Low-Communication Parallel Quantum Multi-Target Preimage Search

Gustavo Banegas[1(✉)] and Daniel J. Bernstein[2(✉)]

[1] Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven, P.O. Box 513,
5600 MB Eindhoven, The Netherlands
gustavo@cryptme.in

[2] Department of Computer Science, University of Illinois at Chicago,
Chicago, IL 60607–7045, USA
djb@cr.yp.to

Abstract. The most important pre-quantum threat to AES-128 is the 1994 van Oorschot–Wiener "parallel rho method", a low-communication parallel pre-quantum multi-target preimage-search algorithm. This algorithm uses a mesh of p small processors, each running for approximately $2^{128}/pt$ fast steps, to find one of t independent AES keys $k_1, \ldots, k_t$, given the ciphertexts $\mathrm{AES}_{k_1}(0), \ldots, \mathrm{AES}_{k_t}(0)$ for a shared plaintext 0.

NIST has claimed a high post-quantum security level for AES-128, starting from the following rationale: "Grover's algorithm requires a long-running serial computation, which is difficult to implement in practice. In a realistic attack, one has to run many smaller instances of the algorithm in parallel, which makes the quantum speedup less dramatic." NIST has also stated that resistance to multi-key attacks is desirable; but, in a realistic parallel setting, a straightforward multi-key application of Grover's algorithm costs more than targeting one key at a time.

This paper introduces a different quantum algorithm for multi-target preimage search. This algorithm shows, in the same realistic parallel setting, that quantum preimage search benefits asymptotically from having multiple targets. The new algorithm requires a revision of NIST's AES-128, AES-192, and AES-256 security claims.

Keywords: Quantum cryptanalysis · Multi-target preimages
Parallel rho method · Grover's algorithm

This project has received funding under the European Union's Horizon 2020 research and innovation programme (grant agreement 645622 PQCRYPTO and Marie Skłodowska-Curie grant agreement 643161 ECRYPT-NET); from the Netherlands Organisation for Scientific Research (NWO grant 639.073.005); and from the U.S. National Science Foundation (grant 1314919). "Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation." Permanent ID of this document: `564c02527d5562810a43e02ec640d604e13a9910`. Date: 2017.08.18.

C. Adams and J. Camenisch (Eds.): SAC 2017, LNCS 10719, pp. 325–335, 2018.
https://doi.org/10.1007/978-3-319-72565-9_16

1 Introduction

Fix a function H. For any element x in the domain of H, the value $H(x)$ is called **the image** of x, and x is called **a preimage** of $H(x)$.

Many attacks can be viewed as searching for preimages of specified functions. Consider, for example, the function H that maps an RSA private key (p, q) to the public key pq. Formally, define P as the set of pairs (p, q) of prime numbers with $p < q$, and define $H : P \to \mathbf{Z}$ as the function $(p, q) \mapsto pq$. Shor's quantum algorithm efficiently finds the private key (p, q) given the public key pq; in other words, it efficiently finds a preimage of pq.

As another example, consider a protocol that uses a secret 128-bit AES key k, and that reveals the encryption under k of a plaintext known to the attacker, say plaintext 0. Define $H(k)$ as this ciphertext $\mathrm{AES}_k(0)$. Given $H(k)$, a simple brute-force attack takes a random key x as a guess for k, computes $H(x)$, and checks whether $H(x) = H(k)$. If $H(x) \neq H(k)$ then the attack tries again, for example replacing x with $x + 1 \bmod 2^{128}$.

Within, e.g., 2^{100} guesses the attack has probability almost 2^{-28} of successfully guessing k. We say "almost" because there could be preimages of $H(k)$ other than k: i.e., it is possible to have $H(x) = H(k)$ with $x \neq k$. This gives the attack more chances to find a preimage, but it means that any particular preimage selected as output is correspondingly less likely to be k. Typical protocols give the attacker a reasonably cheap way to see that these other preimages are not in fact k, and then the attacker can simply continue the attack until finding k.

This brute-force attack is not specific to AES, except for the details of how one computes $\mathrm{AES}_k(0)$ given k. The general strategy for finding preimages of a function is to check many possible preimages. In this paper we focus on faster attacks that work in the same level of generality. Some specific functions, such as the function $(p, q) \mapsto pq$ mentioned above, have extra structure allowing much faster preimage attacks, but we do not discuss those special-purpose attacks further.

