

# Iterated Inhomogeneous Polynomials

Jiaxin Guan\* and Mark Zhandry\*\*

Princeton University and NTT Research, USA

**Abstract.** Let  $p$  be a polynomial, and let  $p^{(i)}(x)$  be the result of iterating the polynomial  $i$  times, starting at an input  $x$ . The case where  $p(x)$  is the homogeneous polynomial  $x^2$  has been extensively studied in cryptography. Due to its associated group structure, iterating this polynomial gives rise to a number of interesting cryptographic applications such as time-lock puzzles and verifiable delay functions. On the other hand, the associated group structure leads to quantum attacks on the applications. In this work, we consider whether *inhomogeneous* polynomials, such as  $2x^2 + 3x + 1$ , can have useful cryptographic applications. We focus on the case of polynomials mod  $2^n$ , due to some useful mathematical properties. The natural group structure no longer exists, so the quantum attacks but also applications no longer immediately apply. We nevertheless show classical polynomial-time attacks on analogs of hard problems from the homogeneous setting. We conclude by proposing new computational assumptions relating to these inhomogeneous polynomials, with cryptographic applications.

## 1 Introduction

Much of modern cryptography is built on group-theoretic tools. Diffie-Hellman uses a group where discrete logarithms—computing  $a$  from  $g, g^a$ —are hard. RSA, on the other hand, uses groups where computing roots is hard, which necessarily requires groups of unknown order. Together, Diffie-Hellman and RSA are the original public key cryptosystems and the backbone of the current public key infrastructure. Since their introduction in the 1970’s, such tools have become an indispensable part of a cryptographer’s toolbox, being used for a variety of applications.

Unfortunately, relatively few suitable cryptographic groups have been studied. Diffie-Hellman was originally proposed using the multiplicative group  $\mathbb{Z}_p^* = \mathbb{Z}/p\mathbb{Z}$  or its subgroups. One can more generally consider multiplicative (sub)groups of finite fields. Elliptic curve groups are now preferred due to efficiency reasons. RSA proposed the group  $\mathbb{Z}_N^*$  for a composite  $N$ , which has unknown order if the factorization of  $N$  is unknown. For some applications, different groups such as the class group of an imaginary quadratic field can be used (e.g. [Wes19, Pie19]). Most other natural groups simply do not have the necessary computational hardness properties needed for cryptography.

---

\* [jiaxin@guan.io](mailto:jiaxin@guan.io)

\*\* [mzhandry@gmail.com](mailto:mzhandry@gmail.com)

The problem with this state of affairs is that new attacks could be found which render these groups insecure. For example, index calculus methods [AD94] give sub-exponential time algorithms for discrete logs on multiplicative groups of finite fields. While not a complete break, these attacks mean parameters have to be set very large to thwart attacks. Such methods also give sub-exponential time attacks on RSA. While other groups such as elliptic curves do not suffer from these weaknesses, it is conceivable that novel attacks may be found which can weaken or completely break security. For that reason, we believe it is important to keep searching for alternative groups, or even alternative group-theoretic structures. In doing so, we can make sure to always have suitable replacements if some structure is broken.

To make matters worse, quantum computers will render discrete logarithms and hidden group orders completely broken. Indeed, Shor’s algorithm [Sho94] can find the order of *any* group with efficient group operations, as well as solve the very discrete logarithm on any group in polynomial time on quantum computers.

To remedy this situation, Couveignes [Cou06] and Rostovtsev and Stolbunov [RS06] propose using group *actions* instead of groups. Group actions break much of the structure of a group, and in particular Shor’s algorithm no longer necessarily applies. This reduction in structure also comes at the cost of applications, though Diffie-Hellman key agreement remains possible.

As in the case for groups, few suitable group actions have been studied. The most prominent example, which was proposed in [Cou06, RS06], is derived from isogeneies over elliptic curves. Group actions can also be derived from suitable non-abelian groups. Here, too, the lack of suitable groups is potentially a problem, as novel attacks may be found. For example, Stickel’s key exchange protocol [Sti05] based on the non-abelian group of matrices was broken [Shp08]. We therefore believe it is critical to search for alternative group actions.

*Motivating Applications.* To motivate our study, we will consider two concrete applications.

- **Delay Functions.** A delay function are functions that take a long time to compute, and cannot be parallelized. A verifiable delay function (VDF), originally proposed by [BBBF18], additionally requires that once computed, it is possible to also compute a short proof  $\pi$ . A verifier can then quickly verify the outcome of the computation using  $\pi$ . The most efficient VDFs [Wes19, Pie19] have the form  $z \mapsto z^{2^t}$ , and security requires that  $z^{2^t}$  cannot be computed in time much faster than  $t$ , which matches what is possible by repeated squaring. Note that this problem is only hard if the group order  $p$  is unknown, since otherwise one can compute  $z^{2^t} = z^{2^t \bmod p}$  in time  $O(\log p) \ll t$ . In particular, these VDFs are not post-quantum secure.
- **Post-Quantum Diffie-Hellman.** In Diffie-Hellman, Alice and Bob establish a shared key over a public channel as follows. Alice chooses a random  $a \in \mathbb{Z}_p$ , and Bob a random  $b \in \mathbb{Z}_p$ , where  $p$  is the order of a cyclic group. Alice and Bob then send  $g^a, g^b$ , respectively. The shared key is then  $g^{ab} = (g^a)^b = (g^b)^a$ .

Couveignes [Cou06] and Rostovtsev and Stolbunov [RS06] observe that Diffie-Hellman can be generalized to group actions. Consider an abelian group  $\mathbb{G}$  acting on a set  $S$ , and fix some  $X \in S$ . Alice chooses a random  $a \in \mathbb{G}$  and Bob a random  $b \in \mathbb{G}$ , and they exchange  $a * X$  and  $b * X$ . The shared key is then  $(ab) * X = b * (a * X) = a * (b * X)$ . At a minimum, security requires that it is hard for an eavesdropper to recover  $a$  from  $a * X$  (and equivalently  $b$  from  $b * X$ ).

*Iterated Inhomogeneous Polynomials.* The value  $z^{2^t}$  in the delay functions/VDFs discussed above is computed by repeated squaring: iteratively apply the *homogeneous* polynomial  $x \mapsto x^2$ , starting at the point  $z$ .

In this work, we ask: what if we replace the homogeneous polynomial  $x^2$  with an *inhomogeneous* polynomial, say  $p(x) = 2x^2 + 3x + 1$ , potentially over a different modulus as well. We can similarly iterate  $p$ , computing  $p^{(i)}(z) = p(p^{(i-1)}(z))$ . By moving to a different polynomial, and in particular an inhomogeneous one, we completely change the structure of the problem. For example, whereas  $x^2$  corresponds to squaring under a natural group operation (namely multiplication), there is no obvious group operation corresponding to general polynomials. This change may help or hurt security, and may result in fewer or more varied cryptographic applications. The goal of our work is to explore these questions.

*Wishlist.* For the iterative application of a polynomial to be useful for cryptography, we may wish for some of the following features:

1. For applications like delay functions, we would like  $p^{(i)}(x)$  to be hard to “shortcut”; that is, it should take time roughly  $i$  to compute  $p^{(i)}(x)$ . A necessary requirement for the hardness of shortcutting is:
2.  $p^{(i)}(x)$  should have a very large period in  $i$ , so that  $p^{(i)}(x) \neq x$  for small  $i$ .
3. If we want *post-quantum* security, the polynomial  $p$  should not correspond to squaring in a natural group law. Indeed, iterated squaring in groups of known order can be shortcutted as discussed above; the group order can easily be computed post-quantumly [Sho94].
4. At the same time, group-based VDFs exploit the group structure for verification. So in the absence of a group structure, we would like some algebraic structure that can be used to efficiently verify.
5. Finally, if it turns out possible to efficiently shortcut  $p^{(i)}(x)$ , then this shortcutting could plausibly give a cryptographic group *action*. If  $p$  has period  $Q$ , then the additive group  $\mathbb{Z}_Q$  acts on elements  $x$  as  $i * x = p^{(i)}(x)$ . Since this action does not necessarily correspond to group multiplication, there is hope for resistance to quantum computers. In this case, we want at a minimum the analog of discrete logarithms to be hard: given  $x, p^{(i)}(x)$ , it should be hard to compute  $i$ .

