

Secure PRNGs from Specialized Polynomial Maps over Any \mathbb{F}_q

Abstract. Berbain, Gilbert, and Patarin presented QUAD, a pseudo random number generator (PRNG) at Eurocrypt 2006. QUAD (as PRNG and stream cipher) may be proved secure based on an interesting hardness assumption about the one-wayness of multivariate quadratic polynomial systems over \mathbb{F}_2 .

The original BGP proof only worked for \mathbb{F}_2 and left a gap to general \mathbb{F}_q . We show that the result can be generalized to any arbitrary finite field \mathbb{F}_q , and thus produces a stream cipher with alphabets in \mathbb{F}_q .

Further, we generalize the underlying hardness assumption to specialized systems in \mathbb{F}_q (including \mathbb{F}_2) that can be evaluated more efficiently. Barring breakthroughs in the current state-of-the-art for system-solving, a rough implementation of a provably secure instance of our new PRNG is twice as fast and takes 1/10 the storage of an instance of QUAD with the same level of provable security.

Recent results on specialization on security are also examined. And we conclude that our ideas are consistent with these new developments and complement them. This gives a clue that we may build secure primitives based on specialized polynomial maps which are more efficient.

Keywords: sparse multivariate polynomial map, PRNG, hash function, provable security

1 Introduction

Cryptographers have used multivariate polynomial maps for primitives since Matsumoto-Imai [26] but there is a dearth of results proving security based on plausible hardness assumptions. Berbain, Gilbert and Patarin presented a breakthrough in Eurocrypt 2006, when they proposed a PRNG/stream cipher that is provably secure provided that the class of multivariate quadratic polynomials is probabilistically one way:

Class $\mathcal{MQ}(q, n, m)$: For given q, n, m , the class $\mathcal{MQ}(q, n, m)$ consists of all systems of m quadratic polynomials in \mathbb{F}_q with n variables. To choose a random system \mathbf{S} from $\mathcal{MQ}(q, n, m)$, we write each polynomial $P_k(\mathbf{x})$ as $\sum_{1 \leq i < j \leq n} a_{ijk} x_i x_j + \sum_{1 \leq i \leq n} b_{ik} x_k + c_k$, where every a_{ijk}, b_{ik}, c_k is chosen uniformly in \mathbb{F}_q . Solving $\mathbf{S}(\mathbf{x}) = \mathbf{b}$ for any \mathcal{MQ} system S is known as the “multivariate quadratic” problem.

It is often claimed that the NP-completeness of this problem [19] is the basis for multivariate public-key cryptosystems. We could take instead P_i 's to be polynomials of degree d instead of quadratic and get the class of “multivariate polynomial systems” $\mathcal{MP}(q, d, n, m)$. This contains $\mathcal{MQ}(q, n, m)$ as a subset, so solving arbitrary $\mathbf{S}(\mathbf{x}) = \mathbf{b}$ for any \mathcal{MP} system S would be no easier. However, it is not easy to base a proof on worst-case hardness; the premise used in [7] is the following average-case hardness assumption:

Assumption \mathcal{MQ} : Given any k and prime power q , For parameters n, m satisfying $m/n = k + o(1)$, no probabilistic polynomial-time algorithm can solve (in poly(n)-time) any fixed $\varepsilon > 0$ proportion of systems \mathbf{S} drawn from $\mathcal{MQ}(q, n, m)$, and a vector $\mathbf{b} = (b_1, b_2, \dots, b_m)$ drawn from $\mathbf{S}(U_n)$, where U_n is uniform distribution over $(\mathbb{F}_q)^n$ such that $\mathbf{S}(\mathbf{x}) = \mathbf{b}$.

With this premise, [7, Theorem 4] proved the QUAD PRNG secure over \mathbb{F}_2 . *However, a looseness factor in its security argument in the security proof means that provably secure QUAD instances over \mathbb{F}_2 are not yet of practical speed. It also does not work for fields larger than \mathbb{F}_2 . A similar result over any \mathbb{F}_q is non-trivial to prove, which we do here with different and more involved techniques. However, instances of QUAD with the same-size state over larger fields are significantly less secure [33].* To increase the difficulty of solving a system of nonlinear polynomial equations, we can plausibly change (a) the field size q , (b) the number of variables n , or (c) the degree d of the system (cf. [3, 4, 31]). Each costs time and space (for a reduction from the \mathcal{MQ} problem in \mathbb{F}_q case to \mathbb{F}_2 case, see [30]). Even with a hardware implementation, an increase in resource consumption is inevitable.

A logical next step is to combine all these approaches but find polynomials that are easier to evaluate. A natural candidate is sparsity in the chosen polynomials. To our knowledge, however, there are no prior positive results for provable security of specialized polynomial systems, and specifically sparse ones.

So the questions we are trying to answer are:

- Can we prove a similar result to [7] allowing for more efficiently evaluated specialized systems?
- What do we know about how these specializations affect complexity of system-solving?

1.1 Our New Ideas and Main Results

Instead of \mathcal{MQ} , we investigate a class $SMP(q, d, n, m, (\eta_2, \dots, \eta_d))$ of sparse polynomial systems with arbitrary affine parts and terms at other degrees with specified density. I.e., $\mathbf{S} = (P_1(\mathbf{x}), P_2(\mathbf{x}), \dots, P_m(\mathbf{x})) \in SMP(q, d, n, m, (\eta_2, \dots, \eta_d))$ consists of m polynomials of degree d in the variables $\mathbf{x} = (x_1, x_2, \dots, x_n)$; each P_i is a degree- d polynomial such that exactly $\eta_i = \eta_i(n)$ nonzero degree- i terms are present for each $i \geq 2$. The affine terms (coefficients) are totally randomly chosen. Also all the operations and coefficients are in \mathbb{F}_q .

To rephrase, the i -th polynomial we can be written as $P_i(\mathbf{x}) = \sum_{j=2}^d Q_j^{(i)}(\mathbf{x}) + \sum_{1 \leq j \leq n} a_{ij}x_j + c_i$ where each $Q_j^{(i)}(\mathbf{x})$ can be written in the form $\sum_{1 \leq \sigma(1) \leq \sigma(2) \leq \dots \leq \sigma(j) \leq n} a_{(\sigma(1), \sigma(2), \dots, \sigma(j))} x_{\sigma(1)} x_{\sigma(2)} \dots x_{\sigma(j)}$, or the sum of η_j monomials with degree j . “A random system from $SMP(q, d, n, m, (\eta_2, \dots, \eta_d))$ ” then has a probability distribution as follows: all a_{ij} , c_i are uniformly chosen from \mathbb{F}_q . To determine each $Q_j^{(i)}(\mathbf{x})$, we firstly uniformly choose η_j out of $\binom{n+j-1}{j}$ coefficients to be nonzero, then uniformly choose each of these nonzero coefficients from $\mathbb{F}_q^* := \mathbb{F}_q \setminus \{0\}$. All the others coefficients will be zero.

We now propose a probabilistic one-wayness assumption to base a security theorem on.

