Reducing the Number of Qubits in Solving LWE

Barbara Jiabao Benedikt

Cryptoplexity, Technische Universität Darmstadt, Darmstadt, Germany
barbara_jiabao.benedikt@tu-darmstadt.de

Abstract. At Crypto 2021, May presented an algorithm solving the ternary Learning-With-Error problem, where the solution is a ternary vector $s \in \{0, \pm 1\}^n$ with a known number of $(+1)$ and (-1) entries. This attack significantly improved the time complexity of $\mathcal{S}^{0.5}$ from previously known algorithms to $\mathcal{S}^{0.25}$, where $\mathcal{S}$ is the size of the key space. Therefore, May exploited that using more representations, i.e., allowing ternary interim results with additional $(+1)$ and (-1) entries, reduces the overall time complexity.
Later, van Hoof et al. (PQCrypto 2021) combined May's algorithm with quantum walks to a new attack that performs in time $\mathcal{S}^{0.19}$. However, this quantum attack requires an exponential amount of qubits. This work investigates whether the ternary LWE problem can also be solved using only $\mathcal{O}(n)$ qubits. Therefore, we look closely into Dicke states, which are an equal superposition over all binary vectors with a fixed Hamming weight. Generalizing Dicke states to ternary vectors makes these states applicable to the ternary LWE problem.
Bärtschi and Eidenbenz (FCT 2019) proposed a quantum circuit to prepare binary Dicke states deterministically in linear time $\mathcal{O}(n)$. Their procedure benefits from the inductive structure of Dicke states, i.e., that a Dicke state of a particular dimension can be built from Dicke states of lower dimensions. Our work proves that this inductive structure is also present in generalized Dicke states with an underlying set other than $\{0, 1\}^n$. Utilizing this structure, we introduce a new algorithm that deterministically prepares generalized Dicke states in linear time, for which we also provide an implementation in Qiskit.
Finally, we apply our generalized Dicke states to the ternary LWE problem, and construct an algorithm that requires $\mathcal{O}(n)$ qubits and classical memory space up to $\mathcal{S}^{0.22}$. We achieve $\mathcal{S}^{0.379}$ as best obtainable time complexity.

Keywords: LWE, SSP, quantum attack, Dicke state, CNS algorithm

1 Introduction

Regev [42] introduced the Learning-With-Error (LWE) problem, which consists of a matrix $A \in \mathbb{Z}_q^{m \times n}$ and a vector of the form $b \coloneqq As - e \in \mathbb{Z}_q^m$, where $q = \text{poly}(n)$ is an integer, and asks for the unknown vectors $s \in \mathbb{Z}_q^n$ and $e \in \mathbb{Z}_q^m$. The vector e is the error vector typically sampled from a discrete Gaussian distribution (centered around 0) and has a small max-norm. Thus, the LWE

problem asks for a solution s of a system of noisy and linear equations (and $e = As - b$ can be easily computed when the solution s is found).

Moreover, Regev proposed a public-key encryption scheme using LWE with (A, b) as the public and (s, e) as the private key. Several works [7,42,39] proved the scheme secure under presumably hard lattice problems. In 2017, the post-quantum secure digital signature scheme CRYSTALS-Dilithium [22] was submitted to the NIST PQC competition and raised the attention of Regev's problem because Dilithium's security is based on LWE. This scheme is one of the reasons why studies about the hardness of LWE are so crucial, especially since NIST selected CRYSTALS-Dilithium in 2022 to become one of the post-quantum cryptographic signature standards [33].

Like May [35], we restrict ourselves to quadratic matrices $A \in \mathbb{Z}_q^{n \times n}$ and *ternary* secret keys $s, e \in \mathcal{T}^n \coloneqq \{0, \pm 1\}^n$, because these parameters are typically used in cryptosystems due to several significant advantages, including more compact keys. Also, the key generation becomes simpler and more efficient using discrete Gaussian sampling only for ternary vectors since in general Gaussian sampling is complicated and error-prone. Moreover, ternary keys allow defining encryption systems that always decrypt correctly—e.g, NTRU [13] and NTRU Prime [4,5]. These are two of the round-3 finalists in the NIST PQC competition, which—although they did not become the new standard—are still used in practice (e.g., NTRU Prime in OpenSSH [37]). Beside that, there are several signature schemes based on LWE which use ternary keys, e.g., GLP [25], BLISS [20,21], and GLYPH [17].

The Meet-in-the-Middle (MitM) attack on ternary LWE, which we consider in this work, goes back to Odlyzko's (fully classical) MitM attack [27], in which the adversary tries to extract the secret key of an NTRU system. Cheon et al. [14] have shown that it can easily adapted to the LWE problem. This MitM attack noticeably improves the asymptotic time complexity from $\mathcal{S}^{0.5}$ (Odlyzko) to $\mathcal{S}^{0.24}$ (Kirshanova and May [35,31]), where $\mathcal{S}$ is the size of the key space.

Furthermore, van Hoof et al. [45] showed how to use a quantum computer to successfully reduce the time complexity of the MitM attack to $\mathcal{S}^{0.19}$. This quantum algorithm tackles the LWE problem with quantum walks [2] in the *QRAQM* model, i.e., quantum memory with quantum random access. However, since these quantum walks require an exponential amount of qubits, we are motivated to consider the problem in other memory models with a limited number of qubits. Van Hoof et al. [45] also presented an algorithm requiring polynomial-sized classical memory and $\mathcal{O}(n)$ qubits that have a time complexity in the range of $[\mathcal{S}^{0.510}, \mathcal{S}^{0.558}]$.

Our work pursue another approach [26] and propose a new algorithm that works with exponential-sized classical space and $\mathcal{O}(n)$ qubits. This memory-setting has not been considered for the ternary LWE problem so far. Therefore, we combine May's classical algorithm [35] with the algorithm of Chailloux et al. [11] (CNS algorithm) which applies Amplitude Amplification [8] as a generalization of Grover's search [24] in a clever way. This algorithm needs adjustments for our new attack, which requires the preparation of "generalized" Dicke states.

1.1 Generalizing and Preparing Dicke States

As a family of highly entangled states, the so-called Dicke states [19] are well known in quantum physics, but these states also receive attention in other areas, e.g., quantum game theory [38], quantum adiabatic evolution [16], and quantum cryptography [15,40]. Dicke States are defined as an equal superposition over binary vectors $x \in \{0,1\}^n$ with a fixed but arbitrary *Hamming weight*, i.e., the number of non-zero entries $\mathsf{wt}(x) \coloneqq \sum_{x_i \neq 0} 1$.

Different solutions exist for preparing *binary* Dicke states, including probabilistic algorithms. Kaye et al. [30] proposed an algorithm with success probability at least $1-\varepsilon$, and Childs et al. [16] described an algorithm with success probability $\Omega(1/\sqrt{n})$. Both algorithms require a circuit of depth and size superlinear in n. Improving over this, Bärtschi and Eidenbenz [3,10] showed how *binary* Dicke states can be prepared deterministically in linear time (see Fig. 1). Later on, their algorithm was experimentally evaluated and optimized in terms of CNOT gate counts [36,1].

Nepomechie and Raveh [41] considered *non-binary* (or *d-ary*, $d \geq 2$) Dicke states in the qu*d*it-setting, $d \geq 1$, i.e., in comparison to *bits* we consider *dits* which can take any value between 0 and $d-1$. Unfortunately, qudits (or *qutrits*, which we need for ternary vectors) are much harder to control in a quantum device and to scale for increasing d. On the one hand, there are researchers [32,43,34] working on making qudits practical, including Ringbauer et al. [43] who recently proposed a quantum processor with trapped ions that supports qudits with $d = 8$ levels. On the other hand, current efforts are mostly concentrated on qubit-based systems (e.g., Google, IBM, Atom Computing). Therefore, it is advantageous to have a solution that uses qubits. Nevertheless, in the current literature, no algorithms prepare these generalized Dicke states based on qubits.

	Bärtschi and Eidenbenz [3]	**[this work]**
Circuit Depth	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Circuit Size	$\mathcal{O}(nk)$	$\mathcal{O}(nk)$
Applicability	binary Dicke states	d-ary Dicke states, $d \geq 2$

Fig. 1: Comparison of the Bärtschi and Eidenbenz (BE) circuit with our new algorithm, where n is the vector length and k is the Hamming weight.

Our first contribution. This paper shows—as its first contribution—how to generalize Dicke states to non-binary vectors in the qubit setting and how to prepare these states on a quantum computer in linear time. Additionally, we provide an implementation in Qiskit, which generates a quantum circuit to prepare generalized Dicke states [1]. An example of a circuit generating 4-ary Dicke states with $n = k = 4$ in `Quirk` [23] can be found *here*.

[1] Accessible via `https://github.com/bj-benedikt/Generalized-Dicke-States`.

1.2 Applying Generalized Dicke States

Having generalized Dicke states and a circuit to prepare them introduced, the second part of the paper focuses on two selected applications—the Subset-Sum problem (SSP) and the ternary LWE problem. However, these states are valuable for solving many different combinatorial problems, and could become an essential tool in cryptography. E.g., Perriello et al. [40] and Chevignard et al. [15] showed that *binary* Dicke states are useful in code-based cryptography.

Our second contribution. We prove that ternary Dicke states are helpful in solving the SSP. In particular, we find a missing piece in quantum algorithms of [26,6] and complete them without changing their time and memory complexities.

The SSP is defined as follows: Find a solution $s \in \{0,1\}^n$ with Hamming weight $\mathsf{wt}(s) = \frac{n}{2}$ for a given vector $a \in \mathbb{Z}_{2^n}^n$ and a target $t \in \mathbb{Z}_{2^n}$ such that $\langle a, s\rangle = t \mod 2^n$. Note that a and t are well-defined in that at least one solution exists. A common way to solve the SSP is via a MitM attack, i.e., define the solution as $s \coloneqq s_1 + s_2$, rewrite the equation as

$$\langle a, s_1\rangle = t - \langle a, s_2\rangle \mod 2^n \tag{1}$$

and search for (s_1, s_2) such that both sides of the equation *meet-in-the-middle*.

One can use the CNS algorithm [11] to find such a pair quantumly. The tricky part of solving SSP using Amplitude Amplification [8] is to come up with a good definition of the domain for both sides of equation 1. One way to do so, is to define the domains of s_1 and s_2 as binary vectors of length n with Hamming weight $\frac{n}{4}$ and then search for a solution $s \in \{0,1\}^n$ with $\mathsf{wt}(s) = \frac{n}{2}$. The algorithm then takes the equal superposition of the domain, which is a binary Dicke state, as the initial state.

The algorithms presented in [26,6] solve the SSP by using *richer* forms of representation, reducing the time complexity in the process. Howgrave-Graham and Joux first introduced this so-called representation technique [28]. Using this technique, we search for ternary interim result vectors via Amplitude Amplification. However, the algorithm requires a ternary Dicke state as its initial state. Unfortunately, they [26,6] leave how to set up this initial state as an open question. Furthermore, they assume that this initial state can be set up with $\mathcal{O}(n)$ qubits and in time, which is negligible for the overall analysis. Our algorithm to prepare generalized Dicke states finally proves their assumption and completes the algorithm in [26,6].

Our third contribution. We present an algorithm that solves ternary LWE with $\mathcal{O}(n)$ qubits using ternary Dicke states. Our algorithm combines May's algorithm [35] with quantum techniques for the SSP [26,6], and uses our quantum circuit for the preparation of a ternary Dicke state.

The resulting algorithm is faster than van Hoof et al.'s memory-efficient version [45] (range $[\mathcal{S}^{0.510}, \mathcal{S}^{0.558}]$) and has $\mathcal{S}^{0.379}$ as the best obtainable time complexity. Accordingly, it is less efficient than van Hoof et al.'s [45] quantum ($\mathcal{S}^{0.19}$) and May's [35] classical ($\mathcal{S}^{0.24}$) MitM attack (see Fig. 2). Nevertheless, our new algorithm is the fastest possible attack in a setting, where only $\mathcal{O}(n)$ qubits and classical memory up to $\mathcal{S}^{0.22}$ are available. This is because our algorithm is the only algorithm suitable for this memory requirement, closing a significant gap in currently available algorithms.

Furthermore, our algorithm allows the adversary to configure the trade-off of classical memory and computational time. This kind of flexibility can not be found for classical algorithms and can only be achieved with $\mathcal{O}(n)$ qubits by our new algorithm. Note that we do not provide an implementation of our algorithm since there is none of May's classical attack.

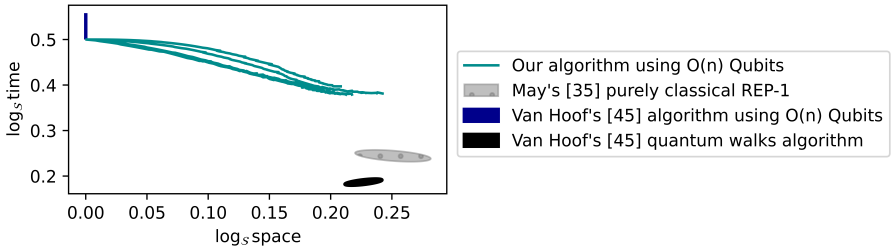


Fig. 2: Time-memory trade-off of our algorithm compared to May's purely classical MitM attack [35] and van Hoof et al.'s memory-efficient variant [45] which uses $\mathcal{O}(n)$ qubits. Obviously, our algorithm is the best option for classical memory requirements in the range of $[\mathcal{S}^0, \mathcal{S}^{0.22}]$.

2 Generalizing Dicke States

We start with the generalization of Dicke states to non-binary vectors in the qubit-setting—commonly these states are defined as follows:

Definition 1 (Dicke States)**.** *A* ***Dicke state*** $|\mathcal{D}_\ell^n\rangle$*,* $\ell \leq n$*, is the equal superposition of all* n*-qubit states* $|x\rangle$ *with Hamming weight* $\mathsf{wt}(x) = \ell$*, i.e.,*

$$|\mathcal{D}_\ell^n\rangle = \binom{n}{\ell}^{-1/2} \sum_{\substack{x \in \{0,1\}^n, \\ \mathsf{wt}(x)=\ell}} |x\rangle \,.$$

If we want to lift this definition to the non-binary setting, we need to take care of two things. First, the number of qubits of a vector component must be increased. For considering ternary Dicke states ($x \in \{0,1,2\}^n$) we require $q = \lceil \log_2 3 \rceil = 2$ qubits per component and $n \cdot q$ qubits to represent an n-dimensional ternary vector. In general, for d-ary Dicke states we need $\lceil \log_2 d \rceil$ qubits per component and $n \cdot \lceil \log_2 d \rceil$ qubits for n-dimensional d-ary vectors.

Second, we need to calculate the new amplitude. For binary Dicke states, the amplitude is $\binom{n}{\ell}^{-1/2}$, where $\binom{n}{\ell}$ is the number of n-dimensional binary vectors with Hamming weight ℓ. In the non-binary setting, the number of all possibilities can be computed with multinomial coefficients (instead of binomial coefficients):

Definition 2 (Multinomial Coefficients)**.** *For positive integers $n, n_1, \ldots, n_k \in \mathbb{N}$, $k \in \mathbb{N}$, such that $n = n_1 + \ldots + n_k$, the* ***multinomial coefficient*** *is*

$$\binom{n}{n_1, \ldots, n_k} \coloneqq \binom{n}{n_1} \cdot \binom{n-n_1}{n_2} \cdot \ldots \cdot \binom{n-\sum_{i<k} n_i}{n_k}$$

and corresponds to the number of combinations of assigning n distinct objects to k distinct buckets such that bucket i contains n_i objects. As a shorthand notation, we write $\binom{n}{n_1, \ldots, n_{k-1}, \cdot}$, where $\cdot$ represents the missing argument $n - \sum_i n_i$.

Having multinomial coefficients introduced, we define generalized *non-binary* Dicke states based on qubits (note for $q = 1$, our new definition coincides with Def. 1):

Definition 3 (Generalized (Qubit-)Dicke States)**.** *A* ***generalized Dicke state*** *$|\mathcal{D}^{q \cdot n}_{\ell_1, \ldots, \ell_{2^q-1}}\rangle$, $\sum_{i \geq 1} \ell_i \leq n$, is the equal superposition of all $n \cdot q$-qubit states $|x\rangle = |x_1\rangle \ldots |x_n\rangle$, where each of the registers $|x_j\rangle$ consists of q qubits and ℓ_i of them are equal to $|i\rangle \coloneqq |i_q\rangle$, $i \in \{0, \ldots, 2^q - 1\}$ with $(\cdot)_q$ as the binary representation in q bits, i.e.,*

$$|\mathcal{D}^{q \cdot n}_{\ell_1, \ldots, \ell_{2^q-1}}\rangle = \binom{n}{\ell_1, \ldots, \ell_{2^q-1}, \cdot}^{-1/2} \sum_{\substack{x \in \{0, \ldots, 2^q-1\}^n, \\ \ell_i \text{ entries equal to } i}} |x\rangle \,.$$

For simplicity, we denote $\mathsf{wt} \coloneqq \sum_{i \geq 1} \ell_i$ as the number of non-zero entries.

Bärtschi and Eidenbenz' [3] algorithm prepares binary Dicke states, by exploiting the fact that these states have an inductive structure. However, it turns out that the generalized Dicke states preserve the inductive structure for $q > 1$, as it is proven in the following lemma. For $q = 1$, this lemma corresponds to Lemma 1 in [3].