1.1. Multiple-Target Preimages. Often an attacker is given many images, say t images $H(x_1), \ldots, H(x_t)$, rather than merely a single image. For example, $x_1, \ldots, x_t$ could be secret AES keys for sessions between t pairs of users, where each key is used to encrypt plaintext 0; or they could be secret keys for one user running a protocol t times; or they could be secrets within a single protocol run.

The **t-target preimage problem** is the problem of finding a preimage of *at least one* of $y_1, \ldots, y_t$; i.e., finding x such that $H(x) \in \{y_1, \ldots, y_t\}$. A solution to this problem often constitutes a break of a protocol; and this problem can be easier than the single-target preimage problem, as discussed below.

Techniques used to attack the t-target preimage problem are also closely related to techniques used to attack the well-known **collision problem**: the problem of finding distinct x, x' with $H(x) = H(x')$.

The obvious way to attack the t-target preimage problem is to choose a random x and see whether $H(x) \in \{y_1, \ldots, y_t\}$. Typically $y_1, \ldots, y_t$ are distinct, and then the probability that $H(x) \in \{y_1, \dots, y_t\}$ is the sum of the probability that $H(x) = y_1$, the probability that $H(x) = y_2$, and so on through the probability that $H(x) = y_t$. If x is a single-target preimage with probability about $1/N$ then x is a t-target preimage with probability about t/N.

Repeating this process for s steps takes a total of s evaluations of H on distinct choices of x, and has probability about st/N of finding a t-target preimage, i.e., high probability after N/t steps. This might sound t times faster than finding a single-target preimage, but there are important overheads in this algorithm, as we discuss next.

1.2. Communication Costs and Parallelism. Real-world implementations show that, as t grows, the algorithm stated above becomes bottlenecked not by the computation of $H(x)$ but rather by the check whether $H(x) \in \{y_1, \dots, y_t\}$.

One might think that this check takes constant time, looking up $H(x)$ in a hash table of $y_1, \dots, y_t$, but the physical reality is that random access to a table of size t becomes slower as t grows. Concretely, when a table of size t is laid out as a $\sqrt{t} \times \sqrt{t}$ mesh in a realistic circuit, looking up a random table entry takes time proportional to $\sqrt{t}$.

Furthermore, for essentially the same cost as a memory circuit capable of storing and retrieving t items, the attacker can build a circuit with t small parallel processors, where the ith processor searches for a preimage of y_i independently of the other processors. Running each processor for N/t fast steps has high success probability of finding a t-target preimage and takes total time N/t, since the processors run in parallel.

The "parallel rho method", introduced by van Oorschot and Wiener in 1994 [13], does better. The van Oorschot–Wiener circuit has size p and reaches high probability after only N/pt fast steps (assuming $p \geq t$; otherwise the circuit does not have enough storage to hold all t targets, and one must reduce t). For example, with $p = t$, this circuit has size t and reaches high probability after only N/t^2 steps.

There are p small parallel processors in this circuit, arranged in a $\sqrt{p} \times \sqrt{p}$ square. There is also a parallel "mesh" network allowing each processor to communicate quickly with the processors adjacent to it in the square. Later, as part of the description of our quantum multi-target preimage-search algorithm, we will review how these resources are used in the parallel rho method. The analysis also shows how large p and t can be compared to N.

1.3. Quantum Attacks. If a random input x has probability $1/N$ of being a preimage of y then brute force finds a preimage of y in about N steps. Quantum computers do better: specifically, Grover's algorithm [7] finds a preimage of y in only about $\sqrt{N}$ steps.

However, increased awareness of communication costs and parallelism has produced increasingly frequent objections to this quantitative speedup claim. For example, NIST's "Submission Requirements and Evaluation Criteria for the

Post-Quantum Cryptography Standardization Process” [11] states security levels for AES-128, AES-192, and AES-256 that provide

> substantially more quantum security than a naïve analysis might suggest. For example, categories 1, 3 and 5 are defined in terms of block ciphers, which can be broken using Grover’s algorithm, with a quadratic quantum speedup. But Grover’s algorithm requires a long-running serial computation, which is difficult to implement in practice. In a realistic attack, one has to run many smaller instances of the algorithm in parallel, which makes the quantum speedup less dramatic.