*Our Setting.* We start with the following observation:

**Theorem 5.**  $2x^2 + 3x + 1$  is a permutation on  $\mathbb{Z}_{2^n}$ , and this permutation is a cycle of length  $2^n$ .

Thus, setting  $p(x) = 2x^2 + 3x + 1$  and  $N = 2^n$ , we satisfy Wishlist Item 2. We also show that  $p(x)$  does not correspond to squaring in any natural group, satisfying Wishlist Item 3.

At first glance, there is no obvious shortcutting attack, potentially giving hope that shortcutting is actually hard (Wishlist Item 1). If so, this would immediately give us a delay function. Now, because of the lack of group structure, it is not immediately clear how one would verify, so more work would be needed to achieve Wishlist Item 4.

On the other hand, if a shortcutting algorithm is found, it may seem plausible that discrete logarithms are also hard (indeed, this was our initial hope!), giving us a group action satisfying Wishlist Item 5. Note that our group action has a smooth order  $2^n$ . For groups, having such a smooth order invalidates security. But for group actions, smooth orders are acceptable, and indeed group actions based on isogenies over elliptic curves have smooth orders.

*Our Results.* Our main results are to dash these hopes, showing both that shortcutting and even discrete logarithms can be computed in classical probabilistic time polynomial in  $n$  (Section 4). Along the way, we explore the structure of the ring of polynomials over  $\mathbb{Z}_{2^n}$ , providing interesting structural results about the ring (Section 3). We conjecture that, despite our attacks above, there may be some interesting hard problem in this ring, and we conclude with some speculative cryptographic applications of this ring (Section 5).

## 2 Preliminaries

### 2.1 Notations

For a polynomial  $p$  and  $i \in \mathbb{N}$ ,  $p^{(i)}$  denotes iterating the polynomial  $i$  times, i.e.  $p^{(i)}(x) = p(p^{(i-1)}(x))$  for  $i \geq 2$  and  $p^{(1)}(x) = p(x)$ . For polynomials  $p$  and  $q$ , we use  $p \circ q$  to denote polynomial composition. We use capital bold letters to denote a matrix  $\mathbf{M}$ . Lowercase bold letters denote vectors  $\mathbf{v}$ .

For  $k \in \mathbb{N}$ , we let  $C_k$  denote the number of factors of 2 within  $k!$ , i.e. the largest integer  $m$  s.t.  $2^m | k!$ . We also denote  $d_k$  as the smallest positive integer  $d$  s.t.  $k | d!$ . Notice that we have  $C_{d_{2^n}} = n$  for  $n \in \mathbb{N}$ .

### 2.2 Cryptographic Background

Here, we recall some basic cryptographic background. For the most part, these results are used to motivate our study of polynomials over  $\mathbb{Z}_{2^n}$ , and are not necessary for our results. The only part that will be necessary for our results is Lemma 1, showing that discrete logarithms are easy in any group of smooth order.

*Cryptographic Groups.* Consider a group  $\mathbb{G}$  with an efficient group operation. We will usually think of  $\mathbb{G}$  as being abelian. For a group element  $g$ , let  $\langle g \rangle \subseteq \mathbb{G}$  be the cyclic group generated by  $g$ . While the group operation will be efficient, other computational problems may be hard. For cryptographic applications, usually we want at a minimum that the discrete logarithm problem (DLog) is hard: given  $g \in \mathbb{G}$  and  $h \in \langle g \rangle$ , find  $a$  such that  $h = g^a$ . The canonical application of a cryptographic group is Diffie-Hellman key agreement:

- Alice and Bob agree on an element  $g$  of order  $p$ .
- Alice chooses a random  $a \in \mathbb{Z}_p$  and sends Bob  $g^a$ . Bob chooses a random  $b \in \mathbb{Z}_p$  and sends Alice  $g^b$ .
- Alice and Bob compute the shared key  $K = g^{ab} = (g^b)^a = (g^a)^b$ .

The security of Diffie-Hellman requires discrete logarithms to be hard, but stronger assumptions are in fact necessary to argue security.

We now recall a necessary (but not sufficient!) property that must be satisfied for DLog to be hard:

**Lemma 1.** *Let  $p$  be the order of  $g$ , and suppose that all of the prime factors of  $p$  are at most  $B$ . In other words,  $p$  is smooth. Then there is an algorithm for DLog running in time polynomial in  $\log p$  and  $B$ .*

*Proof.* This fact is well known, but we include it here for completeness, since we will need the result. Throughout, let  $g, h = g^a$  be a discrete log instance.

Part 1:  $p \leq B$ . First, we consider the case of prime  $p \leq B$ . Then the discrete logarithm can be computed by a brute-force search: simply try all values of  $a = 0, 1, \dots, p-1$  until the correct one is found. Better algorithms are possible, such as Baby-Step Giant-Step, but this basic algorithm suffices for our purposes.

Part 2:  $p = q^k$ . Now consider the case of  $p = q^k$  where  $q$  is a prime and  $q \leq B$ . We compute the discrete logarithm as follows:

- First, compute the discrete logarithm of the pair  $(g' = g^{q^{k-1}}, h' = h^{q^{k-1}})$ . Notice that  $(g')^q = 1$  and  $h' = (g')^a = (g')^{a \bmod q}$ . Therefore, we can use the algorithm from Part 1 to solve for  $a' = a \bmod q$ .
- Next, compute  $g_1 = g^q, h_1 = h \times g^{-a'}$ . Notice that  $h_1 = g^{a-a'} = g^{q \times a_1} = g_1^{a_1}$ , where  $a_1 = (a-a')/q$  is an integer. Hence, we can solve for  $a_1$  by inductively applying the discrete log algorithm for Part 2 to the instance  $(g_1, h_1)$ <sup>1</sup>. Given  $a_1, a'$ , we can then compute  $a = qa_1 + a'$ .

Part 3: General smooth  $p$ . For the case of general smooth  $p$ , we can use the Chinese Remainder Theorem. Analogous to the first step of the  $p = q^k$  case, we can solve for the discrete logarithm mod any prime power dividing  $p$  (as long as the prime is smaller than  $B$ ). Then we can use Chinese Remaindering to combine all the discrete logarithms into the discrete logarithm mod  $p$ .  $\square$

<sup>1</sup> It is smaller since  $g_1$  has order  $q^{k-1}$ , smaller than the order of  $g$ .

*Shor's Algorithm for Discrete Logarithms.* Shor [Sho94] gives a quantum polynomial time algorithm for discrete logarithms in any group.

More generally, Shor's algorithm can solve the Abelian Hidden Subgroup Problem. This problem is defined as follows: let  $\mathbb{G}'$  be a commutative *additive* group, and  $\mathbb{H}$  an unknown subgroup. Suppose there is an efficiently computable function  $f$  with domain  $\mathbb{G}'$  such that  $f(x) = f(y)$  if and only if  $x - y \in \mathbb{H}$ . Then Shor's algorithm can find  $\mathbb{H}$ , or more precisely a set of generators for  $\mathbb{H}$ .

The discrete logarithm problem is a special case of Abelian Hidden Subgroup, as follows. Given  $g, h = g^a$ , let  $\mathbb{G}' = \mathbb{Z}_p^2$  and  $f(\alpha, \beta) = g^\alpha h^{-\beta}$ . Then  $\mathbb{H}$  is exactly the subgroup generated by  $(a, 1)$ , and so  $a$  can be easily recovered from  $\mathbb{H}$  or any set of generators for  $\mathbb{H}$ .

*Group Actions.* Couveignes [Cou06] and Rostovtsev and Stolbunov [RS06] explain how to block Shor's algorithm by using group actions, while still allowing for a version of Diffie-Hellman key agreement.

Given an additive group  $\mathbb{G}''$  acting on a set  $S$ , the new protocol is:

- Alice and Bob agree on an element  $X \in S$
- Alice chooses a random  $a \in \mathbb{G}''$  and sends Bob  $a * X$ . Bob chooses a random  $b$  and sends Alice  $b * X$ .
- Alice and Bob compute the shared key  $K = (ab) * X = a * (b * X) = b * (a * X)$ .

The analog of the discrete log problem is to compute  $a$  from  $X, a * X$ .

