Single Instance Self–Masking via Permutations*

(Preliminary Version)

Asaf Cohen, Paweł Cyprys, and Shlomi Dolev

Ben-Gurion University of the Negev
Beer Sheva, Israel

Abstract. Self–masking allows the masking of success criteria, part of a problem instance (such as the sum in a subset-sum instance) that restricts the number of solutions. Self–masking is used to prevent the leakage of helpful information to attackers; while keeping the original solution valid and, at the same time, not increasing the number of unplanned solutions.

Self–masking can be achieved by xoring the sums of two (or more) independent subset sum instances [DD20, CDM22], and by doing so, eliminate all known attacks that use the value of the sum of the subset to find the subset fast, namely, in a polynomial time; much faster than the naive exponential exhaustive search.

We demonstrate that the concept of self–masking can be applied to a single instance of the subset sum and a single instance of the permuted secret-sharing polynomials.

We further introduce the benefit of permuting the bits of the success criteria, avoiding leakage of information on the value of the i'th bit of the success criteria, in the case of a single instance, or the parity of the i'th bit of the success criteria in the case of several instances.

In the case of several instances, we permute the success criteria bits of each instance prior to xoring them with each other. One basic permutation and its nesting versions (e.g., π^i) are used, keeping the solution space small and at the same time, attempting to create an "all or nothing" effect, where the result of a wrong π trials does not imply much.

1 Introduction

There is a need for an efficient candidate to serve as a one-way function (OWF). Say, a function that is more efficient than SHA, that is, a function that is easy to compute yet has no known attack for easily finding preimage and/or collisions, is important, as such functions can influence the daily computation invested in commitments and signatures. Commitments based on candidates for[1] one-way functions are used in many scenarios, including in obtaining Zero Knowledge

* Partially supported by the Rita Altura Trust Chair in Computer Science and the Israeli Science Foundation (Grant No. 465/22).

[1] Note that a provable one-way function implies $P \neq NP$.

Proofs (ZKP) [GMR85, BM88], whereas OWF is used as a commitment primitive. For a signature example, consider signatures in the style of Lamport's signature [Lam79]. Lamport's signature is an example of using OWF to facilitate a (one-time) digital signature.

There are known candidates for one-way functions for which an algorithm that can be used in practice has successfully inverted them. See the example in [Sha82] for instances of the subset-sum. Indeed, throughout history, one of the main goals of crypto-analysts is to find such breaks, and in many cases, they were successful, e.g., the recent result [PKC22].

Background. Permutations are often used in the design of one-way functions as they provide a source for non-linearity, see, e.g., [SMKG22].

We suggest using the permutation of items in vectors, either the items to choose the sub-set from or the y-values used in reconstructing (secret–sharing) polynomials. Specifically, in the case of the polynomials, we construct a function from the set of all permutations on n elements to a finite field element, the free coefficient of a (secret–sharing) polynomial. Computing the free coefficient requires only an inner product, yet when the permutation is unknown, it is harder to inverse. One piece of evidence for such difficulty is shown in the sequel, proving that random permutations result in an approximately uniform probability on the output space.

To further enhance the challenge, we incorporate self–masking. Roughly speaking, the goal of self–masking is similar to code obfuscation, where the instance is given and defines the solution/functionality but adds a level of pseudo-randomness. The practicality of the masking technique [CDM22, DD20] depends heavily on the hardness of reconstructing the self–masked parts and the number of additional solutions the masking introduces. The success criteria for the reconstruction are encoded to prevent an easy reveal of the self–masked parts.

To make the presentation self–contained and still short, we present only definitions that are explicitly used in our analysis.

Self-masking is a technique to conceal part of a computation task instance(s), for example, the required sum in the subset-sum instance(s), by using a function, possibly bitwise xor, with another analogous part of an (independently chosen) other instance(s). Self-masking may preserve correlation among the solutions, possibly by having sums implied by the same indices of items of the participating instances, and in this way (further) restrict the number of possible unplanned solutions for the combined instances [CDM22, DD20].

Here we further extend the self-masking techniques to the case of a single instance, using the representation of the original instance to conceal a part (e.g., the desired sum of the subset) by applying a function on the part, using the randomness used to produce the instance, for example, the bits of the vector of items (participating in the subset sum). For a more comprehensive background on one-way functions self–masking and related applications, see, e.g., [CDM22, DD20, IN96, HILL99].

In [DD20], it is suggested to produce a random sorted array by randomly choosing n values, choosing the values by which each previous item in the array is incremented (for simplicity, assume integers with no bounds) then randomly permute the items. Both these operations can be performed in $O(n)$ steps. However, sorting the array (i.e., finding the reverse random permutation) in a comparison-based sort requires $\Omega(n \log n)$ steps. It is further suggested in [DD20], to allow other orders beyond the sorted order to be the successful order and to define the success using polynomial P, where $P(i)$, $1 \leq i \leq n$ is the value of a (randomly selected) entry in an array. Then, $P(0)$ and permuted values of the array entries are given, requiring finding the reverse permutation. Here, we significantly extend the basic suggestion of [DD20], e.g., we suggest shuffling the bits of $P(0)$ (without increasing the number of undesired solutions) and proving the uniform distribution of the solving permutations.

