Decoding LTFs in the Generic Group Model

Dennis Hofheinz, Julia Kastner, Akin Ünal, and Bogdan Ursu

Department of Computer Science
ETH Zurich
Zurich, Switzerland
{hofheinz, julia.kastner, akin.uenal, bogdan.ursu}@inf.ethz.ch

Abstract. Lossy trapdoor functions (LTFs) constitute a useful and versatile cryptographic building block. LTFs have found applications in various types of encryption schemes, are closely connected to statistically secure oblivious transfer protocols, and have led to the first constructions of group-based trapdoor functions. However, with one recent exception, all known group-based LTFs are comparatively inefficient, and in particular suffer from large images.
In this work, we attempt to explain this inefficiency, and derive lower bounds for the image size of group-based LTFs. In essence, we find that *purely algebraic* group-based LTFs (i.e., LTFs that use the underlying group in a generic way, without considering group representations) must suffer from a large image size (of an at least super-constant number of group elements). Our results also help to explain the mentioned exceptional group-based LTF with compact images.

Keywords: lossy trapdoor functions · generic group model

1 Introduction

Lossy trapdoor functions. A lossy trapdoor function is a keyed function f_{ek} in which the key ek was generated in one of two modes. In injective mode, an inversion trapdoor ik is sampled alongside ek, such that ik allows to efficiently invert f_{ek}. (In particular, in this mode, f_{ek} must be injective.) But when ek is generated using a lossy mode key generation, the corresponding f_{ek} is highly non-injective (and thus, no inversion trapdoor can exist). To allow for meaningful applications, injective and lossy ek should be computationally indistinguishable.

LTFs have been proposed and investigated by Peikert and Waters [22], who provided a number of LTF applications, and two instantiations from groups and lattices. Later on, a number of additional applications (including deterministic [7], lossy [21, 4], or hedged encryption [3]), and LTF constructions were found [16, 18, 5, 17, 1, 8, 19, 6, 2, 12].

The inefficiency of group-based LTFs. But while there exist a variety of group-based LTF constructions, the LTF from [12] is the only known LTF construction that features compact images (i.e., such that the size of $f_{\mathsf{ek}}(x)$ is not significantly

larger than the size of x). The intuitive reason for this is that most group-based constructions (e.g., from [22, 16]) process their preimage in a bitwise fashion. For instance, a prototypical LTF of this type over a group $\mathbb{G}$ is of the form

$$\mathsf{ek} = \mathbf{g}^M \in \mathbb{G}^{n \times n}$$
$$f_{\mathsf{ek}}(\vec{x}) = \mathbf{g}^{M \cdot \vec{x}} \in \mathbb{G}^{n \times n}$$

where $M \in \mathbb{Z}_p^{n \times n}$ is chosen as a uniformly random full-rank matrix for injective keys, $\vec{x} \in \{0,1\}^n$, and evaluation is performed using eponentiation with the components of $\vec{x}$. Inversion can be performed with the secret (exponent) matrix M^{-1} by computing $\mathbf{g}^{M^{-1} \cdot M \cdot \vec{x}} = \mathbf{g}^x$ through exponentiation, and then reading off the components of x.

In lossy mode, M can be chosen as a low-rank matrix (which results in computationally indistinguishable $\mathsf{ek} = \mathbf{g}^M$ under the DDH or even weaker assumptions [14]). In this case, images $\mathbf{g}^{M \cdot \vec{x}}$ reveal only limited information about $\vec{x}$. Observe that the image of this LTF consists of n group elements, while the input consists only of n bits. In that sense, this construction proceeds in a bitwise fashion and has large outputs.[1] It is possible to work with preimages $\vec{x}$ over a slightly larger domain (e.g., with $\vec{x} \in \mathbb{Z}_\ell$) at the expense of inversion times linear in ℓ. However, this still results in images of $n/\log_2(\ell)$ group elements.

The mentioned LTF of [12] has a much smaller image (close in size in fact to the preimage), and achieves this with a much more complex construction. We give details on their construction in Section 6, but one key ingredient of their construction is that they apply a pseudorandom function to intermediate results of the computation (which are group elements). Jumping ahead, we will see that this is the crucial step that enables compact images.

Our contributions. Indeed, in this work, we are interested in the reason why most group-based LTF constructions do not achieve compact outputs, and what it is that enables compact outputs. In a nutshell, we show that when relying only on hard problems in a group, and when not using the representation of group elements, then LTFs with compact images do not exist. More specifically, we work in Maurer's generic group model [20, 26] (which does not allow access to the representation of group elements), and we assume a LTF with a nontrivial lossiness of $\omega(\log \lambda)$ bits, i.e., such that its image size with lossy ek is superpolynomially smaller than its preimage size. (Since LTF lossiness cannot easily be amplified [23], such a minimal lossiness requirement seems reasonable.)

We offer the following technical results:

1. (Corollary 9.) If preimages are n-bitstrings and images are comprised of k group elements and a m-bitstring of auxiliary information, then we must have
$$k \geq \frac{n-m}{O(\log \lambda)},$$

[1] The constructions in [22, 16] are variations of this prototypical LTF that use different distributions of M. The core idea and the resulting inefficiency, however, is the same as in our example.

where λ denotes the security parameter.
2. (Theorem 26 and Corollary 27.) In the same situation, if k is a constant (that does not depend on λ or n), then we can construct a distinguisher $\mathcal{A}$ between injective and lossy keys. Our $\mathcal{A}$ will be computationally inefficient, but make only a polynomial number of group oracle queries.
3. (Theorem 29.) Similarly, if preimages are comprised only of group elements, we construct a distinguisher $\mathcal{A}$ between injective and lossy keys. In this case, however, $\mathcal{A}$ is even computationally efficient.

Interestingly, the techniques we use for these three results are very different (see below for details), and so are their consequences. Our first result is applicable to the prototypical LTF sketched above and the LTFs from [22, 16], and shows them essentially optimal. Our second result applies to the setting of [12], and shows that their use of the representation of group elements is necessary. (We give a more detailed analysis in Section 6.) Our third result, on the other hand, applies to no known LTF, but closes a potential avenue of group-based LTFs.

We note that we prove our results only for prime-order groups without pairings. However, in view of results like [15] (when ignoring the pairing operation), it is not clear how composite-order groups will be helpful.

A note on related work. We note that [13, 10] offer similar lower bounds for group-based and "sufficiently algebraic" signatures and verifiable random functions. Technically, their results exploit the inherent linearity (or, in the presence of pairings, quadratic nature) of algebraic algorithms in different ways than we do. For instance, [13] use that valid signatures in group-based algebraic signature schemes must satisfy publicly checkable linear equalities in group elements from signature and verification key. They then use these linear relations to construct a generic adversary that linearly combines several existing signatures into a forgery.

Using such a strategy directly will not work in our case, since LTF images may well contain group elements that do not match a known linear equation. In fact, such group elements can be potentially helpful to derive information about the corresponding preimage (e.g., by encoding which of a number of possible linear equations they satisfy). What we show (in our first result above) is that the amount of *efficiently extractable* information of such group elements in LTF images is limited.

1.1 Technical Overview

We now provide an overview over our main ideas and techniques.

First result: preimages are bitstrings, images are (vectors of) group elements. In our modeling of LTFs, preimages and images and can be bitstrings, group elements, or a mix of both. For this exposition, however, let us first consider the case that preimages $\vec{x}$ are (only) bitstrings (of length, say, n), whereas images $f_{\mathsf{ek}}(\vec{x}) = \mathbf{g}^{\vec{y}}$ are (only) group elements. This is in line with our prototypical LTF

above and the LTFs from [22, 16], while the LTF from [12] uses a combination of group elements and bits in the image.

As stated above, we work in Maurer's generic group model [20, 26]. This means that the only way that inversion can obtain information about $\vec{x}$ from a given image $\mathbf{g}^{\vec{y}}$ is through explicit comparisons. In other words, we can think of the inversion algorithm as making a (polynomial) number of affine tests $(\vec{t_i}, s_i)$ (adaptively) and learning whether or not $\langle \vec{y}, \vec{t_i} \rangle = s_i$ holds.

We model the task to invert a given image through such comparisons with a decision tree. At each node N, there is a set X_N of at this point still possible preimages $\vec{x}$. Hence, at the root node root of the tree (which corresponds to the start of the computation), X_{root} contains all 2^n possible preimages. Ideally, each comparison would halve the size of X_N. However, we show that, due to the affine nature of comparisons, each comparison splits the set X_N up in a very imbalanced way. Specifically, most (nontrivial) affine equations checked by such comparisons hold only for few input values. Hence, there must be a very long path along that decision tree (which corresponds to a large worst-case computation time of inversion).

Our first result (in Section 3) formalizes this intuition, and generalizes it to the case when the output also contains a small number of additional bits of auxiliary "helper" information. Essentially, if the auxiliary information is short enough, it cannot reveal too much information about the preimage $\vec{x}$.

Second result: images with large auxiliary information. If the auxiliary (non-group-)information in images is large, the above decision tree-based argument fails. In particular, the LTF from [12] falls into this category. In this case, we will still be able to provide a meaningful lower bound on the number of group elements in the LTF image.

Concretely, we construct a distinguisher $\mathcal{A}$ between injective and lossy ek. Our $\mathcal{A}$ will be inefficient, but use only a polynomial number of (generic) group operations. Intuitively, $\mathcal{A}$ proceeds in two phases, where group oracle queries are only made in the first phase:

1. $\mathcal{A}$ uses group comparison queries to gather enough information about the discrete logarithms of group elements in ek, so that most images can be successfully evaluated even without explicit group oracle queries.
2. Once $\mathcal{A}$ has prepared such an "approximate but oracle-less evaluation key", it evaluates *all* preimages and approximates the image size of f_{ek}. If this image size is small enough, the corresponding ek must be lossy.

We now explain those two steps in a little more detail. We will assume that the evaluation key ek is of the form $\mathsf{ek} = \mathbf{g}^{\vec{z}}$ for a vector $\vec{z}$ of exponents.

First phase: prepare an "approximate but oracle-less" evaluation key. In the first phase, $\mathcal{A}$ starts out with an a-priori distribution on $\vec{z}$ as induced by fresh sampled injective ek. Now $\mathcal{A}$ can consider all affine tests $(\vec{t}, s)$ and the respective probabilities $p_{\vec{t},s}$ for $\langle \vec{z}, \vec{t_i} \rangle = s_i$ (over $\vec{z}$). There are two cases: there is (at least)

one such linear test $(\vec{t}, s)$ with $1 - \frac{1}{2q} < p_{\vec{t},s} < 1 - \frac{1}{2q}$ (for a suitable polynomial q), or all $p_{\vec{t},s}$ are $\frac{1}{2q}$-close to either 0 or 1.

Intuitively, in the latter case, $\mathcal{A}$ has already enough information to approximate the outcome of all affine tests (and thus comparison queries) that could arise during any evaluation $f_{\mathsf{ek}}(\vec{x})$. In this case, we are already done. But in the former case, $\mathcal{A}$ will make an explicit oracle query for one such $(\vec{t}, s)$ with $\frac{1}{2q} < p_{\vec{t},s} < 1 - \frac{1}{2q}$ and condition the distribution of $\vec{z}$ on the result. Intuitively, this way the entropy of $\vec{z}$ will decrease by a nontrivial amount. After a polynomial number of such queries, we will end up in the first case (with all $p_{\vec{t},s}$ close to 0 or 1). (At the latest, this happens once $\vec{z}$ is fully determined.)

Second phase: approximating f_{ek}'s image size. Next, $\mathcal{A}$ uses its knowledge to evaluate all f_{ek} on all inputs $(\vec{x})$ and count the number of different images. Any arising comparison queries made by f_{ek}'s evaluation algorithm in the process correspond to affine tests $(\vec{t}, s)$ and can be answered with the oracle-less evaluation key from the first phase (answering with "match" if and only $p_{\vec{t},s} > 1 - \frac{1}{2q}$). Of course, this results with a few false answers, but we will choose q (polynomially) large enough such that most evaluations can be simulated faithfully.

If this approximated image size is close to the number of all $\vec{x}$, $\mathcal{A}$ will guess that ek is injective, otherwise that ek is lossy. Indeed, by our argument above, in case of an actually injective ek, we will end up with a large overall image size. But even when starting with a lossy ek, its corresponding initial distribution must be stastically close to that of an injective ek, at least when looking at the results of all possible affine tests $(\vec{t}, s)$. (If not, we have found an easy way to distinguish injective and lossy keys using just such a test.) Hence, the resulting approximative evaluation key derived at the end of the first phase will be sufficiently accurate, and in the second phase $\mathcal{A}$ will end up with a small image size.

The full details (which include a lot of fine-tuning of parameters) can be found in Section 4. We do mention, however, that we unfortunately can only show a super-constant lower bound for the number of group elements in the image.

Third result: group elements as input. Finally, we also consider the case in which the preimage is comprised of group elements. This case is somewhat less technical and more straightforward, since in that case, evaluation must essentially be a linear transformation (mapping group elements in evaluation key ek and input $\vec{x}$ to the group elements in $f_{\mathsf{ek}}(\vec{x})$). This transformation can be efficiently inverted using linear algebra.

2 Preliminaries

2.1 Notation

We denote our security parameter by λ. For a distribution $\mathcal{D}$ over a finite set X we denote by $x \xleftarrow{\$} \mathcal{D}$ that x is sampled from X according to $\mathcal{D}$, and we denote

by $x \xleftarrow{\$} X$ that x is sampled uniformly at random from X. We denote group elements $\mathbf{x}$ in boldface, vectors $\vec{v}$ with overhead arrows, and we write the group operation as multiplication.