Lemma 4. *Generalized Dicke states have the inductive sum form*

$$\begin{aligned} |\mathcal{D}^{q \cdot n}_{\ell_1, \ldots, \ell_{2^q-1}}\rangle = \sqrt{\frac{\ell_0}{n}} \, |\mathcal{D}^{q \cdot (n-1)}_{\ell_1, \ldots, \ell_{2^q-1}}\rangle \otimes |0\rangle + \sqrt{\frac{\ell_{j_1}}{n}} \, |\mathcal{D}^{q \cdot (n-1)}_{\ldots, \ell_{j_1}-1, \ldots}\rangle \otimes |j_1\rangle \\ + \ldots + \sqrt{\frac{\ell_{j_m}}{n}} \, |\mathcal{D}^{q \cdot (n-1)}_{\ldots, \ell_{j_m}-1, \ldots}\rangle \otimes |j_m\rangle \,, \end{aligned}$$

where $\mathsf{wt} \leq n$, $\ell_0 \coloneqq n - \mathsf{wt}$, $\{j_1, \ldots, j_m\} \subset \{1, \ldots, 2^q - 1\}$, $m \in \mathbb{N}_0$, denotes the indices with $\ell_{j_i} \neq 0$, $i \in \{1, \ldots, m\}$, in ascending order. Note that the equation above only applies for $\ell_0, m > 0$. Otherwise, the first resp. the last part vanishes.

Proof. First assume $\ell_0, m > 0$. We then use Definition 3 of generalized Dicke states to rewrite (with $j_0 := 0$):

$$\begin{aligned}
|\mathcal{D}^{q\cdot n}_{\ell_1,\ldots,\ell_{2^q-1}}\rangle &= \tbinom{n}{\ell_1,\ldots,\ell_{2^q-1},\cdot}^{-1/2} \sum_{\substack{x\in\{0,\ldots,2^q-1\}^n,\\ \ell_k \text{ entries equal to } k}} |x\rangle \\
&= \tbinom{n}{\ell_1,\ldots,\ell_{2^q-1},\cdot}^{-1/2} \sum_{i=0}^{m} \sum_{\substack{x\in\{0,\ldots,2^q-1\}^{n-1},\\ \ell_k \text{ entries equal to } k \text{ for } k\neq j_i,\\ (\ell_{j_i}-1) \text{ entries equal to } j_i}} |x\rangle\otimes|j_i\rangle \\
&= \sqrt{\frac{\binom{n-1}{\ell_1,\ldots,\ell_{2^q-1},\cdot}}{\binom{n}{\ell_1,\ldots,\ell_{2^q-1},\cdot}}}\, |\mathcal{D}^{q\cdot(n-1)}_{\ell_1,\ldots,\ell_{2^q-1}}\rangle\otimes|0\rangle + \sum_{i=1}^{m}\sqrt{\frac{\binom{n-1}{\ldots,\ell_{j_i}-1,\ldots,\cdot}}{\binom{n}{\ell_1,\ldots,\ell_{2^q-1},\cdot}}}\, |\mathcal{D}^{q\cdot(n-1)}_{\ldots,\ell_{j_i}-1,\ldots}\rangle\otimes|j_i\rangle\,.
\end{aligned}$$

If $\ell_0 = 0$, the first term vanishes; if $m = 0$, the sum has no summands. The claim follows in both cases since

$$\begin{aligned}
\frac{\binom{n-1}{\ell_1,\ldots,\ell_{2^q-1},\ell_0-1}}{\binom{n}{\ell_1,\ldots,\ell_{2^q-1},\ell_0}} &= \frac{\binom{n-1}{\ell_1}\binom{n-1-\ell_1}{\ell_2}\cdots\binom{n-1-\ell_1-\cdots-\ell_{2^q-2}}{\ell_{2^q-1}}\binom{\ell_0-1}{\ell_0-1}}{\binom{n}{\ell_1}\binom{n-\ell_1}{\ell_2}\cdots\binom{n-\ell_1-\cdots-\ell_{2^q-2}}{\ell_{2^q-1}}\binom{\ell_0}{\ell_0}} \\
&= \frac{(n-1)!}{\ell_1!\cdot\ldots\cdot\ell_{2^q-1}!\cdot(n-1-\mathsf{wt})!}\cdot\frac{\ell_1!\cdot\ldots\cdot\ell_{2^q-1}!\cdot(n-\mathsf{wt})!}{n!} \\
&= \frac{n-\mathsf{wt}}{n} = \frac{\ell_0}{n}
\end{aligned}$$

and

$$\begin{aligned}
\frac{\binom{n-1}{\ldots,\ell_j-1,\ldots,\ell_0}}{\binom{n}{\ell_1,\ldots,\ell_{2^q-1},\ell_0}} &= \frac{\binom{n-1}{\ell_1}\cdots\binom{n-1-\ell_1-\ldots-\ell_{j-1}}{\ell_j-1}\binom{n-1-\ell_1-\ldots-(\ell_j-1)}{\ell_{j+1}}\cdots\binom{n-\ell_1-\ldots-\ell_{2^q-2}}{\ell_{2^q-1}}}{\binom{n}{\ell_1}\cdots\binom{n-1-\ell_1-\ldots-\ell_{j-1}}{\ell_j}\binom{n-\ell_1-\ldots-\ell_j}{\ell_{j+1}}\cdots\binom{n-\ell_1-\cdots-\ell_{2^q-2}}{\ell_{2^q-1}}} \\
&= \frac{(n-1)!(n-1-\ell_1-\ldots-(\ell_j-1))!}{\ell_1!\cdot\ell_2!\cdot\ldots\cdot(\ell_j-1)!}\cdot\frac{\ell_1!\cdot\ell_2!\cdot\ldots\cdot\ell_j!}{n!(n-\ell_1-\ldots-\ell_j)!} = \frac{\ell_j}{n}.
\end{aligned}$$

□

3 Preparing Generalized Dicke States

Aiming for a new quantum circuit which prepares generalized Dicke states for $q \geq 1$, we start with defining the desired unitaries, which prepare these generalized Dicke states for a sorted initial state. Hereby, we follow the notation of Bärtschi and Eidenbenz [3] but with a slight modification that we use $|*\rangle$ as a placeholder for ancilla qubits. These ancilla qubits $|*\rangle$ may change after evaluations of unitaries, even if the symbol $|*\rangle$ does not.

Definition 5. *We denote any unitary satisfying the following equation for all $\ell_1,\ldots,\ell_{2^q-1}$ with $\mathsf{wt} \coloneqq \sum_{i\geq 1}\ell_i \leq k$ by $U_{q\cdot n,k}$, $k \leq n$:*

$$U_{q\cdot n,k}\,|0\rangle^{\otimes\ell_0}\,|1\rangle^{\otimes\ell_1}\ldots|2^q-1\rangle^{\otimes\ell_{2^q-1}}\,|*\rangle = |\mathcal{D}^{q\cdot n}_{\ell_1,\ldots,\ell_{2^q-1}}\rangle\,|*\rangle\,,$$

where $\ell_0 \coloneqq n - \mathsf{wt}$ is defined as the number of zero entries.

Given the inductive structure of generalized Dicke states (see Lemma 4), it follows that iterating the two steps—splitting the superposition into $m+1$ parts, and shifting a register $|j_i\rangle$ for each $i \in \{0,\dots,m\}$ to the rightmost position—prepares these states.

In the notation of [3] we call a unitary implementing this transformation a *Split & Cyclic Shift* (SCS) unitary and define it as follows:

Definition 6. *We denote any unitary satisfying the following equation for all* $\ell_1,\dots,\ell_{2^q-1}$ *with* $\mathsf{wt} \le k$, $\ell_0 := n - \mathsf{wt}$ *and* $\{j_1,\dots,j_m\} \subset \{1,\dots,2^q-1\}$, $m \in \mathbb{N}_0$, *the indices with* $\ell_{j_i} \neq 0$, $i \in \{1,\dots,m\}$, *in ascending order by* $SCS_{q\cdot n,k}$, $k \le n$*:*

$$\begin{aligned} SCS_{q\cdot n,k} : |0\rangle^{\otimes \ell_0} |j_1\rangle^{\otimes \ell_{j_1}} |j_2\rangle^{\otimes \ell_{j_2}} \dots |j_m\rangle^{\otimes \ell_{j_m}} |*\rangle \\ \mapsto \sqrt{\frac{\ell_{j_1}}{n}} |0\rangle^{\otimes \ell_0} |j_1\rangle^{\otimes \ell_{j_1}-1} |j_2\rangle^{\otimes \ell_{j_2}} \dots |j_m\rangle^{\otimes \ell_{j_m}} |j_1\rangle |*\rangle \\ + \dots + \sqrt{\frac{\ell_{j_m}}{n}} |0\rangle^{\otimes \ell_0} |j_1\rangle^{\otimes \ell_{j_1}} |j_2\rangle^{\otimes \ell_{j_2}} \dots |j_m\rangle^{\otimes \ell_{j_m}} |*\rangle \\ + \sqrt{\frac{\ell_0}{n}} |0\rangle^{\otimes \ell_0-1} |j_1\rangle^{\otimes \ell_{j_1}} |j_2\rangle^{\otimes \ell_{j_2}} \dots |j_m\rangle^{\otimes \ell_{j_m}} |0\rangle |*\rangle . \end{aligned}$$

Note that the last term only occurs if $\ell_0 > 0$.

The unitary $SCS_{q\cdot n,k}$ can be designed to only act on the last $q\cdot(k+1)$ qubits, because this part—due to the sortedness of the input registers—contains each of the registers $|0\rangle$ and $|j_i\rangle$, $i \in \{1,\dots,m\}$, at least once.

3.1 Construction of $U_{q\cdot n,k}$

Following the approach of Bärtschi and Eidenbenz [3], we use the SCS unitaries to construct $U_{q\cdot n,k}$ inductively:

Lemma 7. *The following construction of* $U_{q\cdot n,k}$, $k \le n$, *satisfies Definition 5:*

$$U_{q\cdot n,k} := \prod_{\ell=2}^{k} (SCS_{q\cdot \ell,(\ell-1)} \otimes Id^{\otimes q\cdot(n-\ell)}) \cdot \prod_{\ell=k+1}^{n} (Id^{\otimes q\cdot(\ell-k-1)} \otimes SCS_{q\cdot \ell,k} \otimes Id^{\otimes q\cdot(n-\ell)}).$$

Proof. We show by induction over n that for all $\ell_1,\dots,\ell_{2^q-1}$ with $\mathsf{wt} \le k$ we have

$$U_{q\cdot n,k} |0\rangle^{\otimes \ell_0} |j_1\rangle^{\otimes \ell_{j_1}} \dots |j_m\rangle^{\otimes \ell_{j_m}} = |\mathcal{D}^{q\cdot n}_{\ell_1,\dots,\ell_{2^q-1}}\rangle ,$$

where $\ell_0 := n - \mathsf{wt}$ and $\{j_1,\dots,j_m\} \subset \{1,\dots,2^q-1\}$, $m \in \mathbb{N}_0$, are the indices with $\ell_{j_i} \neq 0$, $i \in \{1,\dots,m\}$, in ascending order.

Initial Case: The claim is obvious for $n = 1$ (and $k = 1$). For $n = 2$ the unitary is $U_{q\cdot 2,2} = SCS_{q\cdot 2,1}$ for $k = 2$ and $U_{q\cdot 2,1} = SCS_{q\cdot 2,1}$ for $k = 1$. For $i,j \in \{0,\dots,2^q-1\}$ with $i \le j$ we get

$$SCS_{q\cdot 2,1} |i\rangle |j\rangle \mapsto \begin{cases} \frac{1}{\sqrt{2}}(|i\rangle |j\rangle + |j\rangle |i\rangle), & \text{if } i \neq j, \\ \sqrt{\frac{2}{2}} |i\rangle |j\rangle , & \text{otherwise.} \end{cases}$$

Hypothesis: $U_{q\cdot(n-1),k}$ fulfills Def. 5 for arbitrary, but fixed, $n,k\in\mathbb{N}$, $k\leq n$.

Step: **First Case:** $U_{q\cdot(n-1),k}\to U_{q\cdot n,k}$ if $k<n$.
Assuming $U_{q\cdot(n-1),k}$ fulfills the hypothesis, we show in two steps that

$$U_{q\cdot n,k}\left|0\right\rangle^{\otimes\ell_0}\left|j_1\right\rangle^{\otimes\ell_{j_1}}\ldots\left|j_m\right\rangle^{\otimes\ell_{j_m}}=\left|\mathcal{D}^{q\cdot n}_{\ell_1,\ldots,\ell_{2^q-1}}\right\rangle$$

for all $\ell_1,\ldots,\ell_{2^q-1}$ with $\mathsf{wt}\leq k<n$. First, by Def. 6 we have

$$\begin{aligned}
&(\mathrm{Id}^{\otimes q\cdot(n-k-1)}\otimes SCS_{q\cdot n,k})\left|0\right\rangle^{\otimes\ell_0}\left|j_1\right\rangle^{\otimes\ell_{j_1}}\ldots\left|j_m\right\rangle^{\otimes\ell_{j_m}}\\
&\quad=\left|0\right\rangle^{\otimes n-k-1}\otimes\Bigg(\sqrt{\frac{\ell_{j_1}}{n}}\left|0\right\rangle^{\otimes\ell_0-(n-k-1)}\left|j_1\right\rangle^{\otimes\ell_{j_1}-1}\ldots\left|j_m\right\rangle^{\otimes\ell_{j_m}}\left|j_1\right\rangle\\
&\qquad+\ldots+\sqrt{\frac{\ell_{j_m}}{n}}\left|0\right\rangle^{\otimes\ell_0-(n-k-1)}\left|j_1\right\rangle^{\otimes\ell_{j_1}}\ldots\left|j_m\right\rangle^{\otimes\ell_{j_m}}\\
&\qquad+\sqrt{\frac{\ell_0}{n}}\left|0\right\rangle^{\otimes\ell_0-(n-k-1)-1}\left|j_1\right\rangle^{\otimes\ell_{j_1}}\ldots\left|j_m\right\rangle^{\otimes\ell_{j_m}}\left|0\right\rangle\Bigg)=:\left|\mathrm{tmp}\right\rangle.
\end{aligned}$$

By induction hypothesis and Lemma 4:

$$\begin{aligned}
&(U_{q\cdot(n-1),k}\otimes\mathrm{Id}^{\otimes q})\left|\mathrm{tmp}\right\rangle=\sqrt{\frac{\ell_{j_1}}{n}}\left|\mathcal{D}^{q\cdot(n-1)}_{\ldots,\ell_{j_1}-1,\ldots}\right\rangle\left|j_1\right\rangle+\ldots\\
&\quad+\sqrt{\frac{\ell_{j_m}}{n}}\left|\mathcal{D}^{q\cdot(n-1)}_{\ldots,\ell_{j_m}-1,\ldots}\right\rangle\left|j_m\right\rangle+\sqrt{\frac{\ell_0}{n}}\left|\mathcal{D}^{q\cdot(n-1)}_{\ell_1,\ldots,\ell_{2^q-1}}\right\rangle\left|0\right\rangle=\left|\mathcal{D}^{q\cdot n}_{\ell_1,\ldots,\ell_{2^q-1}}\right\rangle.
\end{aligned}$$

Second Case: $U_{q\cdot(k-1),(k-1)}\to U_{q\cdot k,k}$.
Assuming $U_{q\cdot(k-1),(k-1)}$ fulfills the hypothesis, we show that

$$U_{q\cdot k,k}\left|0\right\rangle^{\otimes\ell_0}\left|j_1\right\rangle^{\otimes\ell_{j_1}}\ldots\left|j_m\right\rangle^{\otimes\ell_{j_m}}=\left|\mathcal{D}^{q\cdot k}_{\ell_1,\ldots,\ell_{2^q-1}}\right\rangle$$

for all $\ell_1,\ldots,\ell_{2^q-1}$ with $\mathsf{wt}\leq k$. This follows by hypothesis and Lemma 4:

$$\begin{aligned}
&(U_{q\cdot(k-1),(k-1)}\otimes\mathrm{Id}^{\otimes q})\cdot SCS_{q\cdot k,(k-1)}\left|0\right\rangle^{\otimes\ell_0}\left|j_1\right\rangle^{\otimes\ell_{j_1}}\ldots\left|j_m\right\rangle^{\otimes\ell_{j_m}}\\
&\quad=(U_{q\cdot(k-1),(k-1)}\otimes\mathrm{Id}^{\otimes q})\Bigg(\sum_{i=1}^{m}\sqrt{\tfrac{\ell_{j_i}}{n}}\left|0\right\rangle^{\otimes\ell_0}\ldots\left|j_i\right\rangle^{\otimes\ell_{j_i}-1}\ldots\left|j_m\right\rangle^{\otimes\ell_{j_m}}\left|j_i\right\rangle\\
&\qquad+\sqrt{\tfrac{\ell_0}{n}}\left|0\right\rangle^{\otimes\ell_0-1}\left|j_1\right\rangle^{\otimes\ell_{j_1}}\left|j_2\right\rangle^{\otimes\ell_{j_2}}\ldots\left|j_m\right\rangle^{\otimes\ell_{j_m}}\left|0\right\rangle\Bigg)=\left|\mathcal{D}^{q\cdot k}_{\ell_1,\ldots,\ell_{2^q-1}}\right\rangle.
\end{aligned}$$

□

Hence, we get the following inductive formulas

$$U_{q\cdot n,k}=\begin{cases}\mathrm{Id}^{\otimes q}, & \text{if } n=k=1,\\ (U_{q\cdot(k-1),(k-1)}\otimes\mathrm{Id}^{\otimes q})\cdot SCS_{q\cdot k,(k-1)}, & \text{if } n=k>1,\\ (U_{q\cdot(n-1),k}\otimes\mathrm{Id}^{\otimes q})\cdot(\mathrm{Id}^{\otimes q\cdot(n-k-1)}\otimes SCS_{q\cdot n,k}), & \text{if } n>k.\end{cases}$$

3.2 Construction of $SCS_{q\cdot n,k}$

According to Lemma 7, the preparation of generalized Dicke states only requires a construction of the SCS unitaries. Unfortunately, we can not generalize the construction by Bärtschi and Eidenbenz [3], BE-construction, of $SCS_{1\cdot n,k}$, $k < n$, can not be generalized easily. This has two main reasons. To explain them, we revisit the BE-construction which makes use of the following gate:

Definition 8 (Rotation Gate)**.** *By* $\boxed{\sqrt{i/n}}$*,* $i \in \mathbb{N}$*, we denote the* Y*-rotation*

$$R_y(2\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

around angle $\theta \coloneqq \cos^{-1}\sqrt{\frac{i}{n}}$*. This unitary maps* $|0\rangle \mapsto \sqrt{\frac{i}{n}}\,|0\rangle + \sqrt{\frac{n-i}{n}}\,|1\rangle$*.*

The BE-construction proceeds as follows (see Fig. 3): nothing happens if the *last* qubit (at position n) is $|0\rangle$ because then all other qubits are $|0\rangle$ as well since the vector is sorted and the controlled condition of the XORs and rotation $\boxed{\sqrt{1/n}}$ is false.