Note that using polynomials over items in a finite field and Lagrange interpolation is similar to the secret sharing technique suggested in [Sha79], where ignoring even one item yields a uniform distribution of $P(0)$.

Paper organization. The rest of the paper is organized as follows. Section 2 gives a very short presentation of self–masking ideas in the current literature and exemplifies its use in the subset–sum problem. Section 3 gives the main results for an OWF based on one instance of the secret–sharing polynomial problem. Section 4 depicts an algorithm that incorporates the self–masking ideas with the OWF instance presented herein. Section 5 concludes the paper.

2 Subset Sum with Permutations

We start with a few definitions and settings to make the presentation as self–contained as possible.

Subset Sum. Given a subset-sum instance $A_{n,l} = (a_1, a_2..a_n)$ and b, such that each a_i and b are of ℓ bits. Find a subset of the elements summed up to b (mod 2^{l+1}).

MsbLsb. Is the (long) sequence of $n\ell$ bits, starting from the most significant bit (MSB) of the first number/item, continuing to the first MSB of the second number/item, until all n MSBs bits are used, then turning to use the next to the MSB row collecting in a similar fashion, additional n bits, and so on, until at last, the LSB bits join the created sequence to form a sequence of $n\ell$ bits. We call the obtained sequence *MsbLsb* sequence. Note that other (more sophisticated) constructions for harvesting many randomly chosen bits from the randomly chosen items can be suggested.

Permutations. To harden the reconstruction of the critical parts, say the sum b of the required subset, the self–masking technique presented in [CDM22] can be extended by applying permutation defined by the *MsbLsb* sequence as a permutation index and constructing the actual permutation by using the mapping defined in e.g., [DLH13]. Given an (integer) i index of a permutation in the lexicographical order of the permutations. Unique permutation hashing [DLH13] output the i'th permutation, where i is the index of the permutation in the lexicographic order of permutations, as follows, outputs the first index of the permutation to be the index of the bucket (just as done in bucket sort where buckets are indexes are 1 to n, and each bucket size is $n!/n$ values) in which i is mapped to, say this bucket is j. Next, the scope is the mapping to a bucket in the j'th bucket, partitioned to $n-1$ buckets each of size $(n-1)!/(n-1)$ values, and eliminating the j index from consideration, continuing this way to define the entire permutation explicitly.

We suggest the following particular masking (as an easy example from many possible options). The first ℓ bits in *MsbLsb* are xored with b. Let *mb* (masked b) denote the xor result, concatenate *mb* with the next m bits of *MsbLsb* to form *emb* (extended masked b) a sequence of $\ell + m$ bits.

Use the rest of the bits in *MsbLsb* to choose (almost) uniformly a permutation index in the range 1 to $(\ell + m)!$ (use mod$(\ell + m)!$ as needed) permute *emb* accordingly. And randomly permute $a_1, a_2 ... a_n$. The function's output is the permuted *emb* and the (randomly) permuted $a_1, a_2, ... a_n$.

To reverse the function, one has to produce *emb* that fits the $\ell + m$ bits of a permutation of $a_1, a_2 ... a_n$ (essentially returning to the initial unimportant order of the subset sum order), the main indicator for correct reverting is the m extended bits. There are even more restrictions related to the existing sum in the spirit of [CDM22].

Note that the m bits of *MsbLsb* that extend the *mb* to form *emb*, serve as a success criteria combination, yielding exponentially smaller probability for a collision as m grows. Namely, the longer m is, the smaller the probability of finding more than one permutation that yields a fitting m value.

3 Permuting Secret Sharing Polynomials

In this section, we turn to the scope of polynomials (rather than subset-sum) extending the idea sketched in [DD20], which uses (secret sharing) polynomials over a finite field (introduced in [Sha79]) to create a function that is hard to invert yet easy to verify. The construction is motivated by the following pictorial "story".

3.1 The Combination Lock

Consider an ordered number set $Y = (y_1, y_2, \ldots, y_n)$, that when permuted in a particular combination(s), can open a safe, and y_0 as a challenge associated with Y, which is publicly known and can serve to easily prove that a certain party has the right combination, without opening the safe in practice.

If n is big enough, then a useful lock can be established, together with an easy-to-implement proof of having a key to the safe. In the worst case, one needs to try $n!$ possibilities to open the safe while proving that one has the key that can be linear in n.

The safe lock is opened when the order of the elements in the vector Y corresponds to the particular number, y_0, in the following way: y_0 is the free coefficient in a polynomial of degree $n-1$ defined over a finite field (just like the safe locker is defined over a finite number of possible digits) by the sequence of points $(1, Y_\pi(1)), (2, Y_\pi(2)), \ldots, (n, Y_\pi(n))$, where Y_π is the permuted Y. Note that the first index in each of the above points is regarded as the x coordinate of the point. In the sequel, we prefer to choose the x values of these points to be random rather than the simplest vector $X = (1, 2, \ldots, n)$ of the x values (yet, we make sure while randomly selecting X, that there are no repetitions of x values in X). Note that these x values are exposed and known to all parties.