Assumption SMP : For given q, d , and for $n, m, \eta_2, \dots, \eta_d$ such that $m/n = k + o(1)$ and $\eta_i/n = k_i + o(1)$ (where k, k_2, k_3, \dots are constants) there is no probabilistic algorithm which can solve (in poly(n)-time) any fixed $\varepsilon > 0$ proportion of instances $\mathbf{S}(\mathbf{x})$ drawn from $SMP((q, d, n, m, (\eta_2, \dots, \eta_d))$, and a vector $\mathbf{b} = (b_1, b_2, \dots, b_m)$ drawn from $\mathbf{S}(U_n)$, where U_n is uniform distribution over $(\mathbb{F}_q)^n$ such that $\mathbf{S}(\mathbf{x}) = \mathbf{b}$.

In Secs. 2–3 Assumption SMP is shown to yield a secure PRNG (and hence a probably secure stream cipher), **for any** q . The key to this extension to general \mathbb{F}_q involves a reconstruction over linear polynomials, which is a non-trivial generalization of the Goldreich-Levin hard core bit by Goldreich-Rubinfeld-Sudan [21].

We then check that SMP instances are hard to solve on average (i.e., not just worst case) via the known fastest generic (cf. Sec. 4 and the Appendix B) and special-purpose algorithms. Finally we discuss their practical use. Preliminary implementations of our SPELT (Sparse Polynomials, Every Linear Term) can achieve 5541 and 11744 cycles per byte for a SMP -based *secure* stream cipher over \mathbb{F}_{16} (quartic, 108 variables) and \mathbb{F}_2 (cubic, 208 variables) respectively. The former is at least twice as fast as any other stream ciphers provably secure at the same parameters (cf. Sec. 5.2).

1.2 Previous Work

There had been “provably secure” PRNGs based on discrete log [20], or on hardness of factorization (as in Blum, Blum, and Shub [10]) or a modification thereof [29], or \mathcal{MQ} [7]. But the security proofs always require impractically high parameters for “provable security”, which limit their utility. For example:

- The BBS stream generator at commonly used parameters is not provably secure [23, Sec. 6.1].
- With [29], the specified security level was 2^{70} , today’s cryptographers usually aim for 2^{80} (3DES units).
- Similarly with QUAD there is a gap between the “recommended” instances and the provably secure instances (i.e., the tested instances were unprovable or unproven [33]).
- PRNGs based on decisional Diffie-Hellman assumption have almost no gap between the hardness of breaking the PRNG and solving the underlying intractable problem, but known primitives based on DDH and exponentiation in Z_p [22, 16] are generally slower than those based on other assumptions.

The generic types of methods for solving polynomial systems — Faugère’s \mathbf{F}_4 - \mathbf{F}_5 and XL-derivatives — are not affected drastically by sparsity. In the former, sparsity is quickly lost and tests show that there is no substantial difference in timing when solving SMP instances. Recent versions of XL [33] speeds up proportionally to sparsity. We therefore surveyed the literature for recent results on solving or attacking specialized systems in crypto, listed below. **These results do not contradict our hardness assumption.**

- Aumasson-Meier (ICISC 2007) [1] shows that in some cases sparsity in primarily *underdefined* — *more variables than equations* — *systems* leads to improved attacks. Results are very interesting and takes more study but do not apply to overdetermined systems in general.
- Bard-Courtois-Jefferson [2] tests SAT solvers on uniformly sparse \mathbb{F}_2 equations, and gives numbers.
- Raddum-Samaev [27, 28] attacks “clumped” systems (even though the title says “sparse”). Similarly the Courtois-Pieprzyk XSL attack [13] requires a lot of structures (i.e., “clumping”).

2 PRNG Based on Specialized Polynomial Map in \mathbb{F}_2

This section both provides a recap of past results and extends them to specialized maps over \mathbb{F}_2 . We will start with definitions and models, then give the key results on the provable security level.

Computational Distinguishability: Probability distributions D_1 and D_2 over a finite set Ω are **computationally distinguishable** with computing resources R and advantage ϵ if there exist a probabilistic algorithm A which on any input $x \in \Omega$ outputs answer 1 (accept) or 0 (reject) using computing resources at most R and satisfies $|\Pr_{x \in D_1}(A(x) = 1) - \Pr_{x \in D_2}(A(x) = 1)| > \epsilon$. The above probabilities are not only taken over x values distributed according to D_1 or D_2 , but also over the random choices that are used by algorithm A . Algorithm A is called a distinguisher with advantage ϵ .

If no such algorithm exists, then we say that D_1 and D_2 are computationally indistinguishable with advantage ϵ . If R is not specified, we implicitly mean feasible computing resources (e.g., $< 2^{80}$ simple operations, and reasonable limits [usually polynomially many] in sampling from D_1 and D_2).

PRNG: Let $n < L$ be two integers and $K = \mathbb{F}_q$ be a finite field. The function $G : K^n \rightarrow K^L$ is said to be a Pseudorandom Number Generator (PRNG) if the probability distribution of the random variable $G(\mathbf{x})$, where the vector \mathbf{x} is uniformly random in K^n , is computationally indistinguishable (with distinguisher resource R) from a uniformly random vector in K^L . *Usually $q = 2$ but it is not required.*

Linear polynomial maps: A linear polynomial map $R : (\mathbb{F}_q)^n \rightarrow (\mathbb{F}_q)^m$ means $R(\mathbf{x}) = \sum_{i=1}^n a_i x_i$, where $\mathbf{x} = (x_1, x_2, \dots, x_n)$, and x_1, x_2, \dots, x_n are variables. If we give these variables values in \mathbb{F}_q , by setting $(x_1, x_2, \dots, x_n) = (b_1, b_2, \dots, b_n)$ for $b_i \in \mathbb{F}_q$, denoted as \mathbf{b} , then $R(\mathbf{b}) = \sum_{i=1}^n a_i b_i$ is an element in \mathbb{F}_q .

In the following sections, a “random” linear polynomial map (or form) has the coefficients a_i ’s randomly chosen from \mathbb{F}_q . Also, when we mention R or $R(\mathbf{x})$ refers to the function but when we write $R(\mathbf{b})$, that means the value of the function R with input vector \mathbf{b} .

Instance from SMP (or MQ): If \mathbf{S} is an instance drawn from $SMP(q, d, n, m, (\eta_2, \dots, \eta_d))$, then $\mathbf{S}(\mathbf{x}) = (P_1(\mathbf{x}), P_2(\mathbf{x}), \dots, P_m(\mathbf{x}))$ ($\mathbf{x} = (x_1, x_2, \dots, x_n)$ are variables) is a function that maps $(\mathbb{F}_q)^n \rightarrow (\mathbb{F}_q)^m$ and each $P_i(\mathbf{x})$ has the same probability distribution as that mentioned in section 1.2. For example, if $\mathbf{b} = (b_1, b_2, \dots, b_n)$ is a vector in $(\mathbb{F}_q)^n$, then $\mathbf{S}(\mathbf{b}) = (P_1(\mathbf{b}), P_2(\mathbf{b}), \dots, P_m(\mathbf{b}))$, a value in $(\mathbb{F}_q)^m$.

