Space-Lock Puzzles and Verifiable Space-Hard Functions from Root-Finding in Sparse Polynomials

Nico Döttling[1] [0000−0002−5914−7635], Jesko Dujmovic[1,2] [0000−0003−1043−5094], Antoine Joux[1] [0000−0003−2682−6508]

[1] CISPA Helmholtz Center for Information Security
jesko.dujmovic@cispa.de
[2] Saarland University

Abstract. Timed cryptography has initiated a paradigm shift in the design of cryptographic protocols: Using timed cryptography we can realize tasks *fairly*, which is provably out of range of standard cryptographic concepts. To a certain degree, the success of timed cryptography is rooted in the existence of efficient protocols based on the *sequential squaring assumption*.

In this work, we consider space analogues of timed cryptographic primitives, which we refer to as *space-hard* primitives. Roughly speaking, these notions require honest protocol parties to invest a certain amount of space and provide security against space constrained adversaries. While inefficient generic constructions of timed-primitives from strong assumptions such as indistinguishability obfuscation can be adapted to the space-hard setting, we currently lack concrete and versatile algebraically structured assumptions for space-hard cryptography.

In this work, we initiate the study of space-hard primitives from concrete algebraic assumptions relating to the problem of root-finding of sparse polynomials. Our motivation to study this problem is a candidate construction of VDFs by Boneh et al. (CRYPTO 2018) which are based on the hardness of inverting permutation polynomials. Somewhat anticlimactically, our first contribution is a full break of this candidate. However, we then revise this hardness assumption by dropping the permutation requirement and considering arbitrary sparse high degree polynomials. We argue that this type of assumption is much better suited for space-hardness rather than timed cryptography. We then proceed to construct both space-lock puzzles and verifiable space-hard functions from this assumption.

1 Introduction

Timed Cryptography. Traditionally, in public key cryptography [DH76], the ability to decrypt ciphertexts which have been generated with a public key pk is tied to the possession of a secret key sk corresponding to pk. Likewise, generation of signatures with respect to a verification key vk is tied to the possession of corresponding signing key. Timed cryptography [RSW96] adds a twist to this rigid paradigm: Rather than the possession of a secret key, *investing time* facilitates the decryption of a ciphertext or generation of signature. In other words, time-lock encryption allows to *encrypt to the future*.

This enables new applications both in theory and practice: Timed commitments [BN00] facilitate e.g. fair exchange and fair coin-toss in the two party setting, notions which have been shown to be beyond reach of standard cryptographic notions [Cle86]; Likewise, from a more practical angle time-lock puzzles play a crucial role in the design of public randomness beacons, a crucial component in the design of distributed ledgers (see e.g. [KWJ23]).

Verfiable Delay Functions Boneh et al. [BBBF18] introduced the notion of *verifiable delay functions* (VDFs), which can be loosely thought of as the timed analogue of digital signatures: Computing a VDF output together with a certificate of its validity takes a long time T, whereas verification of a certificate can be performed rapidly, that is in time $\mathsf{poly}(\lambda, \log(T))$. VDFs are likewise powerful tools in the construction of randomness beacons and consensus protocols, as they e.g. facilitate techniques such as *self-selection* [CM19] and proofs of replication [ABBK16].

Boneh et al. [BBBF18] provide both generic and concrete constructions of VDFs. The generic constructions are obtained by combining specific sequential functions, such as iterated hashing, with incrementally verifiable computation [Val08,BCCT13].

Alas, when it comes to concrete assumptions, VDFs in particular and timed cryptography in general rest on a rather narrow foundation; most candidates of time-lock puzzles and verifiable delay functions are tied to the sequential squaring assumption in groups of unknown order. Bitansky et al. [BGJ$^+$16] showed that by relying on indistinguishability obfuscation, timed primitives can be realized assuming the minimal assumption that inherently sequential problems exist. As this construction relies on very heavy theoretical tools, its appeal is currently limited to the domain of pure theory. Another significant candidate are verifiable delay functions from isogenies [DMPS19]. Proofs of sequential [MMV13,CP18,DLM19] can be seen as a more lightweight alternative to VDFs and are achievable from potentially weaker assumptions. However, PoSW lack a uniqueness property, which makes them unsuitable for many of the more advanced applications of VDFs.

The concrete VDF candidates given in [BBBF18] constitute a notable exception from the sequential squaring blueprint. These candidates are based on a novel family of hardness assumptions relating to the inversion of rational functions of high degree.

Space-Hard Cryptography. Conceptually, there is nothing intrinsically special about the computational resource of *time*. Hence, a natural conceptual next step is to consider more general computational resources. In fact, there is a growing body of works investigating the notion of *memory or space-hard functions* [Per09,AS15,AB16,ACP$^+$17,BP17,ABB22,AGP24].

In this work, we are concerned with both *space-lock puzzles*, the space-analogue of time-lock puzzles, and verifiable space-hard functions, the analogue of VDFs.

Syntactically, we define a space-lock puzzle to consist of two algorithm Gen and Solve. Gen takes as input a space parameter S and a message m and outputs a puzzle p, whereas Solve takes a puzzle p and outputs a message m. In terms of efficiency, we require that Gen runs in time and space $\mathsf{poly}(\lambda, \log(S))$, whereas Solve runs in space S. In terms of security, we require that any algorithm running in time $\mathsf{poly}(\lambda, S)$ having access to space of size at most $S^{1-\epsilon}$ has at most negligible advantage guessing an encrypted bit.

A verifiable space-hard function syntactically consists of two algorithms Eval and Verify (potentially along with a setup algorithm producing public parameters). Eval takes a space parameter S and a value x and outputs a value y and a certificate π, whereas Verify takes inputs x, y and a certificate π and outputs either accept or reject. In terms of efficiency, we require that Eval runs in space S, whereas Verify runs in time and space $\mathsf{poly}(\lambda, \log(S))$. In terms of security, we require *computational uniqueness* and *space-hardness*. Computational uniqueness requires that no algorithm running in time and space $\mathsf{poly}(\lambda, S)$ can produce a verifying tuple x, y', π' with $y' \neq y$, where $(y, \pi) = \mathsf{Eval}(S, x)$. Space-hardness requires that no algorithm running in time $\mathsf{poly}(\lambda, S)$ and space $S^{1-\epsilon}$ finds y with non-negligible probability.

In terms of assumptions and constructions, the design-space of space-hard cryptography is comparatively much less explored than that of timed cryptography. In terms of generic constructions, a closer look at the time-lock puzzle construction given in [BGJ+16] reveals that this construction can be adapted to space-lock puzzles, i.e. we can construct a space-lock puzzles assuming iO and, additionally, the minimal assumption that inherently space-hard computations exist.

However critically, there are currently no algebraically structured candidates for efficient space-lock puzzles.

1.1 Our Results

In this work, we take a first step in studying efficient space-hard primitives from algebraic assumptions relating to the solvability of sparse univariate polynomials of large degree. The contributions of our work are two-fold:

1. As a first contribution, we provide an efficient attack against the specific proposal of the "Inverting Injective Rational Maps" Assumption of Boneh et al. [BBBF18]. While we do not break the assumption in its most general form, we demonstrate a full break on their suggested candidate, which indicates that the assumption stands on brittle ground. A major challenge towards instantiating the general assumption is finding suitable families of injective rational maps, and [BBBF18] suggested a family of rational functions of (large) degree d constructed by [GM97]. In [BBBF18] it was conjectured that this family cannot be inverted in time and space $\mathsf{poly}(\log(d))$ (i.e. by

circuits of size $\mathsf{poly}(\log(d))$). We provide and implement an algorithm which inverts these rational functions in time and space $\mathsf{poly}(\log(d))$, thus falsifying the main candidate instantiation of the Inverting Injective Rational Maps assumption.

We remark that this VDF was a *weak* VDF to begin with, i.e. algorithms that run in time $\mathsf{poly}(\log(d))$ and space $O(d^c)$ for some $c \geq 1$ were known and discussed in [BBBF18]. The main innovation of this part of our work is that our attack runs in *both* time and space $\mathsf{poly}(\log(d))$.

2. In Section 4 we introduce and discuss a new algebraically structured space-hardness assumption which we refer to as the *sparse root finding* (SRF) assumption. Building on this, in Section 5 we provide a construction of space-lock puzzles from the SRF assumption, whereas in Section 6 we construct a verifiable space-hard function from the SRF assumption.

1.2 Our Techniques

Invertibility of Guralnick Müller Polynomials As mentioned above, Boneh et al. [BBBF18] provided a concrete candidate for a VDF based on Guralnick-Müller permutation polynomials [GM97]. These are defined via rational functions $f_{\mu,q}$ over a finite field $\mathbb{F}_{p^m}$ and parametrized by an element $\mu \in \mathbb{F}_{p^m}$ and a (large) degree parameter $q = p^r$ (for some $r < m$). Both μ and q need to obey some additional constraints to ensure that the function is a permutation. The function $f_{\mu,q}(X)$ is then given by

$$f_{\mu,q}(X) = \frac{(X^q - \mu X - \mu)(X^q - \mu X + \mu)^q + ((X^q - \mu X + \mu)^2 + 4\mu^2 X)^{(q+1)/2}}{2X^q}.$$

Notice that this function is neither linear nor affine, consequently at a first glance one may reasonably conjecture that it takes space proportional to q to invert it on random inputs. In fact, [BBBF18] provide a survey of cryptanalytic techniques to invert rational functions and argue why these techniques fail for the case of Guralnick-Müller polynomials. This includes inversion of *extremely* sparse polynomials, linear algebraic attacks, as well as attacks against so-called *exceptional polynomials*, which remain permutations when considered as rational functions over the extension field $\mathbb{F}_{p^{m'}}$ *for infinitely many choices of the degree* m'. Boneh et al. [BBBF18] conjecture that inverting a function $f_{\mu,q}$ for a randomly chosen $\mu \in \mathbb{F}_{p^m}$ (under some constraints) takes time polynomial in the degree parameter q.

Yet, our first contribution in this work is a full break of this assumption. Interestingly, we draw the mathematical tools for this attack from the original work of Guralnick and Müller [GM97]. We observe the following: While the function $f_{\mu,q}$ itself is not affine, the problem of inverting $f_{\mu,q}$ on a target $t \in \mathbb{F}_{p^m}$ can be *embedded into* a linear system of higher degree. Specifically, [GM97] provides us with the following property of $f_{\mu,q}$: If θ is $q-1$-st root of μ in the algebraic closure of $\mathbb{F}_{p^m}$, then there exist efficiently computable coefficients

A_0, B_0, B_1, B_2 (depending on μ and t) in an extension of $\mathbb{F}_{p^m}$ such that

$$\prod_{i \in \mathbb{F}_q} (f_{\mu,q}(X + i\theta) - t) = X^{q^3} + B_2 X^{q^2} + B_1 X^q + B_0 X + A_0. \tag{1}$$

Consequently, any solution $\xi \in \mathbb{F}_{p^m}$ of $f_{\mu,q}(\xi) = t$ is also a solution to the right-hand side of equation (1), i.e. such a solution satisfies

$$\xi^{q^3} + B_2 \xi^{q^2} + B_1 \xi^q + B_0 \xi + A_0 = 0. \tag{2}$$

Now observe that equation (2) is in fact a linear equation system (as exponentiation with q is a Frobenius action). Hence we can efficiently compute a solution space using standard linear algebra techniques. In Theorem 2 in Section 3 we show that, except with negligible probability over the choice of t, the system (2) possesses a unique solution, hence our attack succeeds. Furthermore, our experiments in MAGMA (see Section A) demonstrate that this attack is indeed practical.