When we provide the opening criteria, a naive burglar will try all $n!$ possibilities by using Lagrange interpolation until a polynomial with identical y_0 is found, which in turn may take too much time, for the limited time the burglar may afford. However, it is possible that an ingenious burglar can use the free coefficient y_0 to reconstruct the permutation of Y in a much more efficient way. First, in a key part of this paper, we wish to show that while there are less than $n!$ permutations to test, choosing the values of X and Y vectors appropriately results in a system that is hard to invert, in a sense to be rigorously defined later. Moreover, we may incorporate the two instances of self-masking idea [CDM22, DD20], where two combination lockers are installed, and their free coefficients are not given as before, but, instead, the bitwise xor of the free coefficients is given. Now, the burglar can open the safe when she reconstructs the given xor value of the free coefficients.

The safe can be designed to restrict the permutation of one of the lockers to be the opposite permutation of the other. Thus, when the burglar tries one combination in a locker, she has to set the reverse permutation in the second, compute Lagrange for each permuted lock, and xor the y_0s.

To make the challenge harder, we hide the value of each of the y's, hopefully restricting the burglar from finding them in a "blind" fashion[2] so the burglar is caught before opening the safe. We may shuffle/permute the bits of y_0 of the first polynomial, using the (secret) permutation for the numbers of the other polynomial elements, and vice versa, before xoring their bits.

3.2 Presentation of the problem

In this paper, we introduce self-masking for one instance of a problem, i.e., where we have one vector Y, and one instance of y_0. The choice of n, yielding $n!$ possible permutations (though, as mentioned, the actual number may be smaller), should be coordinated with the choice of a field F so that the probability of guessing a permutation that opens the safe among the permutations that yield the y_0 in the single instance problem, or the xor of the shuffled free coefficients (the two-instances problem; in our story, possibly permutations used by other bank managers) is negligible. Thus, we would prefer values of $n!$ coordinated with the value of $\ell = \lg |F|$; this relation can be tuned according to the security parameters (for restricting the number of collisions) required.

The *polynomial permuting* problem of dimensions n and ℓ, to be denoted $PP(n, \ell)$, is defined as follows. Let $F = GF(2^\ell)$ be the finite field of 2^ℓ elements,

[2] We try to enforce exhaustive search as much as we can; obviously, a success to do so in a provable way is the long-standing problem of $P \neq NP$.

and assume some lexicographic order on the elements of F. Let further $X = (x_1, \ldots, x_n)$ be n distinct nonzero elements of F.

Definition 1. *For any X, Y, define by $P_{X,Y}$ the unique polynomial of degree $n-1$ over F, satisfying $P_{X,Y}(x_i) = Y(i), i = 1, \ldots n$.*

Since $P_{X,Y}$ above is unique, there is a unique y_0 in F such that $P_{X,Y}(0) = y_0$. Define $[n] = \{1, 2, \ldots, n\}$ and let Π denote the set of all possible permutations on $[n]$. Further denote by Y_π, $\pi \in \Pi$, the permuted Y according to the permutation π. We can now define $PP(n, \ell)$.

Definition 2. *An input to $PP(n, \ell)$ is a tuple (X, Y, y_0) of length $2n+1$ over $F = GF(2^\ell)$. It is required to decide if there is a permutation $\pi \in \Pi$ s.t. the unique polynomial $P_{X,Y_\pi}(\cdot)$ of degree $n-1$ over F, satisfying $P_{X,Y_\pi}(x_i) = Y_\pi(i), i = 1, \ldots n$, also satisfies $P_{X,Y_\pi}(0) = y_0$.*

That is, $PP(n, \ell)$ requires to determine if under some permutation π, the polynomial P_{X,Y_π} in Definition 1 has y_0 as its free coefficient. Intuitively, we think of X as fixed, and any permutation of Y has some y_0 that satisfies these conditions. In other words, given Y and *only y_0 which satisfies the conditions under some permutation of Y, Y_π, but without giving π itself*, implicitly encodes the permutation π. Our goal is to show that revealing this permutation explicitly is hard.[3] Specifically, define a *function ensemble*, parameterized by X and Y as follows.

Definition 3. *Fix X and Y, both in F^n. Denote by $f_{X,Y} : \Pi \to F$ the function $f_{X,Y}(\pi) = P_{X,Y_\pi}(0)$.*

We wish to show that under some choices of X and Y, and some relationships between l and n, the function $f_{X,Y}(\pi)$ is hard to invert, in the sense that choosing a random $\pi \in \Pi$ results in a uniformly distributed y_0.

3.3 Results

Theorem 1. *Assume X and Y are chosen independently at random, both as uniform i.i.d. vectors of length n over $F = GF(2^\ell)$. Fix a permutation σ. Then randomly choosing a permutation $\pi \in \Pi$ to invert $f_{X,Y}(\sigma)$, that is, to have $f_{X,Y}(\pi) = f_{X,Y}(\sigma)$, has a success probability*

$$Pr\left\{f_{X,Y}(\pi) = f_{X,Y}(\sigma)\right\} \leq (1 - \epsilon_{\ell,n}) \left(\frac{1}{2^\ell} + \frac{2}{e(n-1)!} \right) + \epsilon_{\ell,n}, \tag{1}$$

with $\epsilon_{\ell,n} \to 0$ as $\ell \to \infty$ for any fixed $n > 0$.

