Quantum Time/Memory/Data Tradeoff Attacks

Orr Dunkelman[1,⋆], Nathan Keller[2,⋆⋆],
Eyal Ronen[3,⋆⋆⋆], and Adi Shamir[4]

[1] Computer Science Department, University of Haifa, Israel
orrd@cs.haifa.ac.il
[2] Department of Mathematics, Bar-Ilan University, Israel
Nathan.Keller@biu.ac.il
[3] Computer Science Department, Tel Aviv University, Israel
eyal.ronen@cs.tau.ac.il
[4] Faculty of Mathematics and Computer Science, Weizmann Institute of Science, Israel
adi.shamir@weizmann.ac.il

Abstract. One of the most celebrated and useful cryptanalytic algorithms is Hellman's time/memory tradeoff (and its Rainbow Table variant), which can be used to invert random-looking functions on N possible values with time and space complexities satisfying $TM^2 = N^2$. In this paper we develop new upper bounds on their performance in the quantum setting. As a search problem, one can always apply to it the standard Grover's algorithm, but this algorithm does not benefit from the possible availability of a large memory in which one can store auxiliary advice obtained during a free preprocessing stage. In fact, at FOCS'20 it was rigorously shown that for memory size bounded by $M \leq O(\sqrt{N})$, even quantum advice cannot yield an attack which is better than Grover's algorithm.

Our main result complements this lower bound by showing that in the standard Quantum Accessible Classical Memory (QACM) model of computation, we can improve Hellman's tradeoff curve to $T^{4/3}M^2 = N^2$. When we generalize the cryptanalytic problem to a time/memory/data tradeoff attack (in which one has to invert f for at least one of D given values), we get the generalized curve $T^{4/3}M^2D^2 = N^2$. A typical point on this curve is $D = N^{0.2}$, $M = N^{0.6}$, and $T = N^{0.3}$, whose time is strictly lower than both Grover's algorithm (which requires $T = N^{0.4}$ in this generalized search variant) and the classical Hellman algorithm (which requires $T = N^{0.4}$ for these D and M).

⋆ The first author was supported in part by the Center for Cyber, Law, and Policy in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office and by the Israeli Science Foundation through grants No. 880/18 and 3380/19.

⋆⋆ The second author was supported by the European Research Council under the ERC starting grant agreement n. 757731 (LightCrypt) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

⋆⋆⋆ The third author is partally supported by Len Blavatnik and the Blavatnik Family foundation, the Blavatnik ICRC, and Robert Bosch Technologies Israel Ltd. He is a member of CPIIS.

1 Introduction

The problem of efficiently inverting a random looking and easy to compute function f is a fundamental problem with numerous applications. In particular, it represents the purest form of cryptanalysis, where $f(x)$ is defined as the ciphertext obtained by encrypting some fixed plaintext p under key x. In this context it is natural to consider the variant in which the inversion process can be assisted by *advice* which is stored in a memory of size M and whose precomputation could take an arbitrarily long time, provided that such a one-time effort of analyzing the cryptosystem would make it easier to find any particular key from its associated ciphertext with a lower online time complexity T.

Due to its importance, this problem had received considerable attention. When the function $f : \{1, 2, \ldots, N\} \to \{1, 2, \ldots, N\}$ is a permutation, Hellman [15] had shown that given M memory, one can invert f using time $T = N/M$ (thus satisfying the time/memory tradeoff curve $MT = N$). Later, Yao [22] proved that this attack was optimal up to logarithmic factors.

When f is a random function rather than a random permutation, the situation is more complicated. In 1979 Hellman [15] had published his well known cryptanalytic attack which used time T and memory M satisfying $TM^2 = O(N^2)$, after preprocessing the cryptosystem in time $O(N)$. In 2003, Oechslin offered a different approach for this attack, obtaining the same tradeoff curve ($TM^2 = O(N^2)$), but claimed improved complexity in real-life scenarios. In 2006, Barkan et al. [3] showed that these attacks are indeed optimal in the classical setting (up to logarithmic factors and within a certain natural model of computation which treats f as a black box).

The problem of lower bounding the complexities of such time/memory tradeoff attacks in the quantum setting was first tackled in 2015 by Nayebi et al. [20], who considered only the case of random permutations. Their paper took into consideration the power offered by quantum algorithms, and particularly Grover's algorithm [14], to offer a lower bound of the form $MT^2 = N$, i.e., any quantum algorithm using M memory needs time $T \geq \sqrt{N/M}$ to succeed with a constant probability to invert the permutation f.

This lower bound was extended to the case of random functions by Hhan et al. [16] and Chung et al. [11] to show a similar lower bound of the form $MT^2 = N$. These results show that to invert a function using precomputed quantum advice, we must have $T \geq \sqrt{N/M}$. Finally, Chung et al. [10] proved that no quantum algorithm with quantum advice and memory less than $\sqrt{N}$ can do better than a simple application of Grover's algorithm to this search problem (the actual bound is $MT + T^2 = \Omega(N)$).

In this paper we consider the question of upper bounds. While we cannot fully match this lower bound, we show how to obtain from either the Hellman or the Rainbow attack variants new quantum time/memory tradeoff attacks on the function inversion problem whose time and memory complexities satisfy the relationship $T^{4/3}M^2 = N^2$. Note that this tradeoff attack offers strictly better online time complexity than Grover's algorithm for any $M > N^{2/3}$. Finally, we consider the more general time/memory/data tradeoff attacks, in which we

are given D possible values and have to invert f on at least one of them, and describe a new quantum attack whose complexities satisfy $T^{4/3}M^2D^2 = N^2$. The novelty of our results can be demonstrated by the following quote from [16] which appeared at Asiacrypt 2019:

> "Basically, the best known attacks we are aware of are to just apply quantum attacks without auxiliary inputs (i.e., Grover's algorithm...) or best known classical attacks that make use of auxiliary inputs (i.e., Hellman's attack...). Though quantum attacks possibly exist that utilize classical auxiliary inputs that achieve better bounds than classical ones, we are not aware of such an algorithm for these primitives."