Concretely, Grover’s algorithm has high probability of finding a preimage if it uses p small parallel quantum processors, each running for $\sqrt{N/p}$ steps, as in [8]. The speedup compared to p small parallel non-quantum processors is only $\sqrt{N/p}$, which for reasonable values of p is much smaller than $\sqrt{N}$.

Furthermore, when the actual problem facing the attacker is a t-target preimage problem, the parallel rho machine with p small parallel non-quantum processors reaches high success probability after only N/pt steps. This extra factor t can easily outweigh the $\sqrt{N/p}$ speedup from Grover’s algorithm.

For example, a parallel rho machine of size p finds collisions in only $\sqrt{N}/p$ steps. This is certainly better than running Grover’s algorithm for $\sqrt{N/p}$ steps.

Brassard, Høyer, and Tapp [5] claimed a faster quantum algorithm to find collisions. Their algorithm chooses $t \approx N^{1/3}$, takes t random inputs $x_1, \ldots, x_t$, computes the corresponding images $y_1, \ldots, y_t$, and then builds a new function H' defined as follows: $H'(x) = 0$ if $H(x) \in \{y_1, \ldots, y_t\}$, otherwise $H'(x) = 1$. A random input is an H'-preimage of 0 with probability approximately $1/N^{2/3}$, so Grover’s algorithm finds an H'-preimage of 0 after approximately $N^{1/3}$ steps.

However, Bernstein [4] analyzed the communication costs in this algorithm and in several variants, and concluded that no known quantum collision-finding algorithms were faster than the non-quantum parallel rho method.

1.4. Contributions of This Paper. This paper introduces a quantum algorithm, in the same realistic model mentioned above (p small parallel processors connected by a two-dimensional mesh), that finds a t-target preimage using roughly $\sqrt{N/pt^{1/2}}$ fast steps. If communication were not an issue then $t^{1/2}$ would improve to t. See Fig. 1.4.

Taking $t = 1$ produces a single-target preimage using roughly $\sqrt{N/p}$ steps, as in Grover’s algorithm running on p processors. To save time for larger values of t we combine Grover’s algorithm with the parallel rho method offering a speed up on the quantum attacks. This requires a *reversible* version of the parallel rho method. Reversibility creates a further t^{ϵ} cost explained below compared to pre-quantum attacks. Communication inside the parallel rho method raises further issues that do not show up in simpler applications of Grover’s method; this creates the gap between $t^{1/2}$ and t.

NIST has stated that resistance to multi-key attacks is desirable. Our results show that simply using Grover’s algorithm for single-target preimage search is not optimal in this context. NIST's post-quantum security claims for AES-128, AES-192, and AES-256 assume that it *is* optimal, and therefore need to be revised.

1.5. Open Questions. Our analysis is asymptotic. In this paper we suppress constant factors, logarithmic factors, etc. and focus on asymptotic exponents. We plan to increase the precision of the analysis of the algorithm by measuring the costs (qubits and gates) of an implementation. One major issue is the implementation of AES in a quantum computer; see the cost estimates from [6]. Another major issue is the sorting implementation. Both stages can be efficiently simulated and tested in a non-quantum computer, since both stages are reversible computations without superposition.