Proof. We wish to evaluate $Pr\left(f_{X,Y}(\pi) = f_{X,Y}(\sigma)\right)$, for random i.i.d. X and Y, a fixed σ and a random π, uniform from Π.

[3] We do not know if PP is solvable in polynomial time; however, the applicability of the masking technique is independent of this question.

We remind the reader of the reconstruction of the free coefficient of the reconstructed polynomial. That is,

$$f_{X,Y}(\sigma) = P_{X,Y_\sigma}(0) \tag{2}$$
$$= \sum_{i=1}^{n} Y_\sigma(i) L_i(0), \tag{3}$$

where

$$L_i(x) = \frac{\Pi_{1\le j\le n, j\ne i}(x - x_j)}{\Pi_{1\le j\le n, j\ne i}(x_i - x_j)}. \tag{4}$$

Denote $\mathcal{L} = (L_1(0), \ldots, L_n(0))$, and $\mathcal{Y}_\sigma = (Y_\sigma(1), \ldots, Y_\sigma(n))$. We have,

$$f_{X,Y}(\sigma) = \langle \mathcal{L}, \mathcal{Y}_\sigma \rangle.$$

We thus wish to compute $Pr\left\{\langle \mathcal{L}, \mathcal{Y}_\sigma \rangle = \langle \mathcal{L}, \mathcal{Y}_\pi \rangle\right\}$. Note that since the elements of Y are i.i.d. we can assume, without loss of generality, that σ is the identity permutation. Moreover, due to the linearity of the inner product, we can compute $Pr\left\{\langle \mathcal{L}, \mathcal{Y} - \mathcal{Y}_\pi \rangle = 0\right\}$. To this end, first denote the event $\mathfrak{D}_{X,Y}$ as the event that both X and Y have distinct elements. Note that

$$Pr\left\{\langle \mathcal{L}, \mathcal{Y} - \mathcal{Y}_\pi \rangle = 0\right\} \tag{5}$$
$$= Pr\left\{\langle \mathcal{L}, \mathcal{Y} - \mathcal{Y}_\pi \rangle = 0 | \mathfrak{D}_{X,Y}\right\} Pr\{\mathfrak{D}_{X,Y}\} \tag{6}$$
$$+ Pr\left\{\langle \mathcal{L}, \mathcal{Y} - \mathcal{Y}_\pi \rangle = 0 | \mathfrak{D}^c_{X,Y}\right\} Pr\{\mathfrak{D}^c_{X,Y}\} \tag{7}$$
$$\le Pr\left\{\langle \mathcal{L}, \mathcal{Y} - \mathcal{Y}_\pi \rangle = 0 | \mathfrak{D}_{X,Y}\right\} Pr\{\mathfrak{D}_{X,Y}\} + Pr\{\mathfrak{D}^c_{X,Y}\} \tag{8}$$
$$= Pr\left\{\langle \mathcal{L}, \mathcal{Y} - \mathcal{Y}_\pi \rangle = 0 | \mathfrak{D}_{X,Y}\right\} (1 - \epsilon_{\ell,n}) + \epsilon_{\ell,n}, \tag{9}$$

where $\epsilon_{\ell,n}$ denotes the probability of X and Y not having distinct elements (each), which goes to zero for fixed n yet large enough $|F|$, that is, large enough 2^ℓ.

Thus, we focus on the distinct elements of X and Y. A critical step in the computation is to consider the number of fixed points of the permutation π chosen by the adversary (compared to the original permutation σ, but, as mentioned, compared to the identity permutation without loss of generality). Any fixed point in π results in a zero element in $\mathcal{Y} - \mathcal{Y}_\pi$. However, any entry which does not correspond to a fixed point results in the difference between two uniformly random elements in the field, which is also a uniformly random element. Denote by $\mathfrak{F}_m$ the event that π has less than m fixed points (that is, at least

$n - m$ elements change their place). We have

$$
\begin{aligned}
&Pr\left\{\langle \mathcal{L}, \mathcal{Y} - \mathcal{Y}_\pi \rangle = 0 | \mathfrak{D}_{X,Y}\right\} && (10)\\
&\quad = Pr\left\{\langle \mathcal{L}, \mathcal{Y} - \mathcal{Y}_\pi \rangle = 0 | \mathfrak{D}_{X,Y}, \mathfrak{F}_m\right\} Pr\{\mathfrak{F}_m\} && (11)\\
&\qquad + Pr\left\{\langle \mathcal{L}, \mathcal{Y} - \mathcal{Y}_\pi \rangle = 0 | \mathfrak{D}_{X,Y}, \mathfrak{F}_m^c\right\} Pr\{\mathfrak{F}_m^c\} && (12)\\
&\quad \leq Pr\left\{\langle \mathcal{L}, \mathcal{Y} - \mathcal{Y}_\pi \rangle = 0 | \mathfrak{D}_{X,Y}, \mathfrak{F}_m\right\} + Pr\{\mathfrak{F}_m^c\} && (13)\\
&\quad \leq Pr\left\{\langle \mathcal{L}, \mathcal{Y} - \mathcal{Y}_\pi \rangle = 0 | \mathfrak{D}_{X,Y}, \mathfrak{F}_m\right\} + \sum_{i=m}^{n} \frac{1}{e \cdot i!} && (14)\\
&\quad \leq Pr\left\{\langle \mathcal{L}, \mathcal{Y} - \mathcal{Y}_\pi \rangle = 0 | \mathfrak{D}_{X,Y}, \mathfrak{F}_m\right\} + \frac{n-m+1}{e \cdot m!} && (15)\\
&\quad \leq \frac{|F|^{n-m+1}}{|F|^{n-m}} + \frac{n-m+1}{e \cdot m!} && (16)\\
&\quad = \frac{1}{|F|} + \frac{2}{e(n-1)!}, && (17)
\end{aligned}
$$