Space-hardness from Inverting Sparse Polynomials Guralnick-Müller polynomials are one specific instance of the more general problem of finding roots of sparse polynomials [3], a problem which we will refer to as Sparse Root Finding (SRF).

As [BBBF18] note, their root-finding-based candidate achieves only a mild form of sequentiality to begin with. In fact, a moderate polynomial increase in parallel computation power will enable a solver to find roots significantly faster. On the other hand, however, the space-hardness of these problems seems to be much more robust, as all known algorithms for this type of problem consume a large amount of space, in fact an amount of memory that scales linearly with the degree of the polynomial.

This is the starting point for the constructive results in this work. In a nutshell, we consider the problem of inverting sparse, high degree polynomials but drop the requirement that the polynomial needs to act as a permutation. Hence, the resulting problem carries significantly less structure than e.g. inverting Guralnick-Müller polynomials and does not provide an obvious angle for cryptanalysis.

More importantly, by basing our constructions on the problem of root-finding for general sparse polynomials, we can achieve a win-win scenario:

- If our assumptions hold, we obtain practically efficient candidates for space-hard cryptography
- While we do not provide a worst-to-average case reduction, refuting our assumptions would constitute a considerable advance in the algorithmic state-of-the art of polynomial factorization algorithms, as it is a long open problem to design polynomial factorization algorithms which leverage sparsity (in a non-extreme parameter regime).

[3] More generally, we can consider this as finding roots of structured polynomials which possess a compact representation and can be evaluated quickly

Space-Lock Puzzle Building Space-Lock Puzzles from the assumption that SRF is space-hard in sparse high-degree polynomials is fairly straight forward. To generate a puzzle for a random message m, generate a random sparse polynomial $f(X)$ with high degree and a random constant coefficient. Now, we know that $f(X) - f(m)$ has a root at m and can be output as a space-lock puzzle for m. There are two minor problems with this construction. First, we might want to create a puzzle for a non-random message, which we can resolve by using hybrid encryption. Second, there might be polynomials $f(X)-f(m)$ with multiple roots. We can fix this problem by padding the message and checking for the correct padding after solving the puzzle.

Verifiable Space-Hard Functions. We start by discussing our construction of verifiable space-hard functions from SRF. As we let go of the permutation requirement of the polynomials, we need to work harder to make this function verifiable. Our technical tool to achieve this is a novel and efficient special-purpose proof system for certifying the greatest common divisor between the polynomial $f(X)$ and $X^p - X$. This is sufficient, as given this gcd one can quickly and space-efficiently find the roots of $f(X)$.

For the purpose of this outline, assume that we have cheap way to prove equations over high-degree and possibly dense polynomials. We will later explain how to carry out these checks. We make use of the fact that the greatest common divisor between a polynomial $f(X)$ and $X^p - X$ is constant degree with high probability over the choice of a random sparse polynomial $f(X)$. Note that this gcd allows us to immediately compute the roots of $f(X)$ in $\mathbb{F}_p$. Our proof system establishes that some constant degree polynomial $g(X)$ is the gcd of $f(X)$ and $X^p - X$ in two phases. In the first phase, the prover computes $f'(X) = X^p - X \mod f(X)$ via square and multiply. Each step of this computation is defined by a simple polynomial equation $(X^{2^n} \mod f(X)) \cdot ((X^{2^n} \mod f(X)) = (X^{2^{n+1}} \mod f(X)) + h(X)f(X)$ for some polynomial $h(X)$.

In the second phase, we use that $gcd(X^p - X, f(X)) = gcd(f'(X), f(X))$ and compute $g(X) = gcd(f'(X), f(X))$ together with its Bézout coefficients $a(X), b(X)$ via the extended Euclidean algorithm. The greatest common divisor is unique, up to normalization, hence we require the prover to normalize this polynomial. The verifier can easily check this property by checking that the leading coefficient is 1. Bézouts identity guarantees that for all $\bar{a}(X)$, $\bar{b}(X)$ we have $\bar{a}(X)f'(X) + \bar{b}(X)f(X)$ is a multiple of $gcd(f'(X), f(X))$. Further, the verifier can check whether $g(X) = a(X)f'(X)+b(X)f(X)$ is a divisor, by making sure that $f(X) \mod g(X) = 0$ and $X^p - X \mod g(X) = 0$. Now we have verified that $g(X)$ is a divisor of $f(X)$ and $f'(X)$ and that $g(X)$ is a multiple of the their greatest common divisor, therefore, it is a greatest common divisor.

So far we have skipped over the issue of how we can verify polynomial equations, when the polynomials have representations that are bigger than the verifier's space. Polynomial commitments such as [KZG10] would be the perfect tool for this but its common reference string scales with the degree of the polynomials, which we want to avoid. Instead we commit to evaluation of these polynomials at specific points, which in fact defines a Reed-Solomon code. We then use interactive oracle proofs of proximity to establish that the commitments are indeed close to a codeword of the corresponding Reed-Solomon code. To check the equations we pick a few random positions of the codeword and check whether the equations hold at these positions. We can then use the the Schwartz-Zippel lemma to argue soundness of the polynomial equations.

1.3 Open Problems

We consider it to be an interesting question to investigate whether it is possible to intrinsically and flexibly tie the resources of space and time in a puzzle or verifiable time-space hard function. That is, is it possible to force the puzzle solver to spend S space for T time? Here S and T are adjustable parameters. This concept may be most closely captured by the concept of sustained space complexity [ABP18].

We believe any solution to this problem that goes beyond taking a sequential function that has a scalable domain and generically applying incrementally verifiable computation to it might be of big interest. An example for such a function is sequential squaring over a modulus that scales with the space parameter. More specifically the function could be on input $x \in [2^\lambda]$ compute $h((N^s - x)^{2^T} \bmod N^s)$ where h is some compressing function to reduce the size of the result.

1.4 Related Work

There are many works in memory restricted cryptography such as memory-hard functions and various forms of proof of space. Memory-hard functions are functions that are only computable with a large amount of memory accesses. The measure that many works use is called cumulative memory complexity. These functions are used to reduce the effectiveness of building application specific integrated circuits (ASICs) or field programmable gate arrays (FPGA) for brute force attacks because these excel at computation and do not have a faster way of accessing memory than off-the-shelf CPUs. Memory-hard functions are used in password hashing, proof of work, and other applications where the goal is to make computation expensive. The first memory-hard function was proposed by Percival in 2009 [Per09]. So far, all memory hard functions [Per09,AS15,AB16,ACP$^+$17] use graphs with special structure and iteration of a function to force anyone trying to evaluate the function to do a lot of memory accesses.

Our functions are hard in a slightly different way. We are trying to increase the amount of maximum storage that is necessary to compute the function, which we call space hardness. A space-lock puzzle closely resemble trapdoor memory-hard function [AGP24], asymmetrically memory-hard functions [BP17] and memory-hard puzzles [ABB22], but under the notion of memory hardness.

We introduce space-hard functions and their verifiable counter part. Verifiable space-hard functions are a space-analogue of verifiable delay functions. We heavily deviate from the design space of memory-hard functions as most constructions are based on the random oracle model. Therefore, using incrementally verifiable computation to verify their evaluation requires proving statements over random oracle, which is concretely inefficient and conceptually unsatisfying. Indeed, [ABFG14] show how to verify that the function used much memory, but not that the output is correct. [DFKP15] further extend the notion to proof of space, where the prover executes a memory-hard function and then regularly gets queried to prove that he maintains a large amount of the computation in his memory. For an excellent overview on the topic, we refer to [RD16], as they detail the different notions and their relations.

Our verifiable space-hard function follows a similar design principle as the weak verifiable-delay function suggested by [BBBF18]. They suggest that inverting a fast to evaluate high degree permutation polynomial requires a lot of sequential computation. We instead conjecture and use the space-hardness of the same computation. Because we, however, want to move away from permutation polynomials to less structured polynomials our constructions require a special purpose proof system.

Indeed, verifiable space-hard functions can be though of as a space-analogue of verifiable delay functions [LW17,BBBF18,Wes19,Pie19,HHK$^+$22,HHKK23] and space-lock puzzles as a space-analogue of time-lock puzzles [RSW96].

1.5 Acknowledgments

Nico Döttling and Jesko Dujmovic: Funded by the European Union (ERC, LACONIC, 101041207). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

2 Preliminaries

2.1 Notation

We use the Landau notation to describe the asymptotic behavior of functions. We write $f(x) = O(g(x))$ if there exists a constant c such that $|f(x)| \leq c|g(x)|$ for all x larger than some constant x_0. Further we write $f(x) = o(g(x))$ if $\lim_{x\to\infty} f(x)/g(x) = 0$. We use the notation $[n]$ to denote the set $\{1, \ldots, n\}$. We use the notation poly to denote a polynomial. With negl we denote a negligible function, which is a function that is asymptotically smaller than the inverse of any polynomial. We use the notation λ to denote the security parameter.

2.2 Finite fields

For a prime-power $q = p^k$ we denote the finite field of size q by $\mathbb{F}_q$. We call p the characteristic of $\mathbb{F}_q$. Recall a few basic facts about finite fields. For a field $\mathbb{F}_q$ of characteristic the polynomial functions $x \mapsto x^{p^i}$ are called *Frobenius automorphisms* and it holds that $x^q = x^{p^k} = x$ for all $x \in \mathbb{F}_q$. Hence, the q roots of the polynomial $X^q - X$ are precisely all the elements in $\mathbb{F}_q$. Consequently, if x is in an extension field of $\mathbb{F}_q$ and $x^q - x = 0$, then it must hold that $x \in \mathbb{F}_q$. Likewise, the $q-1$ roots of the polynomial $X^{q-1} - 1$ are exactly all non-zero elements in $\mathbb{F}_q$.

2.3 Polynomials

We call the variable for polynomials X and a polynomial is usually denoted like this $f(X)$ or in explicit form, e.g. $X + X^2$. We call a value x a root of a polynomial $f(X)$ if $f(x) = 0$. The degree of a polynomial $f(X)$ is the highest power of X that appears in $f(X)$. A polynomial is called monic if the coefficient of the highest power of X is 1.

Recall that a univariate polynomial $f(X)$ is square-free if and only if it holds that $\gcd(f(X), f'(X)) = 1$, where $f'(X)$ is the formal derivative of $f(X)$ in X.

Lemma 1 (Polynomial Identity Lemma). *[Sch80,Zip79,DL78] Let $f(X)$ be a polynomial of degree d over a field $\mathbb{F}_q$. Let $S \subseteq \mathbb{F}_q$ be a set of size s. Then for a random $x \leftarrow_\$ S$ we have $f(x) = 0$ with probability at most d/s.*

Lemma 2 (Bézout's Identity for Polynomials). *Let $f(X)$ and $g(X)$ of degree d_1 and d_2 be two polynomials with greatest common divisor $d(X)$. Then there exist polynomials $a(X)$ and $b(X)$ such that $a(X)f(X)+b(X)g(X) = d(X)$ and $\deg(a(X)) < d_2$ and $\deg(b(X)) < d_1$. Moreover, for all $\bar{a}(X)$, $\bar{b}(X)$ then polynomials of the form $\bar{a}(X)f(X)+\bar{b}(X)g(X)$ are exactly the multiples of $d(X)$.*

Lemma 3 (Vandermonde Matrix Invertible). *Over any field $\mathbb{F}$ the Vandermonde matrix*

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{pmatrix}$$

is invertible if the x_i are distinct.