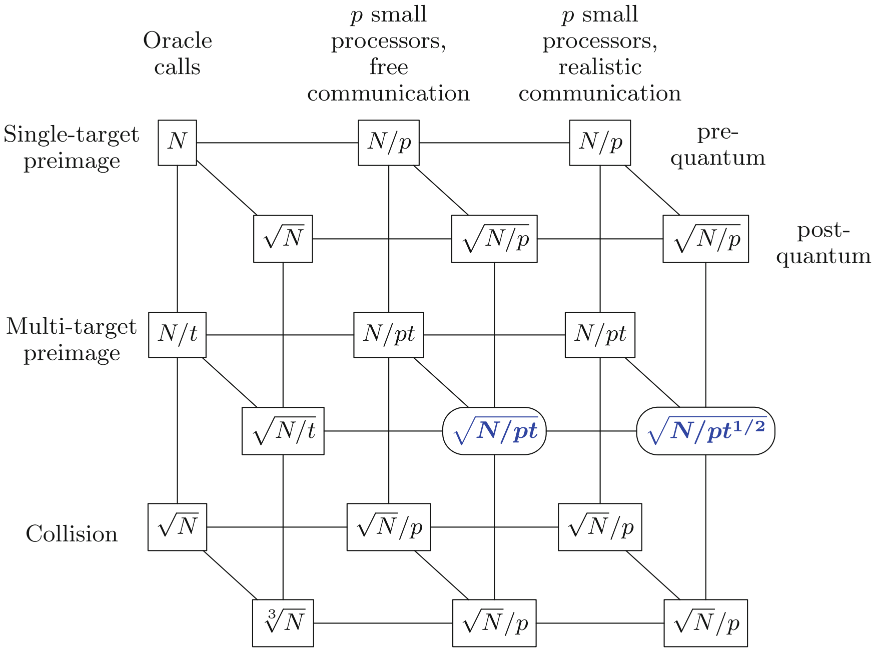

Fig. 1.4. Overview of costs of pre-quantum and post-quantum attacks. Circled blue items are new results in this paper. Lower-order factors are omitted. Pre-quantum single-target preimage attacks: brute force plus simple parallelization. Post-quantum single-target preimage attacks: Grover's algorithm [7] plus simple parallelization [8]. Pre-quantum multi-target preimage attacks: brute force and the parallel rho method [13]. Post-quantum multi-target preimage attacks: [9] for oracle calls, this paper for parallel methods. Pre-quantum collision attacks: the rho method and the parallel rho method. Post-quantum collision attacks: [5] for oracle calls, plus the parallel rho method. (Color figure online)

2 Reversible Computation

A **Toffoli gate** maps bits (x, y, z) to $(x, y, z + xy)$, where $+$ means exclusive-or.

A **reversible n-bit circuit** is an n-bit-to-n-bit function expressed as a composition of a sequence of Toffoli gates on selected bits. We assume that adjacent Toffoli gates on separate bits are carried out in parallel: our model of time for a reversible circuit is the depth of the circuit rather than the total number of gates. To model realistic communication costs, we lay out the n bits in a square, and we require each Toffoli gate to be applied to bits that are laid out within a constant distance of each other.

Let H be a function from $\{0,1\}^b$ to $\{0,1\}^b$, where b is a nonnegative integer. An **a-ancilla reversible circuit for** H is a reversible $(2b + a)$-bit circuit that, for all b-bit strings x and y, maps $(x, y, 0)$ to $(x, y + H(x), 0)$. The behavior of this circuit on more general inputs (x, y, z) is not relevant.

Grover's method, given any reversible circuit for H, produces a quantum preimage-search algorithm. This algorithm uses s serial steps of H computation and negligible overhead, and has probability approximately s^2/N of finding a preimage, if a random input to H has probability $1/N$ of being a preimage.

In subsequent sections we convert the reversible circuit for H into a reversible circuit for a larger function H' using approximately $\sqrt{t}$ steps on t small parallel processors. H' is designed so that

- a random input to H' has probability approximately $t^{5/2}/N$ of being an H'-preimage and
- an H'-preimage produces a t-target H-preimage as desired.

Applying Grover's method to H', with $s \approx \sqrt{N/pt^{3/2}}$, uses overall $\sqrt{N/pt^{1/2}}$ steps on t small parallel processors, and has probability approximately t/p of finding a preimage. A machine with p/t parallel copies of Grover's method has high probability of finding a preimage and uses $\sqrt{N/pt^{1/2}}$ steps on p small parallel processors.

3 Reversible Iteration

As in the previous section, let H be a function from $\{0,1\}^b$ to $\{0,1\}^b$, where b is a nonnegative integer. Assume that we are given a reversible circuit for H using a ancillas and gate depth g (see, e.g., the circuit in [6]). This section reviews the Bennett–Tompa technique [3] to build a reversible circuit for H^n, where n is a positive integer, using $a + O(b \log_2 n)$ ancillas and gate depth $O(gn^{1+\epsilon})$. Here ϵ can be taken as close to 0 as desired, although the O constants depend on ϵ.

As a starting point, consider the following reversible circuit for H^2 using $a+b$ ancillas and depth $3g$:

time 0:	x	y	0	0
time 1:	x	y	$H(x)$	0
time 2:	x	$y + H^2(x)$	$H(x)$	0
time 3:	x	$y + H^2(x)$	0	0

Each step here is a reversible circuit for H, and in particular the last step adds $H(x)$ to $H(x)$, obtaining 0 (recall that + means xor).