where (14) uses the bound for partial derangements, that is, the probability for having m fixed points [Wik23]. The inequality in (16) is since for less than m fixed points, we have more than $n - m$ independent dimensions (not fixed to zero), hence the probability of the random vector $\mathcal{Y} - \mathcal{Y}_\pi$ being orthogonal to an independent random $\mathcal{L}$ is the dimension of the null space of $\mathcal{L}$, divided by the dimension of the entire space. Note that $\mathcal{L}$ is indeed a uniformly distributed random vector over F, as each of its elements is a multiplication of i.i.d. uniformly distributed elements in the field. (17) is by choosing $m = n - 1$.

Theorem 1 asserts that the probability of a particular secret appearing when the y elements are uniformly chosen and then uniformly permuted is approximately uniform across all possible secrets for proper choices of n and l. Thus, there is no a-priori benefit in preferring one secret over another or one permutation over another. Moreover, given a certain instance with a vector Y, the number of permutations that are mapped (collisions in terms of OWF) to any particular secret is approximately uniform across all possible secrets too.

3.4 Numerical Results

We present visualized results of an experimental investigation, which sought to examine the impact of the order of the finite field and the number of points n on the distribution of the $P(0)$ values of the polynomial in multiple experiments. Figures 1 to 5 provide a visual representation of the distribution of the $P(0)$ values of the polynomial defined over a finite field with order 2^ℓ and n number of points, respectively (averaged over several experiments).

As evident from the figures, the average distribution of the $P(0)$ value of the polynomial exhibits a negligible degree of variability for the finite field's order and the number of points used. It is indeed very close to uniform in all experiments.

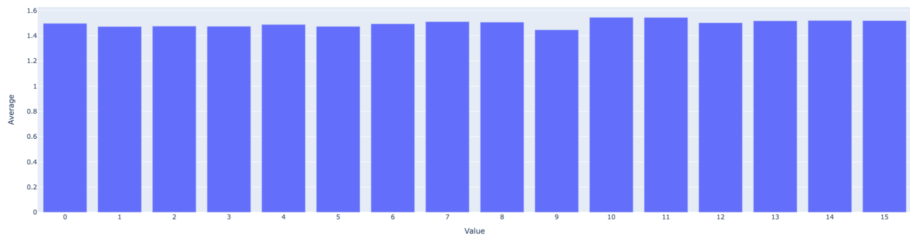

Fig. 1. Averaged distribution of $P(0)$ value over a Finite Field with order 16 and 4 points.

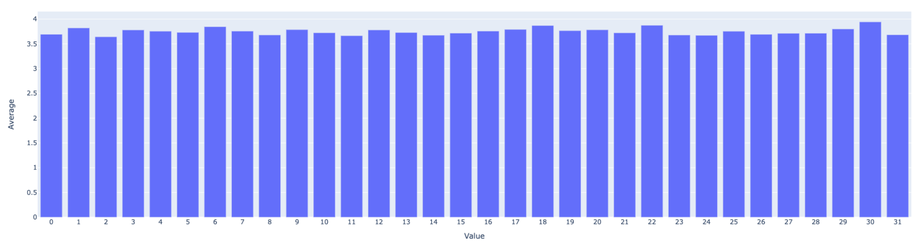

Fig. 2. Averaged distribution of $P(0)$ value over a Finite Field with order 32 and 5 points

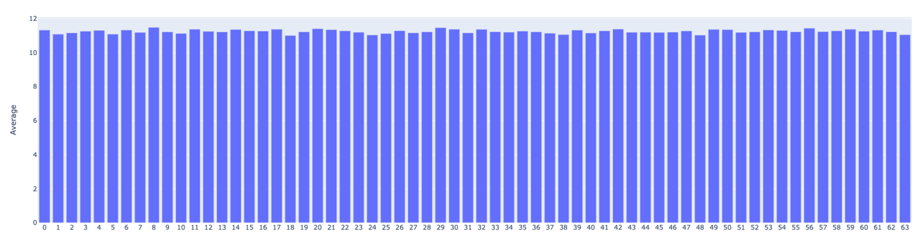

Fig. 3. Averaged distribution of $P(0)$ value over a Finite Field with order 64 and 6 points