2.2 Groups

Definition 1 (Type Safe Generic Group Model [26, 20]). *A generic algorithm in the type safe generic group model has two types of inputs, group element inputs and bit inputs. It can use the following operations on these inputs:*

Bit operations *from any universal set.*

Exponentiations *take as input* $\log p$ *many bit variables and interpret them as an integer* x *and output a group element variable that corresponds to* $\mathbf{g}^x$ *for some fixed generator* $\mathbf{g}$*.*

Group Multiplications *take as input two group element variables encoding* $\mathbf{g}_1, \mathbf{g}_2$ *and output a group element variable encoding* $\mathbf{g}_1 \cdot \mathbf{g}_2$*. Outputs* $\perp$ *if either of the inputs was* $\perp$*.*

Equality checks *Take as input two group element variables and output a bit* 1 *if they are equal, and* 0 *if they are not or if at least one of them is* $\perp$*.*

2.3 Lossy Trapdoor Functions

Definition 2 (Lossy Trapdoor Function(LTF) [22]). *Let* n *be a polynomial,* $k \leq n$*. A* lossy trapdoor function (LTF) *with* domain $\{0,1\}^n$ *consists of three algorithms* $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ *with the following properties:*

Easy to sample injective keys with trapdoors: *In injective mode,* $\mathsf{LGen}(1^\lambda, 1^n, 1)$ *outputs a pair* $(\mathsf{ek}, \mathsf{ik})$ *of an* injective *evaluation key and an inversion key. We assume in this work that injective keys are fully correct, i.e., for each* $x \in \{0,1\}^n$ *in the domain we have*

$$\mathsf{LInv}(\mathsf{ik}, \mathsf{LEv}(\mathsf{ek}, x)) = x.$$

Easy to sample lossy keys: *In* lossy *mode,* $\mathsf{LGen}(1^\lambda, 1^n, 0)$ *outputs* $(\mathsf{ek}, \perp)$ *such that* $\mathsf{LEv}(\mathsf{ek}, \_)$ *is* l-lossy*, i.e.*

$$|\{\mathsf{LEv}(\mathsf{ek}, x) \mid x \in \{0,1\}^n\}| \leq 2^{n-l(\lambda)}.$$

Deterministic evaluation: *For each evaluation key* ek *outputted by* LGen*, the evaluation function* $\mathsf{LEv}(\mathsf{ek}, \_)$ *is computed by a deterministic algorithm.*

Hard to distinguish lossy and injective: *Each PPT algorithm has negligible advantage at distinguishing injective and lossy evaluation keys. Formally, this means that the following two distributions are computationally indistinguishable*

$$\{\mathsf{ek} \mid (\mathsf{ek}, \mathsf{ik}) \xleftarrow{\$} \mathsf{LGen}(1^\lambda, 1^n, 1)\} \overset{c}{\approx} \{\mathsf{ek} \mid (\mathsf{ek}, \perp) \xleftarrow{\$} \mathsf{LGen}(1^\lambda, 1^n, 0)\}$$

2.4 Encoding-Decoding Schemes

Definition 3 (Encoding-Decoding Schemes). *Let $S, Y \subset \{0,1\}^*$ and $k \in \mathbb{N}$. An encoding-decoding scheme is a pair $(\mathsf{Enc}, \mathsf{Dec})$ of deterministic algorithms s.t. Enc maps S to $\mathbb{G}^k \times Y$ and Dec maps $\mathbb{G}^k \times Y$ to S s.t. we have for all $s \in S$:*

$$\mathsf{Dec}(\mathsf{Enc}(s)) = s. \tag{1}$$

We will usually denote an encoding-decoding scheme as

$$S \xrightarrow{\mathsf{Enc}} Y \times \mathbb{G}^k \xrightarrow{\mathsf{Dec}} S.$$

We will call $(\mathsf{Enc}, \mathsf{Dec})$ generic *if Enc and Dec only use generic group operations as described in Definition 3.*

2.5 Mathematical Preliminaries

Lemma 4 (Schwartz-Zippel Lemma [27, 24, 11]). *Let $P \in \mathbb{F}[X_1, \ldots, X_n]$ be a non-zero polynomial of total degree $d \geq 0$ over a field $\mathbb{F}$ and S a finite subset of $\mathbb{F}$. Then*

$$\Pr_{r_1,\ldots,r_n \xleftarrow{\$} S}[P(r_1, \ldots, r_n) = 0] \leq \frac{d}{|S|}.$$

We will often use $\mathbb{F} = \mathbb{Z}_p$ and $S = \mathbb{Z}_p$.

Lemma 5 (Markov's Inequality). *Let X be a random variable over $\mathbb{R}_{\geq 0}$. Denote by $\mathbb{E}[X]$ its expectation. We have for each $\alpha > 1$*

$$\Pr[X > \alpha \cdot \mathbb{E}[X]] \leq \frac{1}{\alpha}.$$

3 Lower Bounds for Generic Encoding-Decoding Schemes

In this section, we give our first result: We prove that an LTF $\mathsf{LEv} : \{0,1\}^n \to \{0,1\}^m \times \mathbb{G}^k$ that outputs fewer output bits than input bits (let's say $m \leq n/2$) must have approximately as many output group elements as input bits, formally $k \in \Omega(n/\log \lambda)$, otherwise the inversion algorithm LInv is not efficient.

Our proof idea essentially boils down to studying how many bits of information LEv can store in a group element it outputs in Maurer's GGM when ensuring that stored bits can be extracted efficiently. To simplify our proof, we construct an encoding-decoding scheme (cf. Definition 3) from the LTF along with its evaluation and inversion key. This basically means fixing an injective key pair ek and ik and letting $\mathsf{Enc}(\cdot) \coloneqq \mathsf{LEv}(\mathsf{ek}, \cdot)$ and $\mathsf{Dec}(\cdot) \coloneqq \mathsf{LInv}(\mathsf{ik}, \cdot)$. (We clarify the connection to lossy trapdoor functions in Theorem 6.) The advantage of this is that we can now argue about a fixed algorithm Dec whose possible inputs are the outputs of Enc. We do not care about lossiness as a property here, but instead are interested solely in the efficiency of the inversion algorithm.

Theorem 6. *1. Each generic LTF* $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ *with*

$$\mathsf{LEv} : \{0,1\}^n \longrightarrow \{0,1\}^m \times \mathbb{G}^k$$

gives rise to a generic encoding-decoding scheme

$$\{0,1\}^n \xrightarrow{\mathsf{Enc}} \{0,1\}^m \times \mathbb{G}^k \xrightarrow{\mathsf{Dec}} \{0,1\}^n$$

where the space, time and group operation complexities of Enc *are upper bounded by the corresponding complexities of* LEv *and the space, time and group operation complexities of* Dec *are upper bounded by the corresponding complexities of* LInv*.*

2. Each efficient generic encoding-decoding scheme

$$\{0,1\}^n \xrightarrow{\mathsf{Enc}} \{0,1\}^m \times \mathbb{G}^k \xrightarrow{\mathsf{Dec}} \{0,1\}^n$$

can be turned (in a non-black box way) into an efficient generic LTF $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ *with*

$$\mathsf{LEv} : \{0,1\}^n \longrightarrow \{0,1\}^m \times \mathbb{G}^k.$$

Proof. 1. Let $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ be a generic LTF $\mathsf{LEv} : \{0,1\}^n \to \{0,1\}^m \times \mathbb{G}^k$. Let $(\mathsf{ek}, \mathsf{ik}) \leftarrow \mathsf{LGen}(1^\lambda, 1^n, 1)$ be any pair injective evaluation key and corresponding inversion key. Define $\mathsf{Enc}, \mathsf{Dec}$ by

$$\mathsf{Enc}(x) := \mathsf{LEv}(\mathsf{ek}, x),$$
$$\mathsf{Dec}(y, \mathbf{g}_1, \ldots, \mathbf{g}_k) := \mathsf{LInv}(\mathsf{ik}, y, \mathbf{g}_1, \ldots, \mathbf{g}_k)$$

for $x \in \{0,1\}^n, y \in \{0,1\}^m, \mathbf{g}_1, \ldots, \mathbf{g}_k \in \mathbb{G}$. The correctness of $(\mathsf{Enc}, \mathsf{Dec})$ follows from the correctness of the injective key ek. The number of operations Enc and Dec need to perform are bounded by the number of operations LEv and LInv need to perform in the worst case.

2. Let $\{0,1\}^n \xrightarrow{\mathsf{Enc}} \{0,1\}^m \times \mathbb{G}^k \xrightarrow{\mathsf{Dec}} \{0,1\}^n$ be a generic encoding-decoding scheme. We define $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ as follows:

   LGen: On input $1^\lambda, 1^n$ and $b \in \{0,1\}$, LGen draws a uniformly random matrix $M_1 \in \mathbb{Z}_p^{k \times k}$ if $b = 1$, and a uniformly random rank-one matrix $M_0 \in \mathbb{Z}_p^{k \times k}$ if $b = 0$. It computes the matrix of group elements of entries of M_b and sets $\mathsf{ek} := \mathbf{g}^{M_b}$. If $b = 1$ it sets $\mathsf{ik} := M_b^{-1}$, otherwise $\mathsf{ik} := \perp$. Finally, it outputs $(\mathsf{ek}, \mathsf{ik})$.

   LEv: On input $\mathsf{ek} = \mathbf{g}^M$ and $x \in \{0,1\}^n$, LEv computes $(y, \mathbf{g}_1, \ldots, \mathbf{g}_k) := \mathsf{Enc}(x)$. Since Enc is purely generic and got no group elements as input, LEv can efficiently extract the exponents $a_1 := \operatorname{dlog} \mathbf{g}_1, \ldots, a_k := \operatorname{dlog} \mathbf{g}_k$ out of the circuit description of Enc. LEv sets $\vec{a} := (a_1, \ldots, a_k)$ and computes $\mathbf{g}^{M \cdot \vec{a}}$. Finally, LEv outputs $(y, \mathbf{g}^{M \cdot \vec{a}})$.

   LInv: On input $(y, \mathbf{g}^{\vec{z}}) \in \{0,1\}^m \times \mathbb{G}^k$ and $\mathsf{ik} = N \in \mathbb{Z}_p^{k \times k}$, LInv computes and outputs $\mathsf{Dec}(y, \mathbf{g}^{N \cdot \vec{z}})$.

The correctness of LTF follows from the correctness of $(\mathsf{Enc}, \mathsf{Dec})$, while the indistinguishability of lossy and injective keys can – by a standard argument that we omit here – be based on the hardness of DDH.
It is clear that the time complexity of LGen is polynomial in λ, n, k and m. The time complexity of LInv equals the time complexity of Dec plus an overhead that is polynomial in k and λ. Now, the time complexity of LEv is dominated by extracting the discrete logarithms of group elements output by $\mathsf{Enc}(x)$. Since Enc is purely generic and efficient, we can assume that analysing its circuit and deducing the exponents of its outputs can be done in an efficient manner. It follows that the runtime of LEv is polynomial in n, k and m.

□

Note that the compactness of group-based encoding-decoding schemes is a static property: it does not depend on distribution of evaluation keys or lossiness. This makes it easier to study and prove lower bounds for it.

Theorem 6 implies that, in order to prove our claim for an LTF, it suffices to look at the encoding-decoding scheme that is derived from it. In fact, we will prove in this section:

Theorem 7. *Let $S \xrightarrow{\mathsf{Enc}} Y \times \mathbb{G}^k \xrightarrow{\mathsf{Dec}} S$ be a generic encoding-decoding scheme for sets $S, Y \subset \{0,1\}^*$. Then, Dec needs to make at least*

$$(|S|/|Y|)^{1/k} - 1$$

equality checks of group elements and

$$\sqrt{2 \cdot (|S|/|Y|)^{1/k} - \frac{7}{4}} - k + \frac{1}{2}$$

group operations in the worst case.

From this theorem, we can directly deduce the following corollary for generic LTFs:

Corollary 8. *For an LTF $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ with*

$$\mathsf{LEv}(\mathsf{ek}, \_) : \{0,1\}^n \longrightarrow \{0,1\}^m \times \mathbb{G}^k,$$

it holds that LInv needs to make at least

$$2^{\frac{n-m}{k}} - 1 \tag{2}$$

equality checks of group elements and

$$\sqrt{2^{\frac{n-m}{k}+1} - \frac{7}{4}} - k + \frac{1}{2}$$

group operations.

Proof. According to Theorem 6, LTF implies an encoding-decoding scheme $\{0,1\}^n \xrightarrow{\mathsf{Enc}} \{0,1\}^m \times \mathbb{G}^k \xrightarrow{\mathsf{Dec}} \{0,1\}^n$ where the number of equality checks Dec makes lower bounds the number equality checks LInv needs to make in the worst case. Denote by t_{Dec} and t_{LInv} the corresponding number of equality checks. From Theorem 7 it follows

$$t_{\mathsf{LInv}} \geq t_{\mathsf{Dec}} \geq \left(\frac{|\{0,1\}^n|}{|\{0,1\}^m|}\right)^{1/k} - 1 = \left(2^{n-m}\right)^{1/k} - 1 = 2^{\frac{n-m}{k}} - 1.$$

Analogously, we can lower bound the number of group operations LInv needs to make in the worst case. □

Now we can deduce that the number of output group elements that $\mathsf{LEv}(\mathsf{ek}, \_) : \{0,1\}^n \to \{0,1\}^m \times \mathbb{G}^k$ outputs is larger than $\Omega\left(\frac{n-m}{\log \lambda}\right)$. It follows that if LEv has noticeably fewer output bits than input bits, then the number of group elements it outputs must be almost linear.

Corollary 9. *For a generic LTF* $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ *where* LInv *runs in polynomial time, and* LEv *maps* n *bits to* k *group elements and* m *bits, we have*

$$\frac{n-m}{k} \in O(\log \lambda).$$

If $m \leq \alpha n$ *for some constant* $\alpha < 1$*, it follows*

$$k \geq \frac{n-m}{O(\log \lambda)} \in \Omega\left(\frac{n}{\log \lambda}\right).$$

Proof. Set the running time of LInv to be $r \in \mathsf{poly}(\lambda)$. Eq. (2) implies that we have $2^{\frac{n-m}{k}} - 1 \leq r$. Now, the claim follows. □

On a technical note, we need to deduce the following lemma to prove Theorems 20 and 26.

Lemma 10. *Let* $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ *be an LTF where* LInv *runs in time* r*. For an evaluation key* ek*, let* $\mathsf{LEv}(\mathsf{ek}, \_) : \{0,1\}^n \to \{0,1\}^m \times \mathbb{G}^k$*, and denote by* $\mathsf{LEv}_1(\mathsf{ek}, \_) : \{0,1\}^n \to \{0,1\}^m$ *the function that only computes the binary output of* $\mathsf{LEv}(\mathsf{ek}, \_)$*. Set*

$$Y_{\mathsf{ek}} := \{\mathsf{LEv}_1(\mathsf{ek}, x) \mid x \in \{0,1\}^n\}.$$

If ek *is injective, we have*

$$|Y_{\mathsf{ek}}| \geq \frac{2^n}{(r+1)^k}.$$

If $r \in \mathsf{poly}(\lambda)$ *and* $k \in O(1)$*, it follows for injective* ek

$$\log |Y_{\mathsf{ek}}| \geq n - O(\log \lambda).$$

We will spend the rest of this section on proving Theorem 7. However, instead of proving Theorem 7 directly, we will instead prove the following simpler lemma:

Lemma 11. *Let $S \xrightarrow{\mathsf{Enc}} Y \times \mathbb{G}^k \xrightarrow{\mathsf{Dec}} S$ be a generic encoding-decoding scheme for sets $S, Y \subset \{0,1\}^*$ s.t. Y is a singleton i.e. $|Y| = 1$. Then,* Dec *needs to make at least*

$$|S|^{1/k} - 1$$

equality checks of group elements and

$$\sqrt{2 \cdot |S|^{1/k} - \frac{7}{4}} - k + \frac{1}{2}$$

group operations in the worst case.