Fig. 3: Building blocks of the BE-construction for preparing binary Dicke states.

Otherwise, the algorithm searches the qubits above until it finds the first qubit equal to $|0\rangle$. Assume this happens after ℓ qubits. Then, the controlled condition of the XORs is false, whereas the condition of the rotation $\boxed{\sqrt{\ell/n}}$ is fulfilled. This rotation $\boxed{\sqrt{\ell/n}}$ together with the following controlled-XOR swap the last qubit (which is equal to $|1\rangle$) with the first qubit equal to $|0\rangle$ (at position $n-\ell$) with the correct amplitude ($\sqrt{\ell/n}$). Since the vector was sorted initially, the qubits at position 1 to $n-\ell-1$ are equal to $|0\rangle$, and the vector stays sorted at position 1 to $n-1$ after the swap.

Two problems occur if we use the BE construction for $q > 1$.

First Problem: We change nothing if the last register is $|0\rangle$. So assume the last register is $|i\rangle \neq |0\rangle$. Similar to the procedure above, we would search the registers until we find a register $\neq |i\rangle$. Assume this happens after ℓ registers. In case the next register is equal to $|0\rangle$, it is as easy as in case $q = 1$ because we know that the remaining $n-\ell$ registers are equal to $|0\rangle$ and we know the correct amplitude. Then:

$$SCS_{q\cdot n,\ell}:|0\rangle^{\otimes n-\ell}\,|i\rangle^{\otimes \ell} \mapsto \sqrt{\tfrac{n-\ell}{n}}\,|0\rangle^{\otimes n-\ell-1}\,|i\rangle^{\otimes \ell}\,|0\rangle + \sqrt{\tfrac{\ell}{n}}\,|0\rangle^{\otimes n-\ell}\,|i\rangle^{\otimes \ell}\,.$$

In case the first register $\neq |i\rangle$ is equal to $|j\rangle \neq |0\rangle$ finding the correct amplitude is more complicated since this value depends on ℓ_j, the number of occurrences of $|j\rangle$ in the complete vector, i.e.,

$$SCS_{q\cdot n,\ell+\ell_j+\dots} : |\dots\rangle |j\rangle^{\otimes \ell_j} |i\rangle^{\otimes \ell} \mapsto \sqrt{\tfrac{\ell_j}{n}} |\dots\rangle |j\rangle^{\otimes \ell_j - 1} |i\rangle^{\otimes \ell} |j\rangle + \dots$$

This means we need to search more registers to learn ℓ_j and save the first register's position equal to $|j\rangle$ (at position $n-\ell$) for the swap. The new vector is no longer sorted if we do not save the position and swap with another register equal to $|j\rangle$. Furthermore, if the next register $\neq |j\rangle$ is non-zero, we face the same problem again.

Second Problem: Our target state is a superposition of vectors, where each occurring value is once at the last position of the vector, and everything else is unchanged. Therefore, we must ensure that no value is missed and that the upper $n-1$ qubits remain sorted. This requires tracking the parts we swapped and the corresponding amplitudes. This new challenge comes with the fact that there may be more than one swap for each SCS unitary.

To summarize, it is more challenging to construct SCS unitaries for $q > 1$ since we need to learn all the number of occurrences $\ell_1, \dots, \ell_{2^q-1}$ and remember all swaps. We solve these issues by swapping all $k+1$ entries and using ancilla qubits to manage the swaps. For illustration purpose, we consider the following example.

Example 9. *Consider the initial state $|x\rangle := |01123\rangle$ for preparing the generalized Dicke state $|\mathcal{D}^{2\cdot 5}_{\ell_1=2,\ell_2=1,\ell_3=1}\rangle$. By the execution of $SCS_{2\cdot 5,4}|x\rangle$ we target—after measurement of possibly used ancilla qubits—the state:*

$$\sqrt{\tfrac{1}{5}} |1123\rangle |0\rangle + \sqrt{\tfrac{2}{5}} |0123\rangle |1\rangle + \sqrt{\tfrac{1}{5}} |0113\rangle |2\rangle + \sqrt{\tfrac{1}{5}} |0112\rangle |3\rangle .$$

We prepare this target state by swapping the last qubit with the second-last qubit, then the last with the third-last, and so on, until we swap with a qubit equal to $|0\rangle$. All these swaps happen with an amplitude, which we will specify in the next step.

Note that we only swap in the part where we swapped previously. Otherwise, there is no guarantee that the state remains sorted, e.g., when we swap the last and the third-last qubit of $|x\rangle$:

$$|01123\rangle \xrightarrow{swap} \sqrt{1-p}\,|0112\rangle |3\rangle + \sqrt{p}\,\underbrace{|0132\rangle}_{unsorted} |1\rangle .$$

This procedure transforms the initial state $|x\rangle$ as follows (we denote the probability of swap i by p_i):

$$\begin{aligned} |01123\rangle |*\rangle &\xrightarrow{swap} \sqrt{1-p_1}\,\underbrace{|0112\rangle |3\rangle}_{no\ swap} |*\rangle + \sqrt{p_1}\,\underbrace{|0113\rangle |2\rangle}_{swap} |*\rangle \\ &\xrightarrow{swap} \sqrt{1-p_1}\,|0112\rangle |3\rangle |*\rangle + \sqrt{p_1}\left(\sqrt{1-p_2}\,\underbrace{|0113\rangle |2\rangle}_{no\ swap} |*\rangle + \sqrt{p_2}\,\underbrace{|0123\rangle |1\rangle}_{swap} |*\rangle\right) \end{aligned}$$

$$\begin{aligned}
&\xrightarrow{swap} \sqrt{1-p_1}\,|0112\rangle\,|3\rangle\,|*\rangle + \sqrt{p_1(1-p_2)}\,|0113\rangle\,|2\rangle\,|*\rangle \\
&\qquad + \sqrt{p_1p_2}\left(\sqrt{1-p_3}\,\underbrace{|0123\rangle\,|1\rangle}_{no\ swap}\,|*\rangle + \sqrt{p_3}\,\underbrace{|0123\rangle\,|1\rangle}_{swap}\,|*\rangle\right) \\
&\xrightarrow{swap} \sqrt{1-p_1}\,|0112\rangle\,|3\rangle\,|*\rangle + \sqrt{p_1(1-p_2)}\,|0113\rangle\,|2\rangle\,|*\rangle + \sqrt{p_1p_2(1-p_3)}\,|0123\rangle\,|1\rangle\,|*\rangle \\
&\qquad + \sqrt{p_1p_2p_3}\left(\sqrt{(1-p_4)}\,|00123\rangle\,|1\rangle\,|*\rangle + \sqrt{p_4}\,|01123\rangle\,|0\rangle\,|*\rangle\right).
\end{aligned}$$

The ancilla register $|*\rangle$ is needed for managing the swaps, especially regarding their probabilities. For each probability, one ancilla qubit is needed, so k qubits in total for $SCS_{q\cdot n,k}$.

Note that the ancilla register is changed after a swap, even if we write the ancilla register as $|*\rangle$ again. Therefore we can not summarize any of the terms since their ancilla register are in different states. Moreover, these ancilla registers are entangled with the other registers, so reusing them among the SCS unitaries is impossible. Therefore we measure these ancilla qubits as soon as we do not need them anymore in order to free resources, which brings us to the state:

$$\begin{aligned}
&\sqrt{p_1p_2p_3p_4}\,|01123\rangle\,|0\rangle + \sqrt{p_1(1-p_2)}\,|0113\rangle\,|2\rangle + \sqrt{1-p_1}\,|0112\rangle\,|3\rangle \\
&\qquad + \underbrace{\sqrt{(\sqrt{p_1p_2(1-p_3)})^2 + (\sqrt{p_1p_2p_3(1-p_4)})^2}}_{=\sqrt{p_1p_2(1-p_3p_4)}}\,|0123\rangle\,|1\rangle.
\end{aligned}$$

A comparison with the target state implies $p_1 = 4/5$, $p_2 = 3/4$ and $p_3p_4 = 1/3$. The same calculation for the initial state $|01233\rangle$ gives us $p_3 = 2/3$ and $p_4 = 1/2$. In general, we can compute the probabilities with the formula $p_i = (n-1-i)/(n-i)$, $i \in \{1,\dots,n-1\}$.

Generalizing the Example 9 gives us the algorithm `PrepareAncAndSwap` (see Fig. 4) which constructs the SCS unitaries.

Theorem 10. *`PrepareAncAndSwap` prepares any generalized Dicke state $|\mathcal{D}^{q\cdot n}_{\ell_1,\dots,\ell_{2^q-1}}\rangle$.*

Proof sketch. Due to Lemma 7 it is sufficient to prove that any $SCS_{q\cdot n,k}$ can be constructed by `PrepareAncAndSwap`. Such a circuit is shown in Fig. 5. The correctness of the implementation can be shown by a calculation similar to Example 9 for two reasons.

- The swaps have the correct amplitude since each $\texttt{anc}[i]$, $i \in \{1,\dots,k\}$, is equal to $|0\rangle$ with probability $p_1 \cdot p_2 \cdot \ldots \cdot p_i$.
- We only swapped in the part of the state that was swapped previously. If $\texttt{anc}[i] = 1$ we neither swap in step i - according to line 4 and 7 - nor in step $i+1$ since - due to the construction in line 6 and 7 - it is ensured that $\texttt{anc}[i : i+1] = 11$.

□

Remark 11. *Since we swap every component, we can weaken Definition 5 of the unitary $U_{q\cdot n,k}$—we prepare a generalized Dicke state $|\mathcal{D}^{q\cdot n}_{\ell_1,\dots,\ell_{2^q-1}}\rangle$ if the initial state is not sorted but has some structure:*

`PrepareAncAndSwap`$(n, k, q, \mathtt{qr}, \mathtt{anc})$:

1 : Require: `anc` contains k qubits initialized as $|0\rangle$
2 : Require: `qr` contains $k+1$ registers with q qubits each
3 : Apply $\boxed{\sqrt{(n-1)/n}}$ on `anc`$[1]$
4 : Swap `qr`$[k+1]$ and `qr`$[k]$ controlled by (`anc`$[1] == 0$)
5 : **for** $i \in \{2, \ldots, k\}$:
6 : Apply NOT-gate on `anc`$[i]$ controlled by (`anc`$[i-1] == 1$)
7 : Apply $\boxed{\sqrt{(n-1-i)/(n-i)}}$ on `anc`$[i]$ controlled by (`anc`$[i-1] == 0$)
8 : Swap `qr`$[k+1]$ and `qr`$[k+1-i]$ controlled by (`anc`$[i] == 0$)
9 : Measure `anc`$[i-1]$ and, **if** $(i == k)$, measure `anc`$[k]$

Fig. 4: Algorithm to prepare the generalized Dicke States. See also our Qiskit implementation which can be found in the supplementary material.

Let be $c \in \{0, \ldots, 2^q - 1\}$ and ℓ_c the occurrence of value c. The unitary $U_{q\cdot n, n-\ell_c}$ prepares $|\mathcal{D}^{q\cdot n}_{\ell_1,\ldots,\ell_{2^q-1}}\rangle$ if the non-c entries are at the end of the initial state and separated from the c entries which are at the beginning, i.e., $\mathtt{qr}[i] = |c\rangle$ for all $i \in \{1, \ldots, \ell_c\}$ and $\mathtt{qr}[i] \neq |c\rangle$ for all $i \in \{\ell_c + 1, \ldots, n\}$.

Theorem 12. *Generalized Dicke states $|\mathcal{D}^{q\cdot n}_{\ell_1,\ldots,\ell_{2^q-1}}\rangle$ can be prepared with a circuit of size $\mathcal{O}(\min\{k, n-\ell_1, \ldots, n-\ell_{2^q-1}\} \cdot n)$ and depth $\mathcal{O}(n)$ using at most $(k+1)$ ancilla qubits at the same time.*

Proof. **Circuit size:** We start the proof by showing that the circuit for preparing the generalized Dicke state $|\mathcal{D}^{q\cdot n}_{\ell_1,\ldots,\ell_{2^q-1}}\rangle$ has size $\mathcal{O}(k \cdot n)$ where $k = \sum_{i\geq 1} \ell_i$. Then it immediately follows from Rem. 11 that $|\mathcal{D}^{q\cdot n}_{\ell_1,\ldots,\ell_{2^q-1}}\rangle$ can be prepared using a circuit of size $\mathcal{O}(\min\{k, n-\ell_1, \ldots, n-\ell_{2^q-1}\} \cdot n)$.

To determine the circuit size, we need to count the required gates to construct $U_{q\cdot n,k}$ as in Theorem 10. The unitary $U_{q\cdot n,k}$ consists of the gates $SCS_{q\cdot i,k}$ with $i = k+1, \ldots, n$ and $SCS_{q\cdot(i+1),i}$ with $i = 1, \ldots, k-1$. In general, the unitary $SCS_{q\cdot(\cdot),k}$ contains $(k+1)$ controlled-swaps, each of which can be implemented with one CCNOT, two CNOT gates, and k ancilla qubits which in turn requires one rotation, $(k-1)$ CNOT gates, and $(k-1)$ controlled-rotations.

Hence, for the implementation of $U_{q\cdot n,k}$ we need $\mathcal{O}(k \cdot n)$ CCNOTs and $\mathcal{O}(k \cdot n)$ CNOTs, as well as $(n-1)$ rotations, $\mathcal{O}(k \cdot n)$ CNOTs, and $\mathcal{O}(k \cdot n)$ controlled-rotations for the ancilla qubits.

Circuit depth: Note that the structure of the swaps (line 4 and 8) of $SCS_{q\cdot n,k}$ is shaped like a stair of width k and height $(k+1)$, where the bottom line interacts with the bottom qubit. These stairs from different SCS unitaries can be "pushed"

together, such that the combined stairs of $SCS_{q\cdot n,k}$ and $SCS_{q\cdot(n-1),k}$ have a width of $(k+2)$, whereas $(k+1)$ is the width of the combined stairs of $SCS_{q\cdot(k+1),k}$ and $SCS_{q\cdot k,(k-1)}$. Hence, the union of all stairs has a width of $(2n-3)$. The ancilla qubits in `anc` are measured at the latest when all swaps are completed. Accordingly, the circuit has a linear depth of $(2n-2)$.

Ancilla qubits: The $SCS_{q\cdot n,k}$ unitary requires k qubits so that the overall number of needed ancilla qubit of $U_{q\cdot n,k}$ is $(n-k)\cdot k+\sum_{i=1}^{k-1} i = \mathcal{O}(k\cdot n)$. However, we can prepare them successively and measure them as soon as they are no longer needed (line 9). Each ancilla qubit controls a swap and is needed to prepare the *(if required) next* ancilla qubit. When both are done, the ancilla qubit can be measured. Accordingly, each SCS unitary hold at most two ancilla qubits. Since we push the SCS stairs together where each of the stairs has at most height $(k+1)$, we consider at most $\lceil \frac{k}{2} \rceil$ stairs simultaneously at any time. Hence, at most $2\cdot\lceil\frac{k}{2}\rceil \leq k+1$ ancilla qubits are required simultaneously. □

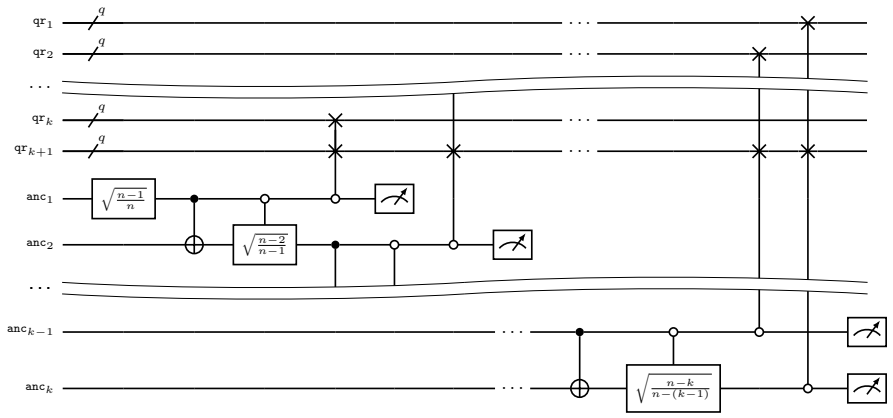


Fig. 5: Circuit for $SCS_{q\cdot n,k}$ applied on the registers $\texttt{qr}[1\ldots k+1]$ and $\texttt{anc}[1\ldots k]$. The wave bar symbolizes the registers in between. An example with $q=2$ and $n=k=4$ in `Quirk` [23] can be found *here*.

