Toward Provable One Way Functions

(Preliminary Version)

Hagar Dolev[1] Shlomi Dolev[2]

[1]School of Computer Science, Hebrew University and Bezalel, Jerusalem, Israel
[2]Department of Computer Science, Ben-Gurion of the Negev, Beer-Sheva, Israel

October 28, 2020

Abstract

The existence of a provable one-way function is a long-standing open problem. This short note presents an example towards the existence a provable one-way function, example in which both directions are polynomial. Namely, we prove that given a sorted array it takes $\Theta(n)$ operations to randomly permute the array values uniformly over the permutation space, while (comparison based) sorting of the permuted array (of big enough values) requires in the worst case (and in the average case) $\Theta(n \log n)$ compare operations.

We also present a candidate cryptosystem based on permutations of random polynomial values.

Keywords: One-way functions, Cryptography, Merkle Puzzles

1 Introduction

One Way functions are functions of the shape $f : x \to f(x)$ such that they are easy to compute, and they are hard to "decipher", meaning given a random image of $f(x)$ it is hard to compute x. Where easy means they have a polynomial time p Turing Machine that given an input x computes $f(x)$ in polynomial time, and hard means that given a random $f(x)$ we cannot compute x in the same polynomial time p.

These functions play an important role in modern cryptography. Unfortunately, there is no provable one-way function, and the currently functions that are used in practice as one-way functions are assumed to be one-way functions, rather than proven to be such functions. Sometimes later revealing a weakness e.g., [6].

2 The One Way from Sorted Array to (uniformly) Unsorted and the Way Back

The lower bound on comparison-based sorting is a basic result in computer science. The number of possible permutations of the inputs is $n!$, thus, the number of decision tree leaves is $\log n!$ which is bigger than $\log(n/2)^{n/2}$ which is in turn $n/2 \log n/2$, thus, $\Omega(n \log n)$. Shuffling a sorted array can be performed by randomly and independently choice of a sequence of n indexes of the array, and swapping the first value of the array with the value appearing first in the indexes sequence, using the resulting array as an input for swapping the second array entry with the value residing in the index that appears second in the indexes sequences, and so on and so forth, yielding $\Theta(n)$ operations, for producing a shuffled array (that is for symmetric reasons) chosen uniformly over all permutation possibilities.

It is known that the average depth of a leaf in binary tree with $n!$ leaves is $\Theta(n \log n)$. Thus, with very high probability sorting a uniformly random chosen instances of shuffled array are as hard as the worst case (the importance of average case hardness as the Shortest Vector Problem over lattices and the permanent are demonstrated when implementing Merkle puzzles schemes e.g., [4, 3]).

There are $\Theta(n)$ sorts that are not comparison based. These sorts, e.g., counting sort or Radix sort, are based on a limited range of the array values, while we can choose input arrays with values of enough number of bits to imply that Radix sort is less efficient than comparison-based sort.

Others, such as bucket sort, are designed for the average case and are based on distribution of the array elements, while we can choose and produce arrays (sorted arrays to be shuffled) with inconvenient distributions. Thus, enforcing the use of comparison-based sorting[1].

3 Merkle Puzzles Use-case

We demonstrate the usefulness of such one-way function by using Merkle puzzles. Alice may receive or produce k sorted arrays. To produce a sorted array Alice may repeatedly chose a random value and add the chosen value to the previous element in an array, where the previous element of the first value is set to be zero.

Alice randomly shuffle each of the arrays, recording the permutation of the shuffle investing $O(kn)$ computation time. The permutation of the shuffle maybe recorded as part of the values, multiplying (possibly by shifting) each element in the array by at least n and adding the original index prior to shuffling. Then removing the index and dividing/shifting back post shuffling and post recording the permutation.

[1] In particular, one can assume a computing device that (is optimized to) supports comparisons rather than operations needed for non-comparison-based sorts, as in many cases in practice.

Note that the permutation can be regarded as a trapdoor t, knowing t implies $\Theta(n)$ operations for "sorting" the shuffled array.