In fact, we can simply deduce Theorem 7 from Lemma 11:

Proof (Theorem 7). Let $S \xrightarrow{\mathsf{Enc}} Y \times \mathbb{G}^k \xrightarrow{\mathsf{Dec}} S$ be a generic encoding-decoding scheme s.t. Y has more than one element. Then, by a pigeon-hole argument, there must exist one $y \in Y$ s.t. the set

$$\widetilde{S} := \left\{ s \in S \subset \{0,1\}^n \mid \mathsf{Enc}(s) \in \{y\} \times \mathbb{G}^k \right\}$$

has at least $|S| \,/\, |Y|$ elements. We can now restrict Enc to $\widetilde{S}$ and Dec to $\{y\} \times \mathbb{G}^k$. This yields a new encoding-decoding scheme

$$\widetilde{S} \xrightarrow{\mathsf{Enc}'} \{y\} \times \mathbb{G}^k \xrightarrow{\mathsf{Dec}'} \widetilde{S}$$

where Enc' and Dec' are the corresponding restrictions of Enc and Dec. From Lemma 11 it now follows that Dec' needs to make at least

$$\left|\widetilde{S}\right|^{1/k} - 1 \geq (|S|/|Y|)^{1/k} - 1$$

equality checks of group elements and

$$\sqrt{2 \cdot \widetilde{S}^{1/k} - \frac{7}{4}} - k + \frac{1}{2} \geq \sqrt{2 \cdot (|S|/|Y|)^{1/k} - \frac{7}{4}} - k + \frac{1}{2}$$

group operations in the worst case. Since Dec' is the restriction of Dec to some inputs, it follows that the same lower bounds also hold for Dec in the worst case. □

Hence, it suffices to prove Lemma 11. Since Y has only one element in Lemma 11, we will tacitly omit it and consider the encoding decoding scheme given by

$$S \xrightarrow{\mathsf{Enc}} \mathbb{G}^k \xrightarrow{\mathsf{Dec}} S.$$

We need to lower bound the number of group element equality checks that Dec may make when executed on some input $\mathbf{g}_1, \ldots, \mathbf{g}_k$. To properly count the

minimum number of necessary equality checks we will model the algorithm Dec as a Turing machine that has oracle access to the group oracle. However, since Dec does not output group elements, it suffices to model the group equality checks made by Dec as *(affine) linear equations* and and ignore other group operations that do not lead to a group equality check. A linear equality check consists of a vector $\vec{c} \in \mathbb{Z}_p^{k+1}$. When the group oracle is queried with such a vector $\vec{c} = (c_1, \ldots, c_{k+1})$ it will respond positively if $\mathbf{g}_1^{c_1} \cdots \mathbf{g}_k^{c_k} \cdot \mathbf{g}^{c_{k+1}} = \mathbf{1}_{\mathbb{G}}$ where $\mathbf{g}$ denotes a fixed generator of $\mathbb{G}$, and $\mathbf{1}_{\mathbb{G}}$ denotes $\mathbb{G}$'s neutral element.

The Turing machine Dec is total and deterministic. It has an input tape on which it receives the number k of the group elements $\mathbf{g}_1, \ldots, \mathbf{g}_k \in \mathbb{G}$. Additionally, it has a working tape, an output tape and an oracle tape where it can write input data for group oracle queries. Besides starting and halting states, the Turing machine has three special states: a state $q_?$ it enters whenever it asks the group oracle to check perform a linear equality check on $\mathbf{g}_1, \ldots, \mathbf{g}_k$ and two corresponding response states q_{Yes} and q_{No}. Whenever Dec wants to perform a linear equality check, it writes a vector $\vec{\gamma} \in \mathbb{Z}_p^{k+1}$ on the oracle tape and then enter $q_?$. The specification of Dec does not contain a transition rule for $q_?$. Instead, whenever Dec enters $q_?$, the group oracle decides the next state of Dec: if the content of the oracle tape specifies a valid vector $\vec{\gamma} \in \mathbb{Z}_p^{k+1}$, and if the equality $\mathbf{g}_1^{\gamma_1} \cdots \mathbf{g}_k^{\gamma_k} \cdot \mathbf{g}^{\gamma_{k+1}} = \mathbf{1}_{\mathbb{G}}$ holds the group oracle moves Dec into the state q_{Yes}. Otherwise, the oracle moves Dec into q_{No}.

Denote by G_{conf} the configuration graph of Dec, that is the graph of all configurations of the Turing machine Dec, which are connected by transitions of Dec. This graph contains all computation paths of Dec that may overlap. Denote by

$$X_{\text{start}} := \left\{ \vec{x} \in \mathbb{Z}_p^k \mid \exists s \in S : \mathsf{Enc}(s) = \mathbf{g}^{\vec{x}} \right\}$$

the set of exponents of inputs of Dec. We recursively construct a binary decision tree $\mathcal{T}_k$ out of G_{conf} s.t. each inner vertex corresponds to a linear equality check performed by Dec. The root of $\mathcal{T}_k$ is the pair $(c_{\text{start}}, X_{\text{start}})$ where c_{start} is the starting configuration c_{start} of Dec on input $\mathbf{g}_1, \ldots, \mathbf{g}_k$. Note that configurations in G_{conf} contain no information about the exponents of $\mathbf{g}_1, \ldots, \mathbf{g}_k$. Hence, for each input $\mathbf{g}^{\vec{x}}, x \in \mathbb{Z}_p^k$,

Dec has the same starting configuration c_{start}. Since Dec is total and deterministic, there must be exactly one configuration $c \in G_{\text{conf}}$ that can be reached from c_{start} and that has no outgoing edges for any input of k group elements. We can distinguish two cases: if the state of c is not $q_?$, then c is a halting configuration and and we have already constructed the entire decision tree $\mathcal{T}_k$.

Otherwise, Dec performs an oracle query at c and there are two configurations c_{Yes} and c_{No} which Dec may enter depending on the response of the group oracle and the exponents of $\mathbf{g}_1, \ldots, \mathbf{g}_k$. Let $\vec{\gamma} \in \mathbb{Z}_p^{k+1}$ be the coefficient vector of the linear check that Dec performs at c. We set

$$X_1 = \left\{ \vec{x} \in X_{\text{start}} \mid \gamma_{k+1} + \sum_{i=1}^{k} \gamma_i \cdot x_i = 0 \mod p \right\}$$

and

$$X_0 = \left\{ \vec{x} \in X_{\text{start}} \mid \gamma_{k+1} + \sum_{i=1}^{k} \gamma_i \cdot x_i \neq 0 \mod p \right\}.$$

Now, X_1 contains all exponents of inputs $\mathbf{g}^{x_1}, \ldots, \mathbf{g}^{x_k}$ s.t. the group oracle will respond positively at c and move Dec into c_{Yes}, while X_0 contains all exponents of inputs s.t. the group oracle will respond negatively and change Dec's configuration into c_{No}. If $X_1 \neq \emptyset$, we add the node (c_{Yes}, X_1) to the graph $\mathcal{T}_k$ together with an edge $(c_{\text{start}}, X_{\text{start}}) \to (c_{\text{Yes}}, X_1)$. Analogously, we add the node (c_{No}, X_0) together with the edge $(c_{\text{start}}, X_{\text{start}}) \to (c_{\text{No}}, X_0)$ to $\mathcal{T}_k$ if $X_0 \neq \emptyset$. For each new node (c, X_{in}) added this way to $\mathcal{T}_k$ we repeat the previous procedure recursively: Let c' be the configuration that is reached from c and has no outgoing edges in G_{conf}. If the state of c' is not $q_?$, then c' is a halting configuration and (c, X_{in}) stay a leaf in $\mathcal{T}_k$. Otherwise, there are two configurations $c_{\text{Yes}}, c_{\text{No}}$ that may be successors of c' depending on the answer of the group oracle. We depict an example of a decision tree in Fig. 1a and a corresponding configuration graph in Fig. 1b.

Let $\vec{\gamma}$ be the linear check performed at c' and set again

$$X_1 = \left\{ \vec{x} \in X_{\mathsf{in}} \mid \gamma_{k+1} + \sum_{i=1}^{k} \gamma_i \cdot x_i = 0 \mod p \right\}$$

and

$$X_0 = \left\{ \vec{x} \in X_{\mathsf{in}} \mid \gamma_{k+1} + \sum_{i=1}^{k} \gamma_i \cdot x_i \neq 0 \mod p \right\}.$$

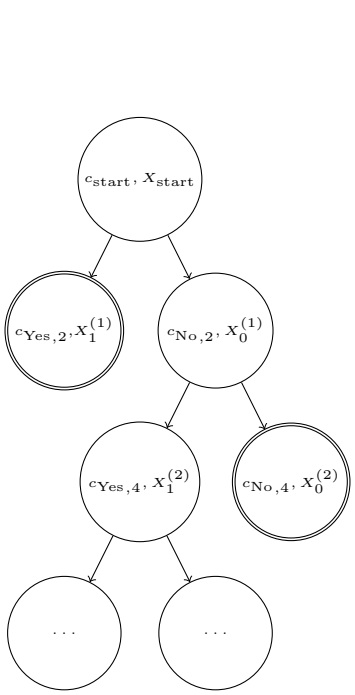
If $X_1 \neq \emptyset$, then we add the node (c_{Yes}, X_1) and the edge $(c, X_{\mathsf{in}}) \to (c_{\text{Yes}}, X_1)$ to $\mathcal{T}_k$. If $X_0 \neq \emptyset$, we add (c_{No}, X_0) and the edge $(c, X_{\mathsf{in}}) \to (c_{\text{No}}, X_0)$ to $\mathcal{T}_k$.

The graph $\mathcal{T}_k$ that we get by this procedure has the following properties:

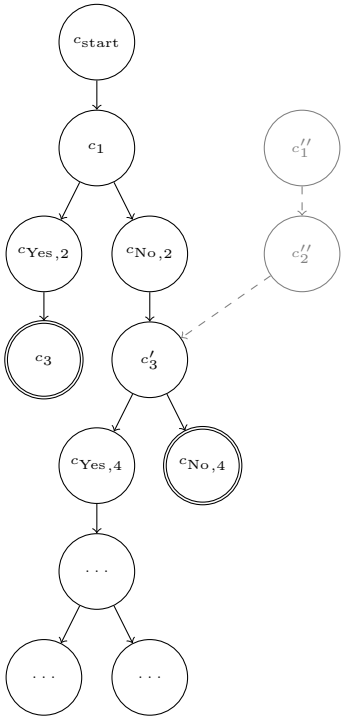
1. Since Dec is deterministic and total, $\mathcal{T}_k$ must be finite and acyclic.
2. If there is a node (c, X_{in}) in $\mathcal{T}_k$, then the configuration c must appear in the run of Dec on input $\mathbf{g}^{\vec{x}}$ for each $x \in X_{\mathsf{in}}$.
3. $\mathcal{T}_k$ is a tree. This is because at each inner vertex the set X_{in} splits into two disjoint sets or one set. Every non-root node has indegree 1.
4. If (c, X_{in}) is a leaf of $\mathcal{T}_k$, then Dec must reach a halting configuration from c without making any linear equality checks.
5. Because of correctness, for each leaf (c, X_{in}) the set X_{in} must be a singleton. Otherwise, there are two different inputs for which Dec will reach the same halting configuration and output the same bitstring.
6. Each path from the root of $\mathcal{T}_k$ to one its leaf corresponds to a run of Dec on one input $\mathbf{g}^{\vec{x}}, x \in X_{\text{start}}$.

To lower bound the number of equality checks that Dec makes in the worst case it suffices now to bound the longest path in $\mathcal{T}_k$.

We prove the statement now by an induction on $k \in \mathbb{N}$. We first show the case for $k = 1$. As described above, we denote by $X_{\text{start}} \subset \mathbb{Z}_p^k$ the set of exponents of $\mathsf{Enc}(S)$ (and by abuse of notation also the inputs to Dec).



(a) A decision tree $\mathcal{T}_k$.-



(b) A configuration graph. This graph also contains configurations that do not occur during the run of the algorithm, here marked in gray.

Fig. 1: The decision tree we build along with the configuration graph.

Proposition 12. *For an encoding-decoding scheme as above with $k = 1$, there exists one $x \in X$ such that* Dec *run on input x makes at least $|S| - 1$ group equality checks.* [2]

Proof. It suffices to prove that the depth of $\mathcal{T}_1$ is at least $|S|$. Because of correctness, we have $|S| = |X_{\text{start}}|$. At each inner vertex (c, X_{in}) of $\mathcal{T}_1$, Dec performs a linear equality check $\vec{\gamma} \in \mathbb{Z}_p^2$ on the input $\mathbf{g}^x$, $x \in X_{\text{start}}$. An input $\mathbf{g}^x$ passes the check if $\gamma_1 \cdot x + \gamma_2 = 0 \mod p$, otherwise it fails the check. Hence, if (c, X_{in}) has two different children (c_{Yes}, X_1) and (c_{No}, X_0), then X_1 must be the singleton $\{-\gamma_2/\gamma_1\}$ and we have $|X_0| = |X_{\mathsf{in}}| - 1$. It follows that each inner vertex (c, X_{in}) in the decision tree $\mathcal{T}_k$ has a child (c_{No}, X_0) with $X_0 \geq |X_{\mathsf{in}}| - 1$. Since each leaf (c, X_{in}) must have a singleton set X_{in}, we can find a path in $\mathcal{T}_1$ of length $|X_{\text{start}}| = |S|$. □

We now turn to the case of $k > 1$. As an induction hypothesis, we assume that the claim of Theorem 7 holds for all encoding-decoding schemes $\widetilde{S} \overset{\mathsf{Enc}'}{\to} \mathbb{G}^k \overset{\mathsf{Dec}'}{\to} \widetilde{S}$. I.e., there exists a $\mathbf{g}^{\vec{x}} \in \mathsf{Enc}'(\widetilde{S})$ such that the run of Dec' on $\mathbf{g}^{\vec{x}}$ performs at least $\left|\widetilde{S}\right|^{\frac{1}{k}} - 1$ group equality checks.

We show the statement for the encoding-decoding scheme $S \overset{\mathsf{Enc}}{\to} \mathbb{G}^{k+1} \overset{\mathsf{Dec}}{\to} S$. Let $\mathcal{T}_{k+1}$ be the corresponding decision tree constructed from the configurations of Dec. As before, for each equality check, we consider the sets $X_{\mathsf{in}}, X_0, X_1$. We note that in this case, an equality check corresponds to checking a linear equation $\vec{\gamma} \in \mathbb{Z}_p^{k+2}$ over the vector of exponents of the input elements $\mathbf{g}_1, \ldots, \mathbf{g}_{k+1}$.

Let (c, X_{in}) be an inner vertex of $\mathcal{T}_{k+1}$ and let $\vec{\gamma} \in \mathbb{Z}_p^{k+2}$ be the linear equality check queried by Dec at this point. We call the equality check $\mathbf{g}_1^{x_1} \cdots \mathbf{g}_{k+1}^{x_{k+1}} \cdot \mathbf{g}^{x_{k+2}} \overset{?}{=} \mathbf{1}_{\mathbb{G}}$ at this point *meaningful* if (c, X_{in}) has two children, otherwise we call it *redundant*. If the check is meaningful, we have for the two children (c_{Yes}, X_1) and (c_{No}, X_0) that X_1 and X_0 are disjoint with $X_{\mathsf{in}} = X_1 \cup X_0$. If the linear equality check is redundant, there is only one child (c_{Yes}, X_1) or (c_{No}, X_0) with $X_1 = X_{\mathsf{in}}$ or $X_0 = X_{\mathsf{in}}$.