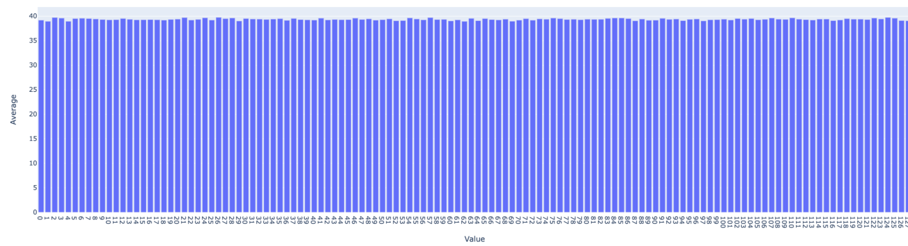

Fig. 4. Averaged distribution of $P(0)$ value over a Finite Field with order 128 and 7 points

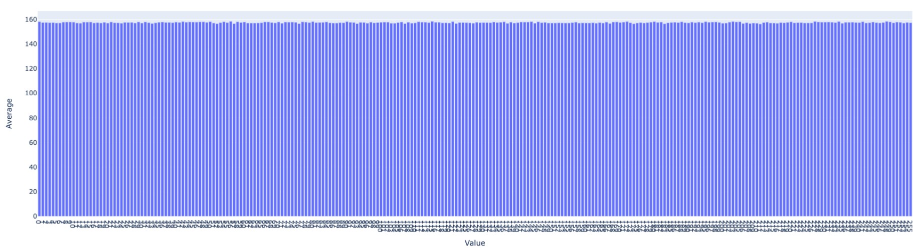

Fig. 5. Averaged distribution of $P(0)$ value over a Finite Field with order 256 and 8 points

To further understand the results, we include the table below. As mentioned, a primary objective of the study was to examine the impact of selecting the order of the finite field F and the number of points n on the observed outcomes. $W = 1000$ denotes the number of experiments conducted. We further explain the table entries.

- F: the order of the finite field. F is used in the proposed algorithm to perform all the arithmetic operations.
- n: the number of points used in the algorithm, n is also related to the order of the finite field in this specific example, $|F| = 2^n$.
- μ_{min} and μ_{max}: the minimal and maximal values of the average counts, respectively. Counts are averaged over the W experiments, each count refers to one permutation leading to the specific secret (a value in F). The difference between μ_{min} and μ_{max} is denoted by Δ_μ.
- σ and σ^2: the standard deviation and variance, respectively, to provide information about the spread of the computed averages.
- $P(A) = P(0)$: the probability that any element from the finite field is a secret (assuming uniform distribution).

- $P(\mu_{min}), P(\mu_{max})$: represents the success probability when the adversary would choose a minimal average value or maximal average value, which is defined as the ratio of μ_{min} to the order of the finite field or μ_{max} respectively.
- $P(\mu_{max})$: the success probability when the adversary would choose a value with a maximal average, which is defined as the ratio of μ_{min} to the order of the finite field.
- $\mathcal{A}_{adv}$: this parameter is defined as a percentage of the maximal potential advantage that an adversary can get by exploiting knowledge of the distribution of $P(0)$, specifically by selecting the average value with the highest number of repetitions. It is calculated as $\mathcal{A}_{adv} = (P(\mu_{max}) - P(\mu_{min})) \cdot 100\%$

Table 1. Results of experiments with varying finite field $\mathbb{F}_{|}$ and n defined as number of used points

$\mathbb{F}_{|}$	n	μ_{min}	μ_{max}	Δ_μ	σ	σ^2	P(A)=P(0)	$P(\mu_{min})$	$P(\mu_{max})$	$\mathcal{A}_{adv}$
16	4	1.447	1.5460	0.0990	0.0007	0.0272	0.0625	0.0904	0.0966	0.6188
32	5	3.643	3.9440	0.3010	0.0048	0.0693	0.0313	0.1138	0.1233	0.9406
64	6	11.009	11.4870	0.4780	0.0115	0.1074	0.0156	0.1720	0.1795	0.7469
128	7	38.955	39.7780	0.8230	0.0344	0.0185	0.0078	0.3043	0.3108	0.6430
256	8	156.324	158.457	2.1330	0.1836	0.4285	0.0039	0.6106	0.6190	0.8332

As evident from Table 1, the observed outcome of the experiment is similar to the choice of finite field and the number of points used. Results of the proposed algorithm were obtained using the SAGE software[4]. Implementation of the interactive charts with the results of our experiments are available for the reader online, along with the numerical values in [CCD23].

4 Algorithm – Techniques Integration Sample

In this section, we present an implementation based on the proposed ideas. The algorithm demonstrates the use of permutations in the scope of self–masking for two (secret sharing) polynomials. Thus, enhancing the single self-masking effect on self–masking a single (secret sharing) polynomial, presented above. The design uses a single secret permutation π that encodes (also in its nested forms) the masking in several masking permutations. The permutations of the elements of Y_1 (according to π) and the elements of Y_2 (according to π^{-1}). Then, the permutation of the bit of y_{01}, and y_{02} (according to different random bits defined by more different nested versions of π). Only then is the mutual xor used. Thus, a single π serves as proof for the commitment but is used in many forms, roughly speaking, similar to the use of a seed.

In order to provide a clear and concise representation of the implementation, we present the pseudo-code below. Note, for the sake of simplicity we omit the

[4] SageMath version 9.7, Release Date: 2022-09-19.

possibility of choosing random x values for the randomly chosen Y_1 and Y_2 coordinates and present the restricted version in which $x = 1, 2, \ldots, n$.

