Journal of Algorithms and Computation

journal homepage: http://jac.ut.ac.ir

A Hybrid Classical-Quantum Rainbow Table Attack on Human Passwords

MA. Khajeian*[1]

[1]School of Engineering Sciences, College of Engineering, University of Tehran

ABSTRACT

Long, human-generated passwords pose significant challenges to both classical and quantum attacks due to their irregular structure and large search space. In this work, we propose an enhanced classical–quantum hybrid attack specifically designed for this scenario. Our approach constructs rainbow tables using dictionary-based password generation augmented with transformation rules that better capture real-world user behavior. These tables are organized into buckets, enabling faster lookup and reduced space complexity. For the search within each bucket, we employ a distributed exact variant of Grover's algorithm. This method provides deterministic success and significantly lower circuit depth, enhancing robustness against noise—particularly depolarizing errors common in near-term quantum devices. Overall, our hybrid framework improves the efficiency and practicality of password recovery for long, human-readable passwords in realistic adversarial settings.

ARTICLE INFO

Article history:
Research paper
Received 18 July 2025
Accepted 03 August 2025
Available online 03 August 2025

Keywords: Rainbow Table Attack, Grover's Algorithm, Human Password Recovery, Quantum Security.

AMS subject Classification: 68P25, 81P68.

*Corresponding author: MA. Khajeian. Email: khajeian@ut.ac.ir

1 Introduction

The growing use of lengthy, user-created passwords has weakened the effectiveness of conventional password-cracking techniques. Such passwords, typically built from recognizable words or easy-to-remember sequences, often evade detection by traditional brute-force methods and standard rainbow table attacks.
In addition, the anticipated rise of quantum computing introduces new challenges and opportunities in password recovery. Although Grover's algorithm offers a quadratic speedup for unstructured search, its direct application remains impractical in many real-world scenarios due to its high circuit depth and probabilistic nature.
To address these challenges, we revisit and extend the use of structured rainbow tables for human-generated passwords. Building on prior work that introduces a "smart dictionary" using dictionary generators and transformation rules [6], we adopt a structured approach. Passwords are broken down into components governed by composition patterns. This modeling captures human tendencies and enables compact, realistic coverage of the password space. As a result, precomputation becomes feasible even for long passwords.
Based on this foundation, we propose a hybrid classical–quantum password cracking framework with two key innovations:

- We partition the smart dictionary-derived rainbow table into buckets based on structural similarity or indexing heuristics. This reduces space complexity and enables fast, targeted lookups—particularly suitable for quantum memory access.
- We use a distributed exact Grover variant [7] to perform quantum search within each bucket. This method offers deterministic success and lower quantum circuit depth, making it more robust to depolarizing noise.

This paper is organized as follows: Section 2 covers fundamentals, Section 3 presents our hybrid quantum-classical approach, Section 4 shows experimental outcomes, with implementation details and conclusions following in Sections 5 and 6.

2 Background

This section covers the rainbow table attack method, human password pattern analysis, and a modified Grover quantum search approach.

2.1 Review of Rainbow Table Attack

Rainbow tables are a time–memory trade-off technique used to invert cryptographic hash functions, particularly for password recovery. Originally introduced by Oechslin [4], they allow attackers to precompute chains of hash outputs while storing only the first and last elements of each chain. This approach significantly reduces memory usage while enabling efficient lookups during the attack phase.

2.1.1 Rainbow Table Structure

A rainbow table is composed of multiple chains, where each chain represents a sequence of alternating hash and reduction operations. The construction begins with a starting plaintext x_0, which is hashed to produce $y_0 = H(x_0)$. This hash is then reduced using a reduction function R_1 to obtain the next plaintext $x_1 = R_1(y_0)$. The process continues for t steps, alternating between hash and reduction functions:

$$x_0 \xrightarrow{H} y_0 \xrightarrow{R_1} x_1 \xrightarrow{H} y_1 \xrightarrow{R_2} \cdots \xrightarrow{R_t} x_t$$

Only the starting plaintext x_0 and the final output x_t are stored, forming the (x_0, x_t) pair that represents the chain. To minimize chain collisions—where distinct inputs lead to the same endpoint—each reduction step uses a different function R_i.