Proposition 13. *For any meaningful equality check at an inner vertex in $\mathcal{T}_{k+1}$, all elements of X_1 lie on an affine hyperplane of dimension at most k in $\mathbb{Z}_p^{k+1}$.*

Proof. Recall that an equality check takes as input an affine linear equation $\vec{\gamma} \in \mathbb{Z}_p^{k+2}$ over the input group element exponents. We have

$$\begin{aligned} X_1 &= \left\{ \vec{x} \in X_{\mathsf{in}} \;\middle|\; \sum_{i=1}^{k+1} \gamma_i \cdot x_i = -\gamma_{k+2} \mod p \right\} \\ &\subseteq \left\{ \vec{x} \in \mathbb{Z}_p^{k+1} \;\middle|\; \sum_{i=1}^{k+1} \gamma_i \cdot x_i = -\gamma_{k+2} \mod p \right\} =: H. \end{aligned}$$

[2] We consider only the behaviour of Dec on inputs $x \in \mathsf{Enc}(S)$.

Now, the dimension of the affine space H is k, if $(x_1, \ldots, x_{k+1}) \neq 0$, and k, otherwise.

As the equality check is meaningful, the set X_0 must be non-empty. This can only be the case if the affine space H has dimension at most k, as an affince space of dimension $k+1$ would cover the entire space $\mathbb{Z}_p^{k+1}$ (i.e. X_1 would cover the entire space, leaving $X_0 = \emptyset$).

It follows that H has dimension k and must be an affine hyperplane. □

Proposition 14. *If for all meaningful equality checks, $|X_1| < \sqrt[k+1]{|S|}^k$, then there is a path in $\mathcal{T}_{k+1}$ that needs to make at least $\geq \sqrt[k+1]{|S|} - 1$ meaningful linear equality checks.*

Proof. Whenever an inner vertex $(c, X_{\mathsf{in}}) \in \mathcal{T}_{k+1}$ has two children (c_{Yes}, X_1) and (c_{No}, X_0), we must have

$$|X_0| = |X_{\mathsf{in}}| - |X_1| > |X_{\mathsf{in}}| - \sqrt[k+1]{|S|}^k.$$

So, at each meaningful check the set X_{in} can lose at most $\sqrt[k+1]{|S|}^k$ elements. Consider now the path of $\mathcal{T}_k$ where each meaningful check gets answered negatively and let t be the number of meaningful equality checks along this graph. Note, that the path has to end in a leaf (c, X_{in}) with $|X_{\mathsf{in}}| = 1$. We have:

$$\begin{aligned}
&|X_{\text{start}}| - t \cdot \sqrt[k+1]{|S|}^k \leq 1 \\
\iff &t \cdot |S|^{k/(k+1)} \geq |X_{\text{start}}| - 1 = |S| - 1 \\
\iff &t \geq \frac{|S| - 1}{|S|^{k/(k+1)}} = \frac{|S|}{|S|^{k/(k+1)}} - \frac{1}{|S|^{k/(k+1)}} \\
\implies &t \geq \sqrt[k+1]{|S|} - 1.
\end{aligned}$$

□

Proposition 15. *If there is a meaningful equality check in $\mathcal{T}_{k+1}$ such that $|X_1| \geq \sqrt[k+1]{|S|}^k$, then there exists a path in the tree that contains $\sqrt[k+1]{|S|} - 1$ equality checks.*

Proof. Let (c, X_{in}) be the inner vertex of $\mathcal{T}_{k+1}$ with children (c_{Yes}, X_1) and (c_{No}, X_0) s.t. $|X_1| \geq \sqrt[k+1]{|S|}^k$.

We want to apply our induction hypothesis here. Recall that by Proposition 13, there exists an affine hyperplane H of dimension k such that $X_1 \subseteq H$ As H is a hyperplane there exists an affine linear bijection $\phi\colon H \mapsto \mathbb{Z}_p^k$. Now, one can apply the ϕ and its inverse ϕ^{-1} on the exponents of group elements by just making $O(k^2)$ group operations and exponentiations. We will denote those maps by $\varphi(\mathbf{g}^{\vec{x}}) := \mathbf{g}^{\phi(\vec{x})}$ and $\varphi^{-1}(\mathbf{g}^{\vec{y}}) := \mathbf{g}^{\phi^{-1}(\vec{y})}$ for $\vec{y} \in \mathbb{Z}_p^k, \vec{x} \in \mathbb{Z}_p^{k+1}$.

Set $\widetilde{S} := \mathsf{Enc}^{-1}(X_1)$. We define a new encoding-decoding scheme $\widetilde{S} \overset{\mathsf{Enc}}{\to} \mathbb{G}^k \overset{\mathsf{Dec}}{\to} \widetilde{S}$ as follows:

$$\begin{aligned}
\mathsf{Enc}'(s) &:= \varphi(\mathsf{Enc}(s)) \in \mathbb{G}^k \\
\mathsf{Dec}'(\mathbf{g}^{\vec{y}}) &:= \mathsf{Dec}(\varphi^{-1}(\mathbf{g}^{\vec{y}})) \in S.
\end{aligned}$$

The correctness of $(\mathsf{Enc}', \mathsf{Dec}')$ follows from the correctness of $(\mathsf{Enc}, \mathsf{Dec})$. Further, the number of meaningful equality checks Dec' needs to make in the worst cases is lower than the number of meaningful equality checks Dec needs to make in the worst case. For the size of $\widetilde{S}$ we have

$$\left|\widetilde{S}\right| = |X_1| \geq \sqrt[k+1]{|S|}^k.$$

Now, let $\mathcal{T}_k$ be the decision tree that we can derive from the configuration graph of Dec'. According to our induction hypothesis, $\mathcal{T}_k$ has a path with at least $\sqrt[k]{\left|\widetilde{S}\right|} - 1$ meaningful equality checks. Hence, there must exist an input $\mathbf{g}^{\vec{x}} \in \mathsf{Enc}(S)$ s.t. $\mathsf{Dec}(\mathbf{g}^{\vec{x}})$ needs to make at least

$$\sqrt[k]{\left|\widetilde{S}\right|} - 1 \geq \sqrt[k]{\sqrt[k+1]{|S|}^k} - 1 = \sqrt[k+1]{|S| - 1}$$

meaningful equality checks.

□

Proposition 14 and Proposition 15 together now imply that Dec needs to make $\sqrt[k+1]{|S|} - 1$ meaningful equality checks in the worst case. This finishes our induction.

For the number of group operations we have:

Lemma 16. *Any deterministic algorithm with input* $(\mathbf{g}_1, \ldots, \mathbf{g}_k)$ *that makes* t *non-redundant equality needs to make* $\alpha \geq \sqrt{2t + \frac{1}{4}} - k + \frac{1}{2}$ *group operations when run on input* $(\mathbf{g}_1, \ldots, \mathbf{g}_k)$.

Proof. Recall that in the type safe generic group model, equality checks are free. However, as we are counting non-redundant equality checks, every equality check needs to have a new pair of input group element vectors $(\mathbf{g}_1, \mathbf{g}_2)$ that have never been compared to each other before. In an algorithm with runtime at most α (measured in the number of cost-inducing group operations), at most $\alpha + k$ distinct group elements can be computed, which means at most $\binom{\alpha+k}{2} = \frac{(\alpha+k)\cdot(\alpha+k-1)}{2}$ pairs of group elements can be used as inputs to equality gates.

This yields the following:

$$\frac{(\alpha + k)(\alpha + k - 1)}{2} \geq t$$

which we can solve for $\alpha \geq \max\left(0, \sqrt{2t + \frac{1}{4}} - k + \frac{1}{2}\right)$. □

4 Lower Bounds for LTFs with Many Bits and Few Group Elements in the Output

In this section, we show upper bounds for the lossiness of LTFs where the output has only few group elements but many bits. From Theorem 7 we cannot derive

useful bounds for these LTFs directly, as the corresponding encoding-decoding scheme could put a lot of information about the input in the bit part of the output. Thus, we have to show that a LTF cannot put too much information in the bit output, at least not if the hardness of distinguishing lossy from injective keys is based on a group assumption. To formalize this intuition, we want to construct an adversary that can distinguish lossy from injective keys if there is too much information about the input in the bit part of the output. As we assume the hardness comes from the group, we allow the adversary to have arbitrary runtime and only restrict its access to the group oracle. The goal of this adversary compute the bit part of the output for all possible input and then check how many different options for this bit part there are (i.e. the adversary wants to compute the size of the set Y_{ek} from Lemma 10).

However, this bit part of the output may be dependent on group elements from the evaluation key, i.e. the bits in the output may be derived from the outcome of group equality checks. Therefore, the adversary needs to find some way to approximate the outcome of such group equality checks. To this end, we first define a class of LTFs in Section 4.1 for which it is easy to find such an approximation because for each possible group equality check, the probability of passing is either very close to 1 or very close to 0. We provide an adversary for this class that makes no group operations to estimate the size of Y_{ek}. This in turn means that if the sizes of Y_{ek} cannot differ too much for lossy and injective keys, and thus the lossiness is bounded as it has to be "contained" in the group element part of the output.

As a second step, we show how for an LTF that does not already fulfill this property, we can narrow down the distribution from which the challenge evaluation key comes such that it fulfills the same property as before. We do this using a statistical binary search which we describe generically in Section 4.2, and then we apply this strategy in Section 4.3 to obtain an adversary that makes only polynomially many group operations to distinguish lossy from injective keys if the size of Y_{ek} differs too much between the key types.

This in turn yields a logarithmic upper bound for the lossiness of group-based LTFs with a constant number of group elements in the output.

4.1 LTFs with Linearly Independent Evaluation Keys

Definition 17 (LTFs with Linearly Independent Evaluation Keys). *We say that a LTF* $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ *has* $q(\lambda)$-linearly independent evaluation keys *if*

- $\{\mathsf{ek} \mid (\mathsf{ek}, \cdot) \xleftarrow{\$} \mathsf{LGen}(1^\lambda, 1^n, \cdot)\} \subset \{0,1\}^{\ell_b} \times \mathbb{G}_{\mathbb{T}}^{\ell_g}$. *Where* $\ell_b(\lambda)$ *and* $\ell_g(\lambda)$ *are polynomials.*
- *For any* $\lambda \in \mathbb{N}$, *any* $\vec{b}' \in \{0,1\}^{\ell_b}$, *any vector* $\vec{v} \in \mathbb{Z}_p^{\ell_g+1} \setminus \{\vec{0}\}$ *and each* $a \in \{0,1\}$, *it holds that*

$$\Pr_{\substack{(\mathsf{ek},\mathsf{ik}) \xleftarrow{\$} \mathsf{LGen}(1^\lambda, 1^n, a) \\ (\vec{b}, \vec{\mathbf{e}}) := \mathsf{ek}}} \left[\mathbf{g}^{v_{\ell_g+1}} \prod_{i=1}^{\ell_g} \mathbf{e}_i^{v_i} = \mathbf{1}_{\mathbb{G}} \middle| \vec{b} = \vec{b}' \right] \notin \left[\frac{1}{2q(\lambda)}, 1 - \frac{1}{2q(\lambda)} \right]$$

for some function $q(\lambda) > 0$.

– *For any* $\lambda \in \mathbb{N}$, *any* $\vec{b}' \in \{0,1\}^{\ell_b}$, *any vector* $\vec{v} \in \mathbb{Z}_p^{\ell_g+1} \setminus \{\vec{0}\}$, *it holds that if*

$$\Pr_{\substack{(\mathsf{ek},\mathsf{ik}) \xleftarrow{\$} \mathsf{LGen}(1^\lambda,1^n,1) \\ (\vec{b},\vec{\mathbf{e}}) := \mathsf{ek}}} \left[\mathbf{g}^{v_{\ell_g+1}} \prod_{i=1}^{\ell_g} \mathbf{e}_i^{v_i} = \mathbf{1}_{\mathbb{G}} \middle| \vec{b} = \vec{b}' \right] < \frac{1}{2q(\lambda)}$$

then

$$\Pr_{\substack{(\mathsf{ek},\perp) \xleftarrow{\$} \mathsf{LGen}(1^\lambda,1^n,0) \\ (\vec{b},\vec{\mathbf{e}}) := \mathsf{ek}}} \left[\mathbf{g}^{v_{\ell_g+1}} \prod_{i=1}^{\ell_g} \mathbf{e}_i^{v_i} = \mathbf{1}_{\mathbb{G}} \middle| \vec{b} = \vec{b}' \right] < \frac{1}{2q(\lambda)}$$

and vice versa.

We give a very natural and common class of LTFs below that fulfill Definition 17.

Definition 18 (LTFs with Uber-like Evaluation Keys). *We say that an LTF* $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ *has* uber-like evaluation keys *if the following holds.*

– *The support of* LGen *is a subset of* $\mathbb{G}^{\ell_g}$ *where* $\ell_g(\lambda)$ *is a polynomial.*
– *There exists a family of polynomials* $P_1^{(\lambda,n)}, \ldots, P_{\ell_g}^{(\lambda,n)}$ *over* $\mathbb{Z}_p$ *in* $f(\lambda, n)$ *variables (where* f *is a polynomial) such that* $\mathsf{LGen}(1^\lambda, 1^n, 1)$ *samples values* $x_1, \ldots, x_{f(\lambda,n)} \xleftarrow{\$} \mathbb{Z}_p$ *and sets* $\mathbf{e}_i := \mathbf{g}^{P_i^{(\lambda,n)}(\vec{x})}$ *for* $i = 1, \ldots, \ell_g$.
– *There exists a family of polynomials* $Q_1^{(\lambda,n)}, \ldots, Q_{\ell_g}^{(\lambda,n)}$ *over* $\mathbb{Z}_p$ *in* $f(\lambda, n)$ *variables (where* f *is a polynomial) such that* $\mathsf{LGen}(1^\lambda, 1^n, 0)$ *samples values* $x_1, \ldots, x_{f(\lambda,n)} \xleftarrow{\$} \mathbb{Z}_p$ *and sets* $\mathbf{e}_i := \mathbf{g}^{Q_i^{(\lambda,n)}(\vec{x})}$ *for* $i = 1, \ldots, \ell_g$.

Lemma 19. *Any LTF* $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ *with uber-like evaluation keys where the polynomials* $P_1^{(\lambda,n)}, \ldots, P_{\ell_g}^{(\lambda,n)}$ *and* $Q_1^{(\lambda,n)}, \ldots, Q_{\ell_g}^{(\lambda,n)}$ *have degree at most* $d(\lambda)$ *has* $q(\lambda)$*-linearly independent evaluation keys (in the sense of Definition 17) for any* $q(\lambda) < \frac{p}{2d(\lambda)}$.

Proof. We show this for the injective case, the lossy case follows analogously. Fix n, λ. Let $\vec{v} \in \mathbb{Z}_p^{\ell_g(\lambda)+1}$. Then, $\overline{P} = v_{\ell_g+1} + \sum_{i=1}^{\ell_g(\lambda)} v_i \cdot P_i^{(\lambda,n)}$ is a polynomial of degree $\leq d(\lambda)$. We consider two cases.