Algorithm 1: Polynomial based self masking algorithm

```
Input: Points_Number = n, Field_Size = GF(2^ℓ)
Result: xor_original, Y1, Y2

1  Function Generate_Points(Points_Number, Field_Size):
2      F = FiniteField(Field_Size)
3      while x < len(Points_Number) do
4          Y1 = sort(F.get_random_elements_without_repetitions())
5          Y2 = sort(F.get_random_elements_without_repetitions())
6      return Y1, Y2
7
8  Function Generate_Input():
9      Y1, Y2 = Generate_Points(n, 2^ℓ)
10     π = Generate_permutation
11     y01 = free_coefficient (lagrange_interpolation(Y1, π))
12     y02 = free_coefficient (lagrange_interpolation(Y2, π^-1))
13     y01_shuffled = shuffle(y01, π^-2, π^-3)
14     y02_shuffled = shuffle(y02, π^2, π^3)
15     xor_original = y01_shuffled ⊕ y02_shuffled
16     return xor_original, Y1, Y2
```

Function *Generate_ Points* (line 1), uses two arguments *Points Number* and *Field_ Size*. At the beginning of this function, the F object is created using the *FiniteField* class (lines 4-5). The constructor of this class takes an integer value as an argument, which determines the size of the field. Then, in the loop, list Y_1 (and independently later list Y_2) is created from *Points Number* distinct values randomly selected from F and then sorted. Lastly, the generated sorted lists are returned.

The function *Generate_ Input* starts with the invocation of the *Generate_ Points* function with the number of points n and the field 2^ℓ. As a result, two sorted lists of numbers Y_1 and Y_2 are returned, each consisting of n distinct numbers in the field. Then, a permutation π is randomly selected (line 10) and applied to Y_1. The permuted numbers in Y_1 are regarded as y coordinates of n points with the n smallest distinct x coordinates. The y coordinates are paired to the x coordinates according to the order of the y's in the permuted Y_1 and the growing order of the x coordinates. Then, Lagrange interpolation is applied to the n points, finding the free coefficient of the polynomial of degree $n - 1$ that they uniquely represent (line 11). The same is done for Y_2, but this time the y coordinates are ordered according to π^{-1}. Note that we prefer to correlate the operation on the two arrays based on knowing the solution (the permutation π) and use different correlations based on the solution, hence the choice of π and π^{-1}.

Using the sequence of the bits defined by (the permutation indexes of) π^2 and π^3 (possibly even π^4..., depending on the number of bits needed to encode a permutation of ℓ bits), we shuffle (permute the bits) of the free coefficient y_{01} (line 13) and similarly, using π^{-2} and π^{-3}, we shuffle the bits of y_{02} (line 14), and bitwise *xor* the resulting y_{01}_*shuffled* with y_{02}_*shuffled* to form *xor_original* (line 15). Lastly, we return the xor results and the two sorted vectors, each consisting of n distinct numbers in the field.

We note that our schemes work when the numbers are not necessarily distinct, and we choose to restrict the use of distinct numbers as an optimization, avoiding equivalent permutations. Also, note that one can tune the field and number of $y's$ to support independent (rather than correlated) permutations for Y_1 and Y_2.

The basic self masking technique for PP is as follows: the input for a masked function $[f]$ is a triple (Y_1, Y_2, π), where Y_1, Y_2 are two independent n-subsets of $GF(2^\ell)$ and $\pi \in SYM(n)$ is a permutation. Note that a symmetric group consists of all possible permutations of a finite set of distinct elements. Let $y_{01} = P_{Y_1,\pi}(0)$ and $y_{02} = P_{Y_2,\pi}(0)$. Then:

$$[f_{n,\ell}]\,(Y_1, Y_2, \pi) = (y_{01} \oplus y_{02}, Y_1, Y_2). \tag{18}$$

That is, the values of y_{01} and y_{02}, corresponding to inputs (Y_1, π) and (Y_2, π), mask each other by $y_{01} \oplus y_{02}$.
Let $N(Y, y)$ denote the number of "collisions" corresponding to $y_0 \in F$ when the input set is Y. i.e.

$$N(Y, y) = |\{\pi : P_{Y,\pi}(0) = y\}|. \tag{19}$$

Ideally, we would like that $N(Y, y)$ is almost the same for all $y \in F$ (i.e., it is either $\lfloor \frac{n!}{2^\ell} \rfloor$ or $\lceil \frac{n!}{2^\ell} \rceil$). Unfortunately, this is not the case for the PP case: for a given set Y of n elements, the distribution of $N(Y, y)$ is not uniform (and is dependent on the set Y).

4.1 Numerical Results

Our numerical results use small toy examples as the hardness of computation, and the needed computation power grows exponentially fast with the size of the example. Next, we present a table with the summary of our experiment and relevant figures for combinations of fields from $\{GF(2^\ell) : \ell = 1, \ldots, 19\}$, and of sequence lengths $n = 4, \ldots, 8$. Of particular interest are combinations for which $0.1 < n!/2^\ell < 1$. The results showed that while the distribution does not look totally random (e.g., the percentage of elements for which $N(Y, y) = 0$ was larger than expected), the values of $N(Y, y)$ for field $GF(2^\ell)$ and $n! < 2^\ell$ are, approximately, linear in ℓ, indicating that self masking could be hard to invert also on this case. The figures below depict distributions of collisions for a random subset of n elements of $GF(2^\ell)$ for various combinations of ℓ and n. The number of collisions for a field element y, $N(Y, y)$. is defined in Equation (19).