Note: Heretofore we will also say $SMP(n, m)$ for short, if no confusion is likely to ensue.

Given any PRNG, there is a standard way to stretch it into an old-fashioned stream cipher (Prop. 1), i.e. stream cipher without IV and key setup. There are ways to set up an initial state securely, such as in Sec. 3.1. Thus we concentrate our efforts on building a PRNG from any MQ family of map from $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$; in order, we need to

1. show that if an instance \mathbf{S} drawn from MQ is *not* a PRNG, then for a (secretly) given vector \mathbf{b} we can predict, with the help of information from the value of \mathbf{S} , $\mathbf{S}(\mathbf{b})$, and any linear form R , the value of $R(\mathbf{b})$ with strictly larger than $1/2 + \epsilon$ probability; then
2. use Goldreich-Levin theorem, which states that the value of any linear function R , $R(\mathbf{b})$ is a hardcore bit of any $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ one-way function \mathbf{S} , $\mathbf{S}(\mathbf{b})$, and R . I.e., *being able to guess with strictly larger than $1/2 + \epsilon$ probability $R(\mathbf{b})$ from \mathbf{S} , $\mathbf{S}(\mathbf{b})$, and R means that we can invert $\mathbf{S}(\mathbf{b})$ with non-negligible probability.*

2.1 From Distinguisher to Predictor

In fact, the following two results are valid for any $K = \mathbb{F}_q$. In [7], the proofs were covered only in the \mathbb{F}_2 case. However, the generalization is nontrivial but straightforward. Therefore, for simplicity, we put the generalized propositions here, though this section is for the \mathbb{F}_2 case.

Proposition 1 ([7]) *Take a stream cipher with $\mathbf{Q} : K^n \rightarrow K^n$ and $\mathbf{P} : K^n \rightarrow K^r$ as the update and output filter functions and random initial state \mathbf{x}_0 , that is, starting from the initial state \mathbf{x}_0 , at each step we update with $\mathbf{x}_{i+1} = \mathbf{Q}(\mathbf{x}_i)$ and output $\mathbf{y}_i = \mathbf{P}(\mathbf{x}_i)$.*

$$\begin{array}{ccccccc} \mathbf{x}_0 & \longrightarrow & \mathbf{x}_1 = \mathbf{Q}(\mathbf{x}_0) & \longrightarrow & \mathbf{x}_2 = \mathbf{Q}(\mathbf{x}_1) & \longrightarrow & \mathbf{x}_3 = \mathbf{Q}(\mathbf{x}_2) \longrightarrow \cdots & \text{(state)} \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & \\ \mathbf{y}_0 = \mathbf{P}(\mathbf{x}_0) & & \mathbf{y}_1 = \mathbf{P}(\mathbf{x}_1) & & \mathbf{y}_2 = \mathbf{P}(\mathbf{x}_2) & & \mathbf{y}_3 = \mathbf{P}(\mathbf{x}_3) \cdots & \text{(output)} \end{array}$$

If we can distinguish between its first λ blocks of output $(\mathbf{y}_0, \mathbf{y}_1, \dots, \mathbf{y}_{\lambda-1})$ and a true random vector in $K^{\lambda r}$ with advantage ϵ in time T , then we can distinguish between the output of a true random vector in K^{n+r} and the output of $\mathbf{S} = (\mathbf{P}, \mathbf{Q})$ in time $T + \lambda T_{\mathbf{S}}$ with advantage ϵ/λ . [Standard Proof is in Appendix A.]

Proposition 2 (an extension of [7]) *Let $K = \mathbb{F}_q$. Suppose there is an algorithm A that given a system $\mathbf{S} : K^n \rightarrow K^m$ chosen from $SMP(q, d, n, m, (\eta_2, \dots, \eta_d))$ distinguishing $\mathbf{S}(U_n)$ from a uniform random distribution U_m , (where U_r means uniform distribution over K^r for the r), with advantage at least ϵ in time T . Then there is an algorithm B that, given (1) a system $\mathbf{S} : K^n \rightarrow K^m$ from $SMP(n, m)$, (2) any $K^n \rightarrow K$ linear form R , and (3) $\mathbf{y} = \mathbf{S}(\mathbf{b})$, where \mathbf{b} is an secret input value randomly chosen from K^n , predicts $R(\mathbf{b})$ with success probability at least $(1 + \epsilon/2)/q$ using at most $T + 2T_{\mathbf{S}}$ operations.*

Proof. Without loss of generality, we may suppose that A has probability at least ϵ higher to return 1 on an input distribution $(\mathbf{S}, \mathbf{S}(U_n))$ than on distribution (\mathbf{S}, U_m) . Define a recentered distinguisher

$$A'(\mathbf{S}, \mathbf{w}) := \begin{cases} A(\mathbf{S}, \mathbf{w}), & \text{probability } \frac{1}{2} \\ 1 - A(\mathbf{S}, \mathbf{u}), & \mathbf{u} \in K^m \text{ uniform random, probability } \frac{1}{2} \end{cases}$$

then A' returns 1 with probability $\frac{1+\epsilon}{2}$ on input $(\mathbf{S}, \mathbf{S}(U_n))$ and with probability $\frac{1}{2}$ on input (\mathbf{S}, U_m) .

Now, given an input \mathbf{S} and $\mathbf{y} \in K^m$, the algorithm B first randomly chooses a value $v \in K$ (representing a guess for $R(\mathbf{b})$), then randomly chooses a vector $\mathbf{u} \in K^m$, and form $\mathbf{S}' := \mathbf{S} + R\mathbf{u} : K^n \rightarrow K^m$. This is equal to S plus a random linear polynomial (see above for the meaning of random linear form) and is hence of $SMP(n, m)$. Define algorithm B as following:

$$B(\mathbf{S}, \mathbf{y}, R) := \begin{cases} v, & \text{if } A'(\mathbf{S}', \mathbf{y} + v\mathbf{u}) = 1; \\ \text{uniformly pick an element from } K \setminus \{v\}, & \text{if } A'(\mathbf{S}', \mathbf{y} + v\mathbf{u}) = 0. \end{cases}$$

If $v = R(\mathbf{b})$, $\mathbf{y} + v\mathbf{u} = \mathbf{S}'(\mathbf{b})$, else $\mathbf{y} + v\mathbf{u}$ is equal to $\mathbf{S}'(\mathbf{b})$ plus a nonzero multiple of the random vector \mathbf{u} , hence is equivalent to being uniformly random. The probability that $B := B(\mathbf{S}, \mathbf{S}(\mathbf{b}), R)$ is the correct guess is hence

$$\begin{aligned} \Pr(B = R(\mathbf{b})) &= \Pr(B = v | v = R(\mathbf{b})) \Pr(v = R(\mathbf{b})) + \Pr(B = R(\mathbf{b}) | v \neq R(\mathbf{b})) \Pr(v \neq R(\mathbf{b})) \\ &= \frac{1}{q} \left(\frac{1}{2} + \frac{\epsilon}{2} \right) + \left(\frac{q-1}{q} \right) \frac{1}{2} \left(\frac{1}{q-1} \right) = \frac{1}{q} \left(1 + \frac{\epsilon}{2} \right). \end{aligned}$$