1. $\overline{P} = 0 \in \mathbb{Z}_p^{\ell_g(\lambda)}$. Then

$$\Pr_{\substack{(\mathsf{ek},\mathsf{ik}) \xleftarrow{\$} \mathsf{LGen}(1^\lambda,1) \\ \mathsf{ek} =: \vec{\mathbf{e}}}} \left[\mathbf{g}^{v_{\ell_g+1}} \prod_{i=1}^{\ell_g} \mathbf{e}_i^{v_i} = \mathbf{1}_{\mathbb{G}} \right] = 1 \notin \left[\frac{1}{2q(\lambda)}, 1 - \frac{1}{2q(\lambda)} \right].$$

2. $\overline{P} \neq 0$. Then, it holds by the Schwartz-Zippel-Lemma (see Lemma 4) that $\Pr_{\vec{x}}[\overline{P}(\vec{x}) = 0] \leq \frac{d}{p}$. As the LGen algorithm samples $\vec{\mathbf{e}}$ by sampling $\vec{x} \xleftarrow{\$}$

$\mathbb{Z}_p^{f(\lambda,n)}$, it holds that

$$\Pr_{\substack{(\mathsf{ek},\mathsf{ik})\xleftarrow{\$}\mathsf{LGen}(1^\lambda,1)\\ \mathsf{ek}=:\vec{\mathbf{e}}}}\left[\mathbf{g}^{v_{\ell_g+1}}\prod_{i=1}^{\ell_g}\mathbf{e}_i^{v_i}=\mathbf{1}_{\mathbb{G}}\right]\leq\frac{d}{p}.$$

Using $q(\lambda)<\frac{p}{2d(\lambda)}$, we obtain $\frac{d(\lambda)}{p}<\frac{1}{2q(\lambda)}$.

We now prove the last item of Definition 17. In the case that the last item of Definition 17 does not hold, there exists a distinguisher between lossy and non-lossy keys that makes a polynomial number of group equality checks (but is otherwise computationally unbounded). The distinguisher computes a vector $\vec{v}$ such that

$$\left|\Pr_{\substack{(\mathsf{ek},\mathsf{ik})\xleftarrow{\$}\mathsf{LGen}(1^\lambda,1^n,1)\\ \vec{\mathbf{e}}:=\mathsf{ek}}}\left[\mathbf{g}^{v_{\ell_g+1}}\prod_{i=1}^{\ell_g}\mathbf{e}_i^{v_i}=\mathbf{1}_{\mathbb{G}}\right]-\Pr_{\substack{(\mathsf{ek},\perp)\xleftarrow{\$}\mathsf{LGen}(1^\lambda,1^n,0)\\ \vec{\mathbf{e}}:=\mathsf{ek}}}\left[\mathbf{g}^{v_{\ell_g+1}}\prod_{i=1}^{\ell_g}\mathbf{e}_i^{v_i}=\mathbf{1}_{\mathbb{G}}\right]\right|\geq 1-\frac{1}{q(\lambda)}.$$

Such a vector must exist as we already showed that the probability for each vector is either smaller than $\frac{1}{2q(\lambda)}$ or larger than $1-\frac{1}{2q(\lambda)}$ for lossy and injective keys separately. Thus, if there is a vector for which the last item does not hold, it must be such that the probability is larger than $1-\frac{1}{2q(\lambda)}$ for one type of key, and smaller than $\frac{1}{2q(\lambda)}$ for the other type. It thus follows that the difference between the probabilities must be larger than $1-\frac{1}{q(\lambda)}$. The algorithm can determine the vector $\vec{v}$ by running the LGen algorithm in its head computing the exponents of all group elements in the evaluation keys without ever querying the group oracle. It then checks if for the evaluation key $\mathsf{ek}=(\mathbf{e}_1,\ldots,\mathbf{e}_{\ell_g})$ it received, it holds that $\mathbf{g}^{v_{\ell_g+1}}\prod_{i=1}^{\ell_g}\mathbf{e}_i^{v_i}=\mathbf{1}_{\mathbb{G}}$. Without loss of generality, we may assume that the probability in the lossy case is the larger one. In this case, the adversary would output that the key is lossy if the equality check returned true, and injective otherwise. We obtain that this adversary has advantage $1-\frac{1}{q(\lambda)}$ at distinguishing lossy from injective keys, and thus, LTF would not be secure.

□

In the following, we show that for schemes with linearly independent evaluation keys, there exists an adversary that issues no group oracle queries (but is computationally unbounded) and breaks the scheme if there is too much lossiness "contained" in the non-group output of LEv.

Theorem 20. *Let* $\mathsf{LTF}=(\mathsf{LGen},\mathsf{LEv},\mathsf{LInv})$ *be a candidate* l*-lossy trapdoor function with* $q(\lambda)$*-linearly independent evaluation keys, where for any* $(\mathsf{ek},\cdot)\xleftarrow{\$}\mathsf{LGen}(1^\lambda,1^n,\cdot)$, $\mathsf{Image}(\mathsf{LEv}(\mathsf{ek},\cdot))\subset\{0,1\}^{\ell_b}\times\mathbb{G}^{\ell_g}$. *Denote by* $Y_{\mathsf{ek}}\subset\{0,1\}^{\ell_b}$ *the set*

$$Y_{\mathsf{ek}}:=\left\{y\in\{0,1\}^{\ell_b}\middle|\exists\vec{\mathbf{h}}\in\mathbb{G}^{\ell_g}\colon(y,\vec{\mathbf{h}})\in\mathsf{Image}(\mathsf{LEv}(\mathsf{ek},\cdot))\right\}.$$

Let $r(\lambda)$ *be an upper bound on the number of group equality checks that the algorithm* LEv *makes. If it holds that for all* $(\mathsf{ek},\mathsf{ik})\in\mathsf{Image}(\mathsf{LGen}(1^\lambda,1^n,1))$ *that*

$|Y_{\mathsf{ek}}| > 2^{n-l} + 2^n \cdot 3 \cdot \frac{2r(\lambda)}{q(\lambda)}$*, then there exists an adversary that does not make any group queries (but is computationally unbounded) that breaks the indistinguishability of lossy and injective keys with advantage*

$$\mathrm{adv}^{\mathsf{LIND}}_{\mathcal{A}}(\lambda) = 1 - \frac{1}{3}.$$

Proof Overview. In the proof, we construct an adversary against the indistinguishability of lossy and injective keys that is unbounded, but makes no queries to the group evaluation and the group equality check oracle. As access to the group is limited, the adversary cannot break any assumption that is hard in Maurer's generic group model. The adversary's strategy is the following: It precomputes the values from Y_{ek} in the theorem statement for all input values. If $|Y_{\mathsf{ek}}| \geq 2^{n-l}$, it outputs that the key was injective (if Y_{ek} is larger than 2^{n-l}, then so is $\mathsf{Image}(\mathsf{LEv}(\mathsf{ek}, \{0,1\}^n))$).

However, some of the bits in the output of LEv may actually depend on group elements, so the adversary cannot straightforwardly compute them without querying the group oracle. To avoid having to query the group oracle, recall that the probability for any linear equation to hold is either close to 0 or close to 1. Our adversary therefore simulates these group equality checks (without actually querying the group oracle) by responding with the most likely output of the group equality check.

Proof. We first provide an adversary that is unbounded in the amount of non-group operations it can make, but bounded in its group operations.

The adversary proceeds as follows: It receives an evaluation key $\mathsf{ek} =: (\vec{b}, \vec{\mathbf{e}}) \in \{0,1\}^{\ell_b} \times \mathbb{G}^{\ell_g}$. For each vector $\vec{v} \in \mathbb{Z}_p^{\ell_g+1}$, compute

$$pr(\vec{v}) = \Pr_{\substack{(\mathsf{ek},\mathsf{ik}) \xleftarrow{\$} \mathsf{LGen}(1^\lambda, 1^n, 1) \\ \mathsf{ek} =: (\vec{b}', \vec{\mathbf{e}}')}} \left[\mathbf{g}^{v_{\ell_g+1}} \prod_{i=1}^{\ell_g} \mathbf{e}_i'^{v_i} = \mathbf{1}_{\mathbb{G}} \middle| \vec{b} = \vec{b}' \right].$$

In the following, the adversary will simulate the group oracle to the evaluation algorithm $\mathsf{LEv}(\mathsf{ek}, \cdot)$. The simulation works as follows. For each group element $\mathbf{e}_i$ from the evaluation key, it internally keeps a formal variable E_i. It furthermore keeps a list L of all "group elements" the algorithm has computed so far. Initially, this list is filled with the formal variables from the evaluation key. For any group operation that $\mathsf{LEv}(\mathsf{ek}, \cdot)$ makes, the adversary adds together the two corresponding list entries from L. It adds the linear combination of the entries to its list. Whenever the algorithm makes a query to check two group elements for equality, it computes the vector $\vec{v} \in \mathbb{Z}_p^{\ell_g+1}$ corresponding to the equality check, i.e. the equality check checks whether $v_{\ell_g+1} + \sum_{i=1}^{\ell_g} E_i \cdot v_i = 0$. If $\vec{v} = \vec{0}$, it returns true. Otherwise, it checks whether $pr(\vec{v}) \leq \frac{1}{2q(\lambda)}$ and returns false if yes, and true otherwise.

The adversary then proceeds as follows to approximate the size of Y_{ek} for the evaluation key ek at hand as follows:

It iterates over all the possible input values of $\mathsf{LEv}(\mathsf{ek}, \cdot)$, i.e. over all $\vec{x} \in \{0,1\}^n$ and calculates the bit part of $\mathsf{LEv}(\mathsf{ek}, \vec{x})$. To do this without making any additional group operations, it simulates all group operations as described above. This yields an approximate set $\tilde{Y}_{\mathsf{ek}}$ for the evaluation key ek at hand. The adversary then compares whether $\left|\tilde{Y}_{\mathsf{ek}}\right| \geq 2^{n-l} + 2^n \cdot 3 \cdot \frac{r(\lambda)}{q(\lambda)}$ in which case it outputs that the key is injective, otherwise that it is lossy.

We now compute the advantage of the adversary $\mathcal{A}$.

Recall that $r(\lambda)$ is an upper bound on the number of equality checks made by LEv. Note that by Lemma 16, this is bounded by the square of the runtime of LEv and thus polynomial in λ.

Let $F_{\vec{x}}$ be the event that at least one of the equality checks in the computation of $\mathsf{LEv}(\mathsf{ek}, \vec{x})$ is simulated wrong by the adversary.

We have

$$\mathbb{E}\left[F_{\vec{x}}\right] \leq r(\lambda) \cdot \frac{1}{q(\lambda)}$$

Let W_1 (W_0) be the random variable that indicates how many bit strings in the evaluation were computed wrong for all $\vec{x} \in \{0,1\}^n$ in the case that the evaluation key was injective (lossy).

We have that

$$\mathbb{E}\left[W_1\right] = \sum_{\vec{x} \in \{0,1\}^n} \mathbb{E}\left[F_{\vec{x}}\right] \leq 2^n \cdot r(\lambda) \cdot \frac{1}{q(\lambda)}.$$

and

$$\mathbb{E}\left[W_0\right] = \sum_{\vec{x} \in \{0,1\}^n} \mathbb{E}\left[F_{\vec{x}}\right] \leq \ 2^n \cdot r(\lambda) \cdot \frac{1}{q(\lambda)}.$$

Using a Markov bound, we obtain for $\mu \in \{1,0\}$

$$\Pr\left[W_\mu \geq 3\mathbb{E}\left[W_\mu\right]\right] \leq \frac{1}{3}.$$

Thus, with probability $1 - \frac{1}{3}$, in case of an injective key

$$\begin{aligned}
\left|\tilde{Y}_{\mathsf{ek}}\right| \geq & |Y_{\mathsf{ek}}| - 3 \cdot \mathbb{E}[W_1] \\
> & 2^{n-l} + 2^n \cdot 3 \cdot \frac{2r(\lambda)}{q(\lambda)} - 3 \cdot 2^n \cdot r(\lambda) \cdot \frac{1}{q(\lambda)} \\
= & 2^{n-l} + 2^n \cdot 3 \cdot \frac{r(\lambda)}{q(\lambda)}.
\end{aligned}$$

Recall that as the function is l-lossy, for any $(\mathsf{ek}, \perp) \xleftarrow{\$} \mathsf{LGen}(1^\lambda, 1^n, 0)$, it holds that $|Y_{\mathsf{ek}}| \leq 2^{n-l}$. Therefore, in case of a lossy key, we get with probability $1 - \frac{1}{3}$

$$\begin{aligned}
\left|\tilde{Y}_{\mathsf{ek}}\right| \leq & |Y_{\mathsf{ek}}| + 3 \cdot \mathbb{E}[W_0] \\
\leq & 2^{n-l} + 3 \cdot 2^n \cdot \frac{r(\lambda)}{q(\lambda)}.
\end{aligned}$$

That is, in both cases, the probability that the adversary outputs the correct bit is $1 - \frac{1}{q(\lambda)}$.

In the end, we obtain that

$$\begin{aligned}
&\mathrm{adv}_{\mathcal{A}}^{\mathsf{LIND}} \\
=&2 \cdot \left(\frac{1}{2} \Pr_{\substack{(\mathsf{ek},\mathsf{ik}) \xleftarrow{\$} \mathsf{LGen}(1^\lambda, 1^n, 1) \\ \mathsf{ek} =: (\vec{b}, \vec{\mathbf{e}})}} [\mathcal{A} \text{ outputs } 1] + \frac{1}{2} \Pr_{\substack{(\mathsf{ek},\perp) \xleftarrow{\$} \mathsf{LGen}(1^\lambda, 1^n, 0) \\ \mathsf{ek} =: (\vec{b}, \vec{\mathbf{e}})}} [\mathcal{A} \text{ outputs } 0] - \frac{1}{2} \right) \\
\geq&2 \cdot \left(\frac{1}{2} \cdot \left(1 - \frac{1}{3} \right) + \frac{1}{2} \cdot \left(1 - \frac{1}{3} \right) - \frac{1}{2} \right) = 1 - \frac{2}{3}.
\end{aligned}$$

□

Corollary 21. *For an* LTF *as in Theorem 20, it holds that for all evaluation keys* ek *with* $(\mathsf{ek}, \mathsf{ik}) \xleftarrow{\$} \mathsf{LGen}(1^\lambda, 1^n, 1)$ *that* $|Y_{\mathsf{ek}}| \leq 2^{n-l} + 2^n \cdot 3 \cdot \frac{2r(\lambda)}{q(\lambda)}$.

Corollary 22. *A candidate lossy trapdoor function* $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ *where* LEv *makes at most* $r(\lambda)$ *many group equality queries with* $q(\lambda)$*-linearly independent evaluation keys where* $q(\lambda) \geq 3\lambda \cdot r(\lambda)$ *and with* k *constant can be at most* l*-lossy for* $l \in O(\log(\lambda))$.