2.4 Codes

A code C is a set of codewords. A code is called linear if it is a vector space over a finite field $\mathbb{F}_q$. The distance of a code is the minimum Hamming distance between any two distinct codewords. The distance of a code is denoted by δ. A code is called an $[n, k, \delta]$ code if it has length n, dimension k, distance δ, and relative distance δ/n. A Reed-Solomon code is a linear code that is defined by evaluating a polynomial at a set of points. The code is defined by a polynomial $f(X)$ of degree $k-1$ and a set of points $x_1, \ldots, x_n$. The codewords are then the evaluations of $f(X)$ at the points $x_1, \ldots, x_n$. The distance of a Reed-Solomon code is $n - k + 1$. We call the Reed-Solomon code evaluated at a set of points $\mathcal{L}$ with polynomials of degree d $\mathsf{RS}[\mathcal{L}, d]$. The field this code is over will be clear from the context.

2.5 Combinatorics

H is family of strongly universal$_t$ hash functions [WC81] that map from X to Y if for any distict $x_1, \ldots, x_t \in X$, and any possibly non-distinct $y_1, \ldots, y_t \in Y$, we have that

$$\mathsf{Pr}_{h \leftarrow_\$ H}[h(x_1) = y_1, \ldots, h(x_t) = y_t] = |Y|^{-t}.$$

We detail the first two Bonferroni inequalities.

Lemma 4 (Bonferroni Inequalities). *[Bon36] Let $A_1, \ldots, A_n$ be events. We have $\sum_{i \in [n]} \mathsf{Pr}[A_i] - \sum_{i \in [n], j \in [i-1]} \mathsf{Pr}[A_i \cap A_j] \leq \mathsf{Pr}[\cup_{i \in [n]} A_i] \leq \sum_{i \in [n]} \mathsf{Pr}[A_i]$.*

2.6 IOP

An interactive oracle proof (IOP) [BCS16] combines the powers of an interactive proof and a probabilistically checkable proof. It is a proof system where a prover and a verifier interactively exchange messages. The prover sends long strings that the verifier does not have to read entirely. The verifier only sends random challenges to the prover and checks the consistency of the responses. Formally:

Definition 1 (IOP). *An IOP with soundness error β for a language L with witness relation R is defined by a tuple of algorithms $\mathsf{IOP} = (\mathsf{P}, \mathsf{V})$ defined with the following two properties:*

Completeness *For all $(x, w) \in R$:*

$$\mathsf{Pr}\left[\mathsf{V}^{\pi_1, \ldots, \pi_t}(x, \rho_1, \ldots, \rho_t) = 1 \middle| \begin{array}{c} \textit{Sample } \rho_1, \ldots, \rho_t \textit{ uniformly at random} \\ \pi_1 \leftarrow \mathsf{P}(x, w) \\ \pi_2 \leftarrow \mathsf{P}(x, w, \rho_1) \\ \vdots \\ \pi_t \leftarrow \mathsf{P}(x, w, \rho_1, \ldots, \rho_{t-1}) \end{array}\right]$$

is 1.

Soundness *For all $x \notin L$ and all unbounded provers P^*:*

$$\mathsf{Pr}\left[\mathsf{V}^{\pi_1, \ldots, \pi_t}(x, \rho_1, \ldots, \rho_t) = 1 \middle| \begin{array}{c} \textit{Sample } \rho_1, \ldots, \rho_t \textit{ uniformly at random} \\ \pi_1 \leftarrow \mathsf{P}^*(x) \\ \pi_2 \leftarrow \mathsf{P}^*(x, \rho_1) \\ \vdots \\ \pi_t \leftarrow \mathsf{P}^*(x, w, \rho_1, \ldots, \rho_{t-1}) \end{array}\right]$$

is $\leq \beta$

IOPP An interactive oracle proof of proximity (IOPP) is an IOP related notion where the prover is given a word of some code to which the verifier has query access. The prover then wants to convince the verifier that the word is close to a codeword of some code C. Formally:

Definition 2 (IOPP). *An IOPP for a code C with distance δ and soundness error β is defined by a tuple of algorithms* $\mathsf{IOPP} = (\mathsf{P}, \mathsf{V})$ *defined with the following two properties:*

Completeness *For all $c \in C$:*

$$\Pr\left[\mathsf{V}^{c,\pi_1,\dots,\pi_t}(w,\rho_1,\dots,\rho_t) = 1 \;\middle|\; \begin{array}{c} \textit{Sample } \rho_1,\dots,\rho_t \textit{ uniformly at random} \\ \pi_1 \leftarrow \mathsf{P}(c) \\ \pi_2 \leftarrow \mathsf{P}(c,\rho_1) \\ \vdots \\ \pi_t \leftarrow \mathsf{P}(c,\rho_1,\dots,\rho_{t-1}) \end{array}\right]$$

is 1.

Soundness *For all c is at distance $\geq \delta$ from C and all unbounded provers P^*:*

$$\Pr\left[\mathsf{V}^{c,\pi_1,\dots,\pi_t}(w,\rho_1,\dots,\rho_t) = 1 \;\middle|\; \begin{array}{c} \textit{Sample } \rho_1,\dots,\rho_t \textit{ uniformly at random} \\ \pi_1 \leftarrow \mathsf{P}^*(w) \\ \pi_2 \leftarrow \mathsf{P}^*(w,\rho_1) \\ \vdots \\ \pi_t \leftarrow \mathsf{P}^*(w,\rho_1,\dots,\rho_{t-1}) \end{array}\right]$$

is $\leq \beta$.

There exist an IOPPs for Reed-Solomon codes [BBHR18,ACY23,ACFY24].

Combiner Beyond IOPPs, we actually use the batch IOPPs as defined in Arnon et al. [ACFY24]. Specifically, we use a procedure $\mathsf{Combine}$ with the following properties. It takes as input a natural number d^*, some random field element r, m functions $f_1,\dots,f_m$ and corresponding degrees $d_1,\dots,d_m$. If f_i is a univariate polynomial of degree d_i and $d_i \leq d^*$ for all $i \in [m]$ then $\mathsf{Combine}(d^*, r, (f_i, d_i)_{i\in[m]})$ is a degree d^* univariate polynomial. Further if for some distance parameter δ, rate ρ, and error err we have if

$$\Pr_{r \leftarrow_\$ \mathbb{F}}[\Delta(\mathsf{Combine}(d^*, r, (f_i)_{i\in[m]}), \mathsf{RS}[\mathcal{L}, d^*]) \leq \delta] > \mathsf{err}$$

then there exists $S \subset \mathcal{L}$ with $|S| \geq (1-\rho)\cdot|\mathcal{L}|$, and for all $i \in [m]$ exists a $u \in \mathsf{RS}[\mathcal{L}, d_i]$ with $f_i(S) = u(S)$. For guidance on how to choose the parameters we refer to [ACFY24].

3 Inverting the Guralnick–Müller permutation polynomial

In 1997, Guralnick and Müller [GM97] introduced a family of permutation polynomials that were later proposed as a candidate for building a verifiable delay function [BBBF18]. In this proposal it is crucial that inverting the polynomial is not easy. Guralnick–Müller polynomials do not seem as easy to invert as permutation polynomials from other families.

In this Section, we show – using the properties established by Guralnick and Müller – that, on the contrary, they can be easily inverted. As a consequence, they should not be used as a building block for verifiable delay functions.

We start by recalling the definition of the Guralnick–Müller polynomials and some of their relevant properties. We use the original notations of [GM97] rather the notations from [BBBF18].

3.1 Notations and known facts

Let K be a finite field of characteristic p. A polynomial in $K[X]$ is called exceptional if it acts as a permutation on infinitely many finite extensions of K.

Theorem 1 ((Part of) Theorem 1.4 of [GM97]). *Let K be a finite field of characteristic p. Let q be a power of p. Given $\mu \in K$, set:*

$$a_\mu(X) = X^{2q} - 2\,\mu X^{q+1} + 2\,\mu X^q + \mu^2 X^2 + 2\,\mu^2 X + \mu^2.$$

Define:

$$f_\mu(X) = \frac{a_\mu(X)^{(q+1)/2} + (X^q - \mu X + \mu)^q (X^q - \mu X - \mu)}{2\,X^q}.$$

Then f_μ is exceptional over K provided that μ is not a $(q-1)$-st power in $K\,\mathbb{F}_q$.

Remark 1. At first glance, $f_\mu(X)$ is defined as a rational fraction. However, its numerator is divisible by X^q, so $f_\mu(X)$ is a polynomial of degree q^2.

We use the following property of f_μ.

Proposition 1 (Proposition 3.4 of [GM97]). *Let t be a variable and set $Y = X^q - \mu X$ and $\delta = (t^2 - 4\,\mu^{2q+1})^{(q-1)/2}$. Let θ denote a $(q-1)$-st root of μ in $\bar{K}$. Then*

$$\prod_{i \in \mathbb{F}_q} (f(X + i\theta) - t) = H(Y),$$

where $H(Y) = Y^{q^2} + A_2 Y^q + A_1 Y + A_0$. The coefficients $A_0, A_1, A_2 \in K$ are given by

$$\begin{aligned} A_2 &= \frac{t^q - \delta t}{2\,\mu^q}, \\ A_1 &= -\mu^q \delta, \\ A_0 &= -\frac{t^q + (t - 2\,\mu^{q+1})\delta}{2} + \mu^{q^2}. \end{aligned}$$

3.2 Inversion in the context of VDFs

In the setting of verifiable delay functions, the authors of [BBBF18] consider the use of the Guralnick–Müller on a fixed field where it acts as a permutation. More precisely, they take $q = p^r$ and consider the family f_μ on $\mathbb{F}_{p^n}$. Without loss of generality we may assume that r and n are co-prime. If they are not let g denote their greatest common divisor. Then both q and p^n are powers of p^g, with coprime exponents. In the sequel, we change r and n and consider the following case:

$$q_0 = p^g, \quad q = q_0^r, \quad \text{and } Q = q_0^n.$$

Assume now that we want to solve the equation $f_\mu(x) = t$ for some given target t in $\mathbb{F}_Q$ and some solution x also in $\mathbb{F}_Q$. As a direct consequence of Proposition 1, x has to be a root of:

$$H(X^q - \mu X) = X^{q^3} + B_2\, X^{q^2} + B_1\, X^q + B_0\, X + A_0, \tag{3}$$

with:

$$B_2 = A_2 - \mu^{q^2}, \quad B_1 = A_1 - \mu^q A_2, \quad \text{and } B_0 = -\mu A_1.$$

Since Equation (3) is affine, it is easy to solve using standard techniques. More precisely, we use the action of Frobenius to amplify this equation in an affine system of n equations in n variables. First, we define each variable X_i as $X^{q_0^i}$. Since we look for a solution in $\mathbb{F}_Q$, we have the constraint $X_n = X_0$ and similarly $X_i = X_{i \bmod n}$ for any i. Let d be an integer modulo n, by raising Equation (3) to the power q_0^d we obtain (by linearity of Frobenius) that:

$$X_{d+3r} + B_2^{q_0^d}\, X_{d+2r} + B_1^{q_0^d}\, X_{d+r} + B_0^{q_0^d}\, X_d = -A_0^{q_0^d}.$$

Taken these n equations together, we obtain a linear system and can recover the desired value x as the value of X_0 in the solution.