4 Reducing the Number of Qubits in LWE

Having Thm. 12 which prepares generalized Dicke states, we finally focus on applications. First, our new circuit can be used to solve SSP, and hereby completes the quantum attacks on SSP in [26,6]. To explain this we revisit the MitM attack on the SSP which searches for a pair (s_1, s_2) such that $s \coloneqq s_1 + s_2$ is a solution of $\langle a, s\rangle = t \mod 2^n$ for given a and t and has Hamming weight $\mathsf{wt}(s) = \frac{n}{2}$.

This identity allows us to rewrite the equation to $\langle a, s_1\rangle = t - \langle a, s_2\rangle \mod 2^n$, so that we successfully transform the SSP to a problem of finding *claws* of functions $f_1(s) := \langle a, s\rangle \mod 2^n$ and $f_2(s) := t - \langle a, s\rangle \mod 2^n$, i.e., finding a pair (s_1, s_2) such that $f_1(s_1) = f_2(s_2)$. This claw search is done via the CNS algorithm which was introduced by Chailloux et al. [11] as an alternative to the BHT algorithm [9] with a reduced number of needed qubits to find collisions.

Let $\mathcal{S}_1, \mathcal{S}_2$ be the domain of f_1 and f_2 respectively. Essentially both algorithms generate a list $L = \{(s, f_1(s)) | s \in \mathcal{S}^*\}$, where $\mathcal{S}^* \subset \mathcal{S}_1$, and use Amplitude Amplification [8] to find a *L-suitable* element $s_2 \in \mathcal{S}_2$, i.e., there exists $s_1 \in \mathcal{S}_1$ such that $(s_1, *) \in L$ and (s_1, s_2) is a claw.

The BHT algorithm stores the list L in qubits, whereas the CNS algorithm realizes it classically and makes classical queries to access list L. According to these differences, both algorithms have varying time and (classical / quantum) memory requirements. The CNS algorithm saves qubits—only $\mathcal{O}(n)$ (vs. $\mathcal{O}\big(2^{n/3}\big)$) qubits are needed—at the cost of time ($\mathcal{O}\big(2^{2n/5}\big)$ vs. $\mathcal{O}\big(2^{n/3}\big)$) and classical memory ($\mathcal{O}\big(2^{n/5}\big)$ vs. none). Nonetheless, it is more efficient than purely classical algorithms with a time complexity of $\mathcal{O}\big(2^{n/2}\big)$. This shows that even the usage of a few qubits is rewarding. However, the CNS algorithm for collisions can be generalized straight-forwardly to an algorithm for claws [26]:

Algorithm 13 (CNS Algorithm [11])**.** *Let be $f_i : \mathcal{S}_i \to \{0,1\}^n$, $|\mathcal{S}_i| \leq 2^n$ and $i \in \{1, 2\}$, two random and efficiently computable (i.e., $T_{f_i} = poly(n)$) functions, and $\mathcal{C} \coloneqq \{(s_1, s_2) \in \mathcal{S}_1 \times \mathcal{S}_2 | f_1(s_1) = f_2(s_2)\}$ the set of claws of size $R \coloneqq |\mathcal{C}|$. Given parameters r,l the next steps eventually output a claw $(s_1, s_2) \in \mathcal{C}$:*

(1) *Generate a sorted list $L \coloneqq \{(s_1, f_1(s_1)) | \pi_r(f_1(s_1)) = 0\}$ containing 2^l candidates for s_1, where $\pi_r(\cdot)$ denotes the projection on the first r components. Let T_1 be this step's time complexity, which requires $\mathcal{O}\big(2^l\big)$ classical memory.*

(2) *Setup $\mathcal{A}$: Prepare an equal superposition of $\mathcal{S}_2$ (in time $T_{\mathcal{S}_2}$ using $M_{\mathcal{S}_2}$ qubits). Apply Grover's search [24] on the resulting state to create an equal superposition of $\mathcal{S}_2^* \coloneqq \{s_2 \in \mathcal{S}_2 | \pi_r(f_2(s_2) = 0)\}$.*
This step uses $\mathcal{O}(\log_2 |\mathcal{S}_2|) + M_{\mathcal{S}_2} = \mathcal{O}(n) + M_{\mathcal{S}_2}$ qubits and can be done in $T_{\mathsf{setup}} = \mathcal{O}\left(T_{f_2} \cdot \sqrt{\mathbb{P}_{s_2 \in \mathcal{S}_2}\big[s_2 \in \mathcal{S}_2^*\big]}^{-1}\right) + T_{\mathcal{S}_2}$.

(3) *Set-Membership Oracle O_{f_L}: The quantum unitary of $f_L : \mathcal{S}_2 \to \{0,1\}$ which outputs 1 for L-suitable elements $s_2 \in \mathcal{S}_2$ can be implemented using $\mathcal{O}(n)$ qubits within time $T_{f_L} = \mathcal{O}\big(T_{f_L} \cdot 2^l\big) = \mathcal{O}\big(n \cdot 2^l\big)$.*

(4) *Amplitude Amplification: Use Setup $\mathcal{A}$ (from (2)) and Oracle function O_{f_L} (from (3)). Then, the algorithm eventually outputs $s_2 \in \mathcal{S}_2$ with $f_L(s_2) = 1$ in* $T_2 = \mathcal{O}\Big((T_{\mathsf{setup}} + T_{\mathcal{S}_2} + T_{f_L}) \cdot \sqrt{\mathbb{P}_{s_2 \in \mathcal{S}_2^*}\big[f_L(s_2) = 1\big]}^{-1}\Big)$.

(5) *Search for $s_1 \in L$ such that $f_1(s_1) = f_2(s_2)$. This can be done in $\mathcal{O}(1)$ since L is sorted.*

We choose the parameters as follows:

- *r is the number of matched components ($\pi_r(f_1(s_1)) = \pi_r(f_2(s_2)) = 0$). We require $r \leq \log \mathcal{R}$ to ensure that there exists at least one (out of $\mathcal{R}$) claw $(s_1, s_2) \in \mathcal{C}$ with $\pi_r(f_1(s_1)) = 0$.*
- *l determines the list size $|L| = 2^l$. We require $2^l \geq |\mathcal{S}_1|/\mathcal{R}$ to ensure that there exists a L-suitable $s_2 \in \mathcal{S}_2^*$. The probability that there exists a $s_2 \in \mathcal{S}_2^*$ for a fixed $s_1 \in \mathcal{S}_1$ with $(s_1, s_2) \in \mathcal{C}$ is $\mathcal{R}/|\mathcal{S}_1|$, since the functions f_1, f_2 are assumed to be random.*

Thus, we get a total runtime of $\max\{T_1, T_2\}$ *using* $\mathcal{O}\big(2^l\big)$ *classical memory and* $\mathcal{O}(n) + M_{\mathcal{S}_2}$ *qubits.*

Step (4) uses Brassard et al.'s Amplitude Amplification [8] that is a generalization of Grover's search [24] which applies the concept of amplifying the amplitude of a particular subspace. In each iteration, the amplitude increases so that the algorithm outputs an element of this particular subspace with overwhelming probability by measurement after enough repetitions. The main difference in contrast to Grover's search [24] is that one may start with another search space than the entire set.

Originally, Amplitude Amplification is only stated for binary input functions $f : \{0,1\}^n \to \{0,1\}$. However, any (arbitrary but finite) set $\mathcal{S}$ can be embedded in a set of long enough binary strings so that the algorithm can be formulated more generally:

Algorithm 14 (Amplitude Amplification, [8])**.** *Let $f : \mathcal{S} \to \{0,1\}$ be a function and $\mathcal{X} \subset \mathcal{S}$ a subset (the search space). The algorithm has given:*

- *Setup $\mathcal{A}$: Produces an equal superposition of $\mathcal{X}$ in time T_{setup}.*
- *Oracle function O_f: The quantum unitary of function f with runtime T_f.*

Then the algorithm outputs element $s \in \mathcal{X}$ such that $f(s) = 1$ in time

$$T = \mathcal{O}\Big(\max\{T_{\mathsf{setup}}, T_f\} \cdot \mathbb{P}_{s \in \mathcal{X}}\big[f(s) = 1\big]^{-1/2}\Big).$$

Let us go back to Helm and May's algorithm [26]. In the notation of Alg. 13 they chose $f_1(s) := \langle a, s\rangle \mod 2^n$, $f_2(s) := t - \langle a, s\rangle \mod 2^n$, and

$$\mathcal{S}_1 = \mathcal{S}_2 = \big\{s \in \{0, \pm 1\}^n |\ s \text{ has } (\tfrac{n}{2} + a) \text{ many } (+1) \text{ and } (a) \text{ many } (-1) \text{ entries}\big\},$$

where a is the number of additional (-1) entries. Step (2) requires an equal superposition of $\mathcal{S}_2$. This state is the generalized Dicke state $|\mathcal{D}^{2\cdot n}_{\frac{n}{2}+a,a,0}\rangle$ where

we identify the underlying set $\{0,1,2,3\}^n$—as it is defined in Def. 3—with $\{0,1,-1,*\}^n$ ($*$ arbitrary). However, Helm and May [26] just assumed that this Dicke state can be prepared in negligible time compared to the other terms. Luckily this assumption turns out to be true since Thm. 12 implies $M_{\mathcal{S}_2} = \frac{n}{2}+1$ and $T_{\mathcal{S}_2} = \mathcal{O}(n)$.

4.1 From SSP to LWE

Let us consider a ternary LWE-based scheme, where the public key is a tuple $(A,b) \in \mathbb{Z}_q^{n\times n} \times \mathbb{Z}_q^n$, consisting of a matrix A and a vector b, such that $As = b+e \mod q$ for some secret vectors $s,e \in \mathcal{T}^n \coloneqq \{0,\pm 1\}^n$, where $q = \text{poly}(n)$ is an integer. Furthermore, we focus on the LWE problem with ternary LWE keys where the vector s has a fixed (Hamming) weight $w = \sum_{s_i \neq 0} 1$ and has as many $(+1)$ entries as (-1) entries. We denote

$$\mathcal{T}^n(w/2) \coloneqq \{s \in \mathcal{T}^n | s \text{ has } w/2 \text{ many } (+1) \text{ and } (-1) \text{ entries each}\}$$

as the set of n-dimensional ternary weight-w vectors.

The choice of n, q, and w varies from scheme to scheme. For q, we require a lower bound $q = \Omega(n)$ (as in all modern NTRU-type systems), which has the advantage that decryption errors are eliminated. For w, we define the relative weight $\omega \coloneqq w/n$. This parameter ω simplifies the comparison of different schemes. Security analyses of NTRU [13,5] yield an optimal relative weight in the range $\omega \in [1/3, 2/3]$. Inspired by NTRU and NTRU Prime, we will analyze our quantum algorithm using the values $\omega \in \{0.375, 0.441, 0.5, 0.621, 0.668\}$.

Multinomial coefficients (Def. 2) enable to determine the size of the key space $|\mathcal{T}^n(\omega n/2)| = \binom{n}{\omega n/2, \omega n/2, \cdot}$ for fixed n and ω. The following lemma can approximate this number, but therefore we first define the Shannon entropy [44]:

Definition 15 (Shannon Entropy)**.** *Let $c_1,\ldots,c_k \in \mathbb{R}_{>0}$, $k \in \mathbb{N}$, be positive real numbers with $\sum_{i=1}^k c_i = 1$. Then the Shannon entropy is defined as follows*

$$H(c_1,\ldots,c_k) \coloneqq -\sum_{i=1}^{k} c_i \cdot \log_2(c_i).$$

As a shorthand notation we write $H(c_1,\ldots,c_{k-1},\cdot)$ instead, where $(\cdot)$ represents the missing argument $1 - c_1 - \ldots - c_{k-1}$.

Lemma 16 (Lemma 2.2 in [18])**.** *Let $D = \{d_1,\ldots,d_k\} \subset \mathbb{Z}_q$ be a set of digits and $s \in \mathbb{Z}_q^n$ a vector having exactly $(c_i n)$ many (d_i) entries, where $\sum_{i=1}^k c_i = 1$ and $c_i > 0$ for all $i \in \{1,\ldots,k\}$. Then*

$$H(c_1,\ldots,c_k)\cdot n - \ln\binom{n+k-1}{k-1} \leq \ln\binom{n}{c_1 n,\ldots,c_k n} \leq H(c_1,\ldots,c_k)\cdot n.$$

In particular this implies $\binom{n}{c_1 n,\ldots,c_k n} = \Theta\big(2^{H(c_1,\ldots,c_{k-1},\cdot)n}\big)$.

Hence, there asymptotically are $2^{H(\omega/2,\omega/2,\cdot)n}$ many n-dimensional ternary weight-ωn vectors. Through we mostly consider multinomial coefficients with $k = 3$ and $c_1 = c_2$, we also write $H3(c)$, $c \in [0, 1/2)$, as a short form of $H(c, c, \cdot)$.

At Crypto 2021, May [35] proposed an algorithm that finds the correct key within the $\mathcal{S} = 2^{H3(\omega/2)n}$ possibilities with time complexity $\mathcal{S}^{0.25}$. Later on, van Hoof et al. [45] combine the REP-1 algorithm with quantum walks [2], which results in a very time-efficient quantum MitM attack with an asymptotic time complexity of $\mathcal{S}^{0.19}$. However, in regards to memory complexities the classical algorithm [35] needs an exponential-sized classical memory, whereas the quantum algorithm [45] requires, in addition to the same amount of classical space, also an exponential number of qubits.

Van Hoof et al. recognize this weak spot and introduce a memory-efficient quantum version that works in the regime of polynomial classical and quantum memory having a time complexity in the range of $[\mathcal{S}^{0.514}, \mathcal{S}^{0.558}]$ for $\omega \in [0.375, 0.668]$. However, so far, no one has considered the scenario of exponential-sized classical memory and polynomial number of qubits, which seems to be a very realistic setting for the coming decades. Classical memory becomes cheaper, and if we assume an adversary who is only restricted to the currently available resources, then she is practically only limited in the number of obtainable qubits (currently, the biggest quantum processor has 1,121 qubits [29]). So, this is precisely the setting for which we construct our algorithm which compensates for the lack of qubits by using additional classical memory.

Therefore we start to explain May's [35] (classical) algorithm, go over to revisit and adjust the CNS algorithm [11], and then present our new qubit efficient algorithm.

4.2 May's Classical Attack

The MitM attack on LWE keys is an algorithm that extracts the secret key s and e for given (A, b). Note that this task is equivalent to finding a vector $s \in \mathcal{T}^n$ such that (As, b) is *1-close*, i.e., a pair (As, b) such that $||(As - b) \mod q||_\infty = 1$. Given the key s, we easily compute the error vector $e = As - b \mod q$.

The essential idea of the MitM attack is to split the ternary key $s \in \mathcal{T}^n(w/2)$ into two parts $s = s_1 + s_2$ with $s_1, s_2 \in \mathcal{T}^n$ and search for 1-close pair $(As_1, b - As_2)$. This can be done by computing As_1 and $b - As_2$ for proper candidates of s_1, s_2. The algorithm succeeds when a match is found, where the corresponding values of s_1 and s_2 build a 1-close pair $(As_1, b - As_2)$.

Odlyzko [27] proposed to choose candidates for s_1, s_2 based on a locality-sensitive hash function. Indeed, this hash function still plays a role in May's attack [35] and is used to assign a binary hash label of As_1 resp. $b - As_2$ for all candidates s_1, s_2 under the so-called Odlyzko's hash function:

$$\begin{aligned} \ell : \mathbb{Z}_q^n &\to \{0, 1, \{0, 1\}\}^n, \\ x = (x_i) &\mapsto \left(\ell(x)_i = \begin{cases} 0, & \text{if } 0 \le x_i < \lfloor q/2 \rfloor - 1 \\ 1, & \text{if} \lfloor q/2 \rfloor \le x_i < q - 1 \\ \{0, 1\}, & \text{otherwise} \end{cases} \right). \end{aligned}$$

Essentially, this hash function maps each component to its most significant bit. The idea is to construct a function that maps 1-close pairs to the same hash value and distant pairs to different hash values. Since 1-close pairs can cause a label flip in Odlyzko's hash function, we assign both labels $\{0,1\}$ to the border values $\lfloor q/2 \rfloor - 1$ and $q-1$.

May [35] used the representation technique [28] in his attack in a similar way as for the SSP. In the so-called REP-0 representation, each non-zero entry of $s = s_1 + s_2 \in \mathcal{T}^n(w/2)$ is assigned to either s_1 or s_2. Since both search spaces of s_1 and s_2 should have the same size to balance the workload for s_1 and s_2, we split the non-zero entries equally between the two components and search for $s_1, s_2 \in \mathcal{T}^n(w/4)$.

In contrast, the REP-1 representation also allows each 0 to be presented as $0+0$ or $\pm 1 \mp 1$. Let a be the number and $\alpha := a/n$ the relative number of additional $(+1)$ and (-1) entries. Hence, we want to represent s by $s_1, s_2 \in \mathcal{T}^n(w/4+a)$. This increases the number of representations:

Proposition 17. *Let $s \in \mathcal{T}^n(w/2)$ be the secret key and a the number of additional $(+1)$ and (-1) entries, then there are*

$$\mathcal{R}^{(0)} := \binom{w/2}{w/4}^2 = \binom{w/2}{w/4, w/4}^2 \approx 2^{\omega n}$$

many REP-0 representations (with $s_1, s_2 \in \mathcal{T}^n(w/4)$) and

$$\mathcal{R}^{(1)} := \binom{w/2}{w/4}^2 \cdot \binom{n-w}{a, a, \cdot} = \binom{w/2}{w/4, w/4}^2 \cdot \binom{n-w}{\frac{\alpha}{1-\omega}\cdot(n-w), \frac{\alpha}{1-\omega}\cdot(n-w), \cdot} \approx 2^{(\omega + (1-\omega)\cdot H3(\frac{\alpha}{1-\omega}))\cdot n}$$

many REP-1 representations (with $s_1, s_2 \in \mathcal{T}^n(w/4+a)$). Hence, there are strictly more REP-1 than REP-0 representations.