Any group yields a group actions, by setting  $X = g$  and  $a * X = g^a$ , in which can the group action discrete log corresponds exactly to group discrete log. However, not every group action gives a group. Importantly, there is no obvious way to adapt Shor's algorithm to general group actions. Concretely, this is because Shor's algorithm needed to multiply two group elements, computing  $g^\alpha h^{-\beta}$ . In the setting of group actions, this would require combining  $\alpha * X$  with  $\beta * X$ . In a general group action, there may not be an efficient way to perform such a combination. In the case of isogenies over elliptic curves, for example, such combinations appear hard, leading to plausible post-quantum resistance.

Another consequence is that the group order of  $\mathbb{G}''$  can actually be smooth prime powers, unlike the group case. This is because the discrete log algorithm in Lemma 1, we had to compute  $h_1 = h \times g^{-a'}$  in order to reduce from a higher power to a lower power. As in the case for Shor's algorithm, this is no longer possible with group actions. In fact, cryptosystems based on isogenies explicitly rely on the hardness of discrete logarithms in smooth prime-power group actions.

### 3 The Polynomial Rings

In this section, we will formalize the rings of polynomials that we work with.

*The Ring  $R$ .* We first define  $R = \mathbb{Z}_{2^n}[x]$ , the ring of polynomials over  $\mathbb{Z}_{2^n}$ .

### 3.1 Equivalence Classes

First, we will define the equivalence classes for such polynomials, i.e., which polynomials are equivalent congruent to  $2^n$ . To do so, we start by examining the polynomials that are equivalent to the zero polynomial (one that is identically zero) mod  $2^n$ .

We first recall the following Lemma on integer-valued polynomials due to Pólya [Pól15].

**Lemma 2 ([Pól15]).** *Inside the polynomial ring  $\mathbb{Q}[x]$  of polynomials with rational number coefficients, the subring of integer-valued polynomials is a free abelian group with a basis of polynomials*

$$P'_k(x) = \frac{x(x-1)\cdots(x-k+1)}{k!}$$

for  $k = 0, 1, 2, \dots$

Using Lemma 2, we are now able to characterize the polynomials that are identically zero mod  $2^n$ .

**Theorem 1.** *In the polynomial ring  $R$ , the subset of polynomials that are equivalent to the zero polynomial modulo  $2^n$  is a free abelian group with a basis of polynomials*

$$P_k(x) = 2^{\max(0, n-C_k)} x(x-1)\cdots(x-k+1)$$

for  $k = 0, 1, 2, \dots$

*Proof.* Let  $p(x)$  be some polynomial that is equivalent to the zero polynomial mod  $2^n$ . This means that  $p(x) \equiv 0 \pmod{2^n}$  for all  $x$ . More specifically, we have

$$p(x) = k_x \cdot 2^n$$

for integer-valued  $k_x$ 's.

Dividing both sides by  $2^n$  yields  $p(x)/2^n = k_x$  for all  $x$ , i.e.  $p(x)/2^n$  is an integer-valued polynomial with rational coefficients. Thus, by Lemma 2,  $p(x)/2^n$  must come from the group with a basis of  $\{P'_k\}_k$ . Correspondingly,  $p(x)$  is formed from the basis of polynomials  $P''_k(x) = 2^n P'_k(x) = (2^n/k!)x(x-1)\cdots(x-k+1)$ . But notice that  $p(x)$  should have coefficients in  $\mathbb{Z}_{2^n}$ , but the coefficients for  $P_k(x)$  are still rationals. Thus we need to scale  $P_k$  to obtain integer coefficients.

Toward that end, consider the leading coefficient  $(2^n/k!)$  of  $2^n P'_k$ . Note that the GCD of  $2^n$  and  $k!$  is  $2^{C_k}$ , so the leading coefficient can be reduced as  $2^{n-C_k} / (k!/2^{C_k})$ . Observe that  $k!/2^{C_k}$  is odd, and therefore invertible mod  $2^n$ . We can therefore scale any polynomial in the basis by  $k!/2^{C_k}$ . Therefore,  $p(x)$  is formed from the basis  $P'''_k(x) = 2^{n-C_k} x(x-1)\cdots(x-k+1)$ . If  $n \geq C_k$ , this is an integer-valued polynomial, and in particular has coefficients in  $\mathbb{Z}_{2^n}$ . If  $n < C_k$ , then the leading coefficient is divided by a power of 2, which is not a unit in  $\mathbb{Z}_{2^n}$ . In the  $n < C_k$  case, we therefore have to multiply by  $2^{C_k-n}$  to get a polynomial with coefficients in the ring. The result is the basis of polynomials:

$$P_k(x) = 2^{\max(0, n-C_k)} x(x-1) \dots (x-k+1).$$

We can also quickly verify that the polynomials in the group formed by this basis are indeed equivalent to zero mod  $2^n$ . This comes directly from the fact that  $P_k(x) \equiv 0 \pmod{2^n}$  for all  $k$  and  $x$ . Notice that since  $k! | x(x-1) \dots (x-k+1)$ ,  $x(x-1) \dots (x-k+1)$  has at least  $C_k$  number of factors of 2's. As a result,  $P_k(x)$  has at least  $(n - C_k) + C_k = n$  number of 2's, and hence a multiple of  $2^n$ , i.e.  $0 \pmod{2^n}$ .  $\square$

Using these zero polynomials, we further define the equivalence classes.

**Corollary 1.** *For polynomials  $p(x), q(x) \in R$ ,  $p(x) \equiv q(x) \pmod{2^n}$  for all  $x \in \mathbb{Z}_{2^n}$  if and only if  $p(x) - q(x)$  is in the free abelian group with  $\{P_k\}_k$  as the basis.*

*The Reduced Ring  $\tilde{R}$ .* We are now ready to define our reduced ring  $\tilde{R}$  as:

$$\tilde{R} = R / \langle \{P_k\} \rangle .$$

In other words,  $\tilde{R}$  is the ring of equivalence classes of  $R$  under the polynomials  $\{P_k\}_k$ .

Corollary 1 means that when given a polynomial  $p(x)$  with integer coefficients, we can always reduce it by subtracting out multiples of  $P_k$  for all  $k \geq d_{2^n}$ , where  $d_{2^n}$  denotes the smallest  $d$  s.t.  $2^n | d!$ . As a result, we will arrive at a polynomial with degree at most  $d_{2^n} - 1$ . We can further reduce the coefficient of each lower degree term (with degree  $k$ ) to be between  $[0, 2^{n-C_k})$ . Notice that by Corollary 1, such polynomials (ones that have degree at most  $d_{2^n} - 1$ , and where for all degrees  $0 \leq k \leq d_{2^n} - 1$ , the coefficient for the degree- $k$  term is between  $[0, 2^{n-C_k})$ ) are all in distinct equivalence classes.

Therefore, we can equivalently define the ring  $\tilde{R}$  as in a more operational manner as vectors of length  $d_{2^n}$  whose degree- $k$  coefficient is in  $[0, 2^{n-C_k})$ . Addition is component-wise and multiplication is a convolution, but then followed by a reduction by the polynomials  $\{P_k\}_k$ .

### 3.2 Group of Units

Now that we have defined the overall polynomial ring  $R$  and its reduced ring  $\tilde{R}$ , we explore the group  $\tilde{R}^\times$  of units in the ring. Recall that a unit in  $\tilde{R}$  is a polynomial  $p \in \tilde{R}$  such that there exists  $p^{-1} \in \tilde{R}$  with  $p(p^{-1}(x)) = p^{-1}(p(x)) = x$  for all  $x \in \mathbb{Z}_{2^n}$ .

One direct observation is that whether a polynomial is a unit is directly related to whether it has a collision in  $\mathbb{Z}_{2^n}$ .

**Theorem 2.** *A polynomial  $p$  is a unit if and only if  $p(x)$  has no collisions in  $\mathbb{Z}_{2^n}$ , i.e.  $\forall x_1 \neq x_2 \in \mathbb{Z}_{2^n}$ ,  $p(x_1) \not\equiv p(x_2) \pmod{2^n}$ .*

*Proof.* First, it's easy to see that if  $p(x)$  has a collision, then  $p$  cannot be a unit. Let  $x_1 \neq x_2$  be such a collision s.t.  $p(x_1) \equiv p(x_2)$ . Assume towards contradiction that  $p$  has a compositional inverse  $p^{-1}$ . Then we'd have  $x_1 \equiv p^{-1}(p(x_1)) \equiv p^{-1}(p(x_2)) \equiv x_2$  as a contradiction.