2.1.2 Table Generation and Parameters

The effectiveness of rainbow tables depends on several key parameters. The *chain length* t determines how many reduction steps are applied per chain. Longer chains reduce the number of required chains but increase lookup time. The *number of chains* m affects the table's coverage—more chains improve success rates but require more storage.
Reduction functions $R_1, R_2, \ldots, R_t$ convert hash outputs back into candidate plaintexts. The generation process is illustrated in Algorithm 1.

Algorithm 1 Ordinary Rainbow Table Generation

Require: Dictionary of starting plaintexts, chain length t
Ensure: Rainbow table entries

```
for all S_start in dictionary do
    S ← S_start
    for i ← 1 to t do
        y ← H(S)                          ▷ Apply hash function
        S ← R_i(y)                        ▷ Apply reduction function R_i
    end for
    Store pair (S_start, S_end) in table
end for
```

2.1.3 Rainbow Table Lookup Process

To look up a key in a rainbow table, the following procedure is used: First, apply the reduction function R_{n-1} to the ciphertext and check if the result matches any endpoint in the table. If a match is found, the corresponding chain can be reconstructed using the starting point. If no match is found, the process continues by applying R_{n-2} followed by f_{n-1} to check if the key appears in the second-to-last column of the table. This process is repeated iteratively, applying $R_{n-3}, f_{n-2}, f_{n-1}$, and so on. The total number of calculations required is $\frac{t(t-1)}{2}$ [4].

2.2 Modeling Human Passwords

Zhang et al. [6] proposed an improved rainbow table attack targeting long, human-readable passwords. Traditional rainbow tables use fixed dictionaries and uniform reduction functions, which fail to capture the irregular patterns found in real-world passwords. To address this, the authors introduced a *smart dictionary* generated from high-frequency words and substrings in leaked datasets, combined with human-like *transformation rules*. These transformations include capitalizing letters, appending digits or years, using leetspeak substitutions (e.g., "o" to "0"), and inserting symbols. The result is a compact yet realistic password candidate set that reflects common user behavior without expanding the search space excessively. In their evaluation, the improved method achieved a success rate of 83% when recovering SHA1 hashes, demonstrating its effectiveness in cracking long, human-structured passwords.

2.3 Distributed Exact Grover's Algorithm

The Distributed Exact Grover's Algorithm (DEGA) [7] partitions the original n-qubit search problem into $\lfloor n/2 \rfloor$ subfunctions $\{g_i\}_{i=0}^{\lfloor n/2 \rfloor - 1}$. Each subfunction g_i is derived from the target Boolean function $f(x)$ by fixing selected groups of input bits.
For $i \in \{0, 1, \ldots, \lfloor n/2 \rfloor - 2\}$, the first $2i$ bits and the last $(n-2(i+1))$ bits of x are fixed. This yields 2^{n-2} subfunctions $f_{i,j} : \{0,1\}^2 \to \{0,1\}$ defined as

$$f_{i,j}(m_i) = f(y_{j,0} \cdots y_{j,2i-1}\, m_i\, y_{j,2i} \cdots y_{j,n-3}),$$

where $m_i \in \{0,1\}^2$ is the variable input and y_j enumerates all $(n-2)$-bit configurations. Each g_i is then constructed as

$$g_i(m_i) = \mathrm{OR}\left(f_{i,0}(m_i), f_{i,1}(m_i), \ldots, f_{i,2^{n-2}-1}(m_i)\right), \tag{1}$$

with

$$\mathrm{OR}(x) = \begin{cases} 1, & |x| \geq 1 \\ 0, & |x| = 0 \end{cases}$$

and $|x|$ denoting the Hamming weight.
For the final case $i = \lfloor n/2 \rfloor - 1$, the first $2i$ bits of x are fixed, and the remaining $n - 2i$ bits constitute the variable input m_i. The subfunction is given by

$$f_{i,j}(m_i) = f(y_{j,0} \cdots y_{j,2i-1}\, m_i),$$

where the dimensionality of m_i depends on the parity of n:

$$n - 2i = \begin{cases} 2, & \text{if } n \text{ is even} \\ 3, & \text{if } n \text{ is odd} \end{cases}.$$