Proof. Follows from Lemma 16. □

May's MitM attack [35] builds a binary search tree out of lists, where the left resp. the right subtree is used to find candidates for s_1 resp. s_2 (see Fig. 6). The algorithm starts by generating the leaf lists and constructing the tree bottom-up (from leaf level d to root level 0).

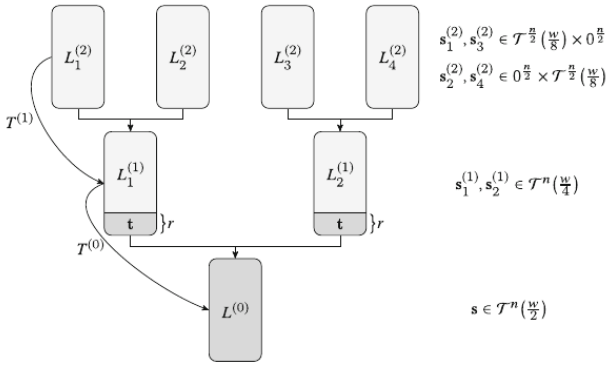


Fig. 6: This figure (from [35]) shows a REP-0 search tree. The darker areas indicate the matching components of list elements, e.g., each $s \in L_1^{(1)}$ fulfills $\pi_r(As + e_1) = t$.

In each level $d' \geq 2$ we merge two neighbored lists $L_i^{(d')}$ and $L_{i+1}^{(d')}$, where $i \in \{1, 3, \ldots, 2^{d'} - 1\}$ odd, by computing all possible sums $s^{(d'-1)} \coloneqq s_i^{(d')} + s_{i+1}^{(d')}$ and applying the so-called *Match-and-Filter approach*. We require that $As^{(d'-1)}$ (or $b - As^{(d'-1)}$) *matches* with a target vector $t \in \mathbb{Z}_q^{r^{(d'-1)}}$ on $r^{(d'-1)} \in \{1, \ldots, n\}$ components (see Fig. 6) and *filter* for the correct (Hamming) weights. The resulting list contains the remaining sums. This allows to consider only the residual $n - r^{(d'-1)}$ components in the next levels, because we have $\pi_{r^{(d'-1)}}(As^{(d'-1)}) = t$ (or $\pi_{r^{(d'-1)}}(b - As^{(d'-1)}) = t$) for all elements $s^{(d'-1)}$ on level $d' - 1$.

However, this matching condition is too strict since we only require 1-closeness in each component. May solves this issue by guessing the error vectors $e_1 \in \mathcal{T}^{r^{(d'-1)}/2} \times 0^{r^{(d'-1)}/2}$ and $e_2 \in 0^{r^{(d'-1)}/2} \times \mathcal{T}^{r^{(d'-1)}/2}$ on these components and using $\pi_{r^{(d'-1)}}(As_1 + e_1)$ (or $\pi_{r^{(d'-1)}}(b - As_2 + e_2)$) to match on t. May chooses multiple targets $t \leftarrow\!\$\, \mathbb{Z}_q^{r^{(d'-1)}}$, $d' \in \{2, \ldots, d\}$, uniformly at random for the different levels and subtrees, but indeed this makes the algorithm for search trees with depth $d > 2$ complex and incomprehensible. For the sake of simplicity, we use $t = 0$ at any point in the tree.

Assume we merged the lists on all levels $d' \geq 2$ and attained to level 1. Then, it only remains to merge both level-1 lists to get the root list which contains pairs $(s_1, s_2) \in (\mathcal{T}^n(w/4))^2$ such that $s = s_1 + s_2 \in \mathcal{T}^n(w/2)$ is the sought LWE key. For this step, we use Odlyzko's hash function.

First, we compute the *Odlyzko-label* $\ell(As^{(1)})$ (or $\ell(b - As^{(1)})$) for all level-1 elements and then compare only those sums, where both summands have *matching* labels. According to this procedure, we get pairs (s_1, s_2) such that $(As_1, b - As_2)$ is 1-close. Finally, we only need to *filter* for pairs, where the sum $s_1 + s_2$ has the correct weight and $A(s_1 + s_2) - b \mod q \in \mathcal{T}^n$ is ternary.

Note that projection $\pi_r(\cdot)$ and Odlyzko's hash function $\ell(\cdot)$, which can be seen as a projection to $\{0,1\}^n$ have a similar role through the attack. Both functions project elements to a smaller space, where *match-and-filter* is easier.

Algorithm 18 (REP-0 with $d = 2$, Thm. 1 in [35])**.** *Let $s \in \mathcal{T}^n(w/2)$ and $e \in \mathcal{T}^n$ be ternary LWE keys.*

(1) Construct the level-2 lists $L_i^{(2)} = \mathcal{T}^{n/2}(w/8) \times 0^{n/2}$ and $L_{i+1}^{(2)} = 0^{n/2} \times \mathcal{T}^{n/2}(w/8)$, $i = 1, 3$, of size

$$|\mathcal{T}^{n/2}(w/8)| = \binom{n/2}{w/8, w/8, \cdot} \approx 2^{H3(w/4)n/2}.$$

(2) Let be $r \coloneqq \lfloor \log_q(\mathcal{R}^{(0)}) \rfloor$. Then guess the error vectors $e_1 \in \mathcal{T}^{r/2} \times 0^{r/2}$ and $e_2 \in 0^{r/2} \times \mathcal{T}^{r/2}$, and construct the $L^{(1)} \subset \mathcal{T}^n(w/4) \times \mathbb{Z}_q^n$ lists

$$L_1^{(1)} = \{(s_1^{(1)} = s_1^{(2)} + s_2^{(2)}, As_1^{(1)}) | \pi_r(As_1^{(1)} + e_1) = 0 \mod q, s_1^{(2)} \in L_1^{(2)}, s_2^{(2)} \in L_2^{(2)}\},$$
$$L_2^{(1)} = \{(s_2^{(1)} = s_3^{(2)} + s_4^{(2)}, b - As_2^{(1)}) | \pi_r(b - As_2^{(1)} + e_2) = 0 \mod q, s_3^{(2)} \in L_3^{(2)}, s_4^{(2)} \in L_4^{(2)}\}.$$

By the choice of parameter r, we expect that these lists contain at least one (out of the $\mathcal{R}^{(0)}$) REP-0 representations (s_1, s_2) of s. Both lists have an

expected size of

$$|L^{(1)}| = \frac{|\mathcal{T}^n(w/4)|}{q^r} \geq \frac{|\mathcal{T}^n(w/4)|}{\mathcal{R}^{(0)}} = \binom{n}{w/4, w/4, \cdot} \binom{w/2}{w/4}^{-2} \approx 2^{(H3(\omega/4)-\omega)n}.$$

Then construct the list via the Match-and-Filter-Approach: search for all candidates $(s_1^{(2)}, s_2^{(2)})$ *matching to* 0 *on the first* r *coordinates and filter for the correct weight. Note that in this case, no filtering is required since all level-1 elements have the correct weight by construction of the level-2 lists.*

(3) Construct the list $L^{(0)}$ *with entries* $(s_1^{(1)}, s_2^{(1)}) \in L_1^{(1)} \times L_2^{(1)}$ *such that* $L^{(0)}$ *contains a LWE key* $s^{(0)} := s_1^{(1)} + s_2^{(1)}$.
Search for all pairs $(s_1^{(1)}, s_2^{(1)}) \in L_1^{(1)} \times L_2^{(1)}$ *with matching Odlyzko label and check whether these pairs additionally fulfill* $s^{(0)} := s_1^{(1)} + s_2^{(1)} \in \mathcal{T}^n(w/2)$ *and* $As^{(0)} - b \mod q \in \mathcal{T}^n$. *If yes, add* $s^{(0)}$ *to* $L^{(0)}$ *and return* $L^{(0)}$.

Following the steps (1) *to* (3)*, the secret key* s *can be found with asymptotic time and classical space complexity* $2^{\mathcal{O}(\max\{H3(\omega/4)/2, H3(\omega/4)-\omega\}n)}$, *where the first term in the exponent comes from* $|L^{(2)}|$ *and the second from* $|L^{(1)}|$.

Remark 19. *Notice that May's classical algorithm [35] demands possibly multiple runs until the error vectors* e_1, e_2 *are guessed correctly. For each* e_1 *resp.* e_2 *we can construct and store the corresponding list* $L_1^{(1)}$ *resp.* $L_2^{(1)}$, *so these lists only need to be computed once.*

Thus, we have an expected guessing complexity of

$$T_{guess} = 3^{r/2} = 2^{\mathcal{O}(n/\log n)},$$

since $q = \Omega(n)$ *implies* $r = \frac{\log_2 \mathcal{R}^{(0)}}{\log_2 q} = \mathcal{O}(n/\log n)$. *Nevertheless, the time complexity of Alg. 18 is fully dominated by* T_{list} *which is the complexity of constructing the lists* $L^{(2)}, L^{(1)}, L^{(0)}$ *and the guessing complexity* T_{guess} *can be neglect in the overall analysis.*

The needed (asymptotic) time and space is in a range of $T_{\text{list}} \in [\mathcal{S}^{0.371}, \mathcal{S}^{0.395}]$ for different choices of $\omega \in [0.375, 0.668]$, where $\mathcal{S} = |\mathcal{T}^n(w/2)| = 2^{H3(w/2)n}$ is the size of the key space. For $\omega \leq 0.578$ the time to construct $L^{(1)}$-lists is the dominating part, for $\omega > 0.578$ the to construct $L^{(2)}$-lists.

Towards REP-1. In comparison to REP-0, we consider depth-d search trees with $d \geq 2$ for REP-1. This enables to start with smaller lists on the lowest level and to use more additional (± 1) entries on the highest level (see Fig. 8), which reduces all in all the needed amount of time and space.

Level 0 contains the root list $L^{(0)}$, a subset of $\mathcal{T}^n(w/2)$. On level $i \in \{1, \ldots, d-1\}$, we have lists $L^{(i)} \subset \mathcal{T}^n(w^{(i)})$ with $(a^{(i-1)})$ additional (± 1) entries, where $w^{(i)} := w^{(i-1)}/2 + a^{(i-1)}$, for $i \geq 1$, $w^{(0)} := w/2$, and $\omega^{(i)} := w^{(i)}/n$ denotes the corresponding relative weight. The lists $L^{(d)}$ on the lowest level are of the form $\mathcal{T}^{n/2}(w^{(d-1)}/2) \times 0^{n/2}$ or vice versa.

Algorithm 20 (REP-1 [35]). *Let $s \in \mathcal{T}^n(w/2)$ and $e \in \mathcal{T}^n$ be ternary LWE key, $d \geq 2$ the depth of the search tree and $a^{(0)}, a^{(1)}, \ldots, a^{(d-2)}$ be the numbers of additional $(+1)$ and (-1) entries in the different levels.*

(1) Construct the level-d lists $L^{(d)}$ of size $|\mathcal{T}^{n/2}(w^{(d-1)}/2)| \approx 2^{H3(w^{(d-1)})n/2}$.
(2) Let be $r \coloneqq r^{(1)} \coloneqq \lfloor \log_q(\mathcal{R}^{(1)}) \rfloor$, where

$$\mathcal{R}^{(1)} = \binom{w^{(0)}}{w^{(0)}/2}^2 \cdot \binom{n-2w^{(0)}}{a^{(0)}, a^{(0)}, \cdot} \approx 2^{\left(2\omega^{(0)} + (1-2\omega^{(0)})H3(\frac{\alpha^{(0)}}{1-2\omega^{(0)}})\right)n}$$

is the number of REP-1 representation of $(s_1, s_2) \in L_1^{(1)} \times L_2^{(1)}$. Then guess the error vectors $e_1 \in \mathcal{T}^{r/2} \times 0^{r/2}$ and $e_2 \in 0^{r/2} \times \mathcal{T}^{r/2}$.
(3) Construct the level-i lists $L^{(i)}$ for $i = d-1, \ldots, 1$ via the Match-and-Filter-Approach: search for elements matching to 0 on the first $r^{(i)} \coloneqq \lfloor \log_q(\mathcal{R}^{(i)}) \rfloor$ coordinates, where $\mathcal{R}^{(i)} \approx 2^{\left(2\omega^{(i-1)} + (1-2\omega^{(i-1)})H3(\frac{\alpha^{(i-1)}}{1-2\omega^{(i-1)}})\right)n}$, and filter for the correct weight.
(4) Use Odlyzko's hash function to construct $L^{(0)}$.

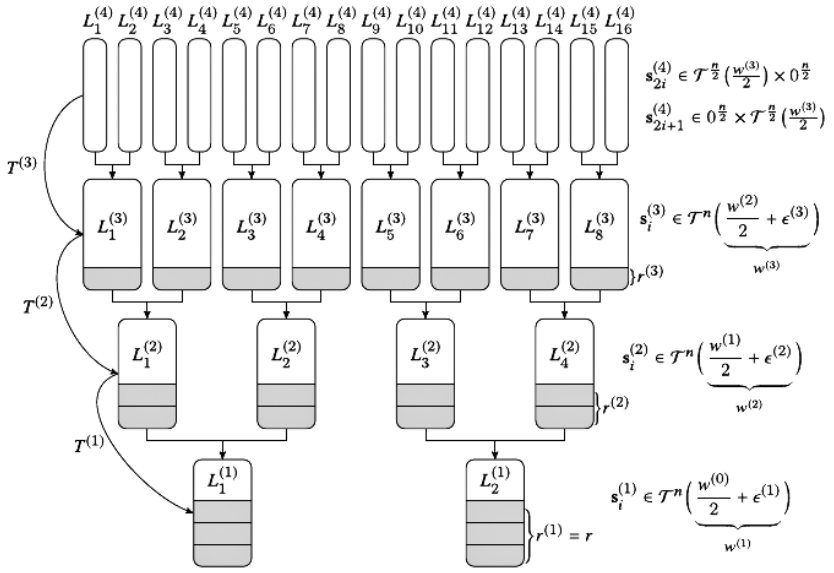


Fig. 7: This figure (from [35]) shows a REP-1 search tree with depth $d = 4$, and $\varepsilon^{(i)} \coloneqq a^{(i-1)}$. The darker areas indicate the matching components of list elements.

Time and Space Analysis of Alg. 20. Let be $i \in \{0, \ldots, d-1\}$, $r^{(0)} \coloneqq n$ and $r^{(4)} \coloneqq 0$. Since elements of $L^{(i+1)}$ already match on $r^{(i+1)}$ coordinates, we only consider the remaining $r^{(i)} - r^{(i+1)}$ coordinates for constructing the $L^{(i)}$ lists on the next level. This can be done in the expected time

$$T^{(i)} \coloneqq |L^{(i+1)}|^2 / q^{r^{(i)} - r^{(i+1)}}.$$

Note that the list size $|L^{(i)}| \leq T^{(i)}$ can be smaller since we need to filter for the pairs $(s_1^{(i+1)}, s_2^{(i+1)})$ with correct weight $s_1^{(i+1)} + s_2^{(i+1)} \in \mathcal{T}^n(w^{(i)})$. The expected size of these lists is

$$|L^{(i)}| = \frac{|\mathcal{T}^n(w^{(i)})|}{\mathcal{R}^{(i)}} \approx 2^{\left(H3(\omega^{(i)}) - 2\omega^{(i-1)} - (1-2\omega^{(i-1)})H3(\frac{\alpha^{(i-1)}}{1-2\omega^{(i-1)}})\right)n}.$$

All together, we find a solution in asymptotic time $T = \max\{T^{(0)}, \ldots, T^{(d-1)}\}$ with classical memory complexity $M = \max\{|L^{(1)}|, \ldots, |L^{(d)}|\}$. □

May [35] pointed out that (heuristically) $\alpha^{(3)} = \ldots = \alpha^{(d-2)} = 0$ holds for any $d \geq 5$ and $\omega \in [0.3, 0.668]$, and concluded that $d \geq 5$ does not give further improvements. Therefore, we only analyze the attack using depth-4 search trees.

In the analysis of REP-1 we list the optimizing parameter $\alpha^{(0)}, \alpha^{(1)}, \alpha^{(2)}$ by following two different strategies. On the one hand, we use these parameters to minimize the time complexity (first row in Fig. 8); on the other hand, to minimize the product of space and time complexity (second row in Fig. 8).

$\boldsymbol{\omega}$	$\log_{\mathcal{S}}$(Time)	$\log_{\mathcal{S}}$(Space)	$\alpha^{(0)}$	$\alpha^{(1)}$	$\alpha^{(2)}$
0.375	0.241	0.274	0.071	0.036	0.011
	0.245	0.234	0.054	0.025	0.007
0.441	0.235	0.288	0.084	0.042	0.015
	0.239	0.233	0.058	0.026	0.007
0.5	0.234	0.275	0.082	0.039	0.013
	0.239	0.231	0.058	0.026	0.007
0.621	0.246	0.249	0.073	0.029	0.007
	0.249	0.237	0.058	0.025	0.006
0.668	0.257	0.246	0.073	0.029	0.007
	0.259	0.242	0.058	0.025	0.006

Fig. 8: Asymptotics of REP-1, for $d = 4$. The parameters $\alpha^{(0)}, \alpha^{(1)}, \alpha^{(2)}$ are chosen to minimize the time (T) complexity in the first line and the space-time (ST) complexity in the second line.

In general, the MitM attack using REP-1 gains a time complexity in the range of $[\mathcal{S}^{0.234}, \mathcal{S}^{0.259}]$ independent of the chosen strategy for values $\omega \in [0.375, 0.668]$. This is a significant improvement compared to the best time of $\mathcal{S}^{0.371}$ we achieved with REP-0.