3.3 Efficiency of the Algorithm

The hardness assumption in [BBBF18] assumes that inverting the Guralnick–Müller polynomial costs at least q^2 operations in $\mathbb{F}_Q$. Our algorithm falsifies this assumption. Indeed, its most time consuming part is the resolution of the linear system. Using Gaussian elimination, it can be achieved in n^3 arithmetic operation over $\mathbb{F}_Q$. Thus, its bit complexity is $\tilde{O}(n^4 \log q_0)$.

Note that a more careful implementation can instead solve a linear system of the same dimension over $\mathbb{F}_{q_0}$, thus reducing the bit complexity to $\tilde{O}(n^3 \log q_0)$.

It would be tempting to consider the use of faster linear algebra in this algorithm. However, this is not really useful, since there exists a faster algorithm that uses only $O(n)$ arithmetic operations in $\mathbb{F}_Q$ to invert the Guralnick–Müller polynomial.

3.4 Correctness of the Algorithm

In order to establish the correctness of our algorithm, we will need the following additional lemma which describes the result of the $\mathbb{F}_Q$-Frobenius action on θ, which is a $q-1$-st root of μ.

Lemma 5. *Let $\mu \in K$ be as in Theorem 1 and let θ be a $q-1$-st root of μ. Let $|\mathbb{F}_Q| = |\mathbb{F}_q \cdot K| = q^n$. Then it holds that*

$$\theta^{q^n} = \mu^{\frac{q^n-1}{q-1}}\theta.$$

Furthermore, given that $\theta \notin \mathbb{F}_q \cdot K$, it holds that $1 \neq \mu^{\frac{q^n-1}{q-1}} \in \mathbb{F}_q$.

Proof. First recall that $\frac{q^n-1}{q-1} = \sum_{i=0}^{n-1} q^i$ is indeed an integer. Furthermore, since θ is a $q-1$-st root of μ it holds that

$$\theta^q = \theta^{q-1} \cdot \theta = \mu \cdot \theta.$$

Hence, by iterating this identity we obtain via induction that

$$\begin{aligned}
\theta^{q^n} &= (\theta^q)^{q^{n-1}} \\
&= (\mu \cdot \theta)^{q^{n-1}} \\
&= \mu^{q^{n-1}} \cdot \theta^{q^{n-1}} \\
&= \mu^{q^{n-1}} \cdot \mu^{\frac{q^{n-1}-1}{q-1}} \cdot \theta \\
&= \mu^{q^{n-1}} \cdot \mu^{\sum_{i=0}^{n-2} q^i} \cdot \theta \\
&= \mu^{\sum_{i=0}^{n-1} q^i} \cdot \theta \\
&= \mu^{\frac{q^n-1}{q-1}} \cdot \theta,
\end{aligned}$$

where the inductive step happens from the third to the fourth equality. For the second part of the statement, observe that as $\mu \in K \subseteq \mathbb{F}_q \cdot K$ it holds that

$$(\mu^{\frac{q^n-1}{q-1}})^{q-1} = \mu^{q^n-1} = 1,$$

i.e. $\mu^{\frac{q^n-1}{q-1}} \in \mathbb{F}_q$. Finally note that $\mu^{\frac{q^n-1}{q-1}} = 1$ would imply that

$$\theta^{q^n} - \theta = 0,$$

i.e. $\theta \in \mathbb{F}_q \cdot K$. But this immediately contradicts the assumption that $\theta \notin \mathbb{F}_q \cdot K$.

The following theorem establishes that our algorithm always returns the correct solution given that the coefficient $A_1 = -\mu^q\delta$ is non-zero. Note that $\delta = (t^2 - 4\,\mu^{2q+1})^{(q-1)/2}$ is non-zero whenever $t \neq \pm 2\,(\sqrt{\mu})^{2q+1}$. If $\sqrt{\mu} \notin K$ this event never happens, otherwise if $\sqrt{\mu} \in K$ the event that $t = \pm 2\,(\sqrt{\mu})^{2q+1}$ happens with negligible probability $2/|K|$, given that t is distributed uniformly on K. Note further that in the [BBBF18] scheme the input t is chosen uniformly random.

Theorem 2. *Let $H(Y) = Y^{q^2} + A_2Y^q + A_1Y + A_0$ as in Proposition 1. Given that the coefficient A_1 of $H(Y)$ is non-zero, the equation $H(X^q - \mu X) = 0$ has a unique solution in K.*

Proof. First off, note that since f is permutation on K, by Proposition 1 the system $H(X^q - \mu X) = 0$ has at least one solution. Observe first that given that $A_1 \neq 0$, the polynomial $H(X^q - \mu X)$ is square free, i.e. over its algebraic closure it splits into distinct linear factors. This holds as its formal derivative is $-\mu A_1$, which is non-zero given that $A_1 \neq 0$.

Assume towards contradiction that $H(X^q - \mu X) = 0$ has two distinct solutions $x \neq x'$ in K. Thus, by Proposition 1 there exist $i, i' \in \mathbb{F}_q$ such that $x + i\theta$ and $x' + i'\theta$ are both roots of $f(X) - t$, i.e.

$$\begin{aligned} f(x + i\theta) - t &= 0 \\ f(x' + i'\theta) - t &= 0. \end{aligned}$$

Observe that not both i and i' can be 0, as otherwise x and x' would be two distinct roots of $f(X) - t$ in K, which contradicts the permutation property of f. Without loss of generality, assume that $i \neq 0$ and set $z = x + i\theta$. We claim that $x + i\mu^{\frac{q^n-1}{q-1}}\theta$ is also a root of $f(X) - t$, as

$$\begin{aligned} 0 &= (f(z) - t)^{q^n} \\ &= f((x + i\theta)^{q^n}) - t \\ &= f(x + i\theta^{q^n}) - t \\ &= f(x + i\mu^{\frac{q^n-1}{q-1}}\theta) - t. \end{aligned}$$

Here the second equality follows as $f(X) - t$ is in $K[X]$, the third equality follows as $x, i \in \mathbb{F}_q \cdot K$, and the last equality follows by the first item of Lemma 5.

Now set $i^* = i \cdot (\mu^{\frac{q^n-1}{q-1}} - 1)$ which is a non-zero element of $\mathbb{F}_q$ by Lemma 5. We claim that z is also a root of $f(X + i^*\theta)$, as

$$f(z + i^*\theta) = f(x + i\theta + i \cdot (\mu^{\frac{q^n-1}{q-1}} - 1)\theta) = f(x + i\mu^{\frac{q^n-1}{q-1}}\theta) = 0.$$

Consequently, z is a zero of both $f(X) - t$ and $f(X + i^*\theta) - t$, and therefore $X - z$ divides both $f(X) - t$ and $f(X + i^*\theta) - t$, and therefore, as $i^* \neq 0$ it holds that $(X - z)^2$ divides $\prod_{j \in \mathbb{F}_q}(f(X + j\theta) - t) = H(X^q - \mu X)$ (by Proposition 1). But this contradicts the square-freeness of $H(X^q - \mu X)$, which concludes the proof.

3.5 Implementation

We provide Magma code for the attack in Appendix A.

4 Space-Hardness of Root-Finding

We conjecture that root-finding for polynomials over a big finite field requires a lot of space. As far as we are aware of, all root finding algorithms [Ber70,CZ81] [VZGS92,Sho93,KS95,KU11,GNU16] in a finite field $\mathbb{F}_p$ for large p and comparatively smaller degree d, start by computing $X^p - X \mod f(X)$. For a discussion of recent results, see [GNU16]. In general, this polynomial $X^p - X \mod f(X)$ is a dense polynomial of degree $d-1$, whose representation requires d elements in $\mathbb{F}_p$. For this reason, we conjecture a minimal space of d for any algorithm with a runtime $o(p)$. Proving this conjecture wrong would greatly advance the state of the art concerning polynomial factorization.

Assumption 1 (Sparse Root-Finding (SRF)) *We define the space-hardness of finding a root in a polynomial from distribution $\mathcal{D}_{\lambda,S}$ as follows: Root-Finding is hard with a gap $\varepsilon < 1$ if there exists a polynomial $\tilde{S}(\cdot)$ such that for all polynomials $S(\cdot) \geq \tilde{S}(\cdot)$ and PPT adversaries $\{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ with space bound $S^\varepsilon(\lambda)$ there exists a negligible function* negl *such that for all $\lambda \in \mathbb{N}$:*

$$\mathsf{Pr}\left[f(x^*) = 0 \middle| \begin{array}{l} f(X) \leftarrow \mathcal{D}_{\lambda,S(\lambda)} \\ x^* \leftarrow \mathcal{A}_\lambda(f(X)) \end{array}\right] = \mathsf{negl}(\lambda)$$

Remark 2. We decided to give the adversary $S^\varepsilon(\lambda)$ space to resemble the guarantees given in time-lock cryptography. But we conjecture the problem to be hard if the adversary has $\varepsilon \cdot S(\lambda)$ space.

We also define a space-hardness assumption for the problem of computing the greatest common divisor of a polynomial and $X^p - X$.

Assumption 2 (Sparse GCD Computation) *We define the space-hardness of gcd computation from distribution $\mathcal{D}_{\lambda,S}$ as follows: gcd computation is hard with a gap $\varepsilon < 1$ if there exists a polynomial $\tilde{S}(\cdot)$ such that for all polynomials $S(\cdot) \geq \tilde{S}(\cdot)$ and PPT adversaries $\{\mathcal{A}_\lambda\}_{\lambda\in\mathbb{N}}$ with space bound $S^\varepsilon(\lambda)$ there exists a negligible function* negl *such that for all $\lambda \in \mathbb{N}$:*

$$\mathsf{Pr}\left[g(X) = gcd(f(X), X^p - X) \middle| \begin{array}{c} f(X) \leftarrow \mathcal{D}_{\lambda,S(\lambda)} \\ g(X) \leftarrow \mathcal{A}_\lambda(f(X)) \\ \deg(g) > 0 \end{array}\right] = \mathsf{negl}(\lambda)$$

Lemma 6. *If root-finding is hard with gap $\varepsilon < 1$ then gcd computation is hard with gap ε.*

Proof. Given an adversary $\mathcal{A}$ that breaks Assumption 2 we construct an adversary $\mathcal{A}'$ that breaks Assumption 1.

$\mathcal{A}'(f(X))$:

- Let $g(X) \leftarrow \mathcal{A}(f(X))$ where $g(X)$ is an n-degree polynomial.
- Factor $g(X)$ into degree 1 polynomials $X - h_1, \ldots, X - h_n$ using the Cantor-Zassenhaus algorithm.

– Return h_1.

The factors of $g(X)$ can only be of degree ≥ 1 because $X^p - X$ only factors of degree 1. h_1 exists because $\deg(g) > 0$. The Cantor-Zassenhaus [CZ81,Sho93] algorithm has space-complexity $O(n \log p)$ and runs in $\mathsf{poly}(n, \log p)$.

We require these assumptions for two different but related applications, which we will detail in later chapters. One is a space lock puzzle and the other a verifiable space-hard function. For the space lock puzzle we only really require that the polynomials by the distributions are fast to evaluate and that root-finding is space-hard.

In general, we will stick with prime order fields because they tend to have less structure, which might protect them against structural attacks. We will also try to impose as little structure as possible on the polynomials. In order to make proofs over the these fields better we choose FFT-friendly fields.