Then g_i is defined as

$$g_i(m_i) = \mathrm{OR}\left(f_{i,0}(m_i), f_{i,1}(m_i), \ldots, f_{i,2^{2i}-1}(m_i)\right). \tag{2}$$

This partitioning guarantees that the global target state τ can be exactly reconstructed from the solutions of the subfunctions $\{g_i\}$, with each g_i isolating a distinct segment of τ. Consequently, we obtain $\lfloor n/2 \rfloor$ subfunctions $g_i(m_i)$ based on $f(x)$ and the parameter n, for $i \in \{0, 1, \ldots, \lfloor n/2 \rfloor - 1\}$. We assume that each Oracle U_{g_i} can be efficiently constructed. For $i \in \{0, 1, \ldots, \lfloor n/2 \rfloor - 2\}$, the Oracle is defined as

$$U_{g_i(x)} : |x\rangle \to (-1)^{g_i(x)}|x\rangle, \tag{3}$$

where $x \in \{0,1\}^2$, and τ_i is the unique input satisfying $g_i(\tau_i) = 1$.
For the final subfunction $g_i : \{0,1\}^{n-2i} \to \{0,1\}$, we distinguish two cases. If n is even, g_i has a 2-bit input and the same Oracle form as in Eq. (3) applies. If n is odd, a phase Oracle is used:

$$R_{g_i(x)} : |x\rangle \to e^{i\phi \cdot g_i(x)}|x\rangle, \tag{4}$$

where $x \in \{0,1\}^3$, $i = \lfloor n/2 \rfloor - 1$, and τ_i satisfies $g_i(\tau_i) = 1$. The phase ϕ is given by

$$\phi = 2 \arcsin\left(\frac{\sin\left(\frac{\pi}{4J+6}\right)}{\sin\theta}\right), \quad J = \left\lfloor \frac{\pi/2 - \theta}{2\theta} \right\rfloor, \quad \theta = \arcsin\left(\sqrt{\frac{1}{2^3}}\right). \tag{5}$$

Algorithm 2 Distributed Exact Grover's Algorithm

Require: The number of qubit $n \geq 2$; Oracle function $f : \{0,1\}^n \to \{0,1\}$ where $f(x) = 0$ for all $x \in \{0,1\}^n$ except τ, for which $f(\tau) = 1$; $\lfloor n/2 \rfloor$ subfunctions $g_i(x)$ as in Eq. (1) and Eq. (2), generated according to $f(x)$ and n, where $i \in \{0, 1, \cdots, \lfloor n/2 \rfloor - 1\}$.
Ensure: Target state $|\tau\rangle$ with certainty

1: Apply $H^{\otimes n}$ to obtain $|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle$
2: **for** $i = 0$ to $\lfloor n/2 \rfloor - 2$ **do**
3: Apply $G_i = -H^{\otimes 2} U_0 H^{\otimes 2} U_{g_i(x)}$
4: where $U_0 = I^{\otimes 2} - 2(|0\rangle\langle 0|)^{\otimes 2}$ and $U_{g_i(x)} = I^{\otimes 2} - 2|\tau_i\rangle\langle\tau_i|$
5: **end for**
6: Let $i = \lfloor n/2 \rfloor - 1$
7: **if** n is even **then**
8: Apply G_i to final pair of qubits
9: **else**
10: Apply $L_i = -H^{\otimes 3} R_0 H^{\otimes 3} R_{g_i(x)}$ twice
11: where $R_0 = I^{\otimes 3} + (e^{i\phi} - 1)(|0\rangle\langle 0|)^{\otimes 3}$ and $R_{g_i(x)} = I^{\otimes 3} + (e^{i\phi} - 1)|\tau_i\rangle\langle\tau_i|$ ▷ See Eq. (5)
12: **end if**
13: Measure all qubits in the computational basis to obtain τ

The Distributed Exact Grover procedure is described in Algorithm 2. To verify the correctness of the Distributed Exact Grover's Algorithm (DEGA), it suffices to show that Algorithm 2 yields the target index string $\tau \in \{0,1\}^n$ exactly [7].