It is interesting to note that in the classical setting, there is no advantage to time/memory(/data) tradeoff attacks over exhaustive search before the memory is sufficiently large (at least $\sqrt{N/D}$). Similarly, in the quantum setting, our algorithms need memory of at least $(N/D)^{2/3}$ in order to offer an attack which is faster than Grover.

A summary of all the known and new results can be found in Table 1.

Algorithm	Time-Memory-Data Curve	Restrictions
Classical Hellman [15,6]	$N^2 = T \cdot M^2 \cdot D^2$	$T \geq D^2$
Classical Rainbow [21,3]	$N^2 = T \cdot M^2 \cdot D^2$	$T \geq D^2$
Grover (Sect. 3.2)	$N^2 = T^4 \cdot D^2$, $M = O(1)$	None
Quantum Hellman (Sect. 4)	$N^2 = T^{4/3} \cdot M^2 \cdot D^2$	$T \geq D^{1.5}$
Quantum Rainbow (Sect. 5)	$N^2 = T^{4/3} \cdot M^2 \cdot D^2$	$T \geq D^{1.5}$

Table 1: Comparison of Time-Memory(-Data) Tradeoff Attacks

1.1 Our Quantum Memory Model

Quantum algorithms can be categorized into one of three main models, depending on the amount and type of memory they used. In addition to standard qubits, one can also consider classical memory which can be probed in a quantum manner. This type of memory is based on quantum RAM (qRAM) gates that support a query (requesting a given memory cell from the memory) which in itself is a superposition, without collapsing the state. In other words, in the qRAM model, one can take a superposition of indices $|i\rangle$ and access an array $a[\cdot]$ using the superposition, to accept a superposition of answers. For example, in [8] the quantum collision finding algorithm tests for each value of x (given in a superposition) whether $f(x)$ appears in a large table $a[\cdot]$, without collapsing the superposition.

Hence, the three main models (we adopt the names and conventions of [19]) are:

- *Low-qubits*: algorithms that require a small amount of quantum bits, e.g., $O(n)$. Any amount of classical memory can be available to the algorithm, but the algorithm has only classical access to it. Results such as Grover's algorithm [14] or those of [13] fall into this category.
- *QACM (quantum-accessible classical memory)*: in addition to a small amount of true qubits, the algorithm can use qRAM gates that allow access to classical data using superposition of a quantum state. As mentioned before, the collision finding algorithm of [8] uses this model. This model is considered more realizable than the next model — QAQM [18].
- *QAQM (quantum-accessible quantum memory):* such algorithms enjoy as many qubits as needed. All the data is thus accessed, stored, and processed in quantum memory. Obviously, this is the strongest possible model. An example to such an algorithm is the unique collision finding algorithm proposed in [1].

The model we use in this paper is the QACM model. This model is used in many quantum-based cryptanalytic papers, such as [4], [8], [9], [13], [17] and [19]. It seems to be the "correct" model for our type of cryptanalytic attacks for the following reasons:

- Hellman-like attacks require a huge amount of precomputed data. Such a large database cannot be handled in the Low-qubits model, and requires an impractical number of high quality qubits (with extremely long decoherence times) in the QAQM model.
- The preparation of such a database requires a huge amount of time, which can only be justified if we amortize this cost among many independent attempts to find multiple cryptographic keys (note that to find a single key, we can simply use Grover's search algorithm to get a square root improvement without performing any preprocessing). When we store this database in an array of qubits in the QAQM model, measuring the database in one attack may collapse its quantum state, stopping us from using it in the next attack. This can not happen if the database is classically stored in the QACM model.
- It may take many years before we have access to sufficiently powerful quantum computers. Powerful attackers can use this interim period to prepare the classical database needed to apply a Hellman-like attack. Such a database can then be used "as is" in a future QACM computer to speed up the actual attacks on particular keys.

1.2 Organization of this paper

This paper is organized as follows: In Section 2 we recall the classical time/memory (/data) tradeoff attacks on cryptographic schemes. In Section 3 we recall the collision finding attack of Brassard et al. and discuss how it can be used to solve multiple-data inversion problem using Grover's algorithm. In Section 4 we develop the quantum variant of Hellman's attack, and in Section 5 we develop the quantum variant of the Rainbow attack. In Section 6, we compare our attacks with Grover's algorithm. Finally, Section 7 concludes this paper.

2 Classical Time-Memory-Data Tradeoff Attacks

The problem of inverting a (pseudo-random) function

$$f : \{1, 2, \ldots, N\} \to \{1, 2, \ldots, N\}$$

has two trivial solutions: exhaustive search (in time $O(N)$ and $O(1)$ memory) and a table attack (in memory $O(N)$ and time $O(1)$, given a preprocessing of $O(N)$ time).[1]

A way to bridge between these complexity extremes was presented by Hellman in [15]. While the precomputation is still $O(N)$, the online time complexity T and the memory complexity M can vary along the curve $TM^2 = N^2$, as long as $M \geq \sqrt{N}$. Hellman's attack is based on precomputing many tables, and then searching them efficiently during the online phase. Later, Oechslin presented a different method for constructing the tables [21], called *Rainbow Tables*, along with a slightly modified way of searching them during the online phase.

We recall Hellman's attack in Section 2.1 and the Rainbow tables in Section 2.2. Finally, we recall the case of multiple data variants of the two attacks in Section 2.3. Readers who are familiar with these techniques can safely skip these three subsections.

2.1 Hellman's Attack

The main tool in Hellman's attack is the use of *chains*, defined by the iterative application of $f(\cdot)$ (i.e., $x, f(x), f(f(x)), \ldots$). In the original work of Hellman, the chains, computed during the preprocessing are of a predetermined length t (we later discuss the value of t). The key idea here is that the adversary can store the starting point and the end point (i.e., $(x, f^t(x))$), and then recover the full chain, if needed, by iteratively applying $f(\cdot)$ to x.

A well known improvement to the algorithm, due to Rivest, called *distinguished points*, stops the chain once the computed value is a special point. This special point needs to have an easy to characterize property, e.g., with $\log_2(t)$ least significant bits equal to 0. It is easy to see that on average the length of the chain is t, and thus for the sake of analysis we assume that indeed this is the case.