Note: *We see that the reasoning can work this way if and only if $\mathbf{S}' = \mathbf{S} + R\mathbf{u}$ have the same distribution as \mathbf{S} . Otherwise, we cannot guarantee the distinguisher A' will output the same distribution.*

2.2 Constructing a PRNG from \mathcal{MQ} (\mathbb{F}_2 Case)

Proposition 3 ([25], [7]) *Suppose there is an algorithm B that given a system $\mathbf{S}(\cdot: \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m)$ from $\mathcal{MQ}(2, n, m)$, a random n -bit to one-bit linear form R and the image $\mathbf{S}(\mathbf{b})$ of a randomly chosen unknown \mathbf{b} , predicts $R(\mathbf{b})$ with probability at least $\frac{1}{2} + \epsilon$ over all possible inputs $(\mathbf{S}, \mathbf{S}(\mathbf{b}), R)$ using time T , then there is an algorithm C that given \mathbf{S} and the m -bit image $\mathbf{S}(\mathbf{b})$ of a randomly chosen n -bit vector \mathbf{b} produces a preimage of $\mathbf{S}(\mathbf{b})$ with probability (over all \mathbf{b} and \mathbf{S}) at least $\epsilon/2$ in time*

$$T' = \frac{8n^2}{\epsilon^2} \left(T + \log \left(\frac{8n}{\epsilon^2} \right) + \frac{8n}{\epsilon^2} T_{\mathbf{S}} \right)$$

Note: This is really the Goldreich-Levin theorem of which we omit the proof here. This essentially states that linear forms are hard-core of any one-way function. In fact, the tighter form [7, Proof of Theorem 3] (using a fast Walsh transform) can be simply followed word-for-word.

This above result (which only holds for \mathbb{F}_2) with Prop. 2 shows that any \mathcal{MQ} family of maps induces PRNGs over \mathbb{F}_2 . To get a useful stream cipher, we can combine Props. 1–3:

Proposition 4 ([25], [7]) *If $\mathbf{S} = (\mathbf{P}, \mathbf{Q})$ is an instance drawn from $\mathcal{MQ}(2, n, n+r)$, where $\mathbf{P}: \mathbb{F}_2^n \rightarrow \mathbb{F}_2^r$, $\mathbf{Q}: \mathbb{F}_2^n \rightarrow \mathbb{F}_2^n$ are the stream cipher as in Prop. 1, then if we can distinguish between λ output blocks of the stream cipher from truly random distribution in T time, we can find \mathbf{b} from $\mathbf{S}(\mathbf{b})$, where \mathbf{b} is a randomly chosen input, with probability at least $\frac{\epsilon}{8\lambda}$ in time*

$$T' = \frac{2^7 n^2 \lambda^2}{\epsilon^2} \left(T + (\lambda + 2) T_{\mathbf{S}} + \log \left(\frac{2^7 n \lambda^2}{\epsilon^2} \right) + 2 \right) + \frac{2^7 n \lambda^2}{\epsilon^2} T_{\mathbf{S}} \quad (1)$$

Note: Roughly this means that if we let $r = n$, want to establish a safety level of 2^{80} multiplications, want $L = \lambda r = 2^{40}$ bits between key refreshes, and can accept $\epsilon = 10^{-2}$, then $T' \lesssim 2^{230}/n$. All we need now is to find a map from $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^{2n}$ which takes this amount of time to invert.

As we see below, unless equation-solving improves greatly for sparse systems, this implies that a handful of cubic terms added to a QUAD system with $n = r = 208$, $q = 2$ can be deemed secure to 2^{80} . There is no sense in going any lower than that, because solving a system with n bit-variables can never take much more effort than 2^n times whatever time it takes to evaluate one equation.

3 PRNG Based on SMP in \mathbb{F}_q

In Proposition 3, [7] transformed the problem into a variation of Goldreich-Levin theorem (in \mathbb{F}_2). *The transformation still works in \mathbb{F}_q ; however, Goldreich-Levin theorem gets stuck in this place.* Here we show a way to extend the main results to \mathbb{F}_q , by using a generalization of the Goldreich-Levin hard-core bit theorem.

Proposition 5 ([25] and our contribution) *Let $K = \mathbb{F}_q$. Suppose there is an algorithm B that given a system $\mathbf{S}(\cdot: K^n \rightarrow K^m)$ from $SMP(n, m)$, a random $K^n \rightarrow K$ linear form R and the image $\mathbf{S}(\mathbf{b})$ of a randomly chosen unknown \mathbf{b} , predicts $R(\mathbf{b})$ with probability at least $\frac{1}{q} + \epsilon$ over all possible inputs $(\mathbf{S}, \mathbf{S}(\mathbf{b}), R)$ using time T , then there is an algorithm C that given \mathbf{S} and the m -bit image $\mathbf{S}(\mathbf{b})$ of a randomly chosen vector $\mathbf{b} \in K^n$ produces a preimage of $\mathbf{S}(\mathbf{b})$ with probability (over all \mathbf{b} and \mathbf{S}) at least $\epsilon/2$ in time*

$$T' \leq 2^{10} \left(\frac{nq}{\epsilon^5} \right) \log^2 \left(\frac{n}{\epsilon} \right) T + \left(1 - \frac{1}{q} \right)^2 \epsilon^{-2} T_{\mathbf{S}}$$

Remark [intuition of why argument in \mathbb{F}_2 cannot be applied in \mathbb{F}_q]: If we know that one out of two exclusive possibilities takes place with probability strictly larger than 50%, then the other one must happen strictly less often 50%. If we know that one of q possibilities takes place with probability strictly greater than $1/q$, we cannot be sure that another possibility does not occur with even higher possibility. Therefore, we can only treat this as a case of learning a linear functional with queries to a highly noisy oracle. **Due to this difference, the order of ϵ in T'/T is as high as ϵ^{-5} in Prop. 5, but only ϵ^{-2} in Prop. 3.**

Proposition 6 ([25] and our contribution) *If $\mathbf{S} = (\mathbf{P}, \mathbf{Q})$ is an instance drawn from a $SMP(n, n+r)$, where $\mathbf{P} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^r$, $\mathbf{Q} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^n$ are the stream cipher as in Prop. 1, then if we can distinguish between λ output blocks of the stream cipher and uniform distribution in T time, we can invert $\mathbf{S}(\mathbf{b})$ with probability at least $\frac{\epsilon}{4q\lambda}$ in time*

$$T' = 2^{15} \frac{nq^6\lambda^5}{\epsilon^5} \log^2 \left(\frac{2qn\lambda}{\epsilon} \right) (T + (\lambda + 2)T_{\mathbf{S}}) + \left(1 - \frac{1}{q}\right)^2 \frac{4q^2\lambda^2}{\epsilon^2} T_{\mathbf{S}} \quad (2)$$

This is a straightforward combination of Props. 1, 2, and 5. In the remainder of this section, we give a proof to Prop. 5 by a variation of the procedure used by Goldreich-Rubinfeld-Sudan [21, Secs. 2 and 4], to give it concrete values that we can derive security proofs from.