3 Methodology

Our methodology builds upon the approach developed by [6], which demonstrated that human-chosen passwords exhibit predictable patterns regardless of length requirements. Based on this foundation, we introduce our improved attack framework.

3.1 Dictionary Generators

Our implementation adopts the dictionary generator framework of [6] to construct optimized rainbow tables for password recovery. This methodology enables efficient processing of lengthy passwords while preserving human memorability patterns. To maximize occurrence probability, we employ three systematic collection approaches: (1) statistical analysis of compromised password datasets, (2) Markov model-generated strings that capture probabilistic character sequences, and (3) fundamental linguistic elements specific to target demographics.
The structural organization follows identifiable *composition patterns*, where each pattern component serves a distinct function. For example, the "WNS" pattern comprises three elements: a word component (W) such as "pass", a numeric segment (N) like "1234", and special symbols (S) including "$$". This structured approach ensures comprehensive coverage of common password constructions.

3.2 Transform Rules

The framework incorporates configurable transformation rules that systematically modify dictionary entries to match prevalent password variations. Common transformations include case shifting operations (e.g., "pass" to "Pass"), special character substitutions (e.g., "E" to "3"), and complete string reversals (e.g., "well" to "llew"). The system's modular design permits seamless integration of additional transformation rules, enabling precise control over dictionary generation parameters while maintaining computational efficiency.

3.3 Improved Rainbow Table Generation

To achieve high success rates, password recovery typically requires extremely large dictionaries, which presents significant storage challenges. While adopting rainbow table concepts can optimize storage, this approach necessitates redefining the core rainbow table computation functions to accommodate generator-set constraints.
A critical component is the *reduction function* $R : H \rightarrow P$, which maps hash values back to plaintext values. Practical implementations commonly decompose $R(\cdot)$ into two sequential operations for improved efficiency:

1. `HashToIndex(·)`: Transforms a hash value into an index or intermediate representation

2. `IndexToPlain(·)`: Converts the index into a valid plaintext within the target domain

The `HashToIndex` function implementation follows the design specified in Algorithm 3, while `IndexToPlain` operates as detailed in Algorithm 4. Using these components, we generate optimized rainbow tables through the process formalized in Algorithm 5.

Algorithm 3 HashToIndex

Require: Hash H, total plaintext candidates $N = |\mathcal{G}_1| \times |\mathcal{G}_2| \times \cdots \times |\mathcal{G}_k|$
Ensure: Index $i \in \{0, 1, \ldots, N-1\}$
$H_{\text{int}} \leftarrow \text{BinaryToInteger}(H)$ ▷ Convert hash to integer
$i \leftarrow H_{\text{int}} \bmod N$ ▷ Map to plaintext space size
return i

Algorithm 4 IndexToPlain

Require: Index i, generator set $\mathcal{G} = \{G_1, G_2, \ldots, G_k\}$, composition pattern P, transform rules $\mathcal{R}$
Ensure: Target plaintext T
Initialize empty plaintext T
for each generator type $G_j \in P$ **do** ▷ Iterate by pattern order (e.g., "WNS")
    $T_{\text{space}} \leftarrow |G_j|$ ▷ Size of generator subset (e.g., $|G_{\text{words}}| = 399$)
    $n \leftarrow \text{ComputeExtensionRatio}(\mathcal{R}, G_j)$ ▷ e.g., $n = 2$ if $\mathcal{R}$ includes case shifting
    $T_{\text{ext}} \leftarrow T_{\text{space}} \times n$
    $\text{subindex} \leftarrow i \bmod T_{\text{ext}}$ ▷ Local index within extended generator space
    $g_{\text{base}} \leftarrow G_j[\text{subindex} \bmod T_{\text{space}}]$ ▷ Select base generator
    $g_{\text{target}} \leftarrow \text{ApplyTransform}(g_{\text{base}}, \mathcal{R})$ ▷ Apply rules (e.g., "pass" → "P@ss")
    $T \leftarrow T \parallel g_{\text{target}}$ ▷ Append to plaintext
end for
return T

3.4 Bucket Creation

Building on [5]'s bucket concept, we constrain Grover's search space by hashing plaintext to k-bit integers and distributing them into buckets, achieving both endpoint distinction and tractable search complexity (Algorithm 6).