We note one technical point with distinguished points — the chains are bounded in length. In other words, if we start from some value x, and compute a chain of some length (it is common to pick $8t$ as this length[2]) without reaching a distinguished point, we call this a failure. If this happens during the preprocessing time, a different starting point may be chosen for generating a new chain. If this happen during the online phase of the attack, then the attack fails for the given input (but in the case of multiple data points, then another

[1] Throughout the paper we disregard logarithmic factors.

[2] The probability of a chain of length $8t$ in a random function to not contain a distinguished point, when a random point is a distinguished point with probability $1/t$ is $(1 - 1/t)^{8t} \approx 1/e^8$. Obviously, picking a larger limit decreases this failure rate.

Algorithm 1: The Preprocessing Algorithm for Hellman's Time-Memory Tradeoff Attack

```
for ℓ = 1 to t do
    Initialize an empty hash table T_ℓ.
    Pick m starting points x_{ℓ,1}, x_{ℓ,2}, ..., x_{ℓ,m}.
    for j = 1 to m do
        Set tmp ← x_{ℓ,j}
        for i = 1 to 8t do
            Set tmp ← f_ℓ(tmp)
            if tmp is a distinguished point then
                Set y_{ℓ,j} = tmp.
                Store (x_{ℓ,j}, y_{ℓ,j}) in the hash table T_ℓ (indexed according to
                  y_{ℓ,j}).
                Break from the loop.
        if tmp is not a distinguished point then
            try a different x_{ℓ,j}
```

point can be tried). While for classical algorithms this issue is a relatively small technical detail, for quantum algorithms, it becomes an important one (as it puts a limit on the depth of the circuit).

Preprocessing In the preprocessing phase, the adversary constructs several tables. Each table is constructed by picking m starting points, $x_1, x_2, \ldots, x_m$. From each starting point, x_i, the adversary computes the chain $x_i \to f(x_i) \to f^2(x_i) = f(f(x_i)) \to \cdots \to y_i = f^t(x_i)$.[3] The pairs (x_i, y_i) are stored in a hash table indexed by the value of y_i.

To avoid collisions between the chains and the tables, Hellman's attack uses t flavors of the function $f(\cdot)$. Namely, let $f_\ell(x) = f(L_\ell(x))$ for some invertible function L_ℓ. There is a table T_ℓ $(1 \leq \ell \leq t)$ for each such function $f_\ell(\cdot)$. The preprocessing phase is given in Algorithm 1.

It is easy to see that the preprocessing takes $N = mt^2$ time, and uses $M = mt$ memory.

The Online Phase In the online phase of the attack, the adversary is given $y = f(x)$ and she wishes to find x. This is done by building a chain from y (under the t different flavors), and checking whether the chain results in an end point y_i stored in the table. Once such an end point is found, the stored starting point x_i is recovered from the table, and the adversary can compute from x_i the chain until reaching y. With constant probability, the chain indeed recovers y. The steps (when using distinguished points) are described in Algorithm 2.

[3] When using distinguished points, the value of y_i is the first distinguished point encountered in the iterative application of $f(\cdot)$.

Algorithm 2: The Online Algorithm for Hellman's Time-Memory Tradeoff Attack

```
for ℓ = 1 to t do
    Set tmp ← y
    for i = 1 to 8t do
        Set tmp ← f_ℓ(tmp)
        if tmp is a distinguished point then
            Break from the loop.
    if tmp is not a distinguished point then
        Output "Failure"
    if tmp is an endpoint stored in T_ℓ then
        Let y_i = y be the end point
        Fetch x_i, the corresponding starting point, from T_ℓ.
        Set tmp ← x_i
        Set tmp2 ← x_i
        Set tmp = f_ℓ(tmp)
        while tmp ≠ y and tmp not a distinguished point do
            Set tmp2 = tmp Set tmp = f_ℓ(tmp)
        if f_ℓ(tmp2) = y then
            Output tmp2
```

Algorithm 3: Constructing the Rainbow Table

```
Pick mt starting points x_1, x_2, ..., x_mt.
for j = 1 to m do
    Compute y_j = f_t(f_{t-1}(··· f_2(f_1(x_j)) ···)).
    Store (x_j, y_j) in the table (indexed according to y_j).
```

The running time of the attack algorithm is $T = t^2$. We recall that the amount of memory is $M = mt$, and that $N = mt^2$. Hence, (up to logarithmic factors) we have $T \cdot M^2 = N^2$. We alert the reader that when $M < \sqrt{N}$ the online running time of this algorithm is worse than exhaustive search.

2.2 Rainbow Tables

In 2003, Oechslin presented a different method to construct the tables [21]. In this method, a single table (called a *Rainbow table*) is constructed. This time, m starting points are chosen, and the constructed chains are of the form $x \to f_1(x) \to f_2(f_1(x)) \to f_3(f_2(f_1(x))) \to \cdots \to f_t(\ldots(f_1(x)))$, where the functions $f_\ell(\cdot)$ are of the same type used in Hellman's scheme (i.e., a small variation of $f(\cdot)$) and t is the number of different functions as in Hellman's attack. The preprocessing step of generating the table in given in Algorithm 3. This technique reduces the effects of false alarms, and allows to cover most of the search space by a single table.

Algorithm 4: The Online Algorithm for Using the Rainbow Table

```
for ℓ = t downto 0 do
    Compute y' = f_t(f_{t-1}(··· f_{ℓ+2}(f_{ℓ+1}(y))···))
    if y' is an end point stored in the table then
        Fetch the starting point x_j from the table
        Set x' = x_j
        for i = 1 to ℓ − 1 do
            Compute x' = f_i(x')
        if f_ℓ(x') = y then
            Output x'
```

The online phase of the attack, described in Algorithm 4, is slightly different than in Hellman's attack. First, the adversary checks whether $f_t(y)$ appears as an endpoint in the table. If not, she computes $f_t(f_{t-1}(y))$ and checks whether this value appears in the table. If not, she computes $f_t(f_{t-1}(f_{t-2}(y)))$, and so forth. Once an endpoint is encountered, the corresponding chain is computed from the respective starting point.

The online running time of the rainbow table attack is about $T = t^2/2$ calls to $f(\cdot)$ (as well as t accesses to the database). The memory requirement of the rainbow table $M = m$. Since, most of the states are covered by the single table, we have $N \approx mt$, and hence the obtained tradeoff curve is $N^2 = 2TM^2$.