A natural candidate is the distribution of random sparse polynomials. We make sure that the lowest two monomials are random for better estimation of number of roots. More formally, we define the distribution $\mathcal{D}_{\lambda,S}$ as follows:

Definition 3 (Random Sparse Polynomial (with Uniform Constant and Linear Coefficient)). *Pick a prime $p \in \Omega(2^\lambda)$, degree $d \in \Omega(S)$, and number of non-zero monomials $k \in \Omega(\lambda)$. Operations happen over $\mathbb{F}_p$. Output the following univariate (the variable is X) polynomial*

$$a_0 + a_1 X + \sum_{i \in [k-2]} a_i X^{e_i} + X^d$$

For uniformly random $a_i \leftarrow_\$ \mathbb{F}_q$ and $e_i \leftarrow_\$ [d-1]$.

We prove these polynomials define a family of strongly universal$_2$ hash functions [WC81].

Lemma 7. *For any polynomial $h(X)$, any distinct $x_1, \ldots, x_t \in \mathbb{F}_p$, and any possibly non-distinct $y_1, \ldots, y_t \in \mathbb{F}_p$, we have that*

$$\mathsf{Pr}_{a_0,\ldots,a_{t-1} \leftarrow_\$ \mathbb{F}_p}[h(x_1) + l(x_1) = y_1, \ldots, h(x_t) + l(x_t) = y_t] = p^{-t}$$

where $l(X) = \sum_{i \in [t]} a_{i-1} X^{i-1}$.

Proof. The statements $h(x_1) + l(x_1) = y_1, \ldots, h(x_t) + l(x_t) = y_t$ are equivalent to the this linear system of equations:

$$\begin{pmatrix} 1 & x_1 & \ldots & x_1^{t-1} & h(x_1) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_t & \ldots & x_t^{t-1} & h(x_t) \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_{t-1} \\ 1 \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_t \end{pmatrix} \Leftrightarrow$$

$$\begin{pmatrix} 1 & x_1 & \ldots & x_1^{t-1} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_t & \ldots & x_t^{t-1} \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_{t-1} \end{pmatrix} = \begin{pmatrix} y_1 - h(x_1) \\ \vdots \\ y_k - h(x_k) \end{pmatrix}$$

The matrix is a Vandermonde matrix, which is invertible. Therefore, multiplication by it is a bijection. Because $a_0, \dots, a_{t-1}$ are uniformly random, the probability of the system of equations to hold is p^{-t}.

Via a inclusion-exclusion argument, we can upper bound the probability of the polynomial having no roots.

Lemma 8. *For any polynomial $h(X)$ and uniform $a_0, a_1 \leftarrow_\$ \mathbb{F}_p$ we have $a_0 + a_1 X + h(X)$ no roots with probability $\leq 1/2$.*

Proof. We have

$$\begin{aligned}
&\mathsf{Pr}_{a_0,a_1}[\vee_{x\in\mathbb{F}_p}(a_0 + a_1 x + h(x) = 0)] \\
\geq &\sum_{x\in\mathbb{F}_p} \mathsf{Pr}_{a_0,a_1}[a_0 + a_1 x + h(x) = 0] \qquad (4) \\
&- \sum_{x_1<x_2\in\mathbb{F}_p} \mathsf{Pr}_{a_0,a_1}[\wedge_{x\in\{x_1,x_2\}}(a_0 + a_1 x + h(x) = 0)] \\
=&1 - \sum_{x_1<x_2\in\mathbb{F}_p} 1/p^2 \qquad (5) \\
\geq&1/2
\end{aligned}$$

Inequality 4 follows from a Bonferroni inequality 4 and equality 5 follows from Lemma 7.

We need the following basic fact.

Lemma 9. *The size of the image of a polynomial $h(X)$ corresponds to the number of $a_0 \in \mathbb{F}_p$ for which the polynomial $h(X) - a_0$ has a root in $\mathbb{F}_p$.*

Proof. If there exists an $x \in \mathbb{F}_p$ such that $h(x) = a_0$ then $h(X) - a_0$ has a root in $\mathbb{F}_p$ and vice versa.

Lemma 10. *For any polynomial $h(X)$ and uniform $a_1 \leftarrow_\$ \mathbb{F}_p$ it holds with probability $> 1 - 1/\sqrt{2}$ that $h(X) + a_1 X$ has a image of size $> p(1 - 1/\sqrt{2})$.*

Proof. Fix a polynomial $h(X)$. By Lemma 8 we know that the number of $a_0, a_1 \in \mathbb{F}_p$ for which the polynomial $f(X) = a_0 + a_1 X + h(X)$ has no root is at most $p^2/2$. Therefore, by a pigeon-hole argument there are $< p/\sqrt{2}$ choices of for a_1 such that there exist $> p/\sqrt{2}$ choices of a_0 such that $a_0 + a_1 X + h(X)$ has no root. This means that there are $> p - p/\sqrt{2}$ choices for a_1 such that $\leq p/\sqrt{2}$ choices for a_0 such that $a_0 + a_1 X + h(X)$ has no root. Therefore, there are $> p - p/\sqrt{2}$ choices for a_1 such that $> p - p/\sqrt{2}$ choices for a_0 such that $a_0 + a_1 X + h(X)$ has a root. The statement follows by Lemma 9.

Lemma 11. *Fix a polynomial $h(X)$. The statistical distance between $(a_1, h(x) + a_1 x)$ and (a_1, y) where a_1, x is uniformly random over $\mathbb{F}_p$ and y is uniformly random from the image of $h(X) + a_1 X$ is $< \sqrt{2} - 0.5$.*

Proof. By Lemma 10 $h(X) + a_1X$ has a image of size $> p(1 - 1/\sqrt{2})$ with probability $1 - 1/\sqrt{2}$. For the rest of this analysis we assume to be in this case.

Picking a random x and evaluating $h(X)+a_1X$ on it is the same as sampling the output uniformly random from the multiset image of $h(X) + a_1X$ (the multiset where each element y has the multiplicity of the number of elements such that the element evaluates to y). This multiset has size p.

We now convert the multiset image to set by enumerating the multiplicities of each element and call this set $\mathcal{M}$. E.g. a multiset $\{8, 8, 8, 13, 55, 55\}$ would turn into a set of tuples $\{(8, 1), (8, 2), (8, 3), (13, 1), (55, 1), (55, 2)\}$. We do the same thing to the image of $h(X) + a_1X$ and call it $\mathcal{D}$. However because it is a set it only ever has multiplicity one. So we would turn the set $\{8, 13, 55\}$ into $\{(8, 1), (13, 1), (55, 1)\}$. By the definition of these sets $\mathcal{D} \subseteq \mathcal{M}$. Because $\mathcal{M}$ is of size p and $\mathcal{D}$ is of size $> p(1-1/\sqrt{2})$ sampling a random element from $\mathcal{M}$ will be in $\mathcal{D}$ with probability $> 1 - 1/\sqrt{2}$. Therefore, for a $1 - 1/\sqrt{2}$ fraction of random choices in sampling x at random and then evaluating $h(x)+a_1x$ behaves exactly as sampling uniformly random from the image. Thus, the statistical distance is $< 1 - (1 - 1/\sqrt{2})^2 = 1 - (1.5 - 2/\sqrt{2}) = \sqrt{2} - 0.5$.

For our verifiable space-hard functions, we also want to have tight bound on the number of roots of these polynomials. A natural candidate for this is a distribution over permutation polynomials. In previous chapters we showed how to efficiently invert the specific set of Guralnick-Müller permutation polynomials [GM97], which [BBBF18] suggested as a time-lock puzzle. As we leveraged the specific structure of these polynomials for our attack, we believe it to be prudent to stay away from permutation polynomials.

If we instead use random sparse polynomials in our verifiable space-hard functions, we would want to have a good bound on the number of roots of these polynomials. The work of [Kel16] conjectures that the number of roots in a random sparse polynomial is $O(k \log p)$, which would be good enough for as this also implies that the probability of sampling a polynomial without roots is not overwhelming.

To have a lower probability of sampling a polynomial without roots that we can even prove, we suggest the distribution of polynomials which are the sum of a dense low-degree polynomial and a single high-degree monomial. We define the distribution $\mathcal{D}_{\lambda,S}$ as follows:

Definition 4 (Low-Degree Dense). *Pick a prime $p \in \Omega(2^\lambda)$, degree $d \in \Omega(S)$, and number of non-zero monomials k such that $k \log k - 2k \geq \lambda$. Operations happen over $\mathbb{F}_p$.*

$$a_0 + \sum_{i \in [k-1]} a_i X^i + X^d$$

For uniformly random $a_i \leftarrow_\$ \mathbb{F}_q$.

Lemma 12. *For a polynomial $f(X)$ sampled from distribution $\mathcal{D}_{\lambda,S}$ we have $f(X)$ has at least k (as in Definition 4) roots with probability $\leq 2^{-\lambda}$.*

Proof. A polynomial having k roots equivalent to the statement that there exists a set of λ distinct points $x_1, \ldots, x_k \in \mathbb{F}_p$ such that the polynomial evaluates to zero at these points. There are $\binom{p}{k}$ much sets. For each of those sets, the probability that the polynomial evaluates to zero at these points is p^{-k} by Lemma 7. Therefore, by union bound the probability that the polynomial evaluates to zero at any set of k distinct points is $\leq \frac{\binom{p}{k}}{p^k} \leq \frac{p(p-1)\cdots(p-k+1)}{k!\cdot p^k} \leq \frac{1}{k!} \leq 2^{-\lambda}$. The last inequality follows from Stirling's approximation and our choice of k relative to λ.

5 Space-Lock Puzzle from SRF

We define space-lock puzzles analogously to time-lock puzzles but the resource we restrict is not sequential time but space.

Definition 5 (Space-Lock Puzzle). *A space-lock puzzle (SLP) with message space $\{0,1\}^n$ is a tuple of three algorithms $\mathsf{SLP} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Solve})$ defined as follows:*

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, S)$ *: The setup algorithm* Setup *takes as input a security parameter 1^λ and a space bound S and outputs public parameters* pp.

$\mathsf{p} \leftarrow \mathsf{Gen}(\mathsf{pp}, \mathsf{m})$ *: The puzzle generation algorithm* Gen *takes as input public parameters* pp *and a message* m *and outputs a puzzle* p.

$\mathsf{m} \leftarrow \mathsf{Solve}(\mathsf{pp}, \mathsf{p})$ *: The solving algorithm* Solve *takes as input a puzzle* p *and outputs a message* m.

Statistical Correctness *:* $\mathsf{SLP} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Solve})$ *is statistically correct if for all polynomials $S(\cdot)$ there exists a negligible function* negl *s.t. for all $n, \lambda \in \mathbb{N}$ and $\mathsf{m} \in \{0,1\}^n$:*

$$\Pr\left[\mathsf{m} \neq \mathsf{Solve}(\mathsf{pp}, \mathsf{p}) \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, S(\lambda)) \\ \mathsf{p} \leftarrow \mathsf{Gen}(\mathsf{pp}, \mathsf{m}) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Efficiency *: There exists a polynomial* poly *such that for all $n, \lambda, S \in \mathbb{N}$, and $\mathsf{m} \in \{0,1\}^n$ the runtime (and therefore space usage) of $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, S)$ and $\mathsf{p} \leftarrow \mathsf{Gen}(\mathsf{pp}, \mathsf{m})$ is $\leq \mathsf{poly}(\lambda, n, \log S)$.*

Security *:* SLP *is secure with gap $\varepsilon < 1$ if there exists a polynomial $\tilde{S}(\cdot)$ such that all polynomials $S(\cdot) \geq \tilde{S}(\cdot)$ and PPT adversaries $\{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ with space bound $S^\varepsilon(\lambda)$ there exists a negligible function* negl *s.t. for all $n, \lambda \in \mathbb{N}$:*

$$\Pr\left[b = \mathcal{A}_\lambda(\mathsf{st}, \mathsf{p}) \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, S(\lambda)) \\ (\mathsf{m}_0, \mathsf{m}_1, \mathsf{st}) \leftarrow \mathcal{A}_\lambda(\mathsf{pp}) \\ b \leftarrow_{\$} \{0,1\} \\ \mathsf{p} \leftarrow \mathsf{Gen}(\mathsf{pp}, m_b) \end{array}\right] \leq 1/2 + \mathsf{negl}(\lambda)$$

We present the first space-lock puzzle based on the space hardness of finding roots of polynomials.