Algorithm 5 Rainbow Table Generation Using Smart Dictionary

Require: Hash function h, generator types $\mathcal{G}$, composition pattern P, transform rules $\mathcal{R}$
Ensure: Rainbow table $\mathcal{T}$

for each chain **do**
  $S \leftarrow$ RandomStartIndex()
  for t iterations **do** ▷ t = chain length
    $T \leftarrow \text{IndexToPlain}(S, \mathcal{G}, P, \mathcal{R})$ ▷ Use Algorithm 4
    $H \leftarrow h(T)$
    $S \leftarrow \text{HashToIndex}(H)$ ▷ Use Algorithm 3
  end for
  Store $(S_{\text{start}}, S_{\text{end}})$ in $\mathcal{T}$
end for
return $\mathcal{T}$

Algorithm 6 Bucket Creation

Require: Plaintext *end*, k-bit hash function k_bit_hash
Ensure: Updated buckets structure

end_hashed $\leftarrow$ k_bit_hash(*end*)
bucket_key $\leftarrow \lfloor \textit{end\_hashed}/k \rfloor$ ▷ Key $\in \{0, \ldots, \lceil 2^k/k \rceil - 1\}$
if *bucket_key* $\notin$ *buckets* **then**
  buckets[*bucket_key*] $\leftarrow$ empty list
end if
buckets[*bucket_key*].append(*end_hashed* mod k) ▷ Offset $\in \{0, \ldots, k-1\}$

3.5 Rainbow Table Search Using DEGA

In a classical rainbow table, the hash is reduced to a plaintext, and a linear search is conducted over a list of plaintexts at the end of the rainbow table chains. In our approach, we also reduce the hash to a plaintext using Algorithms 3 and 4, then proceed to search for it within predefined buckets. First, a classical linear search checks whether the bucket key of the target plaintext exists. If the bucket key is not found in the bucket list, the quantum search is skipped entirely, and the process immediately moves to the previous chain, saving time. If the bucket key exists, we invoke the Distributed Exact Grover's Algorithm (DEGA) to search within the bucket. When DEGA successfully identifies the target, a linear search is performed to locate the corresponding hash, and the chain is reconstructed to retrieve the original plaintext. If DEGA fails to find the result, the process continues with the previous chain. This process repeats until all chains are examined. If no match is found after all iterations, the algorithm concludes that the hash is not present in the rainbow table. The complete rainbow table search using DEGA is outlined in Algorithm 7.

Algorithm 7 Rainbow Table Search with Distributed Exact Grover Search

Require:

H_{target}: Target hash value to recover
$\mathcal{T}$: Rainbow table with:
$\mathcal{T}_{\text{start}}$: Start points $[P_1, \ldots, P_n]$
$\mathcal{T}_{\text{end}}$: End hashes $[h_1, \ldots, h_n]$ (precomputed k-bit hashes)
$\mathcal{B}$: Bucket structure (from Algorithm 6)
k_bit_hash: k-bit hash function
hash: Full cryptographic hash function

Ensure:

Recovered plaintext P or None if not found

$H \leftarrow H_{\text{target}}$
for $i \leftarrow 1$ to max_chain_length **do** ▷ Iterate over possible chain positions
  $h \leftarrow H$
  for $j \leftarrow 0$ to $i - 1$ **do**
    $P \leftarrow \text{IndexToPlain}(\text{HashToIndex}(h), \mathcal{G}, P_{\text{pattern}}, \mathcal{R})$
    if $j < i - 1$ **then**
      $h \leftarrow \text{hash}(P)$
    end if
  end for
  $h_k \leftarrow \text{k\_bit\_hash}(P)$
  bucket_key $\leftarrow h_k \div k$
  if bucket_key $\notin \mathcal{B}$ **then**
    continue
  end if
  lookup $\leftarrow h_k \bmod k$
  result $\leftarrow \text{DistributedExactGroverSearch}(\mathcal{B}[\text{bucket\_key}], \text{lookup})$
  if result = True **then**
    idx $\leftarrow$ index of h_k in $\mathcal{T}_{\text{end\_hashed}}$
    if $P \neq \mathcal{T}_{\text{end}}[\text{idx}]$ **then**
      continue
    end if
    $P_{\text{candidate}} \leftarrow \mathcal{T}_{\text{start}}[\text{idx}]$
    $h_{\text{candidate}} \leftarrow \text{hash}(P_{\text{candidate}})$
    for $m \leftarrow 1$ to max_chain_length $- i$ **do** ▷ Rebuild chain
      $P_{\text{candidate}} \leftarrow \text{IndexToPlain}(\text{HashToIndex}(h_{\text{candidate}}), \mathcal{G}, P_{\text{pattern}}, \mathcal{R})$
      $h_{\text{candidate}} \leftarrow \text{hash}(P_{\text{candidate}})$
      if $h_{\text{candidate}} = H_{\text{target}}$ **then**
        return $P_{\text{candidate}}$
      end if
    end for
  end if
end for
return None

4 Results

This section analyzes our attack's quantum search phase, comparing noise resilience and success rates across Grover's algorithm variants.

4.1 Noise Impact on Grover Variants

To assess the robustness of Grover-based search methods under quantum noise, we evaluate their performance using a depolarizing channel model. Figure 1 compares three approaches: the original Grover's algorithm, a modified variant, and the Distributed Exact Grover's Algorithm (DEGA), testing their performance on bucket searches of size 16.

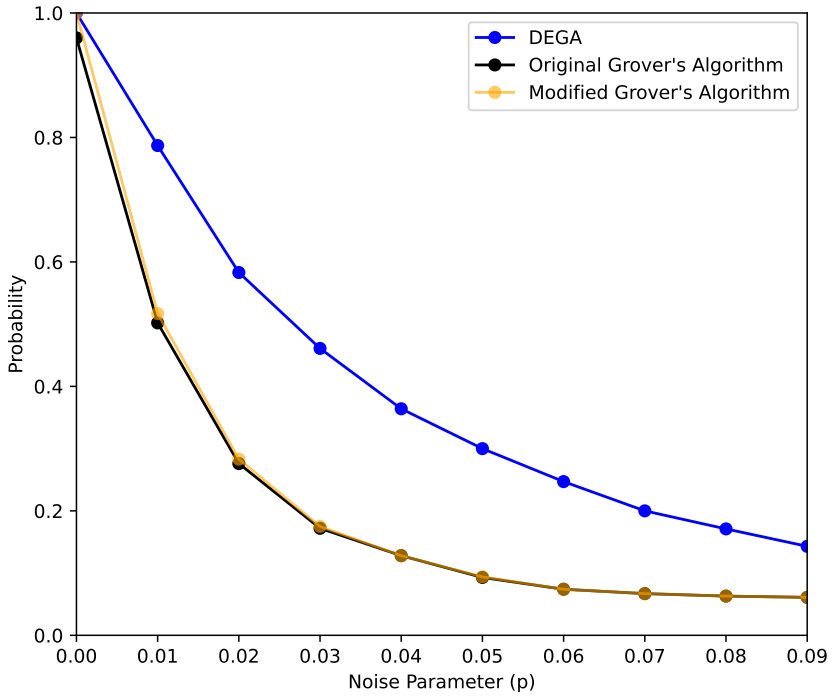

Figure 1: Probability of measuring the target state $\tau = 0011$ for the original Grover's algorithm, modified Grover's algorithm, and DEGA under depolarizing noise.

4.2 Success Probability

Figure 2 compares the success probabilities of the original Grover's algorithm, modified Grover's algorithm, and DEGA across 2–5 qubit systems.