A more complete analysis and comparison of the two attacks is available at [3].

2.3 Time-Memory-Data Tradeoff Attacks

In some cases it may be possible to use multiple data points in time-memory tradeoff attacks. The adversary is given a set of D points $y_1 = f(x_1), y_2 = f(x_2), \ldots, y_D = f(x_D)$, such that $f(x_i) = y_i$, and is asked to find a pre-image of *one* of the y_i's.

The main advantage comes from the fact that the precomputation tries to cover only N/D states, and thus, with constant probability, one of the y_i is covered (either by the Rainbow or the Hellman tables). Then, the online phase of the attack is repeated for any y_i value.[4]

For Hellman's attack, this is done by generating t/D tables [6]. This means that for each of the D data points, the attacker tries t/D flavors, where each such trial takes time t (on average). The resulting online time complexity is thus t^2 (as in Hellman's original attack), but the memory complexity is reduced to $M = mt/D$. Up to the requirement that $t \geq D$ (which stems from the fact that

[4] In some cases related to stream ciphers, this process is repeated only for a single y_i [12].

Algorithm 5: The Preprocessing Algorithm for Time-Memory-Data Rainbow Attack

```
Initialize an empty rainbow table T.
Pick mt/D starting points x_1, x_2, ..., x_{mt}.
for j = 1 to mt do
    Set tmp ← x_j
    for i = 1 to 8t/S do
        Set tmp ← f_S(f_{S-1}(··· f_2(f_1(tmp))···))
        if tmp is a distinguished point then
            Set y_j = tmp.
            Store (x_j, y_j) in the rainbow table T (indexed according to y_j).
            Break the loop.
    if tmp is not a distinguished point then
        try a different x_j
```

there is at least one table[5]), the multiple-data variant of the attack obtains the curve $N^2 = T \cdot M^2 \cdot D^2$ (as long as $T \geq D^2$).

For Rainbow tables, one can use the straightforward approach of reducing the number of functions by a factor of D, proposed in [5]. The resulting curve is $N^2 = T \cdot M^2 \cdot D$, as long as $T \geq D$. However, a more efficient algorithm, offering the curve $N^2 = T \cdot M^2 \cdot D^2$ was proposed in [3]. We now quickly describe the algorithm in its distinguished point version (which is equivalent to the regular description). For more information about the algorithm and the tradeoffs we refer the interested reader to [2].

To achieve the full potential of multiple data in the rainbow attack a *basic unit* is of S flavors (the value of S is discussed later), i.e., $f_S(f_{S-1}(\cdots f_2(f_1(x))\cdots))$. The chains are built as t/S iterations of the basic unit, which consists of t functions in total. Again, as we describe the distinguished point variant of the algorithms, chains end when the output of a basic unit (i.e., of f_S) has $log_2(t/S)$ least significant bits set to 0.

The pre-processing algorithm is quite straightforward adaptation of the rainbow tables one to use the basic unit idea. It is given in Algorithm 5. The online algorithm is slightly different, and due to the use of distinguished points may look similar to the case of Hellman's online phase. Specifically, given a point y, we first apply $f_S(\cdot)$ to it (similarly to the regular Rainbow attack). If the result is not a distinguished point, a series of basic units is applied to $f_S(\cdot)$, until a distinguished point is reached. Once such a point is achieved, we check the stored table whether this distinguished point is an end point. If so, we "jump" to the starting point. Otherwise, we take the *same* point y, and repeat the process after first applying $f_S(f_{S-1}(y))$, if this fails, the algorithms is repeated after applying $f_S(f_{S-1}(f_{S-2}(y)))$, and so forth. The algorithm is described in Algorithm 6.

[5] Reducing the size of a Hellman table below m rows, for $m = N/t^2$ offers sub-optimal attack.

Algorithm 6: The Online Algorithm for Time-Memory-Data Rainbow Attack

Input: $y_1, y_2, \ldots, y_D$ data points.
for *i=1 to D* **do**
  Set $tmp = y_i$
  for $j = S$ *downto* 1 **do**
    Compute $tmp = f_S(f_{S-1}(\cdots f_{j+1}(f_j(tmp))\cdots))$
    if *tmp is not a distinguished point* **then**
      for $k = 1$ *to* $8t/S - 1$ **do**
        Compute $tmp = f_S(f_{S-1}(\cdots f_2(f_1(tmp))\cdots))$
        if *tmp is a distinguished point* **then**
          Break
    if *tmp is an end point stored in the table* **then**
      Fetch the starting point x from the table
      Set $tmp2 = x$
      Compute $tmp2 = f_{j-1}(f_{j-2}(\cdots f_2(f_1(tmp2))\cdots))$
      if $f_j(tmp2) = y$ **then**
        Output $tmp2$
      Compute $tmp2 = f_S(f_{S-1}(\cdots f_{j+2}(f_{j+1}(tmp2))\cdots))$
      while $tmp2$ *is not a distinguished point* **do**
        Compute $tmp2 = f_{j-1}(f_{j-2}(\cdots f_2(f_1(tmp2))\cdots))$
        if $f_j(tmp2) = y$ **then**
          Output $tmp2$
        Compute $tmp2 = f_S(f_{S-1}(\cdots f_{j+2}(f_{j+1}(tmp2))\cdots))$

While the analysis of the pre-processing algorithm is relatively straightforward (m chains of expected length t such that $mt = N/D$ are generated), the analysis of the online algorithm is a bit more involved. We briefly reproduce the analysis suggested in [2,3] as it is needed for understanding the complexity of the quantum variants. Full details are given in [2]. In the online phase, each data point is tested with S different starting positions. For each such starting position, a sequence of functions (from f_j the first function till f_S) is applied, and we check whether the result is a distinguished point. Then, a basic unit (of S functions) is applied, and again we test whether we obtained a distinguished point. It is easy to see that if the distinguishing property of the point is that the $\log_2(t/S)$ least significant bits are 0, then the chain ends after t/S basic units, i.e., after t values were encountered for a given data point. Hence, the online time complexity of the attack is $T = D \cdot t$.