3.1 Conversion to a Modern (IV-Dependent Stream) Cipher

This paper mostly deals with the security of PRNGs, which are essentially old-fashioned stream ciphers. If we have a secure PRNG $\mathbf{S}' = (\mathbf{s}_0, \mathbf{s}_1)$, where both $\mathbf{s}_0, \mathbf{s}_1$ are maps from $K^n \rightarrow K^n$ — note that \mathbf{S}' can be identical to \mathbf{S} — then the following is a standard way to derive the initial state $\mathbf{x}_0 \in K^n$ from the bitstream (key) $\mathbf{c} = (c_1, c_2, \dots, c_{KL}) \in \{0, 1\}^{KL}$ and an initial $\mathbf{u} \in K^n$, where KL is the length of the key:

$$\mathbf{x}_0 := \mathbf{s}_{c_{KL}}(\mathbf{s}_{c_{KL-1}}(\dots(\mathbf{s}_{c_2}(\mathbf{s}_{c_1}(\mathbf{u})))\dots)).$$

This is known as the tree-based construction. From an old-fashioned provably secure stream cipher (i.e., the key is the initial state), the above construction achieves security in the resulting IV-dependent stream cipher, at the cost of some additional looseness in the security proof. A recent example of this is [6].

Thus, all our work really applies to the modern type of stream ciphers which require an IV-dependent setup, except that the security parameters may be slightly different.

3.2 Hardcore Predicate and Learning Polynomials

Let $\mathbf{x} = (x_1, x_2, \dots, x_n)$, $\mathbf{b} = (b_1, b_2, \dots, b_n)$, and x_i, b_i are elements in a finite field $K = \mathbb{F}_q$. Given an arbitrary strong one way function $h(\mathbf{x})$, then $F(\mathbf{x}, \mathbf{b}) = (h(\mathbf{x}), \mathbf{b})$ is also a one way function. Claim $\mathbf{x} \cdot \mathbf{b}$ is the hard-core bit of $F(\mathbf{x}, \mathbf{b})$, where $\mathbf{x} \cdot \mathbf{b}$ means their inner product.

Supposed we have a predictor P which predicts its hardcore $\mathbf{x} \cdot \mathbf{b}$ given $(h(\mathbf{x}), \mathbf{b})$ with probability more than $\frac{1}{q} + \epsilon$, then we can write in the math form:

$$\Pr_{\mathbf{b}, \mathbf{x}} [P(h(\mathbf{x}), \mathbf{b}) = \mathbf{x} \cdot \mathbf{b}] > \frac{1}{q} + \epsilon.$$

By Markov inequality, we know there must be more than $\epsilon/2$ fraction of x such that $\Pr_{\mathbf{b}} [P(h(\mathbf{x}), \mathbf{b}) = \mathbf{x} \cdot \mathbf{b}] > \frac{1}{q} + \frac{\epsilon}{2}$. For this fraction of x , we are trying to find the inverse of $h(\mathbf{x})$ ($F(\mathbf{x})$ as well) through the predictor. Also $\mathbf{x} \cdot \mathbf{b}$ can be written as $\sum b_i x_i$, then

$$\Pr_{\mathbf{b}} \left[P(h(\mathbf{x}), \mathbf{b}) = \sum b_i x_i \right] > \frac{1}{q} + \frac{\epsilon}{2}.$$

What this means in English is that, *if we can find a polynomial which almost matches an arbitrary function P , a predictor function, then we can eventually invert \mathbf{x} from $F(\mathbf{x})$ a non-negligible portion of the time.* Now we try to reconstruct such linear polynomials through the access of the predictor, largely following the footsteps of [21].

3.3 Intuition of Reconstructing Linear Polynomials

Now we are given some oracle accesses to a function $f : K^n \rightarrow K$, where K is a finite field and $|K| = q$. We need to find all linear polynomials which match f with at least $\frac{1}{q} + \epsilon$ fraction of inputs x . Let $p(x_1, x_2, \dots, x_n) = \sum_1^n p_i x_i$, and i -th prefix of p is $\sum_1^i p_j x_j$. The algorithm runs n rounds, and in the i -th round, it extends all possible candidates from the $(i-1)$ -th round with all elements in K and screens them, filtering out most bad prefixes. The pseudocode of the algorithm is presented in Algorithm 2. Since we want the algorithm to be efficient, we must efficiently screen possible prefixes from all extensions. We now introduce a screening algorithm to be called TestPrefix.

Algorithm 1 TestPrefix($f, \epsilon, n, (c_1, c_2, \dots, c_i)$)[25]

```

Repeat  $\text{poly}_1(\frac{n}{\epsilon})$  times:
Pick  $\mathbf{s} = s_{i+1}, \dots, s_n \in_R \text{GF}(q)$ 
Let  $t = \text{poly}_2(\frac{n}{\epsilon})$ 
for  $k = 1$  to  $t$  do
    Pick  $\mathbf{r} = r_1, r_2, \dots, r_i \in_R \text{GF}(q)$ 
     $\sigma^{(k)} = f(\mathbf{r}, \mathbf{s}) - \sum_{j=1}^i c_j r_j$ 
end for
If there is  $\sigma^{(k)} = \sigma$  for at least  $\frac{1}{q} + \frac{\epsilon}{3}$  fraction of the  $k$ 's then output accept and halt
endRepeat
If all iterations were completed without accepting, then reject

```

Algorithm 2 Find All Polynomials(f, ϵ)[25]

```

set a candidate queue  $Q[i]$  which stores all the candidates  $(c_1, c_2, c_3, \dots, c_i)$  in the  $i$ -th round
for  $i = 1$  to  $n$  do
    Pick all elements in  $Q[i]$ 
    TestPrefix( $f, \epsilon, n, (c_1, c_2, \dots, c_i, \alpha)$ ) for all  $\alpha \in F$ 
    If TestPrefix accepts, then push  $(c_1, c_2, \dots, c_i, \alpha)$  into  $Q[i+1]$  i.e. it is a candidate in the  $(i+1)$ -th round
end for

```

Supposed we are testing the i -th prefix (c_1, c_2, \dots, c_i) , we are going to evaluate the quantity of:

$$P_{\mathbf{s}}(\sigma) := \Pr_{r_1, r_2, \dots, r_i \in K} \left[f(\mathbf{r}, \mathbf{s}) = \sum_{j=1}^i c_j r_j + \sigma \right]$$

where $\mathbf{r} = (r_1, r_2, \dots, r_i)$. The value of σ can be thought as a guess of $\sum_{i+1}^n p_j s_j$. For every \mathbf{s} , we can estimate the probability by a sample of several \mathbf{r} 's, and the error rate can be controlled by the times of sampling. If such \mathbf{s} makes the probability significantly larger than $1/q$, then we accept. If no such \mathbf{s} exists, we reject. The detailed algorithm is stated in the Algorithm 1: TestPrefix.

