaisa Nyberg, editor, *Advances in Cryptology – EUROCRYPT'98*, volume 1403 of *Lecture Notes in Computer Science*, pages 334–345, Espoo, Finland, May 31 – June 4, 1998. Springer, Heidelberg, Germany. (Cited on pages 2 and 10.)

[SV03] Amit Sahai and Salil P. Vadhan. A complete problem for statistical zero knowledge. *J. ACM*, 50(2):196–249, 2003. (Cited on page 29.)

[Zha16] Mark Zhandry. The magic of ELFs. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 479–508, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany. (Cited on pages 1 and 2.)

[Zha19] Mark Zhandry. The magic of ELFs. *Journal of Cryptology*, 32(3):825–866, July 2019. (Cited on pages 1 and 2.)

A Lazy-Sampling a Random Permutation

Lazy sampling of random *functions* is quite easy. For each new input x pick a fresh random response $y \leftarrow\$ \{0,1\}^\lambda$, and store y in a table $T[x]$ for value x to answer subsequent requests about x consistently. For lazy-sampling a random *permutation* one often reads that, in this case, y needs to be chosen randomly from $\{0,1\}^\lambda \setminus \{y \mid T[x] \neq \bot\}$. While mathematically precise, this leaves open how this actual sampling should be carried out algorithmically. One option, if the number of previous queries q is sufficiently small, say, $q \leq 2^{\lambda/2}$, is to sample y repeatedly till a fresh value is found. This, however, yields only a lazy sampler with expected run time guarantees. Alternatively, one aborts after a sufficient number of attempts and needs to account for the statistical failure error and the run time influence due to the repeated trials, especially regarding the checks if y is actually fresh.

We present here a different method to lazy-sample a random permutation. The method is based on the Fisher-Yates algorithm, described here in the more efficient version popularized by Durstenfeld [Dur64] and Knuth [Knu98]. The idea of the Fisher-Yates algorithm is to start with an array $A[\,]$ of the integers $A[0] = 0, A[1] = 1, A[2] = 2, \ldots, A[N-1] = N-1$ and to iterate through all array cells, each time swapping the current value with one of the remaining cells: For $i = 0, 1, 2, \ldots, N-1$ pick $j \leftarrow\$ \{i, i+1, \ldots, N-1\}$ at random and exchange the contents of $A[i]$ and $A[j]$. Eventually one obtains the permutation by mapping each value i to $A[i]$. It is easy to see that each value has a chance of $\frac{1}{N}$ to end up at each position r, since it has a probability of $\prod_{i=0}^{r-1} \frac{N-1-i}{N-i} = \frac{N-r}{N}$ of being not picked in the first $r-1$ rounds, times $\frac{1}{N-r}$ of being chosen in the r-th round.

Of course, in our case a random permutation over $\{0,1\}^\lambda$ is too large to be generated at the outset, such that we perform the new assignment on the fly. For this it is convenient to also keep track of the swaps in a table S, but where we use a sparse book keeping of only assigning values $S[j] \neq \bot$ to cells which have been involved in a swap (and setting $S[j] \leftarrow j$ only at the point in time where we access a previously untouched cell).

$\mathsf{Init}()$	$\mathsf{Lazy\text{-}Sample}(x)$
$S[\,], T[\,] \leftarrow \bot$ // of max. size 2^λ	**if** $T[x] \neq \bot$ **then return** $T[x]$
$c \leftarrow 0$ // counter	$j \leftarrow\!\!\$\ \{c, c+1, \ldots, 2^\lambda - 1\}$
return (S, T, c)	// swap values at c and j:
	if $S[j] = \bot$ **then** $S[j] \leftarrow j$
	if $S[c] = \bot$ **then** $S[c] \leftarrow c$
	$temp \leftarrow S[c];\ S[c] \leftarrow S[j];\ S[j] \leftarrow temp$
	// $\mathsf{Int2Str}_\lambda : \{0,1,2,\ldots,2^\lambda - 1\} \to \{0,1\}^\lambda$ canonically
	$T[x] \leftarrow \mathsf{Int2Str}_\lambda(S[c])$
	$c \leftarrow c + 1$
	return $T[x]$

Note that we can store the tables S and T via common structures like search trees or skip lists with logarithmic run time. That is, after q queries the sampling procedures needs at most $\mathcal{O}(\log q)$ steps each to check if $T[x] \neq \bot$ among the at most q non-empty entries in T. This is also true for table S since each sampling sets at most two values in S, such that S contains at most $2q$ entries at this point. But then checking that $S[*] \neq \bot$ and swapping the values can be carried out in $\mathcal{O}(\log q)$ steps as well.

At first glance it seems as if the random sampling $j \leftarrow\!\!\$\ \{c, c+1, \ldots, 2^\lambda - 1\}$ suffers from similar problems as for sampling a fresh y from $\{0,1\}^\lambda \setminus \{y \mid T[x] \neq \bot\}$. But note that the interval $[c, 2^\lambda - 1]$ is consecutive, and it suffices to pick a random integer between 0 and $2^\lambda - 1 - c$ and add c to the sample. Sampling such a random integer can be done, for instance, by choosing a 2λ-bit string *once* and reducing the corresponding integer modulo $2^\lambda - c$. The outcome is statistically close to uniform on $[0, 2^\lambda - 1 - c]$.