The only factor which need to be determined is the value of S. As shown in [2], the optimal value of S is t/D. This follows from the matrix stopping rule — each chain contains t values, and after m chains, there are mt covered points. Each of the t values in the new chain can collide only with mt/S locations (as the collision has to be with the same flavor). The result is a stopping rule of $N = mt \cdot t/S$. As the total cover is $N/D = mt$, we get $mt^2/S = Dmt$. In other

Algorithm 7: $GroverMatch(G, T, (\mathcal{D}, \mathcal{F}lavors))$ the Grover Match Algorithm

Input: A function G, QACM sorted table T of size M free of internal collisions of chain values $(flavor, start_i, end_i)$, a superposition of data-points $d \in \mathcal{D}$ and a superposition of flavors $flavor \in \mathcal{F}lavors$.
Compute $(d, flavor) = Grover(H)$ for

$$H(x) = \begin{cases} 1 & \text{iff there exists } (start_i, end_i) \in T \text{ s.t. } G(d, flavor) = end_i \\ 0 & \text{Otherwise} \end{cases}$$

Output $(d, flavor)$.

words, $S = t/D$ is the requirement for acheiving the time-memory-data tradeoff variant of the Rainbow table.

3 Basic Quantum Algorithms

3.1 Grover Match Algorithm

We take Brassard et al.'s [8] original algorithm (Step 4 of the algorithm, specifically), and re-write it as the *GroverMatch*. This algorithm identifies whether there is a match between a list of values held in a quantum superposition, and the classical list stored in a device which allows quantum access classical memory (QACM). The algorithm is described in Algorithm 7. The analysis as well follows the footsteps of [8].

Let $H(x)$ be the function that tells whether a value $G(x)$ appears in a given database. In other words, $H(x) = 1$ if $G(x)$ is indeed in the database, or otherwise, $H(x) = 0$. When there is a single data-point x, for which $H(x) = 1$, the number of iterations needed by Grover to find that x value is the square root of the size of the search space $\mathcal{D}$. If the function G has a running time $\mathcal{T}$, then the overall run-time complexity of the *GroverMatch* algorithm is $\mathcal{T} \cdot |\mathcal{D}|^{0.5}$.

It should be noted that when the number of points for which $H(x) = 1$ is 0, then Grover is going to suggest in each iteration some random x value (for which $H(x) = 0$. Hence, one can easily detect whether there are no solutions in the table with high probability after a few repetition of Grover's algorithm that result in x values for which $H(x) = 0$). When there are k multiple solutions, as suggested in [7] one should expect each invocation of Grover's algorithm to produce one random x for which $H(x) = 1$ after $\sqrt{N/k}$ iterations. Hence, one could repeat Grover's search about $k \log(k)$ times to obtain all k solutions. As in our cases k is expected to be 1, this results in a logarithmic factor in the total complexity. When k is unknown in advance, we can use the method proposed in [7] that performs binary search-like partitioning looking for k. Again, the increase in the running time is by a logarthmic factor, and thus, we disregard its impact on our analysis.

3.2 Grover with D Targets

It is well known that running Grover on a search space of size N with D possible inputs for which the function $f(x) = 1$ (whereas for the other $N - D$ values, $f(x) = 0$) takes time $O(\sqrt{N/D})$ for finding one possible input. However, in cryptographic settings with multiple data points there are multiple inputs y_i, where only one needs inversion.

These are different y_i, and thus, a straightforward Grover may not be suitable. This follows the fact that the function

$$g(x) = \begin{cases} 1 & f(x) = y_1 \vee y_2 \vee y_3 \vee \ldots \vee y_D \\ 0 & \text{Otherwise} \end{cases}$$

has a circuit size of $O(|f| + |D|)$. Hence, the gain offered by the fact that there are multiple solutions is offset by the increased circuit size.

However, the GroverMatch algorithm can be used almost in a straightforward way to obtain the gain. By storing a table with the D possible y_i's and running GroverMatch on the search space takes time $O(\sqrt{N/D})$ as there are D values for which the matching function evaluates to 1.

4 Quantum Hellman Tables

Our quantum version for the Hellman Tables algorithm provides a significant speed-up over the classical version. In the classical version, we need to separately test each of the D data-points on each of the t/D flavors. However, in the quantum version, we can use the $GroverMatch$ algorithm to test the superposition of the $D \cdot t/D = t$ possible combinations, at a run-time cost of only $\sqrt{t}$.

4.1 Offline Preprocessing Phase

In our quantum algorithm, we prepare the hash tables for the different flavors using the same preprocessing algorithm used in the classical version and described in Algorithm 1. We store the resulting table $T = T_0||T_1||..||T_{t/D}$ in a QACM that can be accessed by our $GroverMatch$ algorithm.

4.2 Finding Distinguished Point

In the classical version, for each data-point d and each flavor ℓ, we iterate over the chain for up to $8t$ function invocations but break when we hit a distinguished point. However, when using a quantum function implemented as a circuit, we must use the same number of function invocations, regardless of the inputs. The function $F_{Hellman}$ described in Algorithm 8 fulfills this requirement in the following way:

1. The function is always iterated $8t$ times regardless of the inputs.

Algorithm 8: $F_{Hellman}(\mathcal{D}, \mathcal{F}lavors)$ The Quantum function used in the Hellman online phase

Input: a superposition of data-points $d \in \mathcal{D}$ and a superposition of flavors $\ell \in \mathcal{F}lavors$.

Set $d' = d$
for $i = 1$ *to* $8t$ **do**
  $tmp = f_\ell(d')$
  if d' *is a distinguished point* **then**
    Set $d' = d'$
  else
    Set $d' = tmp$
Output d'

2. In each iteration, we always apply f_ℓ on the current value but store it in a temporary variable. If the current value is a distinguished point, we keep it. Otherwise, we replace it with the value stored in the temporary variable.[6]

For each data-point and flavor, if the resulting chain of length $8t$ contains a distinguished point with high probability., the function will return that point (or the end of the chain if no distinguished point is found). As the function receives an input which is a superposition of initial data-points and flavors, it returns a superposition of distinguished points with time complexity of $O(t)$.