More generally, if H uses a ancillas and depth g, and H' uses a' ancillas and depth g', then the following reversible circuit for $H' \circ H$ uses $\max\{a, a'\} + b$ ancillas and depth $2g + g'$:

time 0:	x	y	0	0
time 1:	x	y	$H(x)$	0
time 2:	x	$y + H'(H(x))$	$H(x)$	0
time 3:	x	$y + H'(H(x))$	0	0

Bennett now substitutes H^m and H^n for H and H' respectively, obtaining the following reversible circuit for H^{m+n} using $\max\{a_m, a_n\} + b$ ancillas and depth $2g_m + g_n$:

time 0:	x	y	0	0
time 1:	x	y	$H^m(x)$	0
time 2:	x	$y + H^{m+n}(x)$	$H^m(x)$	0
time 3:	x	$y + H^{m+n}(x)$	0	0

Bennett suggests taking $n = m$ or $n = m + 1$, and then it is easy to prove by induction that $a_n = a + \lceil \log_2 n \rceil b$ and $g_n \le 3^{\lceil \log_2 n \rceil} g \le 3n^{\log_2 3} g$. For example, computing $H^{2^k}(x)$ uses $a + kb$ ancillas and depth $3^k g$.

More generally, with credit to Tompa, Bennett suggests a way to reduce the exponent $\log_2 3$ arbitrarily close to 1, at the expense of a constant factor in front of b. For example, one can start from the following reversible circuit for H^3 using $a + 2b$ ancillas and depth $5g$:

time 0:	x	y	0	0	0
time 1:	x	y	$H(x)$	0	0
time 2:	x	y	$H(x)$	$H^2(x)$	0
time 3:	x	$y + H^3(x)$	$H(x)$	$H^2(x)$	0
time 4:	x	$y + H^3(x)$	$H(x)$	0	0
time 5:	x	$y + H^3(x)$	0	0	0

Generalizing straightforwardly from H^3 to $H'' \circ H' \circ H$, and then replacing H, H', H'' with H^ℓ, H^m, H^n, produces a reversible circuit for $H^{\ell+m+n}$ using $\max\{a_\ell + b, a_m + 2b, a_n + 2b\}$ ancillas and depth $2g_\ell + 2g_m + g_n$. Splitting evenly between ℓ, m, n reduces $\log_2 3 \approx 1.58$ to $\log_3 5 \approx 1.46$. (An even split is not optimal: for a given ancilla budget one can afford to take a_ℓ larger than a_m and a_n. See [10] for detailed optimizations along these lines.) By starting with H^4 instead of H^3 one reduces the exponent to $\log_4 7 \approx 1.40$, using, e.g., $a + 9b$ ancillas and depth $567g$ to compute H^{64}. By starting with H^8 one reduces the exponent to $\log_8 15 \approx 1.30$; etc.

4 Reversible Distinguished Points

As above, let H be a function from $\{0,1\}^b$ to $\{0,1\}^b$, where b is a nonnegative integer; and assume that we are given an a-ancilla depth-g reversible circuit for H.

Fix $d \in \{0, 1, \ldots, b\}$. We say that $x \in \{0,1\}^b$ is **distinguished** if its first d bits are 0.

The rho method iterates H until finding a distinguished point or reaching a prespecified limit on the number of iterations, say n iterations. The resulting finite sequence $x, H(x), H^2(x), \ldots, H^m(x)$, either

- containing exactly one distinguished point $H^m(x)$ and having $m \leq n$ or
- containing zero distinguished points and having $m = n$,

is the **chain for** x, and its final entry $H^m(x)$ is the **chain end for** x.

This section explains a reversible circuit for the function that maps x to the chain end for x. This circuit has essentially the same cost as the Bennett–Tompa circuit from the previous section.

Define $H_d : \{0,1\}^b \to \{0,1\}^b$ as follows:

$$H_d(x) = \begin{cases} x & \text{if the first } d \text{ bits of } x \text{ are } 0 \\ H(x) & \text{otherwise.} \end{cases}$$

A reversible circuit for H_d is slightly more costly than a reversible circuit for H, since it needs an "OR" between the first d bits of x and a selection between x and $H(x)$.