Construction 1 (Space-Lock Puzzle) *Let $\mathcal{D}_{\lambda,S}$ be a distribution of polynomial that are fast to evaluate and where finding roots is space-hard. Further, $\mathsf{H} : \mathbb{F}_q \to \{0,1\}^{\lambda+n}$, where n is the size of the message space then the following defines a space-lock puzzle.*

$\mathsf{Setup}(1^\lambda, S)$ *: Return (λ, S).*
$\mathsf{Gen}(\mathsf{pp}, \mathsf{m})$ *:*
- *Sample a degree S polynomial $f(X) \in \mathcal{D}_{\lambda,S}$.*
- *Sample a uniformly random element $z \leftarrow_\$ \mathbb{F}_q$.*
- *Let $y = f(z)$.*
- *Return $(f(X), y, \mathsf{H}(z) \oplus (0^\lambda || m))$.*

$\mathsf{Solve}(\mathsf{p} = (f(x), y, c))$ *:*
- *Compute Z, the set of roots of the polynomial $f(X) - y$.*
- *For $z^* \in Z$:*
  - *Compute $\tilde{m} \leftarrow \mathsf{H}(z^*) \oplus c$.*
  - *If the first λ bits of $\tilde{m}$ are all 0 return the rest of $\tilde{m}$*

Theorem 3. *Construction 1 is a space-lock puzzle under the assumption that the SRF assumption 1 holds for polynomials with uniform linear coefficient.*

Proof. The proof follows from Lemmas 13 to 15.

Lemma 13 (Statistical Correctness). *Construction 1 is statistically correct.*

Proof. Because $f(X) - y$ has degree S it has at most S roots. We know that $f(z) = y$, therefore, z is a root of $f(X) - y$. Because H is a random oracle we have that for all $z^* \in Z \setminus \{z\}$ the probability that the first λ many bits of $\tilde{m}$ are 0^λ is $2^{-\lambda}$. Therefore, Solve outputs m with all but probability $\leq S(\lambda) \cdot 2^{-\lambda}$, which is negligible in λ.

Lemma 14 (Efficiency). *Construction 1 is efficient.*

Proof. The properties of $\mathcal{D}_{\lambda,S}$ tell us that evaluating $f(X)$ on z can be done in $\mathsf{poly}(\lambda, \log S)$. It follows that Gen runs in time and space $\mathsf{poly}(\lambda, \log S)$.

Lemma 15 (Security). *Construction 1 is secure under the SRF assumption 1 for polynomials with uniform linear and constant coefficient.*

Proof. To break security of the Construction 1 the adversary has to compute z given $f(X)$ and y otherwise he has no way of computing $\mathsf{H}(z)$.

We fix an $f(X)$. In Construction 1 the adversary has to compute the root of a polynomial following the distribution $f(X) - f(x)$ for uniformly random x. In the assumption the adversary has to compute a root of a polynomial $f(X) - y$, where y is uniformly at random. Because of the polynomial identity lemma we have $f(X) - f(x)$ can have at most d roots, where d is the degree of $f(X) - f(x)$. Therefore, for every y^* we have $\mathsf{Pr}_{x \leftarrow_\$ \mathbb{F}}[f(x) = y^*] \leq d/|\mathbb{F}| = d \cdot \mathsf{Pr}[y = y^*]$. Therefore, if an adversary compute a root for $f(X) - f(x)$ with probability ε then it can compute a root for $f(X) - y$ with probability $\geq \varepsilon/d$.

6 Verifiable Space-Hard Function from SRF

The definition of verifiable space hard function is similar to the definition of verifiable delay function but instead of a sequential time bound we have a space bound.

Definition 6 (Verifiable Space-Hard Function). *A verifiable space-hard function (VSHF) with domain space $\mathbb{X}$, codomain $\mathbb{Y}$, and proof space Π has the following algorithms:*

$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, S)$ *: The setup algorithm* Setup *takes as input a security parameter λ and a space bound S and outputs public parameters* pp*.*

$(y, \pi) \leftarrow \mathsf{Eval}(\mathsf{pp}, x)$ *: The evaluation deterministic algorithm* Eval *takes as input public parameters* pp *and outputs a function output y and a proof $\pi \in \Pi$.*

$b \leftarrow \mathsf{Verify}(\mathsf{pp}, x, y, \pi)$ *: The verification algorithm* Verify *takes as input public parameters* pp*, a function input $x \in \mathbb{X}$, a function output y and a proof $\pi \in \Pi$ and outputs a bit b.*

it has the following properties:

Correctness *:* $\mathsf{VSHF} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *is correct if for all $\lambda, S \in \mathbb{N}$ and $x \in \mathbb{X}$ we have* $\mathsf{Verify}(\mathsf{pp}, x, y, \pi) = 1$ *for* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, S)$ *and* $(y, \pi) \leftarrow \mathsf{Eval}(\mathsf{pp}, x)$*.*

Space Hardness *:* $\mathsf{VSHF} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *is sound with entropy E and a gap $\varepsilon < 1$ if there exists a polynomial $\tilde{S}(\cdot)$ such that for all polynomials $S(\cdot) \geq \tilde{S}(\cdot)$ and PPT adversaries $\{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ with space bound $S^\varepsilon(\lambda)$ there exists a negligible function* negl *such that for all $\lambda \in \mathbb{N}$, $x \in \mathbb{X}$:*

$$\mathsf{Pr}\left[\mathsf{Eval}(\mathsf{pp}, x) = (y, \pi) \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, S(\lambda)) \\ (x, y) \leftarrow \mathcal{A}_\lambda(\mathsf{pp}) \end{array}\right] \leq 2^{-E} + \mathsf{negl}(\lambda)$$

Computational Uniqueness *:* $\mathsf{VSHF} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *is computationally unique if for all PPT adversaries $\mathcal{A}$ there exists a negligible function* negl *s.t. for all $\lambda \in \mathbb{N}$ and $x \in \mathbb{X}$:*

$$\mathsf{Pr}\left[\mathsf{Verify}(\mathsf{pp}, x, y^*, \pi^*) = 1 \wedge y^* \neq y \middle| \begin{array}{c} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, S(\lambda)) \\ (x, y^*, \pi^*) \leftarrow \mathcal{A}_\lambda(\mathsf{pp}) \\ (y, \pi) \leftarrow \mathsf{Eval}(\mathsf{pp}, x) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Efficiency *: There exists a polynomial* poly *such that for all $\lambda, S \in \mathbb{N}$, $\pi \in \Pi$, and $x \in \mathbb{X}$ the runtime (and therefore space usage) of* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, S)$ *and* $\mathsf{Verify}(\mathsf{pp}, x, y, \pi)$ *is* $\leq \mathsf{poly}(\lambda, \log S)$*.*

We now show how to construct a verifiable space-hard function. We follow the same basic idea as the weak verifiable delay function of [BBBF18] but we require more care because we do not rely on permutation polynomials.

We present the function as an IOP, but the proof can be made non-interactive using the Fiat-Shamir heuristic and Merkle trees [BCS16].

Construction 2 (Verifiable Space-Hard Function) *Let* (P, V) *be an IOPP for Reed-Solomon codes that evaluates the polynomials at points* $\mathcal{L}$ *and has a negligible soundness error. It proves a string* x *is at constant relative distance from a Reed-Solomon codeword (degree* d *polynomials).*

We also use the combine procedure form Section 2.6.

Our verifiable space-hard function does not require any setup.

$\mathsf{Eval}(S, x)$ *:*

- *Let* $f(X) = \mathsf{H}(x)$*.*
- *Let the degree of* $f(X)$ *be* d*.*
- *Let* $(p_i)_{i \in \{0\} \cup [\lfloor \log p \rfloor]}$ *be the binary decomposition of* p*.*
- *Let* $m_0(X) = X$ *and* $v_0(X) = X^{p_0}$*.*
- *For* $i \in [\lfloor \log p \rfloor]$*:*
  - *Compute* $m_i(X) = m_{i-1}(X) \cdot m_{i-1}(X) \mod f(X)$*.*
  - *Compute* $e_i(X) = (m_{i-1}(X) \cdot m_{i-1}(X) - m_i(X))/f(X)$
  - *Compute* $v_i(X) = v_{i-1}(X) \cdot m_i^{p_i}(X) \mod f(X)$*.*
  - *Compute* $w_i(X) = (i_{i-1}(X) \cdot m_i^{p_i}(X)(X))/f(X)$
  - *Send* $m_i(X)$*,* $e_i(X)$*,* $v_i(X)$*, and* $w_i(X)$ *evaluated at* $\mathcal{L}$ *to the verifier.*
- *Compute* $g(X)$*, the gcd of* $v_{\lfloor \log p \rfloor}(X) - X$ *and* $f(X)$ *together with their Bézout coefficients* $a(X), b(X)$*.*
- *Send* $a(X)$ *and* $b(X)$ *evaluated at* $\mathcal{L}$ *to the verifier.*
- *Pick a uniformly random field element* $r \leftarrow_{\$} \mathbb{F}_p$*.*
- *Run* $c(X) \leftarrow \mathsf{Combine}(d,\ r,\ (m_i(X), d-1)_{i \in \{0\} \cup [\lfloor \log p \rfloor]},\ (e_i(X), d-2)_{i \in \{0\} \cup [\lfloor \log p \rfloor]},\ (v_i(X), d-1)_{i \in \{0\} \cup [\lfloor \log p \rfloor]},\ (w_i(X), d-2)_{i \in \{0\} \cup [\lfloor \log p \rfloor]},\ (a(X), d-1),\ (b(X), d-2))$*.*
- *Send* $c(X)$ *evaluated at* $\mathcal{L}$ *to the verifier.*
- *Run* P *to prove that* $c(X)$ *is at a constant relative distance from a polynomial of degree* d*.*
- *Return function output* $y = \frac{g(X)}{\textit{leading coefficient of } g(X)}$*.*

$\mathsf{Verify}(x, y = g(X))$ *:*

- *Let* $f(X) = \mathsf{H}(x)$*.*
- *If* $g(X)$ *is not monic return 0.*
- *Run* V *to verify that* $c(X)$ *is at a constant relative distance from a polynomial of degree* d*.*
- *Let* $(p_i)_{i \in \{0\} \cup [\lfloor \log p \rfloor]}$ *be the binary decomposition of* p*.*
- *Sample a random set* $R \subset \mathcal{L}$ *of size* λ*.*
- *For* $r \in R$*:*
  - *Read* $m_0(r)$ *and* $v_0(r)$ *from the prover string.*
  - *If* $m_0(r) \neq r$ *return 0.*
  - *If* $v_0(r) \neq r^{p_0}$ *return 0.*
- *For* $i \in [\lfloor \log p \rfloor]$*:*
  - *For* $r \in R$*:*
    - *Read* $(m_{i-1},\ m_i(r),\ e_i(r),\ v_{i-1}(r),\ v_i(r),\ w_i(r))$ *from the prover string.*
    - *If* $m_{i-1}(r)^2 \neq m_i(r) + e_i(r) \cdot f(r)$ *return 0.*
    - *If* $v_{i-1}(r) \cdot m_i(r)^{p_i} \neq v_i(r) + w_i(r) \cdot f(r)$ *return 0.*

– *For $r \in R$:*
  - *If $a(r) \cdot (v_{\lfloor \log p \rfloor}(r) - r) + b(r) \cdot f(r) \neq g(r)$ return 0.*
– *If $f(X) \mod g(X) \neq 0$ or $X^p - X \mod g(X) \neq 0$ return 0.*
– *Return 1.*

Remark 3. Note, the above function does not have high output entropy because $f(X)$ does not have any roots with $\leq 1/2$ probability. This is required by many applications and can easily be fixed by repetition.