If a candidate (c_1, c_2, \dots, c_i) passes through the Algorithm 1 for at least one suffix \mathbf{s} , there is a σ such that the estimate of $P_{\mathbf{s}}(\sigma)$ is greater than $\frac{1}{q} + \frac{\epsilon}{3}$. For a correct candidate (c_1, c_2, \dots, c_i) , i.e. (c_1, c_2, \dots, c_i) is the prefix of $p = (p_1, p_2, \dots, p_n)$ which matches f for at least $\frac{1}{q} + \epsilon$, and an arbitrary $\sigma = \sum_{i+1}^n p_j s_j$, it satisfies that $E_{\mathbf{s}}[P_{\mathbf{s}}(\sigma)] \geq \frac{1}{q} + \epsilon$. By Markov's inequality, for at least $\epsilon/2$ fraction of \mathbf{s} and some corresponding σ , it holds that $P_{\mathbf{s}}(\sigma) \geq \frac{1}{q} + \frac{\epsilon}{2}$. In Algorithm 1, we set $\frac{1}{q} + \frac{\epsilon}{3}$ as the passing criteria; thus the correct candidate will pass through the Algorithm 1 with great probability. However, [21, Sec. 4] shows that the total passing number of candidates in each round is limited. In fact, only a small number of candidates will pass the test. This maximum (also given by [21, Sec. 4]) number of prefixes that pass the test is $\leq (1 - \frac{1}{q})^2 \epsilon^{-2}$.

3.4 Giving Concrete Values to “Order of Polynomially Many”

Since there are $\epsilon/2$ fraction of suffix \mathbf{s} such that $P_s(\sigma) \geq \frac{1}{q} + \frac{\epsilon}{2}$, we can randomly choose the suffix polynomially many times (k_1 times) to ensure that we would select such \mathbf{s} with high probability. Also, for such \mathbf{s} , if we choose polynomially many times (k_2 times) of \mathbf{r} , there would be high probability that we would find some α for at least $\frac{1}{q} + \frac{\epsilon}{3}$ fraction. We are estimating how the polynomially many should be as the following:

$$\begin{aligned} & \Pr [\text{TestPrefix fails}] \leq \\ & \Pr [\text{no such } \mathbf{s} \text{ is chosen}] + \Pr \left[\text{no single element exists more than } \frac{1}{q} + \frac{\epsilon}{3} \text{ fraction} \right] \\ & \Pr [\text{no such } \mathbf{s} \text{ is chosen}] \leq (1 - \epsilon/2)^{k_1} \leq e^{-\frac{k_1 \epsilon}{2}} \leq \frac{1}{2} \frac{\epsilon}{\left(1 - \frac{1}{q}\right)^2 \epsilon^{-2} nq} \end{aligned}$$

So, we take k_1 as $O(\frac{1}{\epsilon} \log(\frac{n}{\epsilon})) \approx 3\frac{1}{\epsilon} \log(\frac{n}{\epsilon})$. On the other hand, we want to estimate the probability of there are no σ 's with fraction at least $\frac{1}{q} + \frac{\epsilon}{3}$. For a correct suffix \mathbf{s} , we know for uniform \mathbf{r} , we get that σ with probability more than $\frac{1}{q} + \frac{\epsilon}{2}$. Let X_i be the random variable with value 1 if the i -th trial of \mathbf{r} gets the correct σ , 0 otherwise. Then we have $\Pr[X_i = 1] \geq \frac{1}{q} + \frac{\epsilon}{2}$. Suppose we do k_2 trials:

$$\begin{aligned} \Pr \left[\text{no single element exists more than } \frac{1}{q} + \frac{\epsilon}{3} \text{ fraction} \right] & \leq \Pr \left[\sum_1^{k_2} X_i < \left(\frac{1}{q} + \frac{\epsilon}{3}\right)k_2 \right] \\ & \leq \Pr \left[\left| \frac{\sum_{i=1}^{k_2} X_i}{k_2} - \left(\frac{1}{q} + \frac{\epsilon}{2}\right) \right| \geq \frac{\epsilon}{6} \right], \end{aligned}$$

since these X_i 's are independent, then by Chernoff's bound we have

$$\Pr \left[\left| \frac{\sum_{i=1}^{k_2} X_i}{t} - \left(\frac{1}{q} + \frac{\epsilon}{2}\right) \right| \geq \frac{\epsilon}{6} \right] \leq 2e^{-\frac{k_2 \epsilon^2}{2 \times 36}} \leq \frac{1}{2} \frac{\epsilon}{\left(1 - \frac{1}{q}\right)^2 \epsilon^{-2} nq},$$

$k_2 = O(\frac{\log(n/\epsilon)}{\epsilon^2}) \approx 216 \frac{\log(n/\epsilon)}{\epsilon^2}$ is sufficient to make the inequality hold. Thus, we have

$$\Pr [\text{TestPrefix fails}] \leq \frac{\epsilon}{\left(1 - \frac{1}{q}\right)^2 \epsilon^{-2} nq}.$$

Also, $\Pr [\text{Algorithm 2 fails}] \leq \Pr [\text{one TestPrefix fails}] \leq \sum_{\text{all TestPrefix run}} \Pr [\text{TestPrefix fails}]$

$$\leq \left(\left(1 - \frac{1}{q}\right)^2 \epsilon^{-2} nq \right) \frac{\epsilon}{\left(1 - \frac{1}{q}\right)^2 \epsilon^{-2} nq} = \epsilon$$

Therefore, the algorithm will work with high probability. The worst case running time of algorithm 2 should be: $k_1 k_2 \left(1 - \frac{1}{q}\right)^2 \frac{1}{\epsilon^2} nq = O\left(\frac{n}{\epsilon^3} \log^2\left(\frac{n}{\epsilon}\right)\right) \lesssim 2^{10} \left(\frac{nq}{\epsilon^3}\right) \log^2\left(\frac{n}{\epsilon}\right)$.

Note: $\left(1 - \frac{1}{q}\right)^2 \epsilon^{-2}$ is the maximum number of candidates which pass in each round.

4 On SMP under Generic Solvers

To verify that SMP (and SRQ of Appendix C) represent one way property, we need to show that

1. generic system-solvers do not run substantially faster on them; and
2. there are no specialized solvers that can take advantage of the sparsity.

Here “generic” means the ability to handle any multivariate polynomial system with n variables and m equations in \mathbb{F}_q . There are two well-known types of generic methods for solving polynomial systems, both related to the original Buchberger’s algorithm. One is Faugère’s \mathbf{F}_4 - \mathbf{F}_5 and the other is XL-derivatives. In the former, sparsity is quickly lost and tests show that there are little difference in timing when solving SMP (or SRQ) instances. With recent versions of XL [33], the sparsity results in a proportional decrease in complexity. The effect of sparsity on such generic methods should be predictable and not very drastic, as shown by some testing (cf. Sec. 4.1). We briefly describe what is known about XL and \mathbf{F}_4 - \mathbf{F}_5 in Appendix B.