Next, we show that if  $p(x)$  has no collisions, then  $p$  is a unit. If  $p(x)$  has no collisions, since the domain and range are finite and have the same size, we know  $p(x)$  is a permutation over  $\mathbb{Z}_{2^n}$ . Therefore,  $p$  will consist of one or more cycles. Let the cycle lengths be  $\ell_1, \ell_2, \dots$ . Then the compositional inverse  $p^{-1}$  can be simply computed as  $p^{(\text{lcm}(\ell_1, \ell_2, \dots) - 1)}$  (reduced to  $\tilde{R}$ ). Then we have that  $p^{(\text{lcm}(\ell_1, \ell_2, \dots) - 1)}(x) = p^{(\text{lcm}(\ell_1, \ell_2, \dots))}(x) = x$  for all  $x \in \mathbb{Z}_{2^n}$ .  $\square$

Using this result, we can now specify the group of units  $\tilde{R}^\times$  for the ring  $\tilde{R}$ .

**Theorem 3.** *A polynomial  $p = c_0 + c_1x + c_2x^2 + \dots \in \tilde{R}$  is a unit if and only if all of the following holds:*

- (1)  $c_1 \equiv 1 \pmod{2}$ : *The linear coefficient is odd.*
- (2)  $\sum_{i=1} c_{2i+1} \equiv 0 \pmod{2}$ : *The sum of coefficients for odd-degree terms other than the linear term is even.*
- (3)  $\sum_{i=1} c_{2i} \equiv 0 \pmod{2}$ : *The sum of coefficients for even-degree terms other than the constant term is even.*

*Proof.* First, we show that (1), (2), and (3) are necessary conditions.

First, we claim that if  $p$  has a collision over  $\mathbb{Z}_{2^{n'}}$  for  $n' < n$ , then it must also have a collision over  $\mathbb{Z}_{2^n}$ . To prove this, it suffices to consider  $n' = n - 1$ , and then use induction to get smaller  $n'$ . Suppose we have a collision in  $\mathbb{Z}_{2^{n-1}}$ :  $x_0 \neq x_1 \in \mathbb{Z}_{2^{n-1}}$  such that  $a := p(x_0) = p(x_1) \pmod{2^{n-1}}$ . Now consider the four distinct points  $x_b + c2^{n-1} \pmod{2^n}$  for  $b, c \in \{0, 1\}$ . Note that  $p(x_b + c2^{n-1}) \pmod{2^n} \in \{a, a + 2^{n-1}\}$ . Therefore, we have 4 distinct points mapping to two values, meaning there must be a collision amongst them.

We now look specifically at the case  $n' = 2$ , and conclude that if  $p$  has no collisions over  $\mathbb{Z}_{2^n}$ , then  $p$  has no collision over  $\mathbb{Z}_4$ . Notice that  $x^4 \equiv x^2 \pmod{4}$ . Therefore, we can reduce the polynomial  $p$  to an equivalent polynomial with degree at most 3. During the reduction process, all the coefficients for the odd-degree terms (other than the linear term) will be sum up into the cubic coefficient, and the coefficients for the even-degree terms (other than the constant term) will be sum up into the quadratic coefficient.

Therefore, to prove that (1),(2),(3) are necessary, it suffices to prove the the case  $n = 2$ . A simple but tedious search over all  $4^4 = 256$  polynomials of degree at most 3 in  $\mathbb{Z}_4[x]$  confirms the theorem in this case, namely that a polynomial in  $\mathbb{Z}_4[x]$  is a permutation if and only if it has even cubic and quadratic coefficients, and an odd linear coefficient.

Next, we show that these conditions are sufficient via induction on  $n$ . Suppose  $p$  is a polynomial such that (1), (2), and (3) hold.

For the base case of  $n = 1$ , we have  $x^2 \equiv x \pmod{2}$ . So any polynomial  $p$  satisfying (1), (2), and (3), can be reduced to either  $p(x) = x$  or  $p(x) = x + 1$ , and both of these are indeed units.

For the inductive step, assume that  $p$  is a unit mod  $2^{n-1}$ . Suppose  $p$  is not a unit mod  $2^n$ . Then there is collision  $x \neq y$  such that  $p(x) \equiv p(y) \pmod{2^n}$ . Since this in particular means  $p(x) \equiv p(y) \pmod{2^{n-1}}$ , but  $p$  is a unit mod  $2^{n-1}$ , we must have  $y = x + 2^{n-1}$ .

Plugging this in, we have

$$0 \equiv p(x + 2^{n-1}) - p(x) \equiv \sum_i c_i [(x + 2^{n-1})^i - x^i] \pmod{2^n}$$

If we look at the term  $[(x + 2^{n-1})^i - x^i]$ , we see the following:

1. For  $i = 0$ , we get 0.
2. For  $i = 1$ , we get  $2^{n-1}$ .
3. For  $i > 1$ , if we expand the exponential, we get a term  $i \cdot 2^{n-1} \cdot x^{i-1}$ , together with other terms that all have a factor of  $2^{2(n-1)}$ . Notice that these other terms are simply 0 mod  $2^n$ .

For the  $i > 1$  terms, since we are multiplying by  $2^{n-1}$ , we can reduce everything else in the product mod 2. Thus, for  $i > 1$ ,  $[(x + 2^{n-1})^i - x^i]$  becomes  $2^{n-1} \cdot \text{parity}(i) \cdot x$ . Thus,

$$p(x + 2^{n-1}) - p(x) \equiv 2^{n-1} \left( 1 + x \cdot \sum_{i>1} c_i \cdot \text{parity}(i) \right) \pmod{2^n}$$

By (2),  $\sum_{i>1} c_i \cdot \text{parity}(i)$  is 0. But then we'd have  $p(x + 2^{n-1}) - p(x) = 2^{n-1} \pmod{2^n}$ , contradicting the assumption that  $x + 2^{n-1}$  and  $x$  were a collision over  $\mathbb{Z}_{2^n}$ . So  $p$  doesn't have a collision in  $\mathbb{Z}_{2^n}$ , and is hence a unit.  $\square$

*The subring  $S$ .* Next, we will introduce the subring of polynomials of the form

$$p(x) = \sum_{k>=1} 2^{k-1} \cdot c_k \cdot x^k + c_0$$

where  $c_0, c_1, \dots$  are all integers. Let us denote this subring as  $S$ . Notice that  $S$  is a subring of  $R$  where the  $k$ -th degree coefficient has a  $2^{k-1}$  factor in it. We introduce this subring for two reasons: first, within this subring, we can easily identify the units that correspond to full-length cycles, which is important for security. The other reason is for efficiency, as we can store a polynomial in this subring using less storage.

*The Reduced Subring  $\tilde{S}$ .* Using Corollary 1, we can also reduce the subring  $S$ . We claim that the reduced subring  $\tilde{S}$  also has the characterization that the  $k$ -th degree coefficient has a  $2^{k-1}$  factor.

**Theorem 4.** *The reduced subring  $\tilde{S}$  consists of polynomials of the form*

$$p(x) = \sum_{k=1}^{d_{2^n}-1} 2^{k-1} \cdot c_k \cdot x^k + c_0$$

where  $c_0, c_1, \dots$  are all integers, and  $2^{k-1} \cdot c_k \in [0, 2^{n-C_k})$  for all  $k \in [1, d_{2^n} - 1]$ .

*Proof.* To see this, we show that this invariant is preserved throughout the reduction process. Recall that the reduction process has two phases. In the first phase, we repeatedly subtract multiples of  $P_k(x)$  from the highest degree  $k$  until we reduce the highest degree to be smaller than  $d_{2^n}$ . Notice that in this step, the term we are subtracting out is  $2^{k-1} \cdot c_k \cdot P_k(x)$ , which is a multiple of  $2^{k-1}$ . So in the reduced polynomial, for all the degree- $k'$  terms with  $k' < k$ , they still contain a factor of  $2^{k'-1}$ .