4.3 CNS Algorithm - Revisited and Adapted

Remember we introduced the CNS algorithm (Alg. 13) as an algorithm to find claws. However, the LWE problem is about solving a perturbed system, so the algorithm is required to find an *1-close claw* of the functions

$$f_1 : s_1 \mapsto As_1 \mod q \quad \text{and} \quad f_2 : s_2 \mapsto b - As_2 \mod q.$$

Then, at first sight, we have two technical issues—the functions f_1, f_2 have a non-binary range and are not random. However, this is not really a problem. It suffice to require that f_1, f_2 are quantum-accessible, and that there exists an 1-close claw (s_1, s_2) such that $\pi_r(f_1(s_1)) = 0$ for a properly chosen $r \in \mathbb{N}$. Then, the CNS algorithm can be applied because all needed applications of Amplitude Amplification are legitimate.

The first step is to redefine the set $\mathcal{C}$ such that it contains 1-close claws, i.e.,

$$\mathcal{C}_{LWE} \coloneqq \{(s_1, s_2) \in \mathcal{S}_1 \times \mathcal{S}_2 | (f_1(s_1), f_2(s_2)) \text{ is 1-close}\},$$

with domains (where n, q, w, a are chosen properly)

$$\mathcal{S}_1 = \mathcal{S}_2 = \begin{cases} \mathcal{T}^n(w/4), & \text{if we use REP-0,} \\ \mathcal{T}^n(w/4 + a), & \text{if we use REP-1.} \end{cases}$$

The second step is to ensure, that there exists a pair $(s_1, s_2) \in \mathcal{C}_{LWE}$ with $\pi_r(f_1(s_1)) = 0$. Matrix A and vector b are chosen as the public key, so by construction there exists the corresponding private key (s, e), and $\mathcal{C}_{\text{LWE}} \neq \emptyset$ is not empty. Furthermore all elements in the range have roughly the same number of preimages under the functions f_1, f_2 because of the randomness of matrix A, so we can assume that there is pair $(s_1, s_2) \in \mathcal{C}_{LWE}$ with $\pi_r(f_1(s_1)) = 0$—if r is chosen properly. The search requires $\mathcal{O}(n)$ qubits since $|\mathcal{S}_i| \leq 2^n$, $i = 1, 2$, $n \in \mathbb{N}$, and

$$\mathcal{O}(\log_2 |\mathcal{S}_1|) = \left\{ \begin{matrix} \mathcal{O}(H3(\omega/4)n) \\ \mathcal{O}(H3(\omega/4 + a)n) \end{matrix} \right. \begin{matrix} , \text{ if we use REP-0} \\ , \text{ if we use REP-1} \end{matrix} \Bigg\} = \mathcal{O}(n).$$

To ensure that we can apply the CNS algorithm to f_1, f_2 and $\mathcal{C}_{LWE}$, it only remains to define appropriate setup $\mathcal{A}$ and oracle O. For constructing $\mathcal{A}$, we use our circuit (Thm. 12) to prepare a ternary Dicke state since $\mathcal{S}_1$ is a ternary set with a fixed number of (± 1) entries. However, to build a concrete oracle O we first need to define the oracle function, which depends on the set of "interesting pairs" $\mathcal{C}$ (in our case $\mathcal{C}_{\text{LWE}}$). In any case, the function $f_{\mathcal{C}} : \mathcal{S}_2 \to \{0, 1\}$ is appropriate as oracle function which outputs 1 for L-suitable elements $s_2 \in \mathcal{S}_2$:

$$f_{\mathcal{C}}(s_2) = 1 \quad :\Longleftrightarrow \quad \exists (s_1, *) \in L : (s_1, s_2) \in \mathcal{C}.$$

Note that in the case of the claw finding problem, the function $f_{\mathcal{C}}$ is equivalent to the set-membership function (f_L) in Alg. 13. In our case, we have

$$\begin{aligned} (s_1, s_2) \in \mathcal{C}_{\text{LWE}} \quad &\Longleftrightarrow \quad (f_1(s_1), f_2(s_2)) \text{ is 1-close} \\ &\Longleftrightarrow \quad \exists e \in \mathcal{T}^n : f_1(s_1) = f_2(s_2) + e. \end{aligned}$$

So, we can define the function $f_{\mathcal{C}_{\text{LWE}}}$ as follows:

$$f_{\mathcal{C}_{\text{LWE}}}(s_2) = 1 \quad :\Longleftrightarrow \quad \exists e \in \mathcal{T}^n : (*, f_2(s_2) + e) \in L.$$

Thus, the new *1-closeness oracle* function $f_{\mathcal{C}_{\mathrm{LWE}}}$ is similar to the set-membership oracle with weaker interpretation of "membership"—small errors of (± 1) are allowed. Notice that the 1-closeness oracle has another advantage. In the classical setting, we depended on multiple runs of the attack until the error vector was guessed correctly (see Rem. 19). However, using $f_{\mathcal{C}_{\mathrm{LWE}}}$ finds e and s_2 simultaneously without extra effort.

Moreover, any approach in the quantum setting that involves guessing, would lead us to further problems. Having guessed wrongly, we will not find s_2, and the Amplitude Amplification will fail. Accordingly, we have an algorithm where the reason for failure is not unique—was it a wrong guess or a measurement error? This fact makes it difficult to decide how to react to failures.

Let us take a closer look into the set-membership oracle $\mathcal{O}_{f_L}$ of Alg. 13, and adapt it to our purpose for constructing a 1-closeness oracle $\mathcal{O}_{f_{\mathcal{C}_{\mathrm{LWE}}}}$ afterward. To construct O_{f_L} we use the following operations:

- *Creation:* Takes classical input $x \in \{0,1\}^n$ and n qubits initialized at $|0\rangle$. Then it outputs $|x\rangle$ in time n by constructing each qubit separately.
- *Deletion:* This is the inverse operation of Creation. It takes classical input x and $|x\rangle$. Then it outputs $|0\rangle$.

Algorithm 21 (Set-Membership Oracle O_{f_L}, [11])**.** *Let $L = \{x_i\}_{i=1}^{|L|} \subset \{0,1\}^n$ be a list and $f_L : \{0,1\}^n \to \{0,1\}$ be the function, which outputs $f_L(x) = 1$ iff' $x \in L$ is a member of list L. For computing $O_{f_L}(|x\rangle\,|b\rangle)$, $b \in \{0,1\}$, start from $|\phi_1\rangle \coloneqq |x\rangle\,|b\rangle$ on $n+1$ qubits. For $i = 1, \ldots, |L|$, do:*

(1) Get element x_i from L and construct $|x_i\rangle$ using the Creation *operator. Then concatenate $|x_i\rangle$ with the current state $|\phi_i\rangle$.*
(2) Apply:

$$|x_i\rangle\,|\phi_i\rangle = |x_i\rangle\,|x\rangle\,|b\rangle \mapsto |x_i\rangle \underbrace{|x\rangle\,|b \oplus (\delta_{x_i,x})\rangle}_{\coloneqq |\phi_{i+1}\rangle},$$

where δ is the Kronecker-Delta function, i.e., $\delta_{x_i,x} = 1$ iff' $x_i = x$.
(3) Discard the first register using the Deletion *operator and keep $|\phi_{i+1}\rangle$ as the remaining state.*

Output the final state $|\phi_{|L|+1}\rangle = |x\rangle\,|b \oplus f_L(x)\rangle = O_{f_L}(|x\rangle\,|b\rangle)$.

Time and Space Analysis. Step (1) is done in time n, and step (2) and (3) in constant time. Each $|x_i\rangle$ resp. $|\phi_i\rangle$, $i = 1, \ldots, |L|$, requires n resp. $n+1$ qubits. Since we use Creation and Deletion alternately, we have at most one of each register in parallel. So the oracle function O_{f_L} needs in total $2n+1$ qubits and performs in time $n \cdot |L|$. □

Note that the oracle O_{f_L} implements the function f_L perfectly because of

$$f_L(x) = \bigoplus_{i=1}^{|L|} \delta_{x_i,x}.$$

For each $i = 1, \ldots, |L|$ the oracle compares the register $|x_i\rangle$ with $|x\rangle$, what on the circuit-level means that this is done with each qubit of the registers separately. So $\delta_{x_i,x}$ can be written as $\bigwedge_{j=1}^{n} \delta_{x_i{}^{(j)},x^{(j)}}$, where $x_i^{(j)}$ resp. $x^{(j)}$ is the j^{th} component of the vector x_i resp. x. Note that this is very close to the construction we need for $O_{f_{\mathcal{C}_{\mathrm{LWE}}}}$—there we want to output 1, if the components match except for a small ± 1 error. This corresponds to the operation

$$\delta_{x_i,x}^{\pm 1} \coloneqq \bigwedge_{j=1}^{n} (\delta_{x_i{}^{(j)}-1,x^{(j)}} \vee \delta_{x_i{}^{(j)},x^{(j)}} \vee \delta_{x_i{}^{(j)}+1,x^{(j)}}).$$

Thus, we can write the 1-closeness oracle function as $f_{\mathcal{C}_{\mathrm{LWE}}}(x) = \bigoplus_{i=1}^{|L|} \delta_{x_i,x}^{\pm 1}$, that leads us to:

Algorithm 22 (1-closeness oracle $O_{f_{\mathcal{C}_{\mathrm{LWE}}}}$)**.** *Follow the steps from Alg. 21, but with $\delta_{x_i,x}^{\pm 1}$ instead of $\delta_{x_i,x}$ in step (2), we implement the 1-closeness oracle*

$$O_{f_{\mathcal{C}_{LWE}}}(|x\rangle\,|b\rangle) = |x\rangle\,|b \oplus f_{\mathcal{C}_{LWE}}(x)\rangle\,.$$

Time and Space Analysis. Replacing $\delta_{x_i,x}$ by $\delta_{x_i,x}^{\pm 1}$ only costs a constant factor in the time complexity. Accordingly, the oracle function $O_{f_{\mathcal{C}_{\mathrm{LWE}}}}$ requires $\mathcal{O}(n)$ qubits and performs in time $T_{f_{\mathcal{C}_{\mathrm{LWE}}}} \coloneqq 3 \cdot n \cdot |L| = \mathcal{O}(n \cdot |L|)$. □

This algorithm allows to apply the CNS Algorithm to solve the ternary LWE problem. Noteworthy, this type of adjustment of combining the CNS algorithm with a 1-closeness oracle has not been done before. For instance, Helm and May [26], who applied the CNS algorithm to the subset sum problem, do not need to make such changes since the subset sum problem is considered an unperturbed system. They use the set-membership oracle (see Alg. 21) as Chailloux et al. [11] proposed.

4.4 Quantum LWE with $\mathcal{O}(n)$ Qubits

Finally, we combine May's algorithm [35] and the CNS algorithm [12]—with a setup that prepares a ternary Dicke state and an 1-closeness oracle. The idea is to find the LWE key by applying the algorithm to $\mathcal{C}_{\mathrm{LWE}}$, where $f_1, f_2, \mathcal{S}_1, \mathcal{S}_2$ are defined as before. Therefore, we take the following approach:

(1) Construct a (sorted) *list L of candidates for s_1* and ensure that there exists at least one element $s \in L$ with $(s, *) \in \mathcal{C}_{\mathrm{LWE}}$.
(2) Apply Amplitude Amplification using a setup $\mathcal{A}$ which produces a superposition of a *set $\mathcal{S}_2^* \subset \mathcal{S}_2$ containing candidates for s_2* and $O_{f_{\mathcal{C}_{\mathrm{LWE}}}}$ as oracle. This step eventually outputs a L-suitable element $s_2 \in \mathcal{S}_2^*$.
(3) Find $s_1 \in L$ such that $(s_1, s_2) \in \mathcal{C}_{\mathrm{LWE}}$.

Since step (3) is straightforward because L is sorted, it remains to define and construct list L and set $\mathcal{S}_2^*$. Having May's REP-0 and REP-1 algorithm [35] in mind, even this is not difficult, and we can set $L \coloneqq L_1^{(1)}$ and $\mathcal{S}_2^* \coloneqq L_2^{(1)}$. This ensures that $L_1^{(1)}$ contains at least one element s_1 with $(s_1, *) \in \mathcal{C}_{\mathrm{LWE}}$ and list $L_2^{(1)}$ contains a $L_1^{(1)}$-suitable element s_2.

QREP-0. First, we show how to construct list L and, afterward, the setup algorithm $\mathcal{A}$. In the setting of *unlimited* classical memory and $\mathcal{O}(n)$ qubits, it is a natural idea to use Amplitude Amplification for computing L as well. Indeed, this algorithm is not advantageous for constructing big lists ([26] concluded the same for SSP).

Therefore we reduce the size of list L by shrinking the search space of s_1 by changing the weights to $L_1^{(1)} \subset \mathcal{T}^n(cw/4)$ and $L_2^{(1)} \subset \mathcal{T}^n((2-c)w/4)$, where $c \in [0,1]$. This changes the number of representations:

$$\mathcal{R}_c^{(0)} \coloneqq \binom{w/2}{cw/4}^2 = \binom{w/2}{cw/4,(2-c)w/4}^2 \approx 2^{H(c/2,\cdot)\omega n}.$$

Let be $c \in [0,1]$ and $r \coloneqq \lfloor \log_q \mathcal{R}_c^{(0)} \rfloor$. Use REP-0 (Alg. 18) to construct $L \coloneqq L_1^{(1)}$ classically with weight $cw/4$ and 2^l list elements. According to the CNS algorithm we require at least

$$2^l \geq \frac{|\mathcal{T}^n(cw/4)|}{\mathcal{R}_c^{(0)}} \approx 2^{H3(c\omega/4)-H(c/2,\cdot)\omega n}$$

elements, and according to Alg. 18 this is expected to be the case. This has the time and (classical) space complexity

$$T_1 = M_1 = 2^{\max\{H3(c\omega/4)/2, H3(c\omega/4)-H(c/2,\cdot)\omega\}n}.$$

For the construction of setup $\mathcal{A}$, which generates an equal superposition of $\mathcal{S}_2^* \subset \mathcal{S}_2 \coloneqq \mathcal{T}^n((2-c)w/4)$, we first generate an equal superposition of $\mathcal{S}_2$—using Thm. 10—which is the ternary Dicke State $|\mathcal{D}^{2\cdot n}_{(2-c)w/4,(2-c)w/4,0}\rangle$. Therefore we identify the underlying set $\{0,1,2,3\}^n$ with $\{0,1,-1,*\}^n$ ($*$ arbitrary). This requires linear time and $\mathcal{O}(n)$ qubits.

Notice that $\mathcal{S}_2^*$ is by construction expected to contain one L-suitable element. Thus, by definition of the oracle function $f_{\mathcal{C}_{\mathrm{LWE}}}$ there exists an error vector $e \in \mathcal{T}^n$ such that $\pi_r(f_2(s)+e) = 0 \mod q$. Because of the regularity of f_2, the probability of an element $s \in \mathcal{S}_2$ being in $\mathcal{S}_2^*$ is $p = (3/q)^r$. Hence, the setup algorithm $\mathcal{A}$ has time complexity

$$T_{\mathrm{setup}} = \mathcal{O}(n) + \mathcal{O}\Big(T_{f_2} \cdot (q/3)^{r/2}\Big) = 2^{\widetilde{\mathcal{O}}(H(c/2,\cdot)\omega n/2)},$$

where we use $q^r \leq \mathcal{R}_c^{(0)}$, and neglect the factor $3^{r/2}$ for the same reason as classically (see Rem. 19). The resulting QREP-0 algorithm has space complexity $S = M_1$, and time complexity $T = \max\{T_1, T_2\}$, where

$$T_2 = \mathcal{O}\left(\frac{T_{\mathrm{setup}} + T_{f_{\mathcal{C}_{\mathrm{LWE}}}}}{\sqrt{\mathbb{P}_{s_2 \in \mathcal{S}_2^*}[f_{\mathcal{C}_{\mathrm{LWE}}}(s_2) = 1]}}\right) = \mathcal{O}\left(\frac{2^{\widetilde{\mathcal{O}}(H(c/2,\cdot)\omega n/2)} + 2^l}{\sqrt{\mathbb{P}_{s_2 \in \mathcal{S}_2^*}[f_{\mathcal{C}_{\mathrm{LWE}}}(s_2) = 1]}}\right)$$
$$= 2^{\widetilde{\mathcal{O}}(\max\{H(c/2,\cdot)\omega n/2, H3(c\omega/4)-H(c/2,\cdot)\omega n\}+(H3((2-c)\omega/4)-H(2c,\cdot)\omega)n/2)}$$

is the total time of the adapted CNS algorithm. For calculating the probability, we use the fact that L is expected to contain precisely one proper candidate for s_1. This implies, that there exists exactly one L-suitable $s_2 \in \mathcal{S}_2^*$, and

$$p \coloneqq \mathbb{P}_{s_2 \in \mathcal{S}_2^*}[f_{c_{\mathrm{LWE}}}(s_2) = 1]^{-1/2} = |\mathcal{S}_2^*|^{1/2} = \sqrt{\frac{|\mathcal{T}^n((2-c)w/4)|}{(q/3)^r}} = 2^{\widetilde{\mathcal{O}}((H3((2-c)\omega/4) - H(c/2,\cdot)\omega)n/2)},$$

where we again use Rem. 19 to neglect the factor $3^{r/2}$.

The analysis shows, that the algorithm extracts the LWE key without using classical memory in time $\mathcal{S}^{0.5}$ for any ω. On one hand, this is not very surprising since this corresponds to the complexity of Grover's search. On the other hand, this already shows that QREP-0 achieves better results than van Hoof's memory-efficient versions [45], which has a runtime in the range of $[\mathcal{S}^{0.510}, \mathcal{S}^{0.558}]$.