Proof. From Lemma 10, we know that $|Y_{\mathsf{ek}}|$ is lower bounded by $2^{n-O(\log \lambda)}$. Furthermore, from Corollary 21, we know that $|Y_{\mathsf{ek}}| \leq 2^{n-l} + 2^n \cdot 3 \cdot r(\lambda) \cdot \frac{2}{q(\lambda)}$. Using that $\frac{q(\lambda)}{r(\lambda) \cdot 3} \in \Omega(\lambda)$, we obtain that

$$2^{n-O(\log \lambda)} \leq |Y_{\mathsf{ek}}| \leq 2^{n-l} + 2^{n-\Omega(\log \lambda)}$$

Solving for l yields

$$l \in O(\log \lambda).$$

□

More generally, if $q(\lambda)$ is superpolynomial, we can even bound the lossiness of LTF for a non-constant number of group elements in the image of LEv:

Corollary 23. *A candidate lossy trapdoor function* $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ *with* $q(\lambda)$*-linearly independent evaluation keys where* LEv *makes at most* $r(\lambda)$ *many group equality queries and* LInv *makes at most* $s(\lambda)$ *many group equality queries. If* $q(\lambda) \geq 7 \cdot r(\lambda) \cdot (s(\lambda) + 1)^k$*, then we have for the lossiness* l *of* LTF

$$l \in O(k \log(\lambda)).$$

Proof. Let ek be an injective evaluation key. Again, we deduce from Lemma 10

$$|Y_{\mathsf{ek}}| \geq \frac{2^n}{(s(\lambda) + 1)^k}.$$

From Corollary 21, we have

$$|Y_{\mathsf{ek}}| \leq 2^{n-l} + 2^n \cdot 3 \cdot r(\lambda) \cdot \frac{2}{q(\lambda)}.$$

So, we get in total

$$2^{n-l} + 2^n \cdot r(\lambda) \cdot \frac{6}{q(\lambda)} \geq \frac{2^n}{(s(\lambda)+1)^k}.$$

Solving for l gives us

$$-l \geq \log\left((s(\lambda)+1)^{-k} - 6 \cdot r(\lambda)/q(\lambda)\right),$$

which is equivalent to

$$l \leq \log\left(\frac{q(\lambda) \cdot (s(\lambda)+1)^k}{q(\lambda) - 6 \cdot r(\lambda) \cdot (s(\lambda)+1)^k}\right).$$

We can plug in the inequality $q(\lambda) \geq 7 \cdot r(\lambda) \cdot (s(\lambda)+1)^k$ and get

$$\begin{aligned} l &\leq \log\left(\frac{q(\lambda) \cdot (s(\lambda)+1)^k}{q(\lambda) - 6 \cdot r(\lambda) \cdot (s(\lambda)+1)^k}\right) \\ &\leq \log\left(\frac{7 \cdot r(\lambda) \cdot (s(\lambda)+1)^{2k}}{r(\lambda) \cdot (s(\lambda)+1)^k}\right) \\ &= \log\left(7 \cdot (s(\lambda)+1)^k\right) \in O(k \cdot \log s(\lambda)). \end{aligned}$$

□

If we consider for example LTFs with keys generated as in Definition 18 by polynomials of constant degree, then $q(\lambda)$ lies in $\Omega(p)$. If the number of group elements output by LEv lies in $o(\log(p)/\log(\lambda))$, then it follows that the lossiness of LTF can only be by a logarithmic factor larger than the number of group elements outputted by LEv.

4.2 Statistical Binary Search

In order to generalize our result from the previous subsection to LTFs that do not have already linearly independent keys, we describe a search algorithm that allows to reduce the distribution of a secret element through running a minimal amount of tests. Looking forward, the secret element will be the exponent vector of the group elements in the evaluation key, and the tests will be group equality checks that check some linear equation over the exponent vector.

Concretely, our algorithm—that we denote by $\mathcal{S}$—is in the following situation: An oracle $\mathcal{O}$, with which $\mathcal{S}$ interacts, samples a secret element x together with some information $aux \in \{0,1\}^*$ according to a distribution $\mathcal{D}$. $\mathcal{S}$ receives aux and knows the distribution $\mathcal{D}$ and, additionally, is aware of a finite (but potentially large) set T of tests resp. checks. $\mathcal{S}$ can—multiple times—pick a test

$t \in T$ and query the oracle $\mathcal{O}$ on it. $\mathcal{O}$ will evaluate the test t on x and send the result $t(x) \in \{0,1\}$ to $\mathcal{S}$. While $\mathcal{S}$ is computationally unbounded in principle, we are interested in keeping the number of tests run by $\mathcal{S}$ low. We can imagine that each test performed on x is assigned some fixed cost for $\mathcal{S}$.

We categorize tests into ones that are *meaningful* and *not meaningful* with respect to a given distribution $\mathcal{D}$. Intuitively speaking, a *meaningful* test is one where the algorithm can learn a lot about x because the probability that the test will pass on an element sampled from $\mathcal{D}$ is rather close to $\frac{1}{2}$. A *not meaningful* test on the other hand is one where the probability of passing for an element sampled from $\mathcal{D}$ is either very close to 0 or close to 1. More formally, we introduce a parameter $\varepsilon \in (0, 0.5]$ that specifies how close the probability needs to be to 0 or 1 in order for the test to be considered not meaningful, i.e. a test is meaningful w.r.t. the distribution $\mathcal{D}$ if

$$\Pr_{x' \xleftarrow{\$} \mathcal{D}}[t(x') = 1] \in [\varepsilon, 1 - \varepsilon]$$

and not meaningful otherwise.

We give a formal description of $\mathcal{S}$ in Algorithm 1. The algorithm $\mathcal{S}$ will not always be able to retrieve x, however, we show that if it is not able to retrieve x, it will at least be able to reduce the distribution of x—by a polynomial number of tests—down to a conditional distribution, on which all tests in T are not meaningful any more.

In the following, for a distribution $\mathcal{D}$ over a set X and an element $x \in X$, we will denote by $\mathcal{D}(x)$ the probability that x gets sampled by $\mathcal{D}$, i.e.

$$\mathcal{D}(x) = \Pr_{x' \leftarrow \mathcal{D}}[x = x'].$$

Further, for a non-empty subset $A \subset X$, we will denote by $\mathcal{D}_{|A}$ the conditional distribution of $\mathcal{D}$ on A, i.e. for $x \in X$ we set

$$\mathcal{D}_{|A}(x) := \begin{cases} \dfrac{\mathcal{D}(x)}{\sum_{x' \in A} \mathcal{D}(x')}, & \text{if } x \in A, \\ 0, & \text{if } x \notin A. \end{cases} \tag{3}$$

The idea of $\mathcal{S}$ is to look for a test t that is meaningful, i.e. the probability of x passing t is larger than ε, but smaller than $1 - \varepsilon$. As the adversary is unbounded outside of the test queries, it can compute the probabilities of tests passing without querying the oracle by iteratively sampling from $\mathcal{D}$ using all possible random coins and counting how often each test passes. If there is a meaningful test t, $\mathcal{S}$ queries $\mathcal{O}$ on t and receives the bit $t(x)$. Consequently, it updates its knowledge about x: $\mathcal{S}$ computes the conditional distribution $\mathcal{D}' := \mathcal{D}_{|\{x' \in X \mid t(x') = t(x)\}}$.

This new distribution either has no more meaningful tests, in which case the algorithm terminates, or the algorithm can repeat this step with another test t' that is meaningful w.r.t. $\mathcal{D}'$.

By repeating this procedure multiple times, $\mathcal{S}$ will finally be able to restrict the distribution of x to a distribution on which each test $t \in T$ will pass with very high probability $> 1 - \varepsilon$ or very low probability $< \varepsilon$, i.e. there are no more meaningful tests left. We show that under some natural requirement on $\mathcal{D}$, $\mathcal{S}$ will always terminate after a polynomial number of iterations and thus also using only a polynomial number of test oracle queries (if $\varepsilon = 1/\mathsf{poly}(\lambda)$).

Algorithm 1: Statistical Binary Search Algorithm $\mathcal{S}$
Input: A distribution $\mathcal{D}$ over a set $X \times \{0,1\}^*$, a set T of tests $t : X \to \{0,1\}$, a control parameter $\varepsilon \in (0, 0.5]$, an oracle $\mathcal{O}$ that samples secretly $(x, aux) \leftarrow \mathcal{D}$ and answers queries $t \in T$ with $t(x)$.
Output: A subset $A \subseteq X$.

Receive auxiliary information aux from $\mathcal{O}$
Set $i := 0$
Set $\mathcal{D}_0 := \mathcal{D}_{|X \times \{aux\}}$
Compute $A_0 := \{x' \in X \mid \mathcal{D}_0(x') > 0\}$
while $\exists t \in T : \Pr_{x' \leftarrow \mathcal{D}_i}[t(x') = 1] \in [\varepsilon, 1-\varepsilon]$ **do**
  Choose $t_{i+1} \in T$ s.t. $\Pr_{x' \leftarrow \mathcal{D}_i}[t_{i+1}(x') = 1] \in [\varepsilon, 1-\varepsilon]$
  Query $\mathcal{O}$ on t_{i+1} and receive $b_{i+1} := t_{i+1}(x)$
  Compute $A_{i+1} := \{x' \in A_i \mid t_{i+1}(x') = b_{i+1}\}$
  Set $\mathcal{D}_{i+1} := \mathcal{D}_{i|A_{i+1}}$
  Increment $i := i + 1$
end
Output A_i.

We state and prove the formal guarantees of $\mathcal{S}$:

Theorem 24. *Let $\mathcal{D}$ be a distribution over $X \times \{0,1\}^*$. Let T be a set of tests $t : X \to \{0,1\}$ and let $\varepsilon \in (0, 0.5]$.*

Let $\mathcal{S}$ be the algorithm from Algorithm 1.

1. *If $x \in X$ is the secret sampled by $\mathcal{O}$ together with the auxiliary data aux and if A is the output of $\mathcal{S}(\mathcal{D}, T, \varepsilon, \mathcal{O})$, then x lies in A. Further, we have for each test $t \in T$*

$$\Pr_{x' \leftarrow \mathcal{D}_{|A \times \{aux\}}}[t(x') = 1] \notin [\varepsilon, 1-\varepsilon]. \tag{4}$$

2. *If there is an $\eta \in \mathbb{N}$ s.t. we have for each $x \in X$ and $aux \in \{0,1\}^*$*

$$\mathcal{D}(x, aux) > 0 \implies \mathcal{D}(x, aux) \geq 2^{-\eta}, \tag{5}$$

then algorithm $\mathcal{S}$ on input $\mathcal{D}, T, \varepsilon, \mathcal{O}$ terminates after a finite number of steps and makes at most

$$\left\lfloor \frac{-\eta}{\log(1-\varepsilon)} \right\rfloor \tag{6}$$

calls of the oracle $\mathcal{O}$. If $\varepsilon = 1/q$ for $q \geq 2$, we have $-\eta/\log(1-\varepsilon) \leq q \cdot \eta$.

Proof. It is easy to verify the first point. At the beginning of the algorithm $\mathcal{S}$, x lies in A_0. If in an iteration step x lies in A_i and $\mathcal{S}$ receives the outcome $b_{i+1} = t_{i+1}(x)$ of a test $t_{i+1} \in T$, then x must lie in $A_{i+1} = \{x' \in A_i \mid t_{i+1}(x') = b_{i+1}\}$. Further, it is clear that $\mathcal{S}$ can only terminate if there does not exist a test $t \in T$ s.t. the probability of x passing test t is interesting enough i.e. between ε and $1 - \varepsilon$.

For the second point, we define the i-th *mass* for $i \geq 0$ by

$$\mathfrak{m}_i := \Pr_{x' \leftarrow \mathcal{D}_0}[x' \in A_i] = \sum_{x' \in A_i} \mathcal{D}(x'). \tag{7}$$

Note that we have $1 = \mathfrak{m}_0 > \mathfrak{m}_1 > \ldots > 0$. The number $\mathfrak{m}_i$ measures how much of the original search space we started with remained after the i-th iteration. The reason why we consider the mass of A_i instead of its count is that it may happen that A_0 contains exponentially many elements and each A_{i+1} is only by one element smaller than A_i. We will give an example for such a distribution in *Remark* 25.

We claim that the mass needs to decrease exponentially. Let $b_{i+1} = t_{i+1}(x)$ be the response of $\mathcal{O}$ in the $(i+1)$-th iteration step. Note, that we have

$$\Pr_{x' \leftarrow \mathcal{D}_i}[t_{i+1}(x') = b_{i+1}] = \Pr_{x' \leftarrow \mathcal{D}_0}[t_{i+1}(x') = b_{i+1} \mid x' \in A_i] \tag{8}$$

$$= \frac{\Pr_{x' \leftarrow \mathcal{D}_0}[t_{i+1}(x') = b_{i+1}, x' \in A_i]}{\Pr_{x' \leftarrow \mathcal{D}_0}[x' \in A_i]} \tag{9}$$

$$= \frac{\Pr_{x' \leftarrow \mathcal{D}_0}[x' \in A_{i+1}]}{\Pr_{x' \leftarrow \mathcal{D}_0}[x' \in A_i]} = \frac{\mathfrak{m}_{i+1}}{\mathfrak{m}_i}. \tag{10}$$

We know that $\Pr_{x' \leftarrow \mathcal{D}_i}[t_{i+1}(x') = b_{i+1}]$ must be bounded between ε and $1 - \varepsilon$. It follows, $\mathfrak{m}_{i+1}/\mathfrak{m}_i \leq 1 - \varepsilon$ and, hence

$$\mathfrak{m}_i \leq (1 - \varepsilon)^i. \tag{11}$$

However, since the probability of the smallest possible outcome of $\mathcal{D}_0$ is at least $2^{-\eta}$, $\mathfrak{m}_i$ can never become smaller than $2^{-\eta}$. It follows that the maximum number i of iterations $\mathcal{S}$ can perform must be smaller than $\eta/(-\log(1-\varepsilon))$. □

Remark 25. In Theorem 24, we really need that the smallest positive probability of $\mathcal{D}$ is larger than $2^{-\eta}$ for bounding the number of iterations of $\mathcal{S}$. Consider, for example, the distribution $\mathcal{D}$ over the natural numbers $\{1, 2, \ldots\}$ that outputs 1 with probability $\frac{1}{2}$, 2 with probability $\frac{1}{4}$, 3 with probability $\frac{1}{8}$ et cetera.

If T consists of tests t of the form $t(x) = 1 \iff x \leq \alpha$ for some α, then the runtime of $\mathcal{S}$ may be arbitrarily high. In particular, for $x \in \{1, 2, \ldots\}$, the number of iterations of $\mathcal{S}$ with $\varepsilon = 1/2$ will be exactly x.

Note, that this counter example also shows that we cannot prove Theorem 24 by only considering entropical inequalities. The entropy of the distribution $\mathcal{D}$ of the counter example here is low, however it is equal to the entropy of $\mathcal{D}_{|\{2,3,4,\ldots\}}$. In general, it may happen that—after $\mathcal{S}$ applies an iteration step—the entropy of $\mathcal{D}_{i+1}$ is higher than the entropy of $\mathcal{D}_i$. So, we really need to argue over the mass instead of the entropy.

4.3 Bounding the Lossiness of LTFs with Linearly Dependent Keys

We now use the strategy described in Section 4.2 to extend the results from Section 4.1 to general LTFs where the number of group elements in the output is small.