In the second phase, we go from  $k = d_{2^n} - 1$  all the way to  $k = 1$  and subtract out multiples of  $2^{n-C_k} x(x-1) \cdots (x-k+1)$  so that the coefficient for each degree- $k$  term is between  $[0, 2^{n-C_k})$ . Notice that for degree  $k$ , if  $k-1 \geq n - C_k$ , then the original coefficient  $2^{k-1} \cdot c_k$  is a multiple of  $2^{n-C_k}$ . So the term subtracted will be  $2^{k-1} \cdot c_k \cdot x(x-1) \cdots (x-k+1)$ , a multiple of  $2^{k-1}$ . On the other hand, if  $k-1 < n - C_k$ , then the term subtracted out will be some multiple of  $2^{n-C_k} x(x-1) \cdots (x-k+1)$ , which again is a multiple of  $2^{k-1}$ . By a similar argument as in phase one, all the remaining coefficients still have a factor of  $2^{k-1}$ .

□

Since  $\tilde{S}$  is itself a subring of  $\tilde{R}$ , so the characterization of the group of units in Theorem 2 also extends. More specifically, (2) and (3) are always true for polynomials in  $\tilde{S}$  due to the  $2^{k-1}$  factor. So the only characterization here would be an odd linear coefficient.

**Corollary 2.** *The group of units in the subring  $\tilde{S}$ , denoted as  $\tilde{S}^\times$ , is characterized by an odd linear coefficient, i.e.  $c_1 = 2i + 1$  for some integer  $i$ .*

### 3.3 Number of Zeros

In this section, we bound the number of zeros of the polynomials mod  $2^n$ .

**Lemma 3.** *For any polynomial  $p \in R$  evaluated over  $\mathbb{Z}_{2^n}$ , it is either identically zero, or has at least  $\frac{2^n}{n}$  non-zeros.*

*Proof.* For a polynomial  $p$ , let  $f_n(p)$  denote be the fraction of zeros mod  $2^n$ . Let  $f_n^*$  be the lowest fraction of non-zero entries a not-identically-zero polynomial can have mod  $2^n$ .

Suppose  $f_n^* < f_{n-1}^*$ . Then let  $p(x)$  be a polynomial such that  $f_n^* = f_n(p)$ . Since  $f_{n-1}(p) \leq f_n(p)$ , we must have that  $p$  is identically zero mod  $2^{n-1}$ .

We will need that  $p(x)$  is all zeros on odd  $x$ , but we will actually prove this as part of the proof. Instead we will just assume that  $p$  has at least one even non-zero, which is w.l.o.g by replacing  $p(x)$  with  $p(x+1)$  if necessary.

We now factor out just the odd zero. This means we can write  $p(x) = (x - z_1) \cdots (x - z_t)q(x)$ , where  $z_i$  are odd, and  $q(x)$  is non-zero on all odds.

We first note that  $(x - z_1) \cdots (x - z_t)$  must be zero on all odds. Otherwise, if there was some non-zero odd  $z_{t+1}$ , we could multiply  $p(x)$  by  $(x - z_{t+1})$ . This doesn't change the even zeros (and so the polynomial is still not identically zero)

but increases the number of zeros, contradicting the fact that  $p$  has the smallest number of non-zeros.

Now let  $K(x)$  be the polynomial of degree roughly  $n/2$  that is monic and zero on all the evens. Let  $K'(x) = K(x)(x - z_1) \cdots (x - z_t)$ , which is identically zero everywhere. Therefore, we can assume w.l.o.g. that  $p(x)$  is reduced mod  $K'(x)$ , which is equivalent to assuming that  $q(x)$  is reduced mod  $K(x)$ . In particular, we can assume that  $q(x)$  has degree at most around  $n/2$ .

From what we said before, we know that for all even  $x$ ,  $q(x) = 0 \pmod{2^{n-1}}$ , since  $p(x) = 0 \pmod{2^{n-1}}$  and  $(x - z_1) \cdots (x - z_t)$  is odd when  $x$  is even. Therefore,  $q(2x)$  is identically  $0 \pmod{2^{n-1}}$ . Therefore  $q(2x)$  must be in the span of our basis polynomials. But since  $q(2x)$  has degree at most  $n/2$ , this implies that it is divisible by  $2^v$ , where  $v \sim n/2$ . Note that  $f_n(q(2x)) = 2f_n(p)$ , by our assumption that all the non-zeros of  $p$  are even.

So we let  $p'(x) = q(2x)/2^v$ . Then  $f_{n-v}^* \leq f_{n-v}(p') = f_n(q(2x)) = 2f_n(p) = 2f_n^*$ . Thus, we decrease  $n$  by a factor of around 2, but at most reduce  $f_n^*$  by a factor of 2. This means that  $f_n^*$  is at least  $1/n$ . □

### 3.4 A Class of Full-Length Cycles

Now we consider iterated polynomials of the form

$$p_{k,l,m}(x) = 2kx^2 + (4l + 2k + 1)x + (2m + 1)$$

over  $\mathbb{Z}_{2^n}$ , where  $k, l, m$  take integer values.

First of all, notice that  $p_{k,l,m}$  is a unit, and hence it is a permutation over  $\mathbb{Z}_{2^n}$ .

Next up, we prove an interesting cycling property of such polynomials: when applied iteratively, the outputs of such a polynomial form a full-length cycle of all elements in  $\mathbb{Z}_{2^n}$ , i.e. a cycle of length  $2^n$ .

**Theorem 5.** *Let the sequence  $x_0, x_1, \dots \in \mathbb{Z}$  be defined as*

$$\begin{cases} x_0 = 0 \\ x_i = p_{k,l,m}(x_{i-1}) \text{ for } i > 0. \end{cases}$$

*Then for any  $i$ , the smallest non-zero  $j$  s.t.  $x_i \equiv x_{i+j} \pmod{2^n}$  is equal to  $2^n$ .*

*Proof.* It suffices to prove the case  $i = 0$ . Let  $t \leq n$  be an integer. We prove this via induction on the following two claims:

- (1) If the sequence is reduced mod  $2^t$  (instead of mod  $2^n$ ), it will hit all possible values between 0 and  $(2^t - 1)$  after  $2^t$  steps (and thus we have a cycle).
- (2)  $x_{2^t} = d \cdot 2^t$  for an odd integer  $d$ .

The base case with  $t = 1$  is easy to verify. We will inductively assume the two claims hold for  $t - 1$ .

Now, notice that adding  $2^{t-1}$  to  $x$  has an interesting effect, reduced mod  $2^t$  and  $2^{t+1}$ :

$$\begin{aligned}
p_{k,l,m}(x + 2^{t-1}) &= 2k(x + 2^{t-1})^2 + (4l + 2k + 1)(x + 2^{t-1}) + (2m + 1) \\
&= 2kx^2 + 2^{t+1}kx + 2^{2t-1}k + (4l + 2k + 1)x \\
&\quad + 2^{t+1}l + 2^t k + 2^{t-1} + (2m + 1) \\
&= (2kx^2 + (4l + 2k + 1)x + (2m + 1)) + 2^{t-1} \\
&\quad + 2^t(2kx + 2^{t-1}k + 2l + k) \\
&\equiv p_{k,l,m}(x) + 2^{t-1} \pmod{2^t}
\end{aligned} \tag{a}$$

$$\begin{aligned}
p_{k,l,m}(x + 2^{t-1}) &= 2kx^2 + 2^{t+1}kx + 2^{2t-1}k + (4l + 2k + 1)x \\
&\quad + 2^{t+1}l + 2^t k + 2^{t-1} + (2m + 1) \\
&= (2kx^2 + (4l + 2k + 1)x + (2m + 1)) + 2^t k + 2^{t-1} \\
&\quad + 2^{t+1}(kx + 2^{t-2}k + l) \\
&\equiv p_{k,l,m}(x) + (2k + 1) \cdot 2^{t-1} \pmod{2^{t+1}}
\end{aligned} \tag{b}$$

Now consider the sequence  $x_0, x_1, \dots \pmod{2^t}$ , and go for  $2^t$  steps. By part (2) of the inductive hypothesis, after  $2^{t-1}$  steps we will get an odd multiple of  $2^{t-1}$ , which reduced mod  $2^t$  gives us exactly  $2^{t-1}$ . Thus, after  $2^{t-1}$  steps, we have added  $2^{t-1}$  to the initial value. By equation (a), the remaining  $2^{t-1}$  steps are then just the same as the first  $2^{t-1}$ , but with  $2^{t-1}$  added to them. There can be no collisions between the two halves of the sequence, since if we were to reduce by  $2^{t-1}$ , we'd get a cycle whose length isn't exactly  $2^{t-1}$ .