Theorem 4. *Construction 2 is a verifiable space-hard function.*

Proof. Follows from Lemmas lemmas 16 to 18 and Corollary 1.

Lemma 16. *[Correctness] Construction 2 is correct.*

Proof. Because the prover divides $g(X)$ by its leading coefficient it outputs a monic polynomial. All the checks made by V pass, which follows from the correctness of (P, V). For $i \in \{0\} \cup [\lfloor \log p \rfloor]$ it holds that $m_i(X) = X^{2^i} \mod f(X)$. We also have for $i \in [\lfloor \log p \rfloor]$ it holds $m_{i-1}(X)^2 = m_i(X) + f(X)e_i(X)$. Similarly, for binary decomposition $(p_i)_{i \in \{0\} \cup [\lfloor \log p \rfloor]}$ of p we have

$$v_i(X) = \prod_{i \in \{\} \cup [\lfloor \log p \rfloor]} X^{2^i p_i} \mod f(X)$$

and

$$w_i(X) = \prod_{i \in \{\} \cup [\lfloor \log p \rfloor]} X^{2^i p_i} / v_i(X).$$

Therefore, evaluating all these polynomial at the point r still makes these equations hold.

By definition of a common divisor $g(X)$ divides $v_{\lfloor \log p \rfloor}(X) - X$ and $f(X)$. Because $v_{\lfloor \log p \rfloor}(X) - X = X^p - X \mod f(X)$ we get if $g(X)$ divides $v_{\lfloor \log p \rfloor}(X) - X$ and $f(X)$ it also divides $X^p - X$. The Bézout coefficients have the property that $a(X) \cdot (v_{\lfloor \log p \rfloor}(X) - X) + b(X) \cdot f(X) = g(X)$. Therefore, each check passes if π is honestly generated.

Lemma 17. *[Efficiency] Construction 2 is efficient.*

Proof. We require from $\mathcal{D}_{\lambda,S}$ that with all but negligible probability $f(X)$ has a polynomial number of roots. For polynomials where the λ low order monomials are dense (see Definition 4) this follows from Lemma 12. Therefore, the degree of $g(X)$ is polynomial in λ. Reading the prover string for each polynomial at $\leq 2\lambda$ many locations is in $\mathsf{poly}(\lambda)$. If we have an IOPP for Reed-Solomon codes where the verifier runs in time $\mathsf{poly}(\lambda, \log S)$, then the verifier of the entire protocol runs in time $\mathsf{poly}(\lambda, \log S)$.

Lemma 18. *[Computational Uniqueness] The Construction 2 is computationally unique.*

Proof. Assume there exists an $\mathcal{S} \subset \mathcal{L}$ with $|\mathcal{S}| \geq (1-\delta) \cdot |\mathcal{L}|$ such that

- for all $(m_i(X))_{i \in \{0\} \cup [\lfloor \log p \rfloor]}$, there exists a $\hat{m}_i(X) \in \mathsf{RS}[\mathcal{L}, d-1]$ with $m_i(\mathcal{S}) = \hat{m}_i(S)$,
- for all $(e_i(X))_{i \in \{0\} \cup [\lfloor \log p \rfloor]}$, there exists a $\hat{e}_i(X) \in \mathsf{RS}[\mathcal{L}, d-2]$ with $e_i(\mathcal{S}) = \hat{e}_i(\mathcal{S})$,
- for all $(v_i(X))_{i \in \{0\} \cup [\lfloor \log p \rfloor]}$, there exists a $\hat{v}_i(X) \in \mathsf{RS}[\mathcal{L}, d-1]$ with $v_i(\mathcal{S}) = \hat{v}_i(\mathcal{S})$,
- for all $(w_i(X))_{i \in \{0\} \cup [\lfloor \log p \rfloor]}$, there exists a $\hat{w}_i(X) \in \mathsf{RS}[\mathcal{L}, d-2]$ with $w_i(\mathcal{S}) = \hat{w}_i(\mathcal{S})$,
- there exist $\hat{a}(X) \in \mathsf{RS}[\mathcal{L}, d-1]$ and $\hat{b}(X) \in \mathsf{RS}[\mathcal{L}, d-2]$ with $a(\mathcal{S}) = \hat{a}(\mathcal{S})$ and $b(\mathcal{S}) = \hat{b}(\mathcal{S})$.

First, we lower-bound the number of elements in $\mathcal{S} \cap R$. So we can use them together with the polynomial identity lemma to check the polynomial equations.

By multiplicative Chernoff bound we have that

$$\mathsf{Pr}[|\mathcal{S} \cap R| \leq \lambda(1-\delta)/2] \leq \left(\frac{e^{-1/2}}{(1/2)^{1/2}}\right)^{\lambda(1-\delta)} < 0.9^{\lambda(1-\delta)}$$

. Since δ is constant, we have that with all but negligible probability $|\mathcal{S} \cap R| > \lambda(1-\delta)/2$.

Each polynomial equation we check is at most of degree $2d$. Therefore, we get via polynomial identity lemma 1 if a polynomial equation does not hold then evaluating the polynomial at a random point on $\mathcal{S}$ and then checking holds with probability $2d/|\mathcal{S}|$, which, again, is constant. Because we run this test $|\mathcal{S} \cap R|$ many times we detect it with all but negligible probability.

From these equations follows that $\hat{v}_{\lfloor \log p \rfloor}(X) - X = X^p - X \mod f(X)$. Therefore, $gcd(\hat{v}_{\lfloor \log p \rfloor}(X) - X, f(X)) = gcd(X^p - X, f(X))$. By Bézout's identity for all $a'(X)$, $b'(X)$ we have $a'(X) \cdot (\hat{v}_{\lfloor \log p \rfloor}(X) - X) + b'(X) \cdot f(X)$ is a multiple of $gcd(\hat{v}_{\lfloor \log p \rfloor}(X), f(X))$. So, the check $a(X) \cdot (\hat{v}_{\lfloor \log p \rfloor}(X) - X) + b(X) \cdot f(X) = g(X)$ verifies that $g(X)$ is a multiple of $gcd(X^p - X, f(X))$. The checks $f(X) \mod g(X) = 0$ and $X^p - X \mod g(X) = 0$ verify that $g(X)$ is a divisor of $f(X)$ and $X^p - X$. Therefore, $g(X)$ is a greatest common divisor of $f(X)$ and $X^p - X$. The gcd is unique up to multiplication by a field element, which is why we require $g(X)$ to be monic.

If our initial assumption about the existence of $\mathcal{S}$ does not hold, then by the soundness of $\mathsf{Combine}$, detailed in Section 2.6, we get that $c(X)$ is far from $\mathsf{RS}[\mathcal{L}, d]$. By the soundness of the IOPP that checks that $c(X)$ is close to the code this can only happen with negligible probability.

Remark 4. With access to a extractable polynomial commitment scheme the above construction can be made much simpler by replacing the IOPPs with the polynomial commitment. Then we only need to check the polynomials at one location instead of λ many locations.

Corollary 1 (Space Hardness). *Construction 2 is space-hard.*

Proof. Under Assumption 2 for polynomials that are dense in the low degrees Definition 4 space-hardness with entropy 1 follows directly from computational uniqueness and the fact that the polynomial has a root with probability $> 1/2$ (Lemma 10).

References

AB16. Joël Alwen and Jeremiah Blocki. Efficiently computing data-independent memory-hard functions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 241–271, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

ABB22. Mohammad Hassan Ameri, Alexander R Block, and Jeremiah Blocki. Memory-hard puzzles in the standard model with applications to memory-hard functions and resource-bounded locally decodable codes. In *International Conference on Security and Cryptography for Networks*, pages 45–68. Springer, 2022.

ABBK16. Frederik Armknecht, Ludovic Barman, Jens-Matthias Bohli, and Ghassan O. Karame. Mirror: Enabling proofs of data replication and retrievability in the cloud. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 1051–1068, Austin, TX, USA, August 10–12, 2016. USENIX Association.

ABFG14. Giuseppe Ateniese, Ilario Bonacina, Antonio Faonio, and Nicola Galesi. Proofs of space: When space is of the essence. In Michel Abdalla and Roberto De Prisco, editors, *SCN 14*, volume 8642 of *LNCS*, pages 538–557, Amalfi, Italy, September 3–5, 2014. Springer, Heidelberg, Germany.

ABP18. Joël Alwen, Jeremiah Blocki, and Krzysztof Pietrzak. Sustained space complexity. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 99–130, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

ACFY24. Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. Stir: Reed–solomon proximity testing with fewer queries. *Cryptology ePrint Archive*, 2024.

ACP+17. Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt is maximally memory-hard. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 33–62, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.

ACY23. Gal Arnon, Alessandro Chiesa, and Eylon Yogev. Iops with inverse polynomial soundness error. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 752–761. IEEE, 2023.

AGP24. Benedikt Auerbach, Christoph U Günther, and Krzysztof Pietrzak. Trapdoor memory-hard functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 315–344. Springer, 2024.

AS15. Joël Alwen and Vladimir Serbinenko. High parallel complexity graphs and memory-hard functions. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 595–603, Portland, OR, USA, June 14–17, 2015. ACM Press.

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, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.

BBHR18. Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast reed-solomon interactive oracle proofs of proximity. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPIcs*, pages 14:1–14:17, Prague, Czech Republic, July 9–13, 2018. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

BCCT13. Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 111–120, Palo Alto, CA, USA, June 1–4, 2013. ACM Press.

BCS16. Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 31–60, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

Ber70. Elwyn R Berlekamp. Factoring polynomials over large finite fields. *Mathematics of computation*, 24(111):713–735, 1970.

BGJ+16. Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *ITCS 2016*, pages 345–356, Cambridge, MA, USA, January 14–16, 2016. ACM.

BN00. Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 236–254, Santa Barbara, CA, USA, August 20–24, 2000. Springer, Heidelberg, Germany.

Bon36. Carlo Bonferroni. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R Istituto Superiore di Scienze Economiche e Commericiali di Firenze*, 8:3–62, 1936.

BP17. Alex Biryukov and Léo Perrin. Symmetrically and asymmetrically hard cryptography. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 417–445, Hong Kong, China, December 3–7, 2017. Springer, Heidelberg, Germany.

Cle86. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369, Berkeley, CA, USA, May 28–30, 1986. ACM Press.

CM19. Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

CP18. Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 451–467, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

CZ81. David G Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, 36(154):587–592, 1981.