4.1 Testing the One-Wayness with Generic Solvers

We conducted numerous tests on SMP maps at various degrees and sparsity over the fields \mathbb{F}_2 , \mathbb{F}_{16} , and \mathbb{F}_{256} . For example, Table 1 lists our tests in solving random $\mathcal{MQ}(256, n, m)$ instances where each polynomial only has n quadratic terms [we call these instances $S\mathcal{MQ}(256, n, m, n)$] with \mathbf{F}_4 over $\text{GF}(256)$. It takes almost the same time as solving an \mathcal{MQ} instance of the same size.

$m - n$	D_{XL}	D_{reg}	$n = 9$	$n = 10$	$n = 11$	$n = 12$	$n = 13$
0	2^m	m	6.03	46.69	350.38	3322.21	sigmem
1	m	$\lceil \frac{m+1}{2} \rceil$	1.19	8.91	53.64	413.34	2535.32
2	$\lceil \frac{m+1}{2} \rceil$	$\lceil \frac{m+2-\sqrt{m+2}}{2} \rceil$	0.31	2.20	12.40	88.09	436.10

Table 1. $S\mathcal{MQ}(256, n, m, n)$ timing (sec): MAGMA 2.12, 2GB RAM, Athlon64x2 2.2GHz

For XL variants that use sparse solvers as the last step [33] test results (one of which is shown in Table 2) confirms the natural guess: For SMP instances where the number of non-linear terms is not overly small, the solution degree of XL is unchanged, and the speed naturally goes down as the number of terms, nearly in direct proportion (in Tab. 2, should be close to $n/4$).

n	7	8	9	10	11	12	13
D	5	6	6	7	7	8	8
$S\mathcal{MQ}(256, n, n + 2, n)$	$9.34 \cdot 10^{-2}$	$1.17 \cdot 10^0$	$4.04 \cdot 10^0$	$6.02 \cdot 10^1$	$1.51 \cdot 10^2$	$2.34 \cdot 10^3$	$5.97 \cdot 10^3$
$\mathcal{MQ}(256, n, n + 2)$	$2.06 \cdot 10^{-1}$	$2.92 \cdot 10^0$	$1.10 \cdot 10$	$1.81 \cdot 10^2$	$4.94 \cdot 10^2$	$8.20 \cdot 10^3$	$2.22 \cdot 10^4$
ratio	2.20	2.49	2.73	3.00	3.27	3.50	3.72

Table 2. XL/Wiedemann timing (sec) on Core2Quad 2.4GHz, icc, 4-thread OpenMP, 8GB RAM

For \mathbb{F}_2 , there are many special optimizations made for \mathbf{F}_4 in MAGMA, so we ran tests at various densities of quadratic terms in version 2.12-20 and 2.13-8. Typical results are given in Fig. 1 (cf. Appendix, samples labelled “sparse non-random” are SRQ tests). Most of the time the data points are close to each other. In some tests they overlap each other so closely that no difference in the timing is seen in a diagram.

4.2 A Brief Discussion on Specialization and Security

Since generic system-solvers show no unexpected improvement on our specializations, it remains for us to check that there are no other big improvements in solving specialized systems for. We list below what we

know of recent new attempts on solving or attacking specialized systems in crypto, and show that *our results are consistent with these new results and somewhat complements them.*

- Aumasson-Meier [1] presented several ideas to attack primitives built on sparse polynomial systems, which we sketch separately in Sec. 4.3 below.
 - Raddum-Samaev [27, 28] attacks what they term “sparse” systems, where each equation depend on a small number of variables. Essentially, the authors state that for systems of equations in n bit variables such that each equation depends on only k variables, we can solve the system in time roughly proportional to $2^{(1-\frac{1}{k})n}$ using a relatively small memory footprint. Since XL for cubics and higher degrees over \mathbb{F}_2 is more time-consuming than brute-force, this is fairly impressive. However, the “sparsity” defined by the authors is closer to “input locality” and very different from what people usually denote with this term. The attack is hence not applicable to *SMP*-based stream ciphers.
- In a similar vein is the purported XSL attack on AES [13]. While the S was supposed to stand for Sparse, it really requires Structure – i.e., each equation depending on very few variables. So, whether that attack actually works or not, it does not apply to *SMP*-based systems.
- Bard-Courtois-Jefferson [2] use SAT solvers on uniformly sparse \mathbb{F}_2 equations and give experimental numbers. According to the authors, the methods takes up much less memory than \mathbf{F}_4 or derivatives, but is slower than these traditional methods when they have enough memory.

Some numbers for *very* overdefined and *very* sparse systems shows that converting to a conjunctive normal form and then running a SAT solver can have good results. This seems to be a very intriguing approach, but so far there are no theoretical analysis especially for when the number of equations is a few times the number of variables, which is the case for *SMP* (or SRQ) constructions.

4.3 Solutions and Collisions in Sparse Polynomial Systems

Aumasson-Meier recent published [1] some quite interesting ideas on finding solutions or collisions for primitives using sparse polynomial systems (e.g., hashes proposed in [15]).

They showed that which implies that using sparse polynomials systems of uniform density (*in every degree*) for Merkle-Damgård compression will not be universally collision-free. Some *underdefined* systems that are sparse in the higher degrees can be solved with lower complexity. Their results do not apply to overdetermined systems in general. We summarize relevant results below.

1. Overdetermined higher-degree maps that are sparse of uniform density, or at least sparse in the linear terms, is shown to have high probability of trivial collisions and near-collisions.
It seems that everyone agrees, that linear terms should be totally random when constructing sparse polynomial systems for symmetric primitives.
2. Suppose we have an *underdetermined* higher-degree map sparse in the non-affine part, i.e.,

$$\mathbf{P} : \mathbb{F}_2^{n+r} \rightarrow \mathbb{F}_2^n, \mathbf{P}(\mathbf{x}) = \mathbf{b} + M\mathbf{x} + \mathbf{Q}(\mathbf{x})$$

where \mathbf{Q} has only quadratic or higher terms and is sparse. Aumasson-Meier suggests that we can find $\mathbf{P}^{-1}(\mathbf{y})$ as follows: find a basis for the kernel space of the augmented matrix $[M; \mathbf{b} + \mathbf{y}]$. Collect these basis vectors in a $(n+r+1) \times (r+1)$ matrix M' as a linear code. For an arbitrary $\mathbf{w} \in \mathbb{F}_2^{r+1}$, the codeword $\bar{\mathbf{x}} = M'\mathbf{w}$ will represent a solution to $\mathbf{y} = M\mathbf{x} + \mathbf{b}$ if its last component is 1. Use known methods to find relatively low-weight codewords for the code M' and substitute into $\mathbf{Q}(\mathbf{x})$, expecting it to vanish with non-negligible probability.

Aumasson-Meier proposes to apply this for collisions in Merkle-Damgård hashes with cubic compressor functions. It *does not work* for fields other than \mathbb{F}_2 or *overdetermined* systems. Its exact complexity is unknown and requires some further work.