The basic self–masking $[f]$ of f can be extended by permuting the bits of y_{02} before masking it. First, we use the input permutation $\pi \in SYM(n)$ to define a permutation $\pi' \in SYM(\ell)$. The latter permutation π' is used to shuffle the output bits, e.g., by replacing $y_{01} \oplus y_{02}$ in Equation (18) by $y_{01} \oplus \pi'(y_{02})$. More involved shuffles are also possible, e.g., $\pi'(y_{01}) \oplus [\pi']^2(y_{02})$...

Algorithm 1 applies such shuffles. Simulation results depict that the number of collisions is not increased by applying Algorithm 1 for self–masking as depicted in Figure 6. Note that the depicted results are for the particular settings in which the x values of the points are defined in ascending order $1, \ldots, n$.

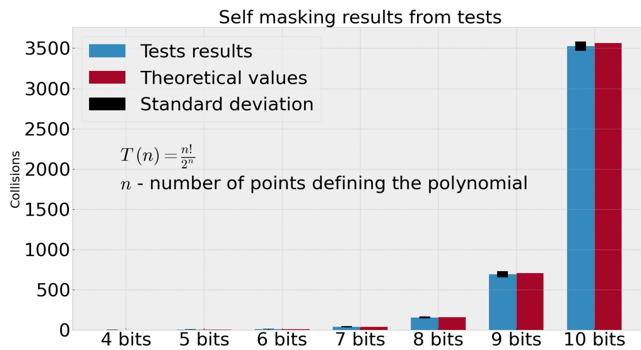

Fig. 6. Collisions as a function of n, when the Finite Field size is 2^n (typically a field different from the 2^n field is chosen. The chosen field should be in the order of $n!$, implying a low number of collisions).

5 Concluding Remarks

Permuting an array in $O(n)$ steps while providing success criteria for reversing the permutation, a permutation selected from the $n!$ possible permutations, is advocated in [DD20]. We extend the xor only approach suggested in [CDM22] to using permutation for a single (and multiple) instance case of the subset–sum problem. We also presented an algorithm building on the concepts of the new one-way function and its self–masking. In [CDM22], the parity of the i'th bit in b_1 and b_2 has been exposed; here, the permutation usage masks the parity of the corresponding bits.

We also analyze the feasibility of such permuting approach in the scope of creating a one-way function from one instance of the permuted (secret–sharing) polynomial problem. The idea can be extended to multiple instances and self–masking. One can xor permuted success criteria of more than two instances (possibly enlarging ℓ as needed) to enhance the effect of mutual permuted one-time-pad.

Acknowledgment. We thank Shlomo Moran for his input throughout the research.

References

[BM88] László Babai and Shlomo Moran. Arthur-merlin games: A randomized proof system, and a hierarchy of complexity classes. *J. Comput. Syst. Sci.*, 36(2):254–276, 1988.

[CCD23] Asaf Cohen, Pawel Cyprys, and Shlomi Dolev. Repository with the charts that represents results of the described experiments. tinyurl.com/5wb2t6cf, 2023.

[CDM22] Pawel Cyprys, Shlomi Dolev, and Shlomo Moran. Self masking for hardering inversions. *IACR Cryptol. ePrint Arch.*, page 1274, 2022.

[DD20] Hagar Dolev and Shlomi Dolev. Toward provable one way functions. *IACR Cryptol. ePrint Arch.*, page 1358, 2020.

[DLH13] Shlomi Dolev, Limor Lahiani, and Yinnon Haviv. Unique permutation hashing. *Theor. Comput. Sci.*, 475:59–65, mar 2013.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems (extended abstract). In Robert Sedgewick, editor, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985.

[HILL99] Johan Håstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal of Computing*, 28:12–24, 1999.

[IN96] Russell Impagliazzo and Moni Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of cryptology*, 9(4):199–216, 1996.

[Lam79] Leslie Lamport. Constructing digital signatures from a one way function. Technical Report CSL-98, October 1979. This paper was published by IEEE in the Proceedings of HICSS-43 in January, 2010.

[PKC22] Ray Perlner, John Kelsey, and David Cooper. Breaking category five sphincs+ with sha-256. Cryptology ePrint Archive, Paper 2022/1061, 2022. https://eprint.iacr.org/2022/1061.

[Sha79] Adi Shamir. How to share a secret. *Communications of The ACM*, 22(11):612–613, 1979.

[Sha82] Adi Shamir. A polynomial time algorithm for breaking the basic merkle-hellman cryptosystem. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *CRYPTO*, pages 279–288. Plenum Press, New York, 1982.

[SMKG22] RK Sharma, PR Mishra, Yogesh Kumar, and Nupur Gupta. Differential δ-uniformity and non-linearity of permutations over zn. *Theoretical Computer Science*, 936:1–12, 2022.

[Wik23] Wikipedia. Rencontres numbers - Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Rencontres_numbers, 2023. [Online; accessed 17-February-2023].