This proves claim (1) holds for  $t$ . Now what's left is to show claim (2) holds for  $t$  too.

Consider the term  $x_{2^{t-1}}$ . By claim (2) for  $(t-1)$ , we know that  $x_{2^{t-1}} \equiv d \cdot 2^{t-1} \pmod{2^{t+1}}$  where  $d \equiv 1$  or  $3 \pmod{4}$ . We break into two cases depending on  $d$ :

– Case 1,  $d \equiv 1 \pmod{4}$ :

By equation (b) we have

$$\begin{aligned}
x_{2^{t-1}+1} &\equiv p_{k,l,m}(x_{2^{t-1}}) \\
&\equiv p_{k,l,m}(0 + 2^{t-1}) \\
&\equiv p_{k,l,m}(0) + (2k + 1) \cdot 2^{t-1} \\
&\equiv x_1 + (2k + 1) \cdot 2^{t-1} \pmod{2^{t+1}}.
\end{aligned}$$

Now we apply (b) once again,

$$\begin{aligned}
x_{2^{t-1}+2} &\equiv p_{k,l,m}(x_{2^{t-1}+1}) \\
&\equiv p_{k,l,m}(x_1 + (2k + 1) \cdot 2^{t-1}) \\
&\equiv x_2 + (2k + 1)^2 \cdot 2^{t-1} \\
&\equiv x_2 + 2^{t-1} \pmod{2^{t+1}}.
\end{aligned}$$

Inductively, we'll have for all even  $u$  that  $x_{2^{t-1}+u} \equiv x_u + 2^{t-1} \pmod{2^{t+1}}$ .  
 Now for all  $t \geq 2$ , we have that  $x_{2^t} \equiv x_{2^{t-1}} + 2^{t-1} \equiv 2^t \pmod{2^{t+1}}$ .

– Case 2,  $d \equiv 3 \pmod{4}$ : It's analogous to case 1. Using equation (b) we have

$$x_{2^{t-1}+1} \equiv x_1 + 3 \cdot (2k + 1) \cdot 2^{t-1} \pmod{2^{t+1}}.$$

Applying (b) again yields

$$x_{2^{t-1}+2} \equiv x_2 + 3 \cdot (2k + 1)^2 \cdot 2^{t-1} \equiv x_2 + 3 \cdot 2^{t-1} \pmod{2^{t+1}}.$$

Inductively, we have  $x_{2^{t-1}+u} \equiv x_u + 3 \cdot 2^{t-1} \pmod{2^{t+1}}$  for all even  $u$ . Now for all  $t \geq 2$ , we have that  $x_{2^t} \equiv x_{2^{t-1}} + 3 \cdot 2^{t-1} \equiv 6 \cdot 2^{t-1} \equiv 2^t \pmod{2^{t+1}}$ .

In either case, we have  $x_{2^t} = 2^t \pmod{2^{t+1}}$ , i.e.  $x_{2^t} = d \cdot 2^t$  for some odd  $d$ , as desired. This completes the inductive step for claim (2) and hence the whole proof.  $\square$

### 3.5 Characterization of Full-Length Cycles in $\tilde{S}^\times$

By generalizing the class of polynomials above, we characterize the units in  $\tilde{S}^\times$  would give us full-length cycles.

**Theorem 6.** *A polynomial  $p \in \tilde{S}^\times$  has a cycle length of  $2^n$  when evaluated mod  $2^n$  if and only if it has the following form:*

$$p(x) = \sum_{j=3}^{d_{2^n}-1} 2^{j-1} \cdot c_j \cdot x^j + 2kx^2 + (4l + 2k + 1)x + (2m + 1)$$

where  $k, l, m, c_3, c_4, \dots$  are all integers, and the coefficient for the degree  $j$  term is in  $[0, 2^{n-C_j})$  for all  $j \in [0, d_{2^n} - 1]$ .

*Proof.* The proof that this characterization is a sufficient condition of a full-length cycle follows the exact same outline as the proof for Theorem 5. In the proof, the base case is where we reduce mod 2, so the higher degree terms simply go away. Hence the base case trivially holds. The proof for the inductive step is based on equations (a) and (b), which still hold here:

$$\begin{aligned}
p(x + 2^{t-1}) - p(x) &= \sum_{j=3}^{d_2^n - 1} 2^{j-1} \cdot c_j \cdot ((x + 2^{t-1})^j - x^j) + 2k((x + 2^{t-1})^2 - x^2) \\
&\quad + (4l + 2k + 1)(x + 2^{t-1} - x) \\
&= \sum_{j=3}^{d_2^n - 1} 2^{j-1} \cdot c_j \cdot \sum_{r=1}^j 2^{(t-1)r} x^{j-r} + 2^{t+1}kx + 2^{2t-1}k \\
&\quad + 2^{t+1}l + 2^t k + 2^{t-1} \\
&= 2^{t+1} \sum_{j=3}^{d_2^n - 1} 2^{j-3} \cdot c_j \cdot \sum_{r=1}^j 2^{(t-1)(r-1)} x^{j-r} + 2^{t+1}kx \\
&\quad + 2^{2t-1}k + 2^{t+1}l + 2^t k + 2^{t-1} \\
&\equiv (2k + 1) \cdot 2^{t-1} \pmod{2^{t+1}} & (b') \\
&\equiv 2^{t-1} \pmod{2^t} & (a')
\end{aligned}$$

Using (a') and (b'), the rest of the proof is identical.

Now we argue why this form is a necessary condition. Notice that if  $p$  is not a full-length cycle mod  $2^t$ , then it is also not a full-length cycle mod  $2^{t+1}$ . This is because when we move from mod  $2^t$  to mod  $2^{t+1}$ , the cycle length of  $p$  can at most double, so it can never reach a cycle length of  $2^{t+1}$  if it wasn't already full-length mod  $2^t$ . The contrapositive of this states that if  $p$  is a full-length cycle mod  $2^n$ , then it must also be a full-length cycle mod  $2^{n'}$  for all  $n' < n$ . So it suffices for us to look at polynomials in  $\tilde{S}^\times \pmod{4}$ . Notice that since we're reducing by mod 4, all the degree-3 terms and beyond don't matter as they have a factor of 4 in the coefficient. And we easily brute-force to see which degree-2 polynomials have full-length cycles mod 4. It turns out that the only possibilities are  $W = \{x + 1, x + 3, 2x^2 + 3x + 1, 2x^2 + 3x + 3\}$ . And it's trivial to see that  $p$  must have the above-mentioned form if  $p \pmod{4} \in W$ .  $\square$

## 4 Attacks

### 4.1 Shortcutting

In this subsection we show how one can compute  $p^{(T)}(x)$  efficiently in time sublinear of  $T$ .

**Theorem 7.** *Given a polynomial  $p \in \tilde{R}$  and an integer  $T$ , there exists an algorithm that computes  $p^{(T)}(x) \pmod{2^n}$  on any input  $x$ , and takes time  $O(n^3 \log T)$ .*

*Proof.* We start by examining how one can compute  $p^{(i+1)}$  given  $p^{(i)}$ . Let  $p^{(i)}(x) = a_0 + a_1x + a_2x^2 + \dots + a_{d_2^n - 1}x^{d_2^n - 1} = (a_0, a_1, \dots, a_{d_2^n - 1}) \cdot (1, x, x^2, \dots, x^{d_2^n - 1})^\top$ .

Notice that the polynomial  $p^{(i)}$  can be represented by the vector  $\mathbf{a} = (a_0, a_1, \dots, a_{d_{2^n}-1})$  which consists of all of its coefficients.

Then we have

$$\begin{aligned} p^{(i+1)}(x) &= p^{(i)}(p(x)) \\ &= (a_0, a_1, \dots, a_{d_{2^n}-1}) \cdot (1, p(x), (p(x))^2, \dots, (p(x))^{d_{2^n}-1})^\top \end{aligned}$$

Now let us examine what the coefficients of  $p^{(i+1)}(x)$  are. Let us denote the coefficients as a vector  $\mathbf{b} = (b_0, b_1, \dots, b_{2d_{2^n}-2})$ . Kindly notice that here the vector is of length  $2d_{2^n}-1$ , instead of  $d_{2^n}$ , since this  $p^{(i+1)}(x)$  is still in the ring  $R$ , not yet reduced to  $\tilde{R}$ .