4.3 Online Phase

The online phase of our quantum Hellman Tables algorithm is described in Algorithm 9. We know that with a constant non-negligible probability we have at least one combination of a data-point d and a flavor ℓ that is covered by table T. That means the distinguished point reached by the chain started at d with flavor ℓ is stored as an endpoint in T. We use the $GroverMatch$ algorithm, with the function $F_{Hellman}$ to find that data-point and flavor. $F_{Hellman}$ converts the superposition of data-points and flavors into a superposition of distinguished points, and the $GroverMatch$ algorithm amplifies the amplitude of the combination of d and ℓ that T covers.

While the $GroverMatch$ algorithm returns the covered data-point and flavor, it does not return the start and end points of the covering chain or the preimage. Our algorithm uses a classical computation to find the chain by simply iterating over the chain until we reach a distinguished point. We then find the corresponding start and end points from table T_ℓ. To recover the preimage for d, we iterate over the chain that begins in the starting point we found until we reach d.

[6] In practice, this is equivalent to constant time implementation of a conditional move operation. This can be implemented in a circuit using only logical gates without a temporary variables.

Algorithm 9: The Quantum Hellman Tables Online Phase

Input: a superposition of data-points to invert $d \in \mathcal{D}$, a superposition of flavors $\ell \in \mathcal{F}lavors$, QACM sorted table T of size M free of internal collisions of chain values $(start_i, end_i)$.

```
// We use GroverMatch to find the data-point d that is covered by
   table T and the specific flavor ℓ that covers the data-point.
```
$(d, \ell) = GroverMatch(F_{Hellman}, T, (\mathcal{D}, \mathcal{F}lavor))$
```
// From this point onwards we only use classical computations.
// We start by finding specific chain in T_ℓ that covers d and
   retriving the corresponding starting point.
```
$d' = d$
for $j = 1$ *to* $8t$ **do**
  if d' *is a distinguished point* **then**
    find $(start_i, end_i) \in T_\ell$ s.t. $d' = end_i$
    $start' = start_i$
    Break
  else
    Set $d' = f_\ell(d')$
```
// Using the recovered starting point and flavor, we can find the
   preimage of the data-point d.
```
for $i = 1$ *to* $8t$ **do**
  if $f_\ell(start') = d$ **then**
    Break
  else
    Set $start' = f_\ell(start')$
  Output $(start', \ell)$

4.4 Complexity of the algorithm

Similar to the classical version of the algorithm, for D data-points, we generate t/D tables. Each table contains m chains of length $8t$, such that omitting constants, $N/D = m \cdot t^2/D$, and the overall memory requirement is $M = m \cdot t/D$.

As inputs to the algorithm, we have a superposition of D data-points and t/D flavors, so the resulting size of the search space for out $GroverMatch$ algorithm is $D \cdot t/D = t$. As the time complexity of the $F_{Hellman}$ function is $O(t)$, the total time complexity of the call to the $GroverMatch$ algorithm is $T = t \cdot \sqrt{t} = t^{1.5}$. The final classical computation time complexity is $O(t)$ and can be neglected.

We get the following constraints:

$$N/D = m \cdot t^2/D$$
$$M = m \cdot t/D$$
$$T = t \cdot \sqrt{t} = t^{1.5}$$

Using these constraints we get the following time memory trade-off curve:

$$
\begin{aligned}
&N/D = m \cdot t^2/D \\
&N^3/D^3 = m^3 \cdot t^6/D^3 = t^3 \cdot (m \cdot t/D)^3 = T^2 \cdot M^3 \\
&N^3 = T^2 \cdot M^3 \cdot D^3 \\
&N^2 = T^{4/3} \cdot M^2 \cdot D^2
\end{aligned}
$$

4.5 Restrictions

Note that as we need to have at least one table, we get the following constraint:

$$
\begin{aligned}
&t/D > 1 \\
&t > D \\
&t^{1.5} > D^{1.5} \\
&T > D^{1.5}
\end{aligned}
$$

5 Quantum Rainbow Tables

Our quantum version of the Rainbow Tables algorithm is based on similar principles as the quantum Hellman Tables algorithm and achieves a similar time complexity improvement. Again, while the classical algorithm needs to test each of the D data-points on each of the t/D flavors separately, using the $GroverMatch$ algorithm, we test the superposition of $D \cdot t/D = t$ combinations, at a run-time cost of only $\sqrt{t}$.

5.1 Offline Preprocessing Phase

In the offline phase, we prepare the hash table using the same preprocessing algorithm used in the classical version and described in Algorithm 5. We store the resulting table T in a QACM that our $GroverMatch$ algorithm can access.

5.2 Finding Distinguished Point

Similar to the $F_{Hellman}$ described in Algorithm 8, we need to use a quantum function that can be implemented in a circuit with a fixed number of function invocations. As in the case of Hellman Table, we need to stop when hitting a distinguished point. Moreover, in the Rainbow Table algorithm, the beginning point of the chain is not fixed but is determined by the flavor. in Algorithm 10, we fulfill these requirements in the following way:

1. The function is always iterated $8t$ times regardless of the inputs.

Algorithm 10: $F_{Rainbow}(\mathcal{D}, \mathcal{F}lavors)$ The Quantum function used in the Rainbow Table Online Phase

Input: a superposition of data-points $d \in \mathcal{D}$ and a superposition of flavors $\ell \in \mathcal{F}lavors$.

```
Set d' = d
for i = 1 to S do
    tmp = f_i(d')
    if i < ℓ then
        Set d' = d'
    else
        Set d' = tmp
for i = 1 to 8t/S do
    tmp = f_S(f_{S-1}(··· f_2(f_1(d')) ···))
    if d' is a distinguished point then
        Set d' = d'
    else
        Set d' = tmp
Output d'
```

2. In the first S invocations, we apply f_i on the current value but store it in a temporary variable. If we didn't reach the starting flavor ℓ, we keep the current value. Otherwise, we replace it with the value stored in the temporary variable.[7]
3. After the first S invocation, we apply the S flavors on the current value using S function invocations but store it in a temporary variable. If the current value is a distinguished point, we keep it. Otherwise, we replace it with the value stored in the temporary variable.