If the chain for x is $x, H(x), H^2(x), \ldots, H^m(x)$ then the iterates

$$x, H_d(x), H_d^2(x), \ldots, H_d^m(x), H_d^{m+1}(x), \ldots, H_d^n(x)$$

are exactly $x, H(x), H^2(x), \ldots, H^m(x), H^m(x), \ldots, H^m(x)$. Hence the chain end for x, namely $H^m(x)$, is exactly $H_d^n(x)$. We compute H_d^n reversibly by substituting H_d for H in the previous section.

If x is chosen randomly and H behaves randomly then one expects each new H output to have chance $1/2^d$ of being distinguished. To have a reasonable chance that the chain end is distinguished, one should take n on the scale of 2^d: e.g., $n = 2^{d+1}$. If n and d are very large then chains will usually fall into loops before reaching distinguished points, but we will later take small n, roughly $\sqrt{t}$ for t-target preimage search.

5 Reversible Parallel Distinguished Points

Define b, H, a, g, d, n as before, and let t be a positive integer. This section explains a reversible circuit for the function that maps a vector $(x_1, \ldots, x_t)$ of b-bit strings to the corresponding vector $(H_d^n(x_1), \ldots, H_d^n(x_t))$ of chain ends.

This circuit is simply t parallel copies of the circuit from the previous section, where the ith copy handles x_i. The depth of the circuit is identical to the depth of the circuit in the previous section. The size of this circuit is t times larger than the size of the circuit in the previous section.

Communication in this circuit is only inside the parallel computations of H etc. There is no communication between the parallel circuits, and there is no dependence of communication costs upon t.

6 Sorting on a Mesh Network

Define $S(c_1, c_2, \ldots, c_t)$, where $c_1, c_2, \ldots, c_t$ are b-bit strings, as $(d_1, d_2, \ldots, d_t)$, where $d_1, d_2, \ldots, d_t$ are the same as $c_1, c_2, \ldots, c_t$ in lexicographic order.

This section presents a reversible computation of S using $O(t(b + (\log t)^2))$ ancillas and $O(t^{1/2}(\log t)^2)$ steps. Each step is a simple local operation on a two-dimensional mesh, repeated many times in parallel. We follow the general sorting strategy from [2] but choose different subroutines.

We start with odd-even mergesort [1]. This algorithm is a **sorting network**: i.e., a sequence of **comparators**, where each comparator sorts two objects. Odd-even mergesort sorts t items using $O((\log t)^2)$ stages, where each stage involves $O(t)$ parallel comparators. For comparison, [2, Table 2] mentions bitonic sort, which is slower than odd-even mergesort, and AKS sort, which is asymptotically faster as $t \to \infty$ but slower for any reasonable size of t.

To make odd-even mergesort reversible, we record for each of the $O(t(\log t)^2)$ comparators whether the inputs were out of order, as in [2, Sect. 2.1]. This uses $O(t(\log t)^2)$ ancillas. The comparators themselves use $O(tb)$ ancillas.

The comparators in odd-even mergesort are not local when items are spread across a two-dimensional mesh. We fix this as in [2, Sect. 2.3]: before each stage, we permute the data so that the stage involves only local comparators. Each of these permutations is a constant determined by the structure of the sorting network; for odd-even mergesort each permutation is essentially a riffle shuffle.

The permutation strategy suggested in [2, Sect. 2.3] is to apply any sorting algorithm built from local operations. For a two-dimensional mesh, [2, Table 2] suggests "Bubble/Insertion sort", but it is not at all clear which two-dimensional algorithm is meant here; the classic forms of bubble sort and insertion sort are not parallelizable. The same table also says that these are "sorting networks", but most of the classic forms of bubble sort and insertion sort include conditional branches. We suggest using the Schnorr–Shamir algorithm [12], which has depth approximately $3\sqrt{t}$. It seems likely that an ad-hoc riffle algorithm would produce a better constant here.

7 Multi-target Preimages

Fix images $y_1, \ldots, y_t$. We build a reversible circuit that performs the following operations:

- Input a vector $(x_1, \ldots, x_t)$.
- Compute, in parallel, the chain ends for $x_1, \ldots, x_t$: i.e., $H_d^n(x_1), \ldots, H_d^n(x_t)$.
- Precompute the chain ends for $y_1, \ldots, y_t$.
- Sort the chain ends for $x_1, \ldots, x_t$ and the chain ends for $y_1, \ldots, y_t$.
- If there is a collision, say a collision between the chain end for x_i and the chain end for y_j: recompute the chain for x_i, checking each chain element to see whether it is a preimage for y_j.
- Output 0 if a preimage was found, otherwise 1.