B Lossy Functions Imply Hard Statistical Zero-Knowledge Problems

We will show that HVPZK, the class of all problems with an honest-verifier perfect zero-knowledge proof, contains hard problems if lossy functions exist. As HVPZK is contained in HVSZK (honest-verifier *statistical* zero-knowledge) and as HVSZK = SZK, this means lossy functions imply hard problems in SZK.

Proposition B.1 *Let* $(\mathsf{Gen}, \mathsf{Eval})$ *be an* ω*-lossy function for* $\omega \geq 1$*. Then the language*

$$L = \left\{ (1^\lambda, \mathsf{pk}) \;\middle|\; \mathsf{pk} \in [\mathsf{Gen}_{\mathsf{inj}}(1^\lambda)] \right\}$$

is in HVPZK \ BPP*. In particular, with the distribution* $\mathcal{D}(1^\lambda)$ *picking a random bit* $b \leftarrow\!\!\$\ \{0,1\}$ *and returning* $\mathsf{pk} \leftarrow\!\!\$\ \mathsf{Gen}_{\mathsf{inj}}(1^\lambda)$ *if* $b = 0$ *resp.* $\mathsf{pk} \leftarrow\!\!\$\ \mathsf{Gen}_{\mathsf{loss}}(1^\lambda)$ *if* $b = 1$*, we get a hard-on-average problem in* HVPZK*.*

Proof. Note that injective public keys $\mathsf{pk} \in [\mathsf{Gen}_{\mathsf{inj}}(1^\lambda)]$ cannot lie in the support of lossy keys $[\mathsf{Gen}_{\mathsf{loss}}(1^\lambda)]$, because the function cannot be injective and lossy at the same time. Hence, if the language L was in BPP, then one could the decision algorithm to decide with error at most 1/3 if a given public key pk is injective or lossy. This, however, contradicts the indistinguishability of keys of the lossy function. This translates accordingly to the defined distribution $\mathcal{D}$ for the hard-on-average problem.

Next, we present our honest-verifier perfect zero-knowledge protocol for L. The input to both parties, the prover and the verifier, is $(1^\lambda, \mathsf{pk})$. The verifier picks a random $x \leftarrow_\$ \{0,1\}^\lambda$ and computes $y \leftarrow \mathsf{Eval}(\mathsf{pk}, x)$ and sends y to the prover. The prover searches for the preimage x^* of y under pk and returns x^*. The verifier accepts if and only if $x = x^*$.

For $(1^\lambda, \mathsf{pk}) \in L$ the key in injective such that the prover finds the correct preimage $x^* = x$, making the verifier accept. For a lossy key pk, however, there are at least two potential preimages for y, each one equally like. Hence, a malicious prover can make the verifier accept with probability at most $1/2$. The simulator for the honest verifier works as follows: It samples $x \leftarrow_\$ \{0,1\}^\lambda$ and computes $y \leftarrow \mathsf{Eval}(\mathsf{pk}, x)$ as the verifier would. Then it pretends that the prover returns x. Note that this view is identical distributed to an actual protocol run between the prover and the verifier (for an injective key). $\square$

C Deferred Proofs

C.1 Proof for Lemma 3.7

Proof. Let us start with one-wayness. It is well-known that a random oracle is exponentially one-way in a distributional sense, i.e., over the choice of the random oracle (see e.g. [MF21] for a full proof):

$$\forall \mathcal{A}, \forall \lambda : \Pr_{\mathcal{O},\mathcal{A}}[\mathcal{A} \text{ wins}] \leq \mathsf{poly}(\lambda) 2^{-\lambda}$$

Let us fix one adversary $\mathcal{A}$. Using the Markov inequality, we get

$$\forall \lambda : \Pr_{\mathcal{O}}\Big[\Pr_{\mathcal{A}}[\mathcal{A} \text{ wins}] \geq \lambda^2 \mathsf{poly}(\lambda) 2^{-\lambda}\Big] \leq \frac{1}{\lambda^2}.$$

As the sum over all $\frac{1}{\lambda^2}$ converges, we can use the Borell-Cantelli lemma to show there only exists a zero-measure of random oracles such that the adversary is successful for infinitely many security parameters. Using the fact that there are only countable many adversaries, the set of random oracles for which some adversary is successful infinitely often is also a zero-measure. Therefore, every random oracle except for this zero-measure set of oracles is one-way.

Similarly to one-wayness, we know that a truncated random oracle is an exponentially-secure collision resistant hash function (as long as the output is still long enough; see [MF21] again for a full proof). Then, by a similar argument, we know that every random oracle except for a zero-measure can be used to construct a collision-resistant hash function.

Finally, let us show the result for one-way product functions. One-way product functions are a set of functions $f_1, \ldots, f_k$ such that any adversary trying to invert $f_1(x_1), \ldots, f_k(x_k)$ for independent, uniform x_i will have a success probability of at most $2^{-kn} \cdot \text{poly}(n)$. In their paper, Holmgren and Lombardi [HL18] do not give an implementation from a random oracle, but it is quite clear one-way product functions can be built relative to a random oracle (with overwhelming probability). For this, note that OWPF can be seen as the combination of two properties: First, every f_i is exponentially-hard to invert, and second, inverting multiple f_i's is as hard as inverting them independently. Assuming every f_i equals the random oracle, we've already shown the first property. The second property follows from the fact that for two different values x_1, x_2, $\mathcal{O}(x_1)$ is completely independent of $\mathcal{O}(x_2)$, so as long as all inputs are different, inverting all f_i at once does not yield any advantage over inverting them independently. Therefore, OWPF can be built from random oracles (with overwhelming probability). Using Borel-Cantelli again yields the proof. $\square$