In general, a growing c increases the list size $|L|$ and the number of representations $\mathcal{R}_c^{(0)}$. Hence, we need more classical space, and the time complexities T_{setup} and $T_{f_{c_{\mathrm{LWE}}}}$ get larger, whereas the probability p decreases. However, there exists a $c_{\max} \in [0,1]$ for which the overhead for the 1-closeness oracle starts to be too large, which results in worse time complexities. For $c = c_{\max}$, we achieve the best obtainable time and worst space complexity (see Fig. 9).

$\boldsymbol{\omega}$	$c_{\max}$	$\log_{\mathcal{S}}$time	$\log_{\mathcal{S}}$space	$\log_{\mathcal{S}} ST$	REP-0: $\log_{\mathcal{S}}$time/ $\log_{\mathcal{S}}$space	REP-0: $\log_{\mathcal{S}} ST$
0.375	0.220	0.477	0.109	0.586	0.383	0.765
0.441	0.236	0.473	0.121	0.595	0.378	0.756
0.5	0.292	0.471	0.150	0.621	0.374	0.748
0.621	0.427	0.471	0.221	0.692	0.381	0.763
0.668	0.485	0.475	0.253	0.728	0.395	0.791

Fig. 9: Best obtainable time complexities for REP-0 and our QREP-0. Note that our new algorithm QREP-0 achieves a better space-time complexity than REP-0 for all $c \in [0, c_{\max}]$.

QREP-1. Our QREP-1 algorithm has—exactly as in the classical REP-1—parameters $\alpha^{(3)} = \ldots = \alpha^{(d-2)} = 0$ for any search tree of depth $d \geq 5$ and for any ω. Therefore, we use the REP-1 depth-4 search tree (see Alg. 20) to construct the list $L \coloneqq L_1^{(1)} \subset \mathcal{T}^n(cw/4 + a^{(0)})$ classically, where $c \in [0,1]$ and $a^{(0)}, a^{(1)}, a^{(2)}$ are the numbers of additional $(+1)$ and (-1) entries. Then we use Thm. 12 to setup an equal superposition of set $\mathcal{S}_2^* \coloneqq L_2^{(1)} \subset \mathcal{T}^n((2-c)w/4 + a^{(0)})$, which is a ternary Dicke state. Afterward, we proceed in the same way we did with QREP-0.

For the construction of lists $L^{(i)} \subset \mathcal{T}^n(w^{(i)})$, $i \geq 1$, we almost use the same weights as for REP-1:

$$w^{(i)} = \frac{w^{(i-1)}}{2} + a^{(i-1)} \text{ and } w^{(0)} = cw/2.$$

Accordingly all the numbers of representations $\mathcal{R}^{(i)}$ on level i for $i \geq 2$ stays the same, except for $i = 1$. On level 1, we have different weights, and the number of representations is

$$\mathcal{R}_c^{(1)} = \binom{w/2}{cw/4}^2 \cdot \binom{n-w}{a^{(0)}, a^{(0)}, \cdot} \approx 2^{(H(c/2,\cdot)\omega + (1-\omega)H3(\frac{\alpha^{(0)}}{1-\omega}))n}.$$

Let be $r \coloneqq \lfloor \log_q \mathcal{R}_c^{(1)} \rfloor$. List L requires at least

$$2^l \geq \frac{|\mathcal{T}^n(cw/4 + a^{(0)})|}{\mathcal{R}_c^{(1)}} \approx 2^{(H3(c\omega/4+\alpha^{(0)})-\widetilde{r})n}$$

elements, where $\widetilde{r} \coloneqq \log_2 \mathcal{R}_c^{(1)}/n$. This step has time and classical memory complexity as in REP-1 ($T = \max\{T^{(0)}, \ldots, T^{(3)}\}$ and $M = \max\{|L^{(1)}|, \ldots, |L^{(4)}|\}$). Adapting the analysis from QREP-0, we get:

$$\begin{aligned}
T_{\text{setup}} &= \mathcal{O}\Big(T_{f_2} \cdot (q/3)^{r/2}\Big) = 2^{\widetilde{\mathcal{O}}(\widetilde{r}n/2)}, \\
T_{f_{c_{\text{LWE}}}} &= \mathcal{O}(n \cdot |L|) = \widetilde{\mathcal{O}}\left(2^{(H3(c\omega/4+\alpha^{(0)})-\widetilde{r})n}\right), \\
p &= \mathbb{P}_{s_2 \in \mathcal{S}_2^*}[f_{c_{\text{LWE}}}(s_2) = 1]^{-1/2} = |\mathcal{S}_2^*|^{1/2} = \sqrt{\frac{|\mathcal{T}^n((2-c)w/4 + a^{(0)})|}{(q/3)^r}} \\
&= 2^{\widetilde{\mathcal{O}}\left((H3((2-c)\omega/4+\alpha^{(0)})-\widetilde{r})n/2\right)}.
\end{aligned}$$

Thus, $T_2 = 2^{\widetilde{\mathcal{O}}\left(\max\{\widetilde{r}n/2, (H3(c\omega/4+\alpha^{(0)})-\widetilde{r})n\} + (H3((2-c)\omega/4+\alpha^{(0)})-\widetilde{r})n/2\right)}$.

$\boldsymbol{\omega}$	$c_{\max}$	$\log_{\mathcal{S}}$time	$\log_{\mathcal{S}}$space	$\log_{\mathcal{S}} ST$	REP-0: $\log_{\mathcal{S}}$time/ $\log_{\mathcal{S}}$space	REP-0: $\log_{\mathcal{S}} ST$
0.375	1.315	0.381	0.242	0.623	0.383	0.765
0.441	1.273	0.379	0.217	0.596	0.378	0.756
0.5	1.330	0.379	0.213	0.592	0.374	0.748
0.621	1.315	0.388	0.203	0.591	0.381	0.763
0.668	1.400	0.397	0.208	0.605	0.395	0.791

Fig. 10: Best obtainable time complexities of our QREP-1 with $c = c_{\max}$ compared with REP-0.

The analysis of QREP-1 points out, that the best obtainable time complexities are in the range of $[\mathcal{S}^{0.379}, \mathcal{S}^{0.397}]$ (see Fig. 10). This is a significant improvement compared to QREP-0, where we achieved time complexities in the range

of $[\mathcal{S}^{0.471}, \mathcal{S}^{0.477}]$. For QREP-1, the parameter $c_{\max}$ is generally bigger than for QREP-0. This is because the additional representations of s reduce the overhead for the 1-closeness oracle so that we can use larger values for c until the overhead starts to be too big. According to the resulting time-memory trade-offs, QREP-1 has useful instantiations for classical memory requirements up to $\mathcal{S}^{0.242}$ and can therefore be applied flexibly.

Surprisingly, for QREP-1 we almost achieve the same time complexities as for REP-0, while it is significantly more (classical) memory-efficient. For instance, QREP-1 with $\omega = 0.375$ runs in time $\mathcal{S}^{0.381}$ using $\mathcal{S}^{0.242}$ classical space, whereas REP-0 has almost the same time complexity ($\mathcal{S}^{0.383}$) but requires $\mathcal{S}^{0.383}$ classical memory.

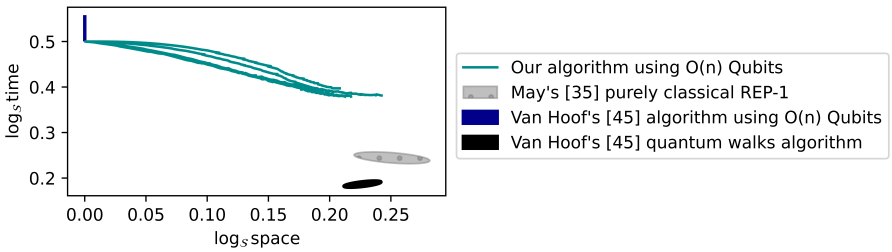


Fig. 11: Time-memory trade-off of our algorithm compared to May's purely classical MitM attack [35] and van Hoof et al.'s memory-efficient variant [45] which uses $\mathcal{O}(n)$ qubits. Obviously, our algorithm is the best option for classical memory requirements in the range of $[\mathcal{S}^{0}, \mathcal{S}^{0.22}]$. The concrete complexity depends on the parameter ω.

Fig. 11 shows the time-memory trade-offs of QREP-1 for some ω-values and compares it with the performance of REP-0, REP-1, QREP-0, van Hoof et al.'s memory-efficient quantum algorithm [45] and van Hoof et al.'s quantum walks algorithm [45]. Moreover, the figure shows that the algorithm QREP-1 closes a previously existing gap. Until now, an adversary (with $\mathcal{O}(n)$ qubits) could not perform better than $\mathcal{S}^{0.51}$ while having not enough classical memory for REP-0 or REP-1 available. Since there is the common starting-point $(\mathcal{S}^{0}, \mathcal{S}^{0.5})$ of QREP-1 for all ω, all instantiations of our algorithm have a better time complexity than $\mathcal{S}^{0.51}$.

Furthermore, our algorithm allows the adversary to reduce the time requirements by deploying more classical memory. Nevertheless, it can not beat the time complexity of REP-1. So QREP-1 is a good solution for an adversary (independent of ω), if she has classical memory up to $\mathcal{S}^{0.22}$ available.

Acknowledgments

I would like to thank Marc Fischlin for helpful conversations and supporting this work. I am grateful to Jan Nöller for engaging discussions on quantum computing. I also thank the anonymous reviewers for their valuable comments.

This research work has been funded by the German Federal Ministry of Education and Research and the Hessian Ministry of Higher Education, Research, Science and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References

1. Aktar, S., Bärtschi, A., Badawy, A.H.A., Eidenbenz, S.: A divide-and-conquer approach to dicke state preparation. IEEE Transactions on Quantum Engineering **3**, 1–16 (2022)
2. Ambainis, A.: Quantum walk algorithm for element distinctness. SIAM Journal on Computing **37**(1), 210–239 (2007)
3. Bärtschi, A., Eidenbenz, S.: Deterministic preparation of dicke states. In: International Symposium on Fundamentals of Computation Theory. pp. 126–139. Springer (2019)
4. Bernstein, D.J., Brumley, B.B., Chen, M.S., Chuengsatiansup, C., Lange, T., Marotzke, A., Peng, B.Y., Tuveri, N., van Vredendaal, C., Yang, B.Y.: NTRU Prime. Tech. rep., National Institute of Standards and Technology (2020), available at `https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions`
5. Bernstein, D.J., Chuengsatiansup, C., Lange, T., Vredendaal, C.v.: Ntru prime: reducing attack surface at low cost. In: International Conference on Selected Areas in Cryptography. pp. 235–260. Springer (2017)
6. Bonnetain, X., Bricout, R., Schrottenloher, A., Shen, Y.: Improved classical and quantum algorithms for subset-sum. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 633–666. Springer, Cham (Dec 2020). `https://doi.org/10.1007/978-3-030-64834-3_22`
7. Brakerski, Z., Langlois, A., Peikert, C., Regev, O., Stehlé, D.: Classical hardness of learning with errors. In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing. pp. 575–584 (2013)
8. Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation (2002)
9. Brassard, G., Høyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: Lucchesi, C.L., Moura, A.V. (eds.) LATIN 1998. LNCS, vol. 1380, pp. 163–169. Springer, Berlin, Heidelberg (Apr 1998). `https://doi.org/10.1007/bfb0054319`
10. Bärtschi, A., Eidenbenz, S.: Short-depth circuits for dicke state preparation. In: 2022 IEEE International Conference on Quantum Computing and Engineering (QCE). pp. 87–96 (2022)
11. Chailloux, A., Naya-Plasencia, M., Schrottenloher, A.: An efficient quantum collision search algorithm and implications on symmetric cryptography. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part II. LNCS, vol. 10625, pp. 211–240. Springer, Cham (Dec 2017). `https://doi.org/10.1007/978-3-319-70697-9_8`

12. Chailloux, A., Naya-Plasencia, M., Schrottenloher, A.: An efficient quantum collision search algorithm and implications on symmetric cryptography. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 211–240. Springer (2017)
13. Chen, C., Danba, O., Hoffstein, J., Hulsing, A., Rijneveld, J., Schanck, J.M., Schwabe, P., Whyte, W., Zhang, Z., Saito, T., Yamakawa, T., Xagawa, K.: NTRU. Tech. rep., National Institute of Standards and Technology (2020), available at https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/round-3-submissions
14. Cheon, J.H., Hhan, M., Hong, S., Son, Y.: A hybrid of dual and meet-in-the-middle attack on sparse and ternary secret lwe. IEEE Access **7**, 89497–89506 (2019)
15. Chevignard, C., Fouque, P.A., Schrottenloher, A.: Reducing the number of qubits in quantum information set decoding. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part VIII. LNCS, vol. 15491, pp. 299–329. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0944-4_10
16. Childs, A.M., Farhi, E., Goldstone, J., Gutmann, S.: Finding cliques by quantum adiabatic evolution. Quantum Information & Computation **2**(3), 181–191 (2002)
17. Chopra, A.: GLYPH: A new insantiation of the GLP digital signature scheme. Cryptology ePrint Archive, Report 2017/766 (2017), https://eprint.iacr.org/2017/766
18. Csiszár, I., Shields, P.C., et al.: Information theory and statistics: A tutorial. Foundations and Trends® in Communications and Information Theory **1**(4), 417–528 (2004)
19. Dicke, R.H.: Coherence in spontaneous radiation processes. Physical review **93**(1), 99 (1954)
20. Ducas, L.: Accelerating bliss: the geometry of ternary polynomials. Cryptology ePrint Archive, Report 2014/874 (2014), https://eprint.iacr.org/2014/874
21. Ducas, L., Durmus, A., Lepoint, T., Lyubashevsky, V.: Lattice signatures and bimodal Gaussians. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 40–56. Springer, Berlin, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_3
22. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: CRYSTALS-Dilithium: A lattice-based digital signature scheme. IACR TCHES **2018**(1), 238–268 (2018). https://doi.org/10.13154/tches.v2018.i1.238-268, https://tches.iacr.org/index.php/TCHES/article/view/839
23. Gidney, C.: Quirk: Quantum circuit simulator. a drag-and-drop quantum circuit simulator., https://algassert.com/quirk, visited 08.10.2024
24. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. pp. 212–219 (1996)
25. Güneysu, T., Lyubashevsky, V., Pöppelmann, T.: Practical lattice-based cryptography: A signature scheme for embedded systems. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 530–547. Springer, Berlin, Heidelberg (Sep 2012). https://doi.org/10.1007/978-3-642-33027-8_31
26. Helm, A., May, A.: The power of few qubits and collisions - subset sum below Grover's bound. In: Ding, J., Tillich, J.P. (eds.) Post-Quantum Cryptography - 11th International Conference, PQCrypto 2020. pp. 445–460. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-44223-1_24
27. Hoffstein, J., Pipher, J., Silverman, J.H.: Ntru: A ring-based public key cryptosystem. In: International algorithmic number theory symposium. pp. 267–288. Springer (1998)

28. Howgrave-Graham, N., Joux, A.: New generic algorithms for hard knapsacks. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 235–256. Springer, Berlin, Heidelberg (May / Jun 2010). https://doi.org/10.1007/978-3-642-13190-5_12
29. IBM: The hardware and software for the era of quantum utility is here (2023), https://www.ibm.com/quantum/blog/quantum-roadmap-2033, visited 08.10.2024
30. Kaye, P., Mosca, M.: Quantum networks for generating arbitrary quantum states. In: Optical Fiber Communication Conference and International Conference on Quantum Information. p. PB28. Optica Publishing Group (2001)
31. Kirshanova, E., May, A.: How to find ternary LWE keys using locality sensitive hashing. In: Paterson, M.B. (ed.) 18th IMA International Conference on Cryptography and Coding. LNCS, vol. 13129, pp. 247–264. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92641-0_12
32. Kues, M., Reimer, C., Roztocki, P., Cortés, L.R., Sciara, S., Wetzel, B., Zhang, Y., Cino, A., Chu, S.T., Little, B.E., et al.: On-chip generation of high-dimensional entangled quantum states and their coherent control. Nature **546**(7660), 622–626 (2017)
33. Lyubashevsky, V., Ducas, L., Kiltz, E., Lepoint, T., Schwabe, P., Seiler, G., Stehlé, D., Bai, S.: CRYSTALS-DILITHIUM. Tech. rep., National Institute of Standards and Technology (2022), available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022
34. Mahmudlu, H., Johanning, R., van Rees, A., Khodadad Kashi, A., Epping, J.P., Haldar, R., Boller, K.J., Kues, M.: Fully on-chip photonic turnkey quantum source for entangled qubit/qudit state generation. Nature Photonics pp. 1–7 (2023)
35. May, A.: How to meet ternary LWE keys. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 701–731. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84245-1_24
36. Mukherjee, C.S., Maitra, S., Gaurav, V., Roy, D.: Preparing dicke states on a quantum computer. IEEE Transactions on Quantum Engineering **1**, 1–17 (2020)
37. OpenSSH: Openssh release notes 9.0 (2022), https://www.openssh.com/txt/release-9.0, visited 08.10.2024
38. Özdemir, S.K., Shimamura, J., Imoto, N.: A necessary and sufficient condition to play games in quantum mechanical settings. New Journal of Physics **9**(2), 43 (2007)
39. Peikert, C.: Public-key cryptosystems from the worst-case shortest vector problem. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 333–342 (2009)
40. Perriello, S., Barenghi, A., Pelosi, G.: A complete quantum circuit to solve the information set decoding problem. In: 2021 IEEE International Conference on Quantum Computing and Engineering (QCE). pp. 366–377 (2021). https://doi.org/10.1109/QCE52317.2021.00056
41. Raveh, D., Nepomechie, R.I.: q-analog qudit dicke states. Journal of Physics A: Mathematical and Theoretical (2024)
42. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
43. Ringbauer, M., Meth, M., Postler, L., Stricker, R., Blatt, R., Schindler, P., Monz, T.: A universal qudit quantum processor with trapped ions. Nature Physics **18**(9), 1053–1057 (2022)
44. Shannon, C.E.: A mathematical theory of communication. The Bell System Technical Journal **27**(3), 379–423 (1948)

45. Van Hoof, I., Kirshanova, E., May, A.: Quantum key search for ternary LWE. In: Cheon, J.H., Tillich, J.P. (eds.) Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021. pp. 117–132. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-81293-5_7

S Supplementary Material

This Jupyter Notebook contains an algorithm to generate a circuit which prepares the generalized Dicke states and an example of usage.