DFKP15. Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. Proofs of space. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 585–605, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

DH76. Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

DL78. Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.

DLM19. Nico Döttling, Russell W. F. Lai, and Giulio Malavolta. Incremental proofs of sequential work. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 292–323, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

DMPS19. Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 248–277, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany.

GM97. Robert M Guralnick and Peter Müller. Exceptional polynomials of affine type. *Journal of Algebra*, 194(2):429–454, 1997.

GNU16. Zeyu Guo, Anand Kumar Narayanan, and Chris Umans. Algebraic problems equivalent to beating exponent 3/2 for polynomial factorization over finite fields. *arXiv preprint arXiv:1606.04592*, 2016.

HHK+22. Charlotte Hoffmann, Pavel Hubácek, Chethan Kamath, Karen Klein, and Krzysztof Pietrzak. Practical statistically-sound proofs of exponentiation in any group. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 370–399, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Heidelberg, Germany.

HHKK23. Charlotte Hoffmann, Pavel Hubáček, Chethan Kamath, and Tomáš Krňák. (verifiable) delay functions from lucas sequences. *Cryptology ePrint Archive*, 2023.

Kel16. Zander Kelley. Roots of sparse polynomials over a finite field. *LMS Journal of Computation and Mathematics*, 19(A):196–204, 2016.

KS95. Erich Kaltofen and Victor Shoup. Subquadratic-time factoring of polynomials over finite fields. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 398–406, 1995.

KU11. Kiran S Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM Journal on Computing*, 40(6):1767–1802, 2011.

KWJ23. Alireza Kavousi, Zhipeng Wang, and Philipp Jovanovic. Sok: Public randomness. Cryptology ePrint Archive, Paper 2023/1121, 2023. `https://eprint.iacr.org/2023/1121`.

KZG10. Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194, Singapore, December 5–9, 2010. Springer, Heidelberg, Germany.

LW17. Arjen K Lenstra and Benjamin Wesolowski. Trustworthy public randomness with sloth, unicorn, and trx. *International Journal of Applied Cryptography*, 3(4):330–343, 2017.

MMV13. Mohammad Mahmoody, Tal Moran, and Salil P. Vadhan. Publicly verifiable proofs of sequential work. In Robert D. Kleinberg, editor, *ITCS 2013*, pages 373–388, Berkeley, CA, USA, January 9–12, 2013. ACM.

Per09. Colin Percival. Stronger key derivation via sequential memory-hard functions, 2009.

Pie19. Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15, San Diego, CA, USA, January 10–12, 2019. LIPIcs.

RD16. Ling Ren and Srinivas Devadas. Proof of space from stacked expanders. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 262–285, Beijing, China, October 31 – November 3, 2016. Springer, Heidelberg, Germany.

RSW96. Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. *Technical Report*, 1996.

Sch80. Jacob T Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717, 1980.

Sho93. Victor Shoup. Factoring polynomials over finite fields: asymptotic complexity vs. reality. In *Proc. IMACS Symposium, Lille, France*. Citeseer, 1993.

Val08. Paul Valiant. Incrementally verifiable computation or proofs of knowledge imply time/space efficiency. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 1–18, San Francisco, CA, USA, March 19–21, 2008. Springer, Heidelberg, Germany.

VZGS92. Joachim Von Zur Gathen and Victor Shoup. Computing frobenius maps and factoring polynomials. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 97–105, 1992.

WC81. Mark N Wegman and J Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of computer and system sciences*, 22(3):265–279, 1981.

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, Darmstadt, Germany, May 19–23, 2019. Springer, Heidelberg, Germany.

Zip79. Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International symposium on symbolic and algebraic manipulation*, pages 216–226. Springer, 1979.

A Implemented Attack

It follows an implementation of the attack we describe in Section 3. The code is written in Magma.

```
//Parameters
p:=101;
s:=p^3;
n:=11;
q:=p^n;

GFq<a>:=GF(q);
GFpol<X>:=PolynomialRing(GFq);

//Definition of the polynomial
function PermEval(mu,x)
  a:=x^(2*s)-2*mu*x^(s+1)+2*mu*x^s+mu^2*x^2+2*mu^2*x+mu^2;
  return (a^((s+1) div 2)+(x^s-mu*x+mu)^s*(x^s-mu*x-mu))/(2*x^s);
```

```
end function;

//Generating the polynomial
//Checking mu is correct (this part is slow).
//Indeed, this is not even part of the attack;
//this defines the function we are trying to invert.
//Can be skipped as mu is likely to be correct.
notfound:=true;
while notfound do
  mu:=Random(GFq);
  if #Roots(X^(s-1)-mu) eq 0 then
    notfound:=false;
  end if;
end while;
input:=Random(GFq);
target:=PermEval(mu,input);

//Attack starts here
delta:=(target^2-4*mu^(2*s+1))^((s-1) div 2);
A2:=(target^s-delta*target)/(2*mu^s);
A1:=-mu^s*delta;
A0:=-target^s/2-((target-2*mu^(s+1))/2)*delta+mu^(s^2);
B2:=A2-mu^(s^2);
B1:=A1-A2*mu^s;
B0:=-mu*A1;
M:=ZeroMatrix(GFq,n,n);
for i:=0 to n-1 do
    M[i+1,i+1]:=B0^(s^i);
    M[i+1,(i+1) mod n+1]:=B1^(s^i);
    M[i+1,(i+2) mod n+1]:=B2^(s^i);
    M[i+1,(i+3) mod n+1]:=1;
end for;
Q:=[];
for i:=0 to n-1 do
  Append(~Q,-A0^(s^i));
end for;
V:=Vector(Q);
M:=Transpose(M);
S:=Solution(M,V);

if (S[1] eq input) then
  print "Correct input recovered!";
end if;
```

B Improvement of the attack complexity

The attack described in Section 3 is polynomial time but can still be improved. In its most basic form, it requires to perform linear algebra on a $n \times n$ system defined over the large field $\mathbb{F}_Q$. A first easy improvement is to remark that because the linear system is Frobenius invariant, one can use Weil descent to instead consider a system over $\mathbb{F}_{q_0}$. Alternatively, we remark that the system is sparse which can also lead to algorithm speed-up by relying on an iterative algorithm. Unfortunately, the linear system over the small field is not sparse which reduces the total gain we can achieve with this approach.

In the present section, we show a different approach to solve the system using only $O(n)$ arithmetic operation over $\mathbb{F}_Q$. For this, we proceed in two steps:

1. We solve the equation $Y^{q^2} + A_2 Y^q + A_1 Y + A_0 = 0$ provided by Theorem 2.
2. We recover X by solving $X^q - \mu X - Y = 0$.

We use the same technique to solve both systems. Since the equation $X^q - \mu X - Y = 0$ has fewer terms, it is easier to start by describing the second step.

Let y_0 be a given value in $\mathbb{F}_Q$. We want to recover an element X in $\mathbb{F}_Q$, such that $X^q = \mu X + y_0$. As a consequence of X being in $\mathbb{F}_Q$ we also know that $X^{q^n} - X = 0$. Raising our first equation to the power q^i, we can remark that:

$$\begin{pmatrix} X^{q^{i+1}} \\ 1 \end{pmatrix} = \begin{pmatrix} \mu^{q^i} & y_0^{q^i} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} X^{q^i} \\ 1 \end{pmatrix}.$$

Let M_i denote the 2×2 matrix in the above equation. As a consequence, we have:

$$\begin{pmatrix} X^{q^n} \\ 1 \end{pmatrix} = \prod_{i=1}^{n} M_{n-i} \cdot \begin{pmatrix} X \\ 1 \end{pmatrix}.$$

In particular:

$$X^{q^n} = \begin{pmatrix} 1 & 0 \end{pmatrix} \cdot \prod_{i=1}^{n} M_{n-i} \cdot \begin{pmatrix} X \\ 1 \end{pmatrix}.$$

Thus, the vector

$$\begin{pmatrix} 1 & 0 \end{pmatrix} \cdot \prod_{i=1}^{n} M_{n-i}$$

provides an expression of X^{q^n} as a linear combination of X and 1. After replacing X^{q^n} in the equation $X^{q^n} - X = 0$, we obtain a linear equation in X and recover its value.

In the first step we solve the equation $Y^{q^2} + A_2 Y^q + A_1 Y + A_0 = 0$. We proceed in a similar fashion by noting that:

$$\begin{pmatrix} Y^{q^{i+2}} \\ Y^{q^{i+1}} \\ 1 \end{pmatrix} = \begin{pmatrix} -A_2^{q^i} & -A_1^{q^i} & -A_0^{q^i} \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} Y^{q^{i+1}} \\ Y^{q^i} \\ 1 \end{pmatrix}. \tag{6}$$

Using the same argument, we can express Y^{q^n} as a linear combination of Y^q, Y and 1. This yields an equation $a\,Y^q + b\,Y + c = 0$. Raising to the power q gives $a^q\,Y^{q^2} + b^q\,Y^q + c^q = 0$. Putting the two equations with $Y^{q^2} + A_2\,Y^q + A_1\,Y + A_0 = 0$ yields an easy to solve linear system of dimension 3.

By inspection, one can see that the complete process only requires $O(n)$ arithmetic operations in $\mathbb{F}_Q$.

B.1 Code for Improved Attack

Below, we provide the magma code for this improved attack.

```
// Y then X approach
p:=101;
s:=p^4;
n:=11;
q:=p^n;
GFq<a>:=GF(q);
GFpol<X>:=PolynomialRing(GFq);

function PermEval(mu,x)
  a:=x^(2*s)-2*mu*x^(s+1)+2*mu*x^s+mu^2*x^2+2*mu^2*x+mu^2;
  return (a^((s+1) div 2)+(x^s-mu*x+mu)^s*(x^s-mu*x-mu))/(2*x^s);
end function;

//Don't test mu for now
mu:=Random(GFq);
input:=Random(GFq);
target:=PermEval(mu,input);

delta:=(target^2-4*mu^(2*s+1))^((s-1) div 2);
A2:=(target^s-delta*target)/(2*mu^s);
A1:=-mu^s*delta;
A0:=-target^s/2-((target-2*mu^(s+1))/2)*delta+mu^(s^2);

//Y^(s^2) + A2 Y^s + A1 Y + A0
//Y^(s^n)-Y
//Get simplified Y^(s^n) by many small matrix mults [Express in terms of Y^(s^i), Y^(s^(i-1)) and 1 until i=1]
V:=Vector([GFq!1,0,0]); //Trivial init
for i:=n-2 to 0 by -1 do
  M:=Matrix(GFq,3,3,[-A2^(s^(i)),-A1^(s^(i)),-A0^(s^(i)),1,0,0,0,0,1]);
  V:=V*M;
end for;
V[2]:=V[2]-1;
W:=V/V[1];
VecBase:=Vector([1,A2,A1,A0]);
VecWs:=Vector([1,W[2]^s,0,W[3]^s]);
VecW:=Vector([0,1,W[2],W[3]]);
VecFinal:=VecBase-VecWs-(A2-W[2]^s)*VecW;
Yval:=-VecFinal[4]/VecFinal[3];

// Go back from Y to X -- X^(s^n)-X=0 and X^s-mu*X-Yval=0
V:=Vector([GFq!1,0]); //Trivial init
for i:=n-1 to 0 by -1 do
  M:=Matrix(GFq,2,2,[mu^(s^(i)),Yval^(s^(i)),0,1]);
  V:=V*M;
end for;
V[1]:=V[1]-1;
```

```
recovered:=−V[2]/V[1];
```