Theorem 26. *Let* $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ *be a candidate* l*-lossy trapdoor function , where for any* $(\mathsf{ek}, \cdot) \xleftarrow{\$} \mathsf{LGen}(1^\lambda, 1^n, \cdot)$*,* $\mathsf{Image}(\mathsf{LEv}(\mathsf{ek}, \cdot)) \subset \{0,1\}^{\ell_b} \times \mathbb{G}^{\ell_g}$*. Denote by* $Y_{\mathsf{ek}} \subset \{0,1\}^{\ell_b}$

$$Y_{\mathsf{ek}} \coloneqq \left\{ y \in \{0,1\}^{\ell_b} \middle| \exists \vec{\mathbf{h}} \in \mathbb{G}^{\ell_g} \colon (y, \vec{\mathbf{h}}) \in \mathsf{Image}(\mathsf{LEv}(\mathsf{ek}, \cdot)) \right\}.$$

Let $r(\lambda)$ *be an upper bound on the number of group equality checks that the algorithm* LEv *makes. Let* $q(\lambda) > 6r(\lambda)$ *be a polynomial. If it holds that for all* $(\mathsf{ek}, \mathsf{ik}) \in \mathsf{Image}(\mathsf{LGen}(1^\lambda, 1^n, 1))$ *that* $|Y_{\mathsf{ek}}| > 2^{n-l} + 2^n \cdot 3 \cdot \frac{2r(\lambda)}{q(\lambda)}$ *then there exists an adversary that is bounded to polynomially many group evaluation queries, but unbounded otherwise that breaks the indistinguishability of lossy and injective keys with advantage*

$$\mathrm{adv}^{\mathsf{LIND}}_{\mathcal{A}}(\lambda) = 1 - \frac{1}{3}$$

Proof overview. We construct an adversary as follows: The adversary first applies the statistical binary search from Section 4.2 with the tests being group equality checks for linear equations that hold with a probability larger than $\frac{1}{2q(\lambda)}$ and smaller than $\frac{1}{2q(\lambda)}$. This yields a new distribution over the evaluation keys conditioned on the results of the group equality checks. This distribution has at least one of the following properties.

- It is a distribution over only polynomially many options of keys. In this case, the adversary will iterate over the candidate keys and check which key it is. It can then use knowledge of all the discrete logarithms of the key to determine whether it is lossy or not.
- The distribution has a property equivalent to that of the linearly independent keys property. Thus, the adversary applies the same strategy as the adversary in Theorem 20 to estimate the size of the set Y_{ek} for the key in question. It then uses this estimate to decide whether or not the key is lossy.

Proof. We describe the adversary's strategy in more detail.

We define the distribution $\mathcal{D}_0$ samples vectors in $\mathbb{Z}_p^{\ell_g}$ as follows:

00 Draw random coins $c \xleftarrow{\$} \{0,1\}$
01 Run $(\mathsf{ek}, \cdot) := \mathsf{LGen}(1^\lambda, 1^n, c)$
02 Parse $\mathsf{ek} =: (\vec{b'}, \vec{\mathbf{e}'})$
03 If $\vec{b} \neq \vec{b'}$, go back to start
04 Otherwise extract the discrete logarithms $\vec{e'} \coloneqq \mathrm{dlog}(\vec{\mathbf{e}'})$ from $\mathsf{LGen}(1^\lambda, 1^n, c)$
05 Output $\vec{e'}$

We note that since we assume the algorithm LGen to be generic in the sense of Definition 1, anyone running this algorithm can extract the discrete logarithms of all group elements contained in the evaluation key, and in fact, the adversary can simulate the LGen algorithm without ever making a group oracle query to obtain the discrete logarithms of the keys.

In the first phase, the adversary runs group equality checks to narrow down the distribution from which the evaluation key it go may have come from to one that either fulfills the linear independence property from Definition 17, or supports only polynomially many evaluation keys. Denote $\mathsf{ek} =: (\vec{b}, \vec{\mathbf{e}})$. The adversary computes the probabilities

$$pr_0(\vec{v}) = \Pr_{\vec{e} \xleftarrow{\$} \mathcal{D}_0} \left[v_0 + \sum_{i=1}^{\ell_g} e_i \cdot v_i = 0 \right]$$

for each $\vec{v} \in \mathbb{Z}_p^{\ell_g+1}$.

It then iteratively computes a new distribution $\mathcal{D}_j$ as follows: If $|\mathrm{supp}(\mathcal{D}_{j-1})|$ is not polynomial in λ, and there exists $\vec{v}$ such that $pr_{j-1}(\vec{v}) := \Pr_{\vec{e} \xleftarrow{\$} \mathcal{D}_{j-1}} \left[\sum_{i=1}^{\ell_g} e_i \cdot v_i = 0 \right] \in \left[\frac{1}{q(\lambda)}, 1 - \frac{1}{q(\lambda)} \right]$, then it picks a vector $\vec{v}$ with $pr_{j-1}(\vec{v}) \in \left[\frac{1}{q(\lambda)}, 1 - \frac{1}{q(\lambda)} \right]$. It then queries the group equality oracle on $\prod_{i=1}^{\ell_g} \mathbf{e}_i^{v_i} \stackrel{?}{=} \mathbf{1}_{\mathbb{G}}$. It computes the sets

$$A_j^{=} := \left\{ \vec{e} \in \mathrm{supp}(\mathcal{D}_{j-1}) \middle| \sum_{i=1}^{\ell_g} e_i \cdot \vec{v} = 0 \right\}$$

and

$$A_j^{\neq} := \left\{ \vec{e} \in \mathrm{supp}(\mathcal{D}_{j-1}) \middle| \sum_{i=1}^{\ell_g} e_i \cdot \vec{v} \neq 0 \right\}.$$

It then computes the distribution $\mathcal{D}_j := \mathcal{D}_{j-1|A_j^{=}}$ if the equality check returned true, and $\mathcal{D}_j := \mathcal{D}_{j-1|A_j^{\neq}}$ if it returned false.

Once this iterative process stops, there are two possible scenarios for the adversary.

1. $|\mathrm{supp}(\mathcal{D}_j)| < \mathsf{poly}(\lambda)$ for a fixed polynomial poly. In this case, the adversary tests for each $\vec{e} \in \mathrm{supp}(\mathcal{D}_j)$ whether $(\mathbf{g}^{e_1}, \ldots, \mathbf{g}^{e_{\ell_g}}) = \vec{\mathbf{e}}$. This requires only polynomially many group operations and group equality checks as there are only polynomially many candidate keys at this point.
2. For each vector $\vec{v} \in \mathbb{Z}_p^{\ell_g}$, $pr_j(\vec{v}) \notin \left[\frac{1}{q(\lambda)}, 1 - \frac{1}{q(\lambda)} \right]$. In this case, the adversary proceeds the same way as the adversary in Theorem 20. In this case, it succeeds with probability $1 - \frac{2}{3}$.

We analyse how many group operations this adversary makes. For any $(\mathsf{ek}, \cdot) \in \mathsf{Image}(\mathsf{LGen}(1^\lambda, 1^n, \cdot))$, with $\mathsf{ek} =: (\vec{b}, \vec{\mathbf{e}})$, $\Pr_{\vec{e} \xleftarrow{\$} \mathcal{D}_0} \left[(\mathbf{g}^{e_1}, \ldots, \mathbf{g}^{e_{\ell_g}}) = \vec{\mathbf{e}} \right] \geq 2^{-\rho(\lambda)}$

where $\rho(\lambda)$ is an upper bound on the length of the random input the algorithm $\mathsf{LGen}(1^\lambda, 1^n, \cdot)$ uses. In particular $\rho(\lambda)$ is polynomial in λ. We apply Theorem 24 with $\varepsilon = \frac{1}{q(\lambda)}$, $\eta = \rho(\lambda)$ and obtain that the adversary makes at most $\left\lfloor \frac{-\rho(\lambda)}{\log(1-\frac{1}{q(\lambda)})} \right\rfloor \leq q(\lambda) \cdot \rho(\lambda)$ queries. For $q(\lambda)$ polynomial, this is polynomial.

□

Corollary 27. *Let* $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ *be a lossy trapdoor function with a constant number* k *of group elements in its output. Then,* LTF *can at most be* l*-lossy for* $l \in O(\log \lambda)$*.*

Proof. We set $q(\lambda) = 6\lambda \cdot r(\lambda)$. We get that $|Y_{\mathsf{ek}}| \leq 2^{n-l} + 2^n \cdot \frac{1}{\lambda} = 2^{n-l} + 2^{n-\log(\lambda)}$. Using Lemma 10, we furthermore obtain $|Y_{\mathsf{ek}}| \geq 2^{n-O(\log(\lambda))}$ and thus

$$2^{n-O(\log(\lambda))} \leq |Y_{\mathsf{ek}}| \leq 2^{n-l} + 2^{n-\log(\lambda)}$$

Solving for l yields $l \in O(\log(\lambda))$. □

5 Impossibility of LTFs with Group Elements as Input

In this section we consider LTFs that take group elements as inputs and output group elements as well. For LTFs where LEv is restricted to generic group operations as in Definition 1, we show that they can lose only less than one group element worth of information, or none at all if LEv makes no use of group equality checks.

Definition 28 (LTFs with Group Elements as Input). *Let* $\mathsf{LTF} = (\mathsf{LGen}, \mathsf{LEv}, \mathsf{LInv})$ *be a lossy trapdoor function that has the following input and output spaces:*

LGen**:** *Takes as input a bitstring* 1^λ *corresponding to the security parameter, a bitstring* 1^n *corresponding to the input length, and a single bit* b *indicating whether to generate a lossy key or an injective key pair. It outputs an evaluation key* ek *consisting of bits* $z_1, \ldots, z_{\ell_b}$ *and group elements* $\mathbf{g}_1, \ldots, \mathbf{g}_{\ell_g}$*. Without loss of generality, we consider inversion keys to be bitstrings (any group element inside the inversion key can also be expressed through its exponent to a fixed generator* $\mathbf{g}$*)*

LEv**:** *Takes as input an evaluation key* ek *as above, and an input bitstring which is parsed as group elements* $\mathbf{h}_1, \ldots, \mathbf{h}_n$*. It outputs a group element string* $\mathbf{y}_1, \ldots, \mathbf{y}_k$*.*

LInv**:** *Takes as input an inversion key* ik *as above and a group element string* $\mathbf{y}_1, \ldots, \mathbf{y}_k$ *and outputs a string of group elements* $\mathbf{h}_1, \ldots \mathbf{h}_n$*.*

Theorem 29. *Let* LTF *be a lossy trapdoor function as in Definition 28 with perfect correctness where the* LEv *algorithm makes at most* $r(\lambda)$ *many group equality queries. If* LTF *is* l*-lossy for some integer* $l > \log p - \log(p - r(\lambda))$*, then lossy keys and injective keys can be distinguished by a probabilistic polynomial-time algorithm.*

Proof.

Evaluation algorithm with no equality queries. As a warm-up, we prove the result on a restricted class of evaluation algorithms LEv, which are limited to making no equality queries. In this case, as LEv is generic in the sense of Definition 1, it can only apply the group operation polynomially many times to compute the output group elements. Thus, for each position $j \in k$ in the output, there is a vector $\vec{v_j} \in \mathbb{Z}_p^{\ell_g}$ and a vector $\vec{w_j} \in \mathbb{Z}_p^n$ along with a scalar u_j such that for input $\mathbf{h}_1, \ldots, \mathbf{h}_n$ and evaluation key group elements $\mathbf{g}_1, \ldots, \mathbf{g}_{\ell_g}$, the output $\mathbf{y}_j$ is computed as:

$$\mathbf{y}_j \coloneqq \mathbf{g}^{u_j} \prod_{i=1}^{\ell_g} \mathbf{g}_i^{v_{j,i}} \cdot \prod_{i=1}^{n} \mathbf{h}_i^{w_{j,i}}.$$

Note that the vectors $\vec{v_j}$ and $\vec{w_j}$ may be derived from the bits in the evaluation key through bit operations, but as we assume the algorithm makes no equality checks, they can be computed without making any group queries. We describe how to distinguish injective from lossy keys in this case. The vectors $\vec{w_j}$ give rise to a matrix W and the evaluation function can be seen as doing a matrix multiplication of $(\mathbf{h}_1, \ldots, \mathbf{h}_n)$ with W in the exponent and then multiplying component-wise with the vector $\left(\mathbf{g}^{u_1} \prod_{i=1}^{\ell_g} \mathbf{g}_i^{v_{1,i}}, \ldots, \mathbf{g}^{u_k} \prod_{i=1}^{\ell_g} \mathbf{g}_i^{v_{k,i}}\right)$. This operation is injective if and only if W has full rank so it suffices to check for this to distinguish lossy from non-lossy keys.

Evaluation algorithm with equality queries. We now turn to the general case where the evaluation algorithm may make equality check queries as well. Our goal is to compute the matrix W again. However, the entries of the matrix W may depend on the output of group equality checks. We describe a "simulated" version of the algorithm LEv that will yield an "approximate" matrix $\tilde{W}$. This simulated version of LEv replaces some of the group equality checks with estimations. We distinguish two types of group equality checks that can occur during a run of LEv. As in the previous sections, we note that group equality checks boil down to checking a linear equation in the exponents.

1. The linear equation contains only group elements $\mathbf{g}, \mathbf{g}_1, \ldots, \mathbf{g}_{\ell_g}$ with non-zero coefficients. In this case, the outcome of the equation is completely derived from the evaluation key. Our simulated variant of LEv makes the according equality check query to the real group oracle.
2. The linear equation contains also group elements $\mathbf{h}_1, \ldots, \mathbf{h}_n$ with non-zero coefficients. In this case, the simulated version of LEv assumes that the group equality check oracle returns 0.

Using the above simulation combined with the technique for the case with no equality checks, we obtain the approximate matrix $\tilde{W}$ along with values $\tilde{u_j}$ and vectors $\tilde{\vec{v_j}}$. We note that by construction, these values are independent of the input. We want to argue however, that for a large fraction of the input, these values model the evaluation correctly (namely the set of input vectors for which every equality check from the run of $\mathsf{LEv}(\mathsf{ek}, \cdot)$ returns 0 is large).

Denote by

$$\mathcal{H} \coloneqq \left\{ \vec{\mathbf{h}} \in \mathbb{G}^n \middle| \mathsf{LEv}(\mathsf{ek}, \vec{\mathbf{h}}) \neq \vec{h}^{\tilde{W}} \circ \left(\mathbf{g}^{u_1} \prod_{i=1}^{\ell_g} \mathbf{g}_i^{v_{1,i}}, \ldots, \mathbf{g}^{u_k} \prod_{i=1}^{\ell_g} \mathbf{g}_i^{v_{k,i}} \right) \right\}$$

the set of possible inputs for which the simulated evaluation deviates from the real evaluation where by $\circ$ we denote the component-wise product. We want to argue that $|\mathcal{H}| \leq r(\lambda) \cdot p^{n-1}$ where $r(\lambda)$ is an upper bound on the number of group equality checks that LEv makes. We note that each linear equality check as in the second item checks for membership of $\vec{\mathbf{h}}$ in a subspace of dimension at most $n-1$. This subspace contains $\leq p^{n-1}$ many input vectors. Therefore, each equality check can add at most $\leq p^{n-1}$ "deviating" input vectors to $\mathcal{H}$. As the algorithm is performed as if each equality check of the second type returned 0, it makes at most $r(\lambda)$ many equality checks (we do not perform equality checks that would occur in other "branches" of the algorithm that can only be reached if an equality check returned 1).

As each such equality check only covers a subspace of size at most p^{n-1}, the union of all these subspaces is of size at most $r(\lambda) \cdot p^{n-1}$.