This note is accessible via https://github.com/bj-benedikt/Generalized-Dicke-States.

S.1 Preparing Generalized Dicke States

[1]:
```
import numpy as np
from qiskit import *
from qiskit.visualization import plot_distribution, state_visualization
from qiskit.circuit.library import RYGate
from qiskit.quantum_info import Statevector
from IPython.display import Latex
```

Now we start to introduce the different Gates and Unitaries as in the paper.

Rotation Gate (Definition 8)

input: *num* - numerator, *den* - denominator

output: rotation gate $\boxed{\sqrt{num/den}}$

[2]:
```
def ry_gate(num, den):
    return RYGate(2*np.arccos(np.sqrt(num/den)),label=str(num)+ '/'+str(den))
```

The unitary SCS (implemented as in Theorem 10 and stated in PrepareAncAndSwap)

input: n - vector length, k - Hamming weight, q - number of qubits per vector component, `circ` - quantum circuit, `qr` - quantum register, `anc` - ancilla qubits, $\texttt{qr}_{offset}$ - clarifies on which registers of `qr` the swap-algorithm should be applied, $\texttt{anc}_{offset}$ - clarifies which ancilla qubits the swap-algorithm should use, `measurement` - boolean that turns the measurement of `anc` on/off

output: SCS-unitary for the given parameters

Our implementation provides the boolean `measurement` to turn the measurement of `anc` on and off. This is necessary for the analysis since some of qiskit's functionalities insist on "unmeasured" qubits.

[3]:
```
def SCS(n,k,circ,qr,anc,cl_anc,qr_offset,anc_offset,q, measurement = True):
    # Prepare the first ancilla qubit (Line 3 of PrepareAncAndSwap).
    circ.append(ry_gate(n-1,n),[anc[anc_offset]])
    for j in range(q):
        circ.cswap(anc[anc_offset], qr[qr_offset + j], qr[qr_offset + j + 2],␣
↪ctrl_state='0')
    for i in range(k-2):
        # Prepare the next ancilla qubit (Line 6,7 of PrepareAncAndSwap).
        circ.cx(anc[anc_offset + i],anc[anc_offset + i+1])
```

```
        circ.append(ry_gate(n-i-2,n-i-1).control(1,␣
↪ctrl_state='0'),[anc[anc_offset + i],anc[anc_offset +1+ i]])
        for j in range(q):
            # Next line corresponds to Line 4 of PrepareAncAndSwap
            circ.cswap(anc[anc_offset +1 + i], qr[qr_offset + j], qr[qr_offset␣
↪+ j +2*(i + 1) + 2], ctrl_state='0')
        if (measurement == True):
            # Measure the "used" ancilla qubits (Line 9 of PrepareAncAndSwap)
            circ.measure(anc[anc_offset + i], cl_anc[anc_offset + i])
    if (measurement == True):
        circ.measure(anc[anc_offset + k - 2], cl_anc[anc_offset + k - 2])
    return circ
```

S.2 Example of Usage

In the following we show an example of usage, how to prepare a generalized 4-dimensional Dicke state with $q = 2$. Feel free to change the initial state by adjusting the parameter ℓ_1, ℓ_2 and ℓ_3.

[4]:
```
n = 4
k = 4
ell_1 = 2
ell_2 = 0
ell_3 = 1

if (k < ell_1 + ell_2 + ell_3):
    print("You've chosen the wrong parameter!")
    print("The Hamming weight of the vector is bigger than k!")
```

Hence we want to build an equal superposition over a set with $\binom{n}{\ell_1,\ell_2,\ell_3} = \binom{4}{2,0,1} = 12$ elements, where each of the vectors has probability $1/12$.

However, the number of required ancilla qubits needs to be known in advance for preparing the Dicke state.

In proof of Theorem 12 we derived the formula: $\#$ Needed-Qubits $= (n-k)k + \sum_{i=1}^{k-1} i$.

input: n - vector length, k - Hamming weight

output: number of needed ancilla qubits of $U_{q\cdot n,k}$

[5]:
```
def number_anc(n,k):
    result = (n-k)*(k)
    for i in range(k):
        result += i
    return result
```

Prepare the initial state $|x\rangle$ according to the parameter ℓ_1, ℓ_2, ℓ_3.

[6]:
```
init = ''
for i in range(n - ell_1 - ell_2 - ell_3):
    init += '00'
for i in range(ell_1):
    init += '01'
for i in range(ell_2):
    init += '10'
for i in range(ell_3):
    init += '11'
```

Build the test circuit:

(1) Prepare the needed quantum register for the Dicke state `qr` and for the ancilla qubits `anc`, as well as the corresponding classical register `cl_qr` and `cl_anc` for the measurement at the end.

(2) Apply the unitary $\mathcal{U}_{2\cdot 4,4} = (SCS_{q\cdot 2,1} \otimes Id^{\otimes 4}) \cdot (SCS_{q\cdot 3,2} \otimes Id^{\otimes 2}) \cdot SCS_{q\cdot 4,3}$ (Lemma 7) to prepare the Dicke state $|\mathcal{D}^{2\cdot 4}_{\ell_1,\ell_2,\ell_3}\rangle$.

Again, our test circuit provides a boolean parameter `measurement` to turn the measurement of `qr` and `anc` on and off.

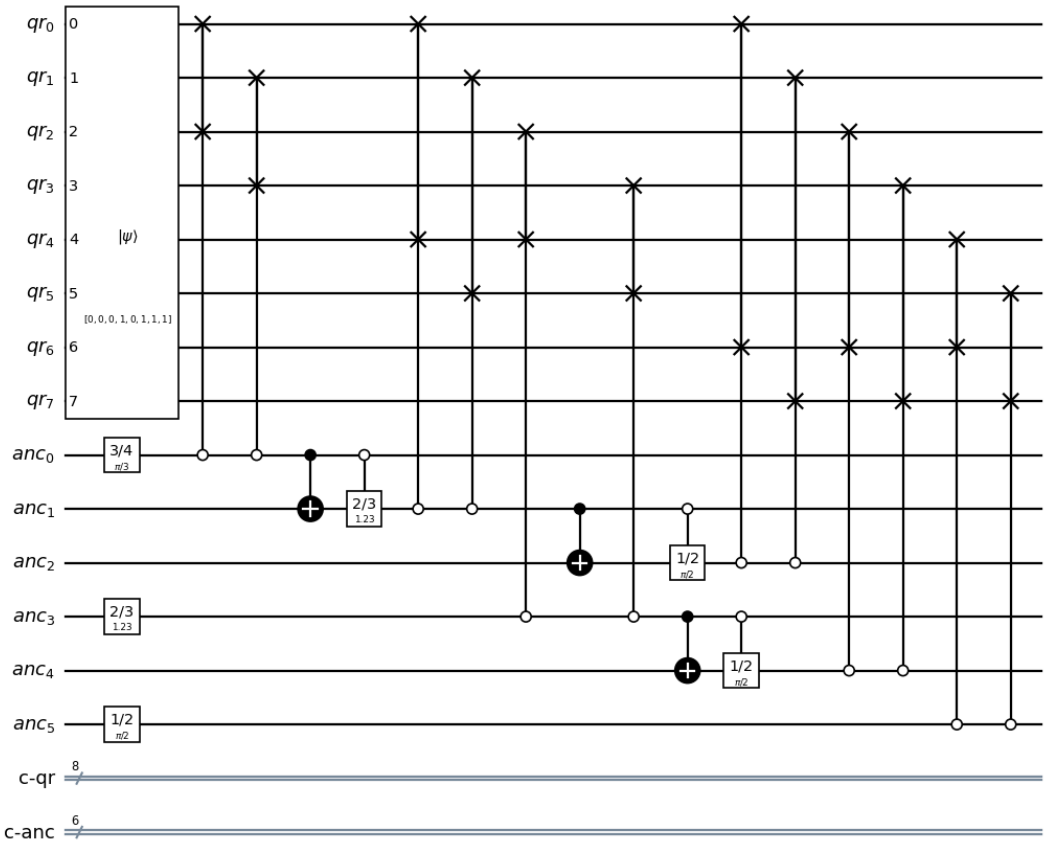
[7]:
```
def build_circuit(measurement):
    qr = QuantumRegister(2*n, 'qr')
    anc = QuantumRegister(number_anc(n,k), 'anc')
    # Classical register for measurements
    cl_qr = ClassicalRegister(2*n, 'c-qr')
    cl_anc = ClassicalRegister(number_anc(n,k), 'c-anc')
    circ = QuantumCircuit(qr, anc, cl_qr, cl_anc)
    # Intialize the register qr with init = |x>.
    circ.initialize(init, qr[:])
    # Set anc_offset
    anc_offset=0
    for i in range(n-1):
        circ = SCS(n-i, np.minimum(k+1, n-i),circ, qr, anc, cl_anc, 2*i,␣
↪anc_offset, 2, measurement=measurement)
        # Update anc_offset, and use "fresh" ancilla qubits in the next␣
↪iteration
        anc_offset += np.minimum(k, n-1-i)
    if (measurement == True):
        for i in range(2*n):
            circ.measure(qr[i],cl_qr[i])
    return circ
```

S.2.1 Build the Circuit and Compute the Statevector

[8]:
```
circ = build_circuit(measurement = False)
```

[9]:
```
circ.draw('mpl', cregbundle = True, reverse_bits=False, scale=0.8, style = "bw")
```

[9]:



[10]:
```
state = Statevector.from_int(0, 2**(2*n+number_anc(n,k)))
state = state.evolve(circ)
state_to_latex = state_visualization._state_to_latex_ket(state.data, max_size =␣
↪256)
Latex("$" + state_to_latex + "$")
```

[10]:
$$\frac{\sqrt{6}}{12}|00000011010100\rangle + \frac{\sqrt{6}}{12}|00010011010001\rangle + \frac{\sqrt{6}}{12}|00011011010001\rangle + \frac{\sqrt{6}}{12}|00011101010011\rangle + \frac{\sqrt{6}}{12}|01000011010100\rangle + \frac{\sqrt{6}}{12}|01010011000101\rangle + \frac{\sqrt{6}}{12}|01011011000101\rangle + \frac{\sqrt{6}}{12}|01011101000111\rangle + \frac{\sqrt{6}}{12}|01100001011100\rangle + \frac{\sqrt{6}}{12}|01110001001101\rangle + \frac{\sqrt{6}}{12}|01111001001101\rangle + \frac{\sqrt{6}}{12}|01111101000111\rangle + \frac{\sqrt{6}}{12}|10000001110100\rangle + \frac{\sqrt{6}}{12}|10010001110001\rangle + \frac{\sqrt{6}}{12}|10011001110001\rangle + \frac{\sqrt{6}}{12}|10011101010011\rangle + \frac{\sqrt{6}}{12}|11000001110100\rangle + \frac{\sqrt{6}}{12}|11010000110101\rangle + \frac{\sqrt{6}}{12}|11011000110101\rangle + \frac{\sqrt{6}}{12}|11011100010111\rangle + \frac{\sqrt{6}}{12}|11100001011100\rangle + \frac{\sqrt{6}}{12}|11110000011101\rangle + \frac{\sqrt{6}}{12}|11111000011101\rangle + \frac{\sqrt{6}}{12}|11111100010111\rangle$$

The result register contains the quantum register `qr` and `anc` so that we can ignore the first 6 qubits in each term. We are only interested in the last 8 qubits which contains the Dicke state (in `qr`). Each of the vectors - containing (ℓ_i) many i-entries, $i = 1, 2, 3$ - is twice in the state, so each of this

vectors has - as expected - the amplitude $\sqrt{(\frac{\sqrt{6}}{12})^2 + (\frac{\sqrt{6}}{12})^2} = \frac{\sqrt{12}}{12} = \frac{1}{\sqrt{12}}$ and the probability $\frac{1}{12}$.

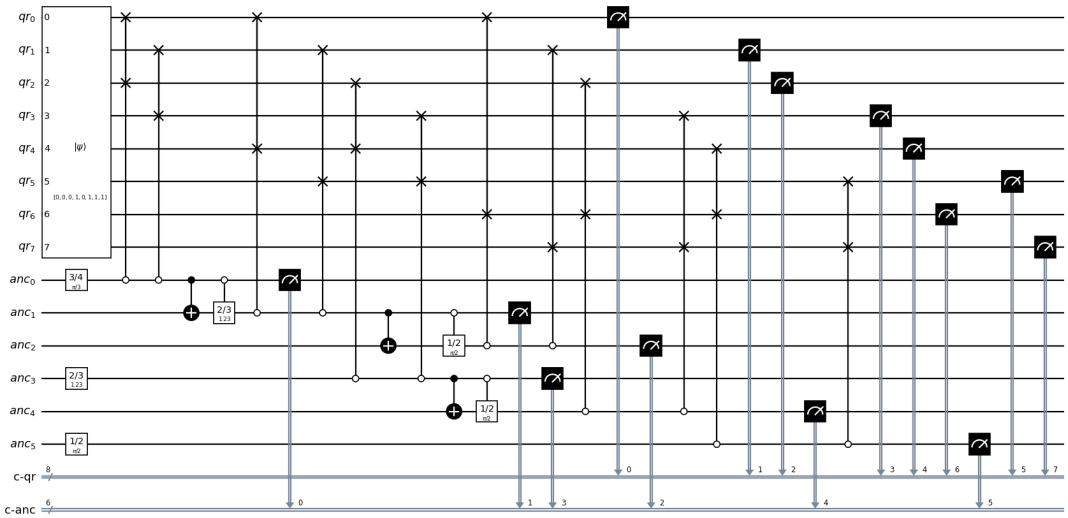
S.2.2 Build the Circuit and Measure the Statevector

Note that the result of the experiment is not "perfect", since we can not run the circuit infinitely often.

[11]:
```
circ = build_circuit(measurement = True)
```

[12]:
```
circ.draw('mpl', cregbundle = True, reverse_bits=False, scale=0.8, fold = 32,␣
↪style = "bw")
```

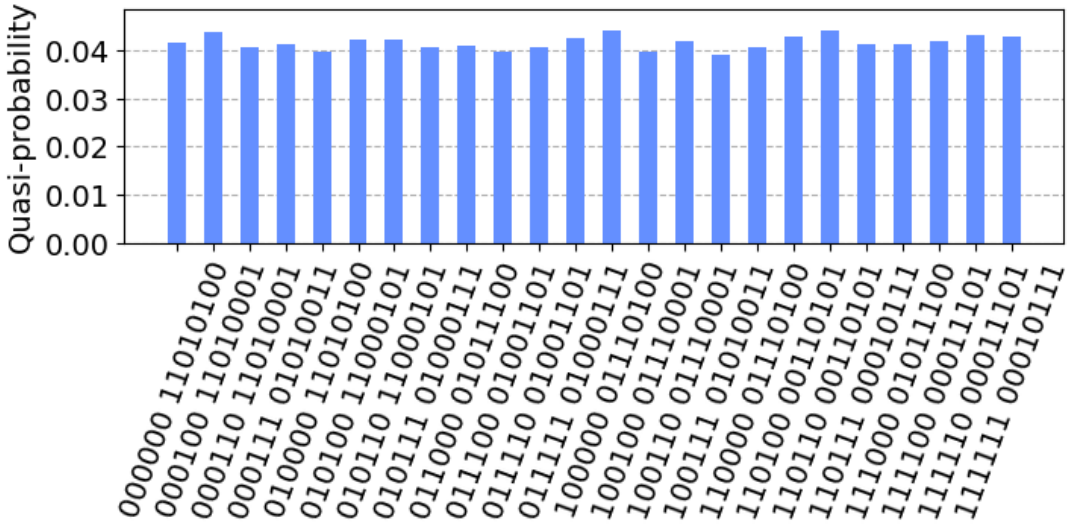
[12]:



The following code snippet builds and measures the circuit 25,000 - times. Again, the result contains the quantum register `qr` and `anc` and we only consider the last 8 qubits which contains the Dicke state. Each of the vectors - containing (ℓ_i) many i-entries, $i = 1, 2, 3$ - is twice in the histogram, so each of this vectors has - as expected - the (quasi-)probability $\frac{1}{12} \approx 0.0833$.

[13]:
```
# shots - number of experiment repetitions:
job = execute(circ, backend=Aer.get_backend('qasm_simulator'), shots=25000)
result = job.result()
count = result.get_counts(circ)
plot_distribution(count, figsize=(8,2), bar_labels=False)
```

[13]:


Quasi-probability
0.00
0.01
0.02
0.03
0.04
000000 11010100
000100 11010001
000110 11010001
000111 01010011
010000 11010100
010100 11000101
010110 11000101
010111 01000111
011000 01011100
011100 01001101
011110 01001101
011111 01000111
100000 01110100
100100 01110001
100110 01110001
100111 01010011
110000 01110100
110100 00110101
110110 00110101
110111 00010111
111000 01011100
111100 00011101
111110 00011101
111111 00010111