This circuit uses $O(a+b\log_2 n+tb+t(\log t)^2)$ ancillas. The chain computation has depth $O(gn^{1+\epsilon})$, and the sorting has depth $O(t^{1/2}(\log t)^2 \log b)$, where $O(\log b)$ accounts for the cost of a b-bit comparator.

If a chain for x_i ends with a distinguished point, and the chain includes a preimage (before this distinguished point) for y_j, then the chain for y_j will end with the same distinguished point. The recomputation will then find this preimage. The number of such chains is proportional to t (with a constant-factor loss for chains that end before a distinguished point), so the number of elements in the chains is proportional to nt (with a constant factor reflecting the length of chains before distinguished points); the chance of a particular preimage being one of these elements is $1/N$; and there are t preimages, for an overall chance roughly nt^2/N.

We take $n \approx \sqrt{t}$, so the circuit uses $O(a+tb+t(\log t)^2)$ ancillas and has depth $O(gt^{1/2+\epsilon/2}+t^{1/2}(\log t)^2 \log b)$; one can also incorporate b, g, ϵ into the choice of n to better balance the two terms in this depth formula. The chance that the circuit finds a preimage is roughly $t^{5/2}/N$, as mentioned earlier. Finally, we apply p/t parallel copies of Grover's method to this circuit, each copy using approximately $\sqrt{N/pt^{3/2}}$ iterations, i.e., depth $O(\sqrt{N/pt^{1/2}}(gt^{\epsilon/2} + (\log t)^2 \log b))$, to reach a high probability of finding a t-target preimage.

References

1. Batcher, K.E.: Sorting networks and their applications. In: Proceedings of the Spring Joint Computer Conference, AFIPS 1968 (Spring), 30 April–2 May 1968, pp. 307–314. ACM, New York (1968)
2. Beals, R., Brierley, S., Gray, O., Harrow, A.W., Kutin, S., Linden, N., Shepherd, D., Stather, M.: Efficient distributed quantum computing. Proc. R. Soc. Lond. Ser. A Math. Phys. Eng. Sci. **469**(2153), 20120686, 20 (2013). ISSN: 1364-5021
3. Bennett, C.H.: Time/space trade-offs for reversible computation. SIAM J. Comput. **18**(4), 766–776 (1989)
4. Bernstein, D.J.: Cost analysis of hash collisions: Will quantum computers make SHARCS obsolete? In: SHARCS 2009 Special-purpose Hardware for Attacking Cryptographic Systems, p. 105 (2009)
5. 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, Heidelberg (1998). https://doi.org/10.1007/BFb0054319
6. Grassl, M., Langenberg, B., Roetteler, M., Steinwandt, R.: Applying Grover's algorithm to AES: quantum resource estimates. In: Takagi, T. (ed.) PQCrypto 2016. LNCS, vol. 9606, pp. 29–43. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29360-8_3
7. Grover, L.: A fast quantum mechanical algorithm for database search. In: Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing, pp. 212–219. ACM (1996)
8. Grover, L., Rudolph, T.: How significant are the known collision and element distinctness quantum algorithms? arXiv preprint arXiv:quant-ph/0309123 (2003)

9. Hülsing, A., Rijneveld, J., Song, F.: Mitigating multi-target attacks in hash-based signatures. In: Cheng, C.-M., Chung, K.-M., Persiano, G., Yang, B.-Y. (eds.) PKC 2016. LNCS, vol. 9614, pp. 387–416. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49384-7_15
10. Knill, E.: An analysis of Bennett's pebble game. CoRR, abs/math/9508218 (1995)
11. NIST: Submission requirements and evaluation criteria for the post-quantum cryptography standardization process (2016). http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf
12. Schnorr, C.-P., Shamir, A.: An optimal sorting algorithm for mesh connected computers. In: Hartmanis, J. (ed.) Proceedings of the 18th Annual ACM Symposium on Theory of Computing, 28–30 May 1986, Berkeley, California, USA, pp. 255–263. ACM (1986)
13. Van Oorschot, P.C., Wiener, M.J.: Parallel collision search with application to hash functions and discrete logarithms. In: Proceedings of the 2nd ACM Conference on Computer and Communications Security, pp. 210–218. ACM (1994)