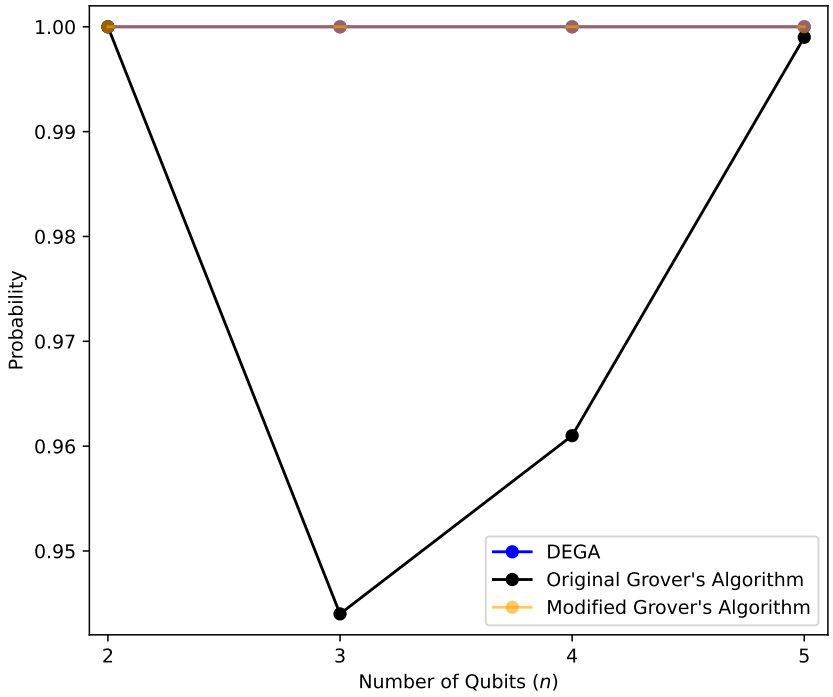

Figure 2: Probability of measuring target states $\tau \in \{11, 001, 1100, 01011\}$ across Grover variants. Both DEGA and the modified Grover's algorithm demonstrate improved exact-match performance compared to the original algorithm.

5 Code Availability

We provide an implementation of the Distributed Exact Grover's Algorithm (DEGA) for our quantum search phase, developed using the PennyLane framework [1]. The implementation includes simulations under both ideal and noisy conditions to evaluate performance impacts on success probability. The complete source code is available at: `https://github.com/w0h4w4d4li/distributed-exact-grover-algorithm`

6 Conclusion

In this work, we introduced a classical-quantum hybrid approach to password recovery that effectively addresses the challenges posed by long, human-generated passwords. By constructing structured rainbow tables using dictionary-based generation and transformation rules, we more accurately capture real-world password patterns. These tables are efficiently organized into buckets, enabling faster and more scalable search operations. To enhance quantum search within each bucket, we employed a distributed, exact variant of Grover's algorithm, which provides deterministic success and reduced circuit depth. This design not only lowers overall quantum resource requirements but also improves resilience

to noise in near-term quantum devices. Our results demonstrate that integrating structured rainbow tables with optimized quantum search significantly improves the efficiency and practicality of password recovery in both classical and quantum settings.

References

[1] Ville Bergholm, Josh Izaac, Maria Schuld, Christian Gogolin, Shahnawaz Ahmed, Vishnu Ajith, M Sohaib Alam, Guillermo Alonso-Linaje, B AkashNarayanan, Ali Asadi, et al. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968*, 2018.

[2] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219, 1996.

[3] Gui-Lu Long. Grover algorithm with zero theoretical failure rate. *Physical Review A*, 64(2):022307, 2001.

[4] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Annual International Cryptology Conference*, pages 617–630. Springer, 2003.

[5] Lee Jun Quan, Tan Jia Ye, Goh Geok Ling, and Vivek Balachandran. Qiris: Quantum implementation of rainbow table attacks. In *International Conference on Information Systems Security*, pages 213–222. Springer, 2024.

[6] Lijun Zhang, Cheng Tan, and Fei Yu. An improved rainbow table attack for long passwords. *Procedia Computer Science*, 107:47–52, 2017.

[7] Xu Zhou, Daowen Qiu, and Le Luo. Distributed exact grover's algorithm. *Frontiers of Physics*, 18(5):51305, 2023.