Then Alice sends to Bob the k arrays, Bob randomly selects one, and sorts it in $O(n \log n)$ time, Bob uses the same technique to record the sorting permutation. Now, say, Bob bit-wise xors all the even indices in the sorted array entries and the even indices (padded concatenated to form a sequence that is broken to the length bits of the array entries) of the permutation and send to Alice as a mean to identify his chosen array, without revealing the actual permutation to Eve. Then both use a function (possibly bitwise xor) over the index of the chosen array and the permutation as a shared symmetric key. The symmetric key is secure for a while.

Alice and Bob can repeat the procedure, this time encrypting the arrays with the previous key(s), so that Eve cannot start solving the new set of puzzles prior to revealing the puzzle (array) chosen previously by Bob (See [4, 3]).

We note, that one approach is to explore hard in average instances of complexity class that is provable higher than P and NP as suggested in [2], and randomly use (short) instances as puzzles, however unlike in the case of one-way functions, in such cases Alice invest significant computation power (even more than Eve) prior to sending the puzzles to Bob.

4 More Secret Permutations and Secret Shared Polynomials

The number of possible permutations is exponential in n. Choosing an arbitrary random permutation rather than the sorted permutation as the secret may yield a greater gap in the computation of the one way versus the way back direction.

There is a possibility to describe a permutation of n values by a polynomial of degree $n-1$, where the x of a value in the array is the index in the array, and the y is the actual value. One can randomly choose a polynomial p of degree $n-1$ over a finite field, and use the free coefficient as done in secret sharing, so n points on the polynomial are needed to reveal the free coefficient. Then calculate the values in the array according the randomly chosen polynomial p.

The obtained array can then be randomly permuted. Then, requiring re-ordered to fit a polynomial with the free coefficient as the one of p. Furthermore, to enforce considering many (preferably all exponential number of) permutation and avoid information leakage on the free coefficient, two such arrays based on two random polynomials, can be independently constructed. The free coefficients of the two arrays can be bitwise xored (just like one-time pad for each other).

The elements of the two arrays maybe shuffled together to form a permutation of $2n$ elements, or in fact, n, if each array consists of $n/2$ elements that are defined by a random polynomial of degree $n/2-1$.

Furthermore, one can build a symmetric key cryptosystem, where the random shuffling

permutation is the shared key, and the message is bitwise xored with the two free coefficients. Thus, coping with leakage of a free coefficient in case of known-plaintext attack.

The permutation secret key can be coordinately replaced even in every communication, by using the randomness of the polynomials.

Note that, it maybe feasible to prove, under (reasonable) computation model restriction as suggested in e.g., [7], or under computation time limitation as suggested in e.g., [5], that there is a computation gap between the two directions of computing of the suggested one way functions.

5 Conclusions

We believe that the approach in which random function is used for one way, implying the need to cancel the randomization effect is a promising direction for provable one-way function also for higher complexity classes.

Acknowledgments. We thank Moti Yung for correspondence on the subject.

References

[1] Alfred V Aho, John E Hopcroft, Jeffrey D Ullman, *The Design and Analysis of Computer Algorithms*, 1974.

[2] Shlomi Dolev, Nova Fandina, Dan Gutfreund, "Succinct Permanent Is NEXP-Hard with Many Hard Instances", *CIAC* 2013: 183-196.

[3] Shlomi Dolev, Nova Fandina, Ximing Li, "Nested Merkle's Puzzles against Sampling Attacks" *Inscrypt* 2012: 157-174.

[4] Shlomi Dolev, Ephraim Korach, Ximing Li, Yin Li, Galit Uzan, "Magnifying computing gaps: Establishing encrypted communication over unidirectional channels," *Theor. Comput. Sci.* 636: 17-26 (2016).

[5] Mrinal Kumar and Ramprasad Saptharishi, "An exponential lower bound for homogeneous depth-5 circuits over finite fields," *Proceedings of the 32nd Computational Complexity Conference*, July 2017 Pages 1–30.

[6] Adi Shamir, "A Polynomial-Time Algorithm for Breaking the Basic Merkle-Hellman Cryptosystem," *IEEE Transaction on Information Theory*, Vol, IT-30, No. 5, 1984.

[7] Victor Shoup and Roman Smolensky, "Lower bounds for polynomial evaluation and interpolation problems," *computational complexity*, volume 6, pages 301–311 (1996).