Therefore, the set of "deviating" input vectors $\mathbb{H}$ has size at most $\mathcal{H} \leq p^{n-1} \cdot r(\lambda)$. Thus, if the matrix $\tilde{W}$ is of full rank, the function evaluates injectively on all but $r(\lambda) \cdot p^{n-1}$ possible input values.

This means that if lossy and injective keys are indistinguishable, the function can have a lossiness of at most

$$l \leq \log p - \log(p - r(\lambda))$$

in terms of bits where we consider that the LTF maps from a space of size $2^{\log(p) \cdot n}$ to one of size $2^{\log(p) \cdot n - l}$ The lossiness in terms of group elements can be expressed as $l' \leq 1 - \log_p(p - r(\lambda))$ where we consider that the LTF maps from a space of size p^n to one of size $p^{n-l'}$.

□

6 The LTF of Döttling, Garg, Ishai, Malavolta, Mour and Ostrovsky

In this section, we recall the LTF by [12]. As this LTF is a modular construction, it will be helpful to unroll it here for easier understanding. The construction steps in the original work are the following.

- Construct a trapdoor hash function (TDH) that takes bitstrings as inputs and outputs group elements. A TDH allows the key generator to generate encoding keys (along with trapdoors). Using an encoding key, the evaluator of the TDH then can create a "hint" that allows the key generator to retrieve a single bit of the input using the trapdoor. Which bit is retrieved depends on the trapdoor, and is not known to the evaluator. This TDH needs one group element per bit as a "hint" to decode the bit.

- Compactify the "hinting" group elements $\mathbf{e_j}$ into bits e_j using a distance function Dist. This distance function uses the representation of a group element interpreted as a bit string and thus cannot be efficiently computed in a generic group model as in Definition 1.
- Construct weak string OT from trapdoor hash functions by applying the TDH to the concatenation of the two strings and creating hints, where the encoding keys either allow to retrieve the first message or the second. Weak string OT allows for some correctness error.
- As the distance function from above introduced an error, apply an error correcting code to input before hashing and after retrieving the bits to obtain string OT from weak string OT.
- Finally, construct an LTF as follows: Generate the first message of string OT, where if the setup is injective, request to see the first secret, otherwise the second. Fix a second secret. The evaluation key consists of the first message of string OT along with the fixed second secret. When evaluating the LTF, generate the second OT message where the first secret is the input, and the second secret is the previously fixed secret from the eval key. To invert, use the internal state of the OT receiver to retrieve the first secret, i.e. the input to the LTF. If the LTF is in lossy mode, this would only allow to retrieve the second (fixed) secret.

Concretely, the LTF uses the following ingredients:

- A group $\mathbb{G}$ where the DDH problem is hard.
- A PRF PRF with input space $\mathbb{G}$ and output space $\{0,1\}^{\lceil \log(2\lambda) \rceil}$
- An error correcting code $\mathsf{Code} = (\mathsf{C.Enc}, \mathsf{C.Dec})$ with rate $1 - \frac{1}{\lambda}$. Denote by $m(n)$ the length of codewords of Code when plaintexts have length n.

We first recall the distance function Dist (borrowed from [9]) for an input group element $\mathbf{x}$, a step width $\mathbf{y}$, and a PRF key K (see Fig. 2). (We plugged in the parameter choices from the original work, namely input range $M = 1$ and error probability $\delta = \frac{1}{\lambda}$). [12] use the following property of the distance function.

$\underline{\mathsf{Dist}_{K,\mathbf{y}}(\mathbf{x})}$
06 $i := 0$
07 while $i \leq [2 \log_e(2\lambda)] \cdot \lambda$
08 if $\mathsf{PRF}_K(\mathbf{x} \cdot \mathbf{y}^i) = 0^{\lceil \log(2\lambda) \rceil}$
09 output $\mathsf{LSB}(i)$
10 else
11 $i := i + 1$.
12 output $\mathsf{LSB}(i)$.

Fig. 2: The distance function from [9] with the values $M = 1$ and $\delta = \frac{1}{\lambda}$ plugged in.

Proposition 30 (Proposition 3.2 in [9]). *Let $\mathbb{G}$ be a multiplicative cyclic group of prime order p with $[2\log_e(2\lambda)] \cdot \lambda < p$, let $\mathbf{y} \in \mathbb{G}$, and let PRF be a pseudo-random function. Then for any $\mathbf{x} \in \mathbb{G}$ it holds that*

$$\Pr_K \left[\mathsf{Dist}_{K,\mathbf{y}}(\mathbf{x}) \oplus \mathsf{Dist}_{K,\mathbf{y}}(\mathbf{xy}) = 1\right] \geq 1 - \frac{1}{\lambda}$$

We can now describe the LTF over input strings of length n.

$\mathsf{LGen}(1^\lambda, 1^n, b)$**:** Sample a matrix $A = \begin{pmatrix} \mathbf{g}_{1,0} \cdots \mathbf{g}_{2m(n),0} \\ \mathbf{g}_{1,1} \cdots \mathbf{g}_{2m(n),1} \end{pmatrix}$.

For $j = 1, \ldots, m(n)$, sample a pair $s_j, t_j \xleftarrow{\$} \mathbb{Z}_p$. Set $B_j = \begin{pmatrix} \mathbf{g}^s_{1,0} \cdots \mathbf{g}^s_{2m(n),0} \\ \mathbf{g}^s_{1,1} \cdots \mathbf{g}^s_{2m(n),1} \end{pmatrix}$.

If $b = 1$, replace the entry $\mathbf{g}_{j,1}^{s_j}$ in B_j by $\mathbf{g_{j,1}}^{s_j} \cdot \mathbf{g}^{t_j}$. If $b = 0$, replace the entry $\mathbf{g}_{m(n)+j,1}^{s_j}$ by $\mathbf{g}_{m(n)+j,1}^{s_j} \cdot \mathbf{g}^{t_j}$. Sample a value $\rho \xleftarrow{\$} \{0,1\}^n$ and a PRF key K. Set $\mathsf{ek} = (A, B_1, \ldots B_{m(n)}, \mathbf{g}^{t_1}, \ldots, \mathbf{g}^{t_{m(n)}}, \rho, K)$ and (if $b = 1$) $\mathsf{ik} = ((s_j, t_j)_{j=1}^{m(n)}, K)$.

$\mathsf{LEv}(\mathsf{ek}, x)$**:** Parse the evaluation key ek as above. Then compute $x' \coloneqq \mathsf{C.Enc}(x)$ and $\rho' \coloneqq \mathsf{C.Enc}(\rho)$. Compute

$$\mathbf{h} \coloneqq \prod_{i=1}^{m(n)} \mathbf{a}_{i,x'[i]} \cdot \prod_{i=1}^{2m(n)} \mathbf{a}_{i+m(n),\rho'[i]}$$

and for $j = 1, \ldots, m(n)$

$$e_j \coloneqq \mathsf{Dist}_{K,\mathbf{g}^{t_j}} \left(\prod_{i=1}^{m(n)} \mathbf{b}_{j,i,x'[i]} \cdot \prod_{i=1}^{m(n)} \mathbf{b}_{j,i+m(n),\rho'[i]} \right).$$

Output $(\mathbf{h}, e_1, \ldots, e_{m(n)})$.

$\mathsf{LInv}(\mathsf{ik}, (\mathbf{h}, e_1, \ldots, e_{m(n)}))$**:** Parse ik as above. For $j = 1, \ldots, m(n)$ compute

$$e_{j,0} \coloneqq \mathsf{Dist}_{K,\mathbf{g}^{t_j}} (\mathbf{h}^{s_j})$$

and

$$e_{j,1} \coloneqq \mathsf{Dist}_{K,\mathbf{g}^{t_j}} \left(\mathbf{h}^{s_j} \mathbf{g}^{t_j}\right).$$

If $e_j = e_{j,0}$ set $x''[j] \coloneqq 0$ and if $e_j = e_{j,1}$, set $x''[j] \coloneqq 1$.
Output $x \coloneqq \mathsf{C.Dec}(x'')$.

Remark 31 (Coverage by our Model). The LTF of [12] cannot be formalized in our model (i.e., within the constraints of Definition 1) as it applies a PRF to group elements to shorten the output. In particular, our results contrast rather than contradict the results from [12], and reveal an interesting difference between Maurer's and Shoup's variants of generic groups. (Our GGM formalization is essentially Maurer's [20], while Shoup's formalization [25] would seem to allow to formalize the LTF of [12].)

Remark 32 (Probabilistic LTFs). As an aside, we note that in [12], the LTF evaluation is in fact probabilistic, as it chooses a random exponent r and multiplies $\mathbf{h}$ by $\mathbf{g}^r$ and applies the equivalent multiplication by $\mathbf{g}^{s_j \cdot r}$ when computing the e_j. This leads to a non-standard variant of LTFs, and is a result of the modular construction. (It seems possible to convert this type of LTF into an ordinary LTF, although this also seems to imply a reduction in lossiness.)

References

1. Alwen, J., Krenn, S., Pietrzak, K., Wichs, D.: Learning with rounding, revisited - new reduction, properties and applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 57–74. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4˙4
2. Auerbach, B., Kiltz, E., Poettering, B., Schoenen, S.: Lossy trapdoor permutations with improved lossiness. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 230–250. Springer, Heidelberg (Mar 2019). https://doi.org/10.1007/978-3-030-12612-4˙12
3. Bellare, M., Brakerski, Z., Naor, M., Ristenpart, T., Segev, G., Shacham, H., Yilek, S.: Hedged public-key encryption: How to protect against bad randomness. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 232–249. Springer, Heidelberg (Dec 2009). https://doi.org/10.1007/978-3-642-10366-7˙14
4. Bellare, M., Hofheinz, D., Yilek, S.: Possibility and impossibility results for encryption and commitment secure under selective opening. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 1–35. Springer, Heidelberg (Apr 2009). https://doi.org/10.1007/978-3-642-01001-9˙1
5. Bellare, M., Kiltz, E., Peikert, C., Waters, B.: Identity-based (lossy) trapdoor functions and applications. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 228–245. Springer, Heidelberg (Apr 2012). https://doi.org/10.1007/978-3-642-29011-4˙15
6. Benhamouda, F., Herranz, J., Joye, M., Libert, B.: Efficient cryptosystems from 2^k-th power residue symbols. Journal of Cryptology **30**(2), 519–549 (Apr 2017). https://doi.org/10.1007/s00145-016-9229-5
7. Boldyreva, A., Fehr, S., O'Neill, A.: On notions of security for deterministic encryption, and efficient constructions without random oracles. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 335–359. Springer, Heidelberg (Aug 2008). https://doi.org/10.1007/978-3-540-85174-5˙19
8. Boyen, X., Li, Q.: All-but-many lossy trapdoor functions from lattices and applications. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 298–331. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63697-9˙11
9. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 509–539. Springer, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4˙19
10. Brandt, N., Hofheinz, D., Kastner, J., Ünal, A.: The price of verifiability: Lower bounds for verifiable random functions. In: Kiltz, E., Vaikuntanathan, V. (eds.) Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022, Proceedings, Part II. Lecture Notes in Computer

Science, vol. 13748, pp. 747–776. Springer (2022). https://doi.org/10.1007/978-3-031-22365-5“˙26, https://doi.org/10.1007/978-3-031-22365-5_26
11. Demillo, R.A., Lipton, R.J.: A probabilistic remark on algebraic program testing. Information Processing Letters **7**(4), 193–195 (1978). https://doi.org/https://doi.org/10.1016/0020-0190(78)90067-4, https://www.sciencedirect.com/science/article/pii/0020019078900674
12. Döttling, N., Garg, S., Ishai, Y., Malavolta, G., Mour, T., Ostrovsky, R.: Trapdoor hash functions and their applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 3–32. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-26954-8˙1
13. Döttling, N., Hartmann, D., Hofheinz, D., Kiltz, E., Schäge, S., Ursu, B.: On the impossibility of purely algebraic signatures. In: Nissim, K., Waters, B. (eds.) TCC 2021, Part III. LNCS, vol. 13044, pp. 317–349. Springer, Heidelberg (Nov 2021). https://doi.org/10.1007/978-3-030-90456-2˙11
14. Escala, A., Herold, G., Kiltz, E., Ràfols, C., Villar, J.: An algebraic framework for Diffie-Hellman assumptions. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 129–147. Springer, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40084-1˙8
15. Freeman, D.M.: Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 44–61. Springer, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5˙3
16. Freeman, D.M., Goldreich, O., Kiltz, E., Rosen, A., Segev, G.: More constructions of lossy and correlation-secure trapdoor functions. In: Nguyen, P.Q., Pointcheval, D. (eds.) PKC 2010. LNCS, vol. 6056, pp. 279–295. Springer, Heidelberg (May 2010). https://doi.org/10.1007/978-3-642-13013-7˙17
17. Hemenway, B., Ostrovsky, R.: Extended-DDH and lossy trapdoor functions. In: Fischlin, M., Buchmann, J., Manulis, M. (eds.) PKC 2012. LNCS, vol. 7293, pp. 627–643. Springer, Heidelberg (May 2012). https://doi.org/10.1007/978-3-642-30057-8˙37
18. Kiltz, E., O'Neill, A., Smith, A.: Instantiability of RSA-OAEP under chosen-plaintext attack. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 295–313. Springer, Heidelberg (Aug 2010). https://doi.org/10.1007/978-3-642-14623-7˙16
19. Libert, B., Sakzad, A., Stehlé, D., Steinfeld, R.: All-but-many lossy trapdoor functions and selective opening chosen-ciphertext security from LWE. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part III. LNCS, vol. 10403, pp. 332–364. Springer, Heidelberg (Aug 2017). https://doi.org/10.1007/978-3-319-63697-9˙12
20. Maurer, U.M.: Abstract models of computation in cryptography (invited paper). In: Smart, N.P. (ed.) 10th IMA International Conference on Cryptography and Coding. LNCS, vol. 3796, pp. 1–12. Springer, Heidelberg (Dec 2005)
21. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (Aug 2008). https://doi.org/10.1007/978-3-540-85174-5˙31
22. Peikert, C., Waters, B.: Lossy trapdoor functions and their applications. In: Ladner, R.E., Dwork, C. (eds.) 40th ACM STOC. pp. 187–196. ACM Press (May 2008). https://doi.org/10.1145/1374376.1374406
23. Pietrzak, K., Rosen, A., Segev, G.: Lossy functions do not amplify well. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 458–475. Springer, Heidelberg (Mar 2012). https://doi.org/10.1007/978-3-642-28914-9˙26

24. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. J. ACM **27**(4), 701–717 (oct 1980). https://doi.org/10.1145/322217.322225, https://doi.org/10.1145/322217.322225
25. Shoup, V.: Lower bounds for discrete logarithms and related problems. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 256–266. Springer, Heidelberg (May 1997). https://doi.org/10.1007/3-540-69053-0˙18
26. Zhandry, M.: To label, or not to label (in generic groups). In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part III. LNCS, vol. 13509, pp. 66–96. Springer, Heidelberg (Aug 2022). https://doi.org/10.1007/978-3-031-15982-4˙3
27. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: Ng, E.W. (ed.) Symbolic and Algebraic Computation. pp. 216–226. Springer Berlin Heidelberg, Berlin, Heidelberg (1979)