It's easy to see that we have  $b_0 = \sum_{j=0}^{d_{2^n}-1} a_j$ . With some effort, we can also compute  $b_1 = \sum_{j=0}^{d_{2^n}-1} \binom{j}{1} c_1 a_j$  where  $c_1$  is the linear coefficient of the original  $p$ . Similarly, we have  $b_2 = \sum_{j=0}^{d_{2^n}-1} \left( \binom{j}{2} c_2 + \binom{j}{2} c_1^2 \right) a_j$ , so on and so forth. But the important thing to notice here is that  $b_0, b_1, b_2, \dots$  are all *linear* transformations on the vector  $\mathbf{a}$ . Therefore, we can compute  $\mathbf{b} = \mathbf{a} \cdot \mathbf{M}'$ , where  $\mathbf{M}'$  is a transformation matrix of dimension  $d_{2^n} \times (2d_{2^n}-1)$ .

The next thing to do is to reduce this  $p^{(i+1)}$  to  $\tilde{R}$ . Recall the reduction process from 1. We will start from the highest degree, and repeatedly subtract out multiples of the falling factorial. Notice that this process is also strictly *linear* in terms of the coefficients  $\mathbf{b}$ . Therefore, after we reduce the polynomial  $p^{(i+1)}$  to  $\tilde{R}$ , the new reduced coefficient vector  $\tilde{\mathbf{b}}$  is also a linear transformation on  $\mathbf{a}$  and can be computed as

$$\tilde{\mathbf{b}} = \mathbf{a} \cdot \mathbf{M}$$

where  $\mathbf{M}$  is a transformation matrix of dimension  $d_{2^n} \times d_{2^n}$ .

Now notice here that the matrix  $\mathbf{M}$  is independent of  $i, T$ , or  $x$ . So we can pre-compute the matrix  $\mathbf{M}$  for the given polynomial  $p$  (which only takes time polynomial in  $n$  anyway), and then to compute the coefficients for  $p^{(T)}(x)$ , we simply compute it as  $(0, 1, 0, \dots) \cdot \mathbf{M}^T$ . And we can expedite the matrix powering part  $\mathbf{M}^T$  with repeated squaring, which requires only  $O(\log T)$  squaring of a square matrix with dimension  $d_{2^n} < 2n = O(n)$ . So the overall runtime is bounded by  $O(n^3 \log T)$  as desired.  $\square$

## 4.2 Discrete Log

Unfortunately, discrete logarithms can also be efficiently computed in the ring:

**Theorem 8.** *Suppose  $p$  is a full length cycle in  $R$ . Then there exists a randomized algorithm running in time polynomial in  $n$ , which computes  $i$  given  $x, y = p^{(i)}(x) \in \mathbb{Z}_{2^n}$  and the polynomial  $p$ .*

*Proof.* The algorithm works in three steps.

*Step 1: Generate Samples of  $(\alpha, p^{(i)}(\alpha))$ .* In the first step, we will show how to generate random  $\alpha$  together with  $p^{(i)}(\alpha)$ , without (yet) knowing  $i$  or  $p^{(i)}$ .

To do so, we choose several random  $j \in \mathbb{Z}_{2^n}$ , and compute  $\alpha_j = p^{(j)}(x)$ ,  $\beta_j = p^{(j)}(y)$ . Then since  $p$  is a full-length cycle polynomial, each  $\alpha_j$  will be uniformly random. Moreover, we have that:

$$\beta_j = p^{(j)}(p^{(i)}(x)) = p^{(i+j)}(x) = p^{(i)}(p^{(j)}(x)) = p^{(i)}(\alpha_j).$$

*Step 2: Interpolate  $p^{(i)}$ .* In the second step, we use the samples  $(\alpha, \beta)$  from Step 1 to interpolate  $p$ .

Concretely, we generate  $T = O(n^3)$  evaluation points  $(\alpha_j, \beta_j)$ . We write

$$p^{(i)}(x) = \sum_{i=0}^d g_i x^i$$

for unknowns  $g_i$ , where  $d = d_{2^n} - 1$  is an upper bound on the degree of polynomials in  $\tilde{R}$ . Each sample  $(\alpha, \beta)$  then gives us an equation

$$\beta = \sum_{i=0}^d g_i \alpha^i \pmod{2^n}$$

After collecting many samples, we obtain a system of linear equations on the  $g_i$ , which we then solve. Notice that, since the  $g_i$  are unconstrained, there are actually multiple  $g_i$  which correspond to the polynomials equivalent to  $p^{(i)}$ ; let this space of “correct”  $g_i$  be  $C$ . We just need to find some solution in  $C$ , and then we can reduce in  $\tilde{R}$ .

So it remains to show that, for a sufficient number of samples, with high probability the solution space of  $g_i$  is *exactly*  $C$ . This follows from Lemma 3. Indeed, consider a  $q \notin C$ . Since  $q \neq p^{(i)}$ , by Lemma 3,  $p(\alpha) \neq q(\alpha)$  for at least a  $1/n$  fraction of  $\alpha$ . Therefore, if we take  $T$  samples, the probability that  $q$  remains in the solution space is at most  $(1 - 1/n)^T$ . A union bound over all  $2^{O(n^2)}$  polynomials  $q$  shows that the solution space will be exactly  $C$ , except with probability

$$2^{O(n^2)}(1 - 1/n)^T = 2^{O(n^2)}e^{-O(T/n)} = 2^{-O(n^2)}$$

Thus, for an appropriate constant in the  $T = O(n^3)$  notation, the probability that  $C$  contains an incorrect solution is negligible.

*Step 3: Compute Discrete Logarithms In Groups of Smooth Order.* Once we know  $p^{(i)}$ , we can now compute the discrete logarithm of  $p^{(i)}$  relative to  $p$  in the group  $\tilde{R}^\times$ . Concretely, since the cyclic subgroup generated by  $p$  has smooth order  $2^n$ , we can solve discrete logarithms classically by Lemma 1.  $\square$

## 5 Applications/Open Problems

We conclude by proposing some speculative applications of polynomials mod  $2^n$  to cryptography. All of these applications rely on new untested computational assumptions. However, we find the constructions interesting, and hope that future work will lead to further confidence.

### 5.1 Non-abelian Cryptosystems

First, the group of units in the ring  $R$  is non-abelian. As such, it can potentially be used as a drop-in replacement for non-abelian groups used in various cryptographic protocols.

As one example, we can adapt Stickel's key agreement protocol [Sti05] to our group. The protocol works as follows:

- Alice and Bob publicly agree on polynomials  $p_1, p_2$  that are full-length cycles over  $\mathbb{Z}_{2^n}$  and which do not commute.
- Alice chooses a random  $a_1, a_2 \in \mathbb{Z}_{2^n}$ , and sends the polynomial  $A = p_1^{(a_1)} \circ p_2^{(a_2)}$ .
- Bob chooses a random  $b_1, b_2 \in \mathbb{Z}_{2^n}$ , and sends the polynomial  $B = p_1^{(b_1)} \circ p_2^{(b_2)}$ .
- The shared key is  $K = p_1^{(a_1+b_1)} \circ p_2^{(a_2+b_2)} = p_1^{(a_1)} \circ B \circ p_2^{(a_2)} = p_1^{(b_1)} \circ A \circ p_2^{(b_2)}$ .

The security of the protocol relies on the following assumption:

**Assumption 1.** There is no algorithm running in polynomial time in  $n$ , which given  $A = p_1^{(a_1)} \circ p_2^{(a_2)}$  and  $B = p_1^{(b_1)} \circ p_2^{(b_2)}$  for random  $a_1, a_2, b_1, b_2 \in \mathbb{Z}_{2^n}$ , outputs  $K = p_1^{(a_1+b_1)} \circ p_2^{(a_2+b_2)}$  with non-negligible probability.

Note that, even if discrete logarithms are easy (as in our group), this assumption may be plausible. In particular, the most obvious way to break the assumption is to compute  $a_1, a_2$  from  $A$  (or equivalently  $b_1, b_2$  from  $B$ ). Even with the ability to solve discrete logarithms such computations may be infeasible.