5.3 Online Phase

The online phase of our quantum Rainbow Tables algorithm is described in Algorithm 11. We know that with a constant non-negligible probability we have at least one data-point d that is covered by table T. That means the distinguished point reached by the chain started at d starting with some flavor ℓ is stored as an endpoint in T. We use the $GroverMatch$ algorithm, with the function $F_{Rainbow}$ to find that data-point and flavor. $F_{Rainbow}$ converts the superposition of data-points and flavors into a superposition of distinguished points, and the $GroverMatch$ algorithm amplifies the amplitude of the combination of d and staring flavor ℓ that T covers.

As before, while the $GroverMatch$ algorithm returns the covered data-point and flavor, it does not return the start and end points of the covering chain or the

[7] As described in Section 4.2, this can be implemented in a circuit using only logical gates without a temporary variables.

Algorithm 11: The Quantum Rainbow Table Online Phase

Input: a superposition of data-points to invert $d \in \mathcal{D}$, a superposition of flavors $\ell \in \mathcal{F}lavors$, QACM sorted table T of size M free of internal collisions of chain values $(start_i, end_i)$.

```
// We use GroverMatch to find the data-point d that is covered by
   table T and the specific flavor ℓ offset of the data-point in
   the covering chain.
```
$(d, \ell) = GroverMatch(F_{Rainbow}, T, (\mathcal{D}, \mathcal{F}lavor))$
```
// From this point onwards we only use classical computations.
// We start by finding specific chain in T that covers d and
   retriving the corresponding starting point.
```
$d' = d$
for $i = \ell$ *to* S **do**
  $d' = f_i(d')$
for $i = 1$ *to* $8t/S$ **do**
  if d' *is a distinguished point* **then**
    find $(start_i, end_i) \in T$ s.t. $d' = end_i$
    $start' = start_i$
    Break
  else
    Set $d' = f_S(f_{S-1}(\cdots f_2(f_1(d'))\cdots))$
```
// Using the recovered starting point and flavor, we can find the
   preimage of the data-point d.
```
for $i = 1$ *to* ℓ **do**
  $start' = f_i(start')$
for $i = 1$ *to* $8t/S$ **do**
  for $j = 1$ *to* S **do**
    $tmp = f_j(start')$ **if** $tmp = d$ **then**
      Break `// Only occurs when` $j = \ell$.
  else
    Set $start' = f_S(f_{S-1}(\cdots f_2(f_1(start'))\cdots))$

Output $(start', \ell)$

preimage. Our algorithm uses a classical computation to find the chain by simply iterating over the chain starting from flavor ℓ until we reach a distinguished point. We then find the corresponding start and end points from table T. To recover the preimage for d, we iterate over the chain that begins in the starting point we found until we reach d.

5.4 Complexity of the algorithm

Similar to the classical version of the algorithm, for D data-points, we generate a single table. The table contains m chains of length $8t$ with $S = t/D$ different

flavors. Omitting constants, we get that $N/D = m \cdot t$, and the overall memory requirement is $M = m$.

As inputs to the algorithm, we have a superposition of D data-points and t/D flavors, so the resulting size of the search space for out $GroverMatch$ algorithm is $D \cdot t/D = t$. As the time complexity of the $F_{Rainbow}$ function is $O(t)$, the total time complexity of the call to the $GroverMatch$ algorithm is $T = t \cdot \sqrt{t} = t^{1.5}$. The final classical computation time complexity is $O(t)$ and can be neglected.

We get the following constraints:

$$\begin{aligned} N/D &= m \cdot t \\ M &= m \\ T &= t \cdot \sqrt{t} = t^{1.5} \end{aligned}$$

Using these constraints we get the following time memory trade-off curve:

$$\begin{aligned} N/D &= m \cdot t \\ N^3/D^3 &= t^3 \cdot m^3 = T^2 \cdot M^3 \\ N^3 &= T^2 \cdot M^3 \cdot D^3 \\ N^2 &= T^{4/3} \cdot M^2 \cdot D^2 \end{aligned}$$

5.5 Restrictions

Note that as we need to have at least one flavor, resulting in the same constraint described in Section 4.5:

$$\begin{aligned} t/D &> 1 \\ t &> D \\ t^{1.5} &> D^{1.5} \\ T &> D^{1.5} \end{aligned}$$

6 Comparison with Grover's Algorithm

In classical settings, we have to make sure that the complexities of tradeoff attacks are better than the generic exhaustive search. In the quantum setting, we have to compare the complexities of the tradeoff attacks with the complexity of generic Grover search instead.

Although our quantum algorithms are valid at any point where $T > D^{3/2}$, we are only interested in the range of parameters where we are faster than Grover. As Grover does not use any memory, we can start by finding the point where the time complexity of both algorithm is the same. As was explained in

Section 3, Grover's time/data tradeoff is $N^2 = T^4 \cdot D^2$. Our algorithm's tradeoff is $N^2 = T^{4/3} \cdot M^2 \cdot D^2$. Equating the time complexities we get:

$$
\begin{aligned}
&N^2 = T^4 \cdot D^2 \\
&T = (N/D)^{0.5} \\
&N^2 = T^{4/3} \cdot M^2 \cdot D^2 = (N/D)^{2/3} \cdot M^2 \cdot D^2 \\
&M^2 = (N/D)^{4/3} \\
&M = (N/D)^{2/3}
\end{aligned}
$$

As the time complexity improves when M increases, our algorithm's time complexity is better than Grover's when $M > (N/D)^{2/3}$. For comparison, in classical settings $M > (N/D)^{1/2}$ is required to have online time complexity faster than exhaustive search.

We now calculate the respective data and time complexities at the point $M = (N/D)^{2/3}$, in which our algorithm's time complexity matches that of Grover's algorithm. When taking the maximal number of data points, which results in the minimal online complexity we obtain:

$$
\begin{aligned}
&T = D^{3/2} = (N/D)^{1/2} \\
&D^{3/2} = (N/D)^{1/2} \\
&D^2 = N^{1/2} \\
&D = N^{1/4} \\
&T = N^{3/8} \\
&M = (N/D)^{2/3} = N^{1/2}
\end{aligned}
$$

For comparison, in the classical setting, when $M = (N/D)^{1/2}$ (which is the transition point with respect to exhaustive search), $D = N^{1/3}$ and $T = N^{2/3}$.