### 5.2 A Simple PRG

We now propose a simple PRG based on a very different conjectured hardness property.

Consider the collection of cyclic subgroups of order  $2^n$ . Consider the subset of subgroups which have as a generator a degree-2 polynomial (or more generally, degree  $d$  for any  $d \ll n$ ). We claim that this collection of degree-2-generated subgroups is sparse. Indeed, the set of cycles of length  $2^n$  has size  $2^{\Theta(n^2)}$ , while there are only  $2^{\Theta(nd)}$  degree-2 polynomials which generate cycles of length  $2^n$ .

We therefore propose the following computational assumption:

**Assumption 2.** Let  $U$  be the uniform distribution over full-length-cycle polynomials  $p$  in  $\tilde{S}$ . Let  $D$  be the distribution which selects a random degree- $d$  full-length-cycle polynomial  $p_0$ , and then chooses a random odd  $i \in \mathbb{Z}_{2^n}$ , and sets  $p = p_0^{(i)}$ . Then there is no polynomial time algorithm which distinguishes  $U$  from  $D$  with non-negligible probability.

Because discrete logarithms are easy, one can compute  $i$  from  $p$  and  $p_0$ . However, if  $p_0$  is not provided (as in the assumption above), then there is no obvious way to find  $i$  or  $p_0$ . The assumption states that not only is the mapping one-way, but that the result is pseudorandom amongst the full-length-cycle polynomials.

Note that the distribution  $D$  has very low entropy relative to  $U$ :  $O(nd)$  vs  $O(n^2)$ . As such, we get a very simple PRG  $G$ . The seed is an index  $i$  and a degree-2 full-length cycle polynomial  $p_0$ , and the output is  $p_0^{(i)}$ .

### 5.3 A Candidate Public Key Encryption Scheme

We now give a candidate public key encryption scheme that is very different from the one based on Stickel's scheme above.

- **Key generation:** Choose  $k \geq 3$  random full-length cycle polynomials  $p_1, \dots, p_k \in \tilde{S}^\times$ , whose composition is the identity:  $p_1 \circ \dots \circ p_k = x$ . Sample such polynomials by choosing random  $p_1, \dots, p_{k-1}$ , and setting  $p_k = (p_1 \circ \dots \circ p_{k-1})^{-1}$ . If  $p_k$  is not a full-length cycle polynomial, then discard everything and try again. Since full-length cycle polynomials represent a constant fraction of  $\tilde{S}^\times$ , only a constant number of iterations will be needed, in expectation.
- Additionally, choose random odd indices  $i_1, \dots, i_k \in \mathbb{Z}_{2^n}$ . The secret key is  $\{(p_j, i_j)\}_j$  and the public key is  $\{q_j = p_j^{(i_j)}\}_j$ .
- **Encryption:** To encrypt a message bit  $b$ , do the following.
  - If  $b = 0$ , choose a random unit  $r$ , and compute  $c_j = r \circ q_j \circ r^{-1}$ . Output  $\{c_j\}_j$ .
  - If  $b = 1$ , output  $k$  random full-length cycle polynomials.
- **Decryption:** Given  $\{c_j\}_j$ , compute

$$d = c_1^{(i_1^{-1})} \circ \dots \circ c_k^{(i_k^{-1})}$$

where  $i_j^{-1}$  is computed in  $\mathbb{Z}_{2^n}$ . If  $d$  is the identity polynomial, output 0, otherwise output 1.

If  $b = 0$ , then  $d = (r \circ p_1 \circ r^{-1}) \circ \dots \circ (r \circ p_k \circ r^{-1}) = r \circ (p_1 \circ \dots \circ p_k) \circ r^{-1} = x$ .

If  $b = 1$ , then  $d$  will essentially be a random polynomial, and is unlikely to be the identity. Hence the scheme is correct.

The scheme appears hard to crypt-analyze. Finding the secret key given the public key means trying to solve for the  $i_j$ 's, without knowing the base polynomials. For similar reasons to our PRG, this seems hard. One may also try to directly attack the scheme by trying to recover the encryption randomness

$r$ . The straightforward approach would be to consider the polynomials  $r, r^{-1}$  as polynomials with formal variables for coefficients, and try to solve for the coefficients. However, the equations in this system will have degree  $O(n)$  in the  $O(n)$  variables of  $r^{-1}$ . In general, such equations have super-polynomially-many monomials, so straightforward linearization techniques will be insufficient.

## 5.4 Homomorphic Trapdoor Functions

Let  $p$  be a polynomial. Suppose we encrypt a polynomial  $q$  by conjugation with  $p$ :  $\text{Enc}(p, q) = p \circ q \circ p^{-1}$ .

Conjugation commutes with polynomial composition, so this encryption scheme is homomorphic with respect to  $\circ$ .

If we want to obtain a trapdoor permutation, we need a way to encrypt  $q$  without explicitly giving  $p$ . There are a couple ways this could plausibly be done:

- Provide as the public key, a collection of pairs  $(q_i, r_i = p \circ q_i \circ p^{-1})$ . If one can write  $q$  as  $q_{i_1} \circ q_{i_2} \circ \dots$ , then the encryption of  $q$  is just  $r_{i_1} \circ r_{i_2} \circ \dots$ .

We note that the basis  $q_i$  cannot be too large, as the pairs  $(q_i, r_i)$  give polynomial equations on the coefficients of  $p$  and  $p^{-1}$ . If there are too many such equations, the coefficients can be solved by linearization techniques.

- If  $p$  has constant degree, say degree  $d = 2$ , here is another possible way. Write  $q = \sum_i \alpha_i x^i$  where the  $\alpha_i$  are formal variables. Then the coefficients of  $p \circ q \circ p^{-1}$  are degree- $d$  polynomials in the  $\alpha_i$ . The public key could contain this list of polynomials. Then to encrypt  $q$ , simply apply the polynomials to the coefficients.

The security of this last method is questionable, as there are only  $d+1$  degrees of freedom in the polynomial  $p$ , but the public key consists of a list of  $O(n)$  polynomials with  $O(n^d)$  monomials each. The public key therefore contains  $O(n^{d+1})$  coefficients derived from  $p$ . This massively over-determined system could potentially be solvable efficiently. However, we did not immediately see how to do so.

Note that  $\circ$  is non-abelian. It is known that a group homomorphic encryption scheme which is CPA secure can be compiled into a fully homomorphic encryption scheme [OS07]. Of course, our trapdoor functions are not CPA-secure, since in particular they are deterministic. However, an interesting direction for future research is to see if there is some way to obtain a CPA-secure variant of our construction, which can then be compiled to give fully homomorphic encryption.

## References

- [AD94] Leonard M. Adleman and Jonathan DeMarrais. A subexponential algorithm for discrete logarithms over all finite fields. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 147–158. Springer, Heidelberg, August 1994.

- [BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.
- [Cou06] Jean-Marc Couveignes. Hard homogeneous spaces. *Cryptology ePrint Archive*, Report 2006/291, 2006. <http://eprint.iacr.org/2006/291>.
- [OS07] Rafail Ostrovsky and William E. Skeith III. Algebraic lower bounds for computing on encrypted data. *Cryptology ePrint Archive*, Report 2007/064, 2007. <http://eprint.iacr.org/2007/064>.
- [Pie19] Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019.
- [Pó115] Georg Pólya. Über ganzwertige ganze funktionen. *Rendiconti del Circolo Matematico di Palermo (1884-1940)*, 40(1):1–16, 1915.
- [RS06] Alexander Rostovtsev and Anton Stolbunov. Public-Key Cryptosystem Based On Isogenies. *Cryptology ePrint Archive*, Report 2006/145, 2006. <http://eprint.iacr.org/2006/145>.
- [Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th FOCS*, pages 124–134. IEEE Computer Society Press, November 1994.
- [Shp08] Vladimir Shpilrain. Cryptanalysis of stickel’s key exchange scheme. In Edward A. Hirsch, Alexander A. Razborov, Alexei Semenov, and Anatol Slissenko, editors, *Computer Science – Theory and Applications*, pages 283–288, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [Sti05] E. Stickel. A new method for exchanging secret keys. In *Third International Conference on Information Technology and Applications (ICITA’05)*, volume 2, pages 426–430, 2005.
- [Wes19] Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.