7 Summary and Open Problems

In this paper we studied how to adapt the Hellman and Rainbow time-memory(-data) tradeoff attacks to the quantum world. We developed quantum variants of these attacks which follow the improved curve $T^{4/3}M^2D^2 = N^2$ compared to the classical curve of $TM^2D^2 = N^2$, using the standard model of quantum-access classical-memory (QACM). As these algorithms have to compete with a straightforward Grover search, the memory size in our algorithms must be at least $N^{2/3}$ (this bound is slightly larger than the previously known result that no improvement is possible when the size of the quantum advice is less than $N^{1/2}$). Another corollary is that while for the classical attacks each doubling of the memory reduces the time by a factor of 4, in our quantum setting the time is reduced only by a factor of $2\sqrt{2} \approx 2.82$.

While these results improve both on the classical time-memory(-data) tradeoff attacks and on the Grover's search algorithm, there is still a gap between

them and the lower bounds proved in [10]. Hence, a natural question to ask is whether one can reduce the gap, either by improving our attacks or by improving the lower bounds.

Aknowledgements

We thank the following people for the insightful discussions: Rotem Arnon-Friedman, Gustavo Banegas, Daniel J. Bernstein, Tal Mor, and María Naya-Plasencia.

References

1. Ambainis, A.: Quantum walk algorithm for element distinctness. SIAM J. Comput. **37**(1), 210–239 (2007). https://doi.org/10.1137/S0097539705447311, https://doi.org/10.1137/S0097539705447311
2. Barkan, E.: Cryptanalysis of Ciphers and Protocols. Ph.D. thesis, Technion, Israel (2006)
3. Barkan, E., Biham, E., Shamir, A.: Rigorous Bounds on Cryptanalytic Time/Memory Tradeoffs. In: Dwork, C. (ed.) Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4117, pp. 1–21. Springer (2006)
4. Bernstein, D.J., Jeffery, S., Lange, T., Meurer, A.: Quantum Algorithms for the Subset-Sum Problem. In: Gaborit, P. (ed.) Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7932, pp. 16–33. Springer (2013). https://doi.org/10.1007/978-3-642-38616-9_2, https://doi.org/10.1007/978-3-642-38616-9_2
5. Biryukov, A., Mukhopadhyay, S., Sarkar, P.: Improved time-memory trade-offs with multiple data. In: Selected Areas in Cryptography. Lecture Notes in Computer Science, vol. 3897, pp. 110–127. Springer (2005)
6. Biryukov, A., Shamir, A.: Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In: Okamoto, T. (ed.) Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1976, pp. 1–13. Springer (2000)
7. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight Bounds on Quantum Searching. Fortschritte der Physik **46**(4-5), 493–505 (1998)
8. Brassard, G., Høyer, P., Tapp, A.: Quantum cryptanalysis of hash and claw-free functions. In: LATIN. Lecture Notes in Computer Science, vol. 1380, pp. 163–169. Springer (1998)
9. Chailloux, A., Naya-Plasencia, M., Schrottenloher, A.: An Efficient Quantum Collision Search Algorithm and Implications on Symmetric Cryptography. In: Takagi, T., Peyrin, T. (eds.) Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10625, pp. 211–240. Springer (2017). https://doi.org/10.1007/978-3-319-70697-9_8, https://doi.org/10.1007/978-3-319-70697-9_8

10. Chung, K., Guo, S., Liu, Q., Qian, L.: Tight Quantum Time-Space Tradeoffs for Function Inversion. In: 61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020. pp. 673–684 (2020). https://doi.org/10.1109/FOCS46700.2020.00068
11. Chung, K., Liao, T., Qian, L.: Lower Bounds for Function Inversion with Quantum Advice. In: Kalai, Y.T., Smith, A.D., Wichs, D. (eds.) 1st Conference on Information-Theoretic Cryptography, ITC 2020, June 17-19, 2020, Boston, MA, USA. LIPIcs, vol. 163, pp. 8:1–8:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
12. Dunkelman, O., Keller, N.: Treatment of the initial value in time-memory-data tradeoff attacks on stream ciphers. Inf. Process. Lett. **107**(5), 133–137 (2008)
13. Grassi, L., Naya-Plasencia, M., Schrottenloher, A.: Quantum Algorithms for the k-xor Problem. In: Peyrin, T., Galbraith, S.D. (eds.) Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11272, pp. 527–559. Springer (2018)
14. Grover, L.K.: A Fast Quantum Mechanical Algorithm for Database Search. In: STOC. pp. 212–219. ACM (1996)
15. Hellman, M.E.: A cryptanalytic time-memory trade-off. IEEE Trans. Inf. Theory **26**(4), 401–406 (1980)
16. Hhan, M., Xagawa, K., Yamakawa, T.: Quantum random oracle model with auxiliary input. In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 11921, pp. 584–614. Springer (2019)
17. Kaluderovic, N., Kleinjung, T., Kostic, D.: Improved key recovery on the Legendre PRF. IACR Cryptol. ePrint Arch. p. 98 (2020), `https://eprint.iacr.org/2020/098`
18. Kuperberg, G.: Another Subexponential-time Quantum Algorithm for the Dihedral Hidden Subgroup Problem. In: Severini, S., Brandão, F.G.S.L. (eds.) 8th Conference on the Theory of Quantum Computation, Communication and Cryptography, TQC 2013, May 21-23, 2013, Guelph, Canada. LIPIcs, vol. 22, pp. 20–34. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013)
19. Naya-Plasencia, M., Schrottenloher, A.: Optimal Merging in Quantum k-xor and k-xor-sum Algorithms. In: Canteaut, A., Ishai, Y. (eds.) Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12106, pp. 311–340. Springer (2020)
20. Nayebi, A., Aaronson, S., Belovs, A., Trevisan, L.: Quantum lower bound for inverting a permutation with advice. Quantum Inf. Comput. **15**(11&12), 901–913 (2015)
21. Oechslin, P.: Making a Faster Cryptanalytic Time-Memory Trade-Off. In: Boneh, D. (ed.) Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2729, pp. 617–630. Springer (2003)
22. Yao, A.C.: Coherent Functions and Program Checkers (Extended Abstract). In: Ortiz, H. (ed.) Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA. pp. 84–94. ACM (1990)