3. Conversely, it has been suggested if we have an *overdetermined* higher-degree map

$$\mathbf{P} : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^{n+r}, \mathbf{P}(\mathbf{x}) = \mathbf{b} + M\mathbf{x} + \mathbf{Q}(\mathbf{x})$$

where \mathbf{Q} has only quadratic or higher terms and is *extremely* sparse, we can consider $\mathbf{P}(\mathbf{x}) = \mathbf{y}$ as $\mathbf{M}\mathbf{x} = (\mathbf{y}+\mathbf{b}) + \mathbf{perturbation}$, and use known methods for decoding attacks, i.e., solving overdetermined linear equations with perturbation. However, $SM\mathcal{P}$ maps with a moderate number of quadratic terms will be intractible.

We note that other specialized polynomials can be constructed that are also easier to evaluate such as the SRQ construction (cf. Appendix C) which also can carry through the same arguments as $SM\mathcal{P}$, so our process is more general than it looks.

5 Summary of Uses for Specialized Polynomial Systems

All information seems to point to the conclusion that we always use totally random linear terms, no matter what else we do. With that taken into account, specialized random systems (such as $SM\mathcal{P}$) represent improvements over generic systems in terms of storage and (likely) speed.

5.1 The Secure Stream Ciphers SPELT

We build a stream cipher called $SPELT(q, d, n, r, (\eta_2, \dots, \eta_d))$, which resembles the construction in section 2:

We specify a prime power q (usually a power of 2), positive integers n and r , a degree d . We have “update function” $\mathbf{Q}_i = (Q_{i,1}, Q_{i,2}, \dots, Q_{i,n}) : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^n$ and “output filter” $\mathbf{P}_i = (P_{i,1}, P_{i,2}, \dots, P_{i,r}) : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^r$, for $i \in \{0, 1\}$. We still do $\mathbf{y}_n = \mathbf{P}(\mathbf{x}_n)$ [output]; $\mathbf{x}_{n+1} = \mathbf{Q}(\mathbf{x}_n)$ [transition], iterated according to the initial vector. To repeat, every polynomial here is of degree d . Affine (constant and linear) term or coefficient are still uniformly random. But terms of each degree are selected according to different densities of terms, such that **the degree- i terms are sparse to the point of having only η_i terms.** *The difference between Eq. 1 and Eq. 2, which governs the maximum provable security levels we can get, affects our parameter choices quite a bit, as seen below.*

By Eq. 2, if $L = \lambda n \lg q$ is the desired keystream length, the looseness factor T'/T is roughly

$$\frac{2^{15} q^6 (L/\epsilon)^5}{n^4 \lg^5 q} \lg^2 \left(\frac{2qL}{\epsilon \lg q} \right)$$

If we let $q = 16$, $r = n$, want a safety level of $T = 2^{80}$ multiplications, $L = 2^{40}$ bits between key refreshes, and can accept $\epsilon = 10^{-2}$, then $T' \lesssim 2^{354}/n^4$. We propose the following instances:

- SPELT using $q = 16$, $d = 3$ (cubics), $n = r = 160$, 20 quadratic and 15 cubic terms per equation. Projected XL degree is 54, storage requirement is 2^{184} bytes. T' is about 2^{346} multiplications, which guarantees $\gtrsim 2^{88}$ multiplications security. This runs at 6875 cycles/byte.
- SPELT using $d = 4$ (quartics), $n = r = 108$, 20 quadratic, 15 cubic, and 10 quartic terms per equation. Projected XL degree is 65, storage requirement is 2^{174} bytes. T' is about 2^{339} multiplications guaranteeing $\gtrsim 2^{81}$ multiplications security at a preliminary 5541 cycles/byte.
- SPELT using $q = 2$, $n = r = 208$, $d = 3$ (cubics), with 20 cubic terms each equation. Preliminary tests achieve 11744 cycles/byte. The expected complexity for solving 208 variables and 416 equations is $\sim 2^{224}$ (by brute-force trials, which is much faster than XL here), which translates to a 2^{82} proven security level.

5.2 Comparisons: A Case for SPELT

All modern-day microprocessor are capable of doing 64-bit arithmetic at least, and there is a natural way to implement QUAD that runs very fast over \mathbb{F}_2 , limited only by the ability to stream data. However, as number of variables goes up, the storage needed for QUAD goes up cubically, and for parameter choices that are secure, the dataset overflows even the massive caches of an Intel Core 2. That is what slows down $QUAD(2, 320, 320)$ — tests on a borrowed ia64 server shows that it is almost exactly the same speed as the $SPELT(2, 3, 208, 208, [480, 20])$. Looking at the numbers, it seems that the idea of specialized polynomials

Stream Cipher	Block	Storage	Cycles/Byte	Security Level
SPELT (2,3,208,208,[480,20])	208b	0.43 MB	11744	2^{82} Proven
SPELT (16,4,108,108,[20,15,10])	864b	48 kB	5541	2^{80} Proven
QUAD (2,320,320)	320b	3.92 MB	13646	2^{82} Proven
QUAD (2,160,160)	160b	0.98 MB	2081	2^{140} Best Attack
SPELT (16,4,32,32,[10,8,5])	128b	8.6 kB	1244	2^{152} Best Attack

Table 3. Point-by-Point, SPELT vs. QUAD on a K8 or C2

is a good complement to the approach of using polynomial maps for symmetric primitives introduced by Berbain-Gilbert-Patarin.

We hasten to add that our programming is quite primitive, and may not match the more polished implementations (e.g., [5]). We are still working to improve our programming and parameter choices. Also, in hardware implementations, the power of sparsity should be even more pronounced.

5.3 For Possible Use in Hashes

In [8] Billet *et al* proposes to use two-staged constructions with a random 192-bit to 464-bit expanding quadratic map followed by a 464-bit to 384-bit quadratic contraction. They show that in general a PRNG followed by a one-way compression function is a one-way function.

In [15] the same construction is proposed but with SRQ quadratics (see Appendix C) and no proof. Now we see that the abovementioned results from [8] and Prop. 6, which justify the design up to a point. This is an area that still takes some study, and perhaps require extra ideas, such as having a hybrid construction with a sparse polynomial expansion stage and a different kind of contraction stage.

References

1. J.-P. Aumasson and W. Meier. Analysis of multivariate hash functions. In K.-H. Nam and G. Rhee, editors, *ICISC*, volume 4817 of *Lecture Notes in Computer Science*, pages 309–323. Springer, 2007.
2. G. V. Bard, N. T. Courtois, and C. Jefferson. Efficient methods for conversion and solution of sparse systems of low-degree multivariate polynomials over $gf(2)$ via sat-solvers. Cryptology ePrint Archive, Report 2007/024, 2007. <http://eprint.iacr.org/>.
3. M. Bardet, J.-C. Faugère, and B. Salvy. On the complexity of Gröbner basis computation of semi-regular overdetermined algebraic equations. In *Proceedings of the International Conference on Polynomial System Solving*, pages 71–74, 2004. Previously INRIA report RR-5049.
4. M. Bardet, J.-C. Faugère, B. Salvy, and B.-Y. Yang. Asymptotic expansion of the degree of regularity for semi-regular systems of equations. In P. Gianni, editor, *MEGA 2005 Sardinia (Italy)*, 2005.
5. C. Berbain, O. Billet, and H. Gilbert. Efficient implementations of multivariate quadratic systems. In E. Biham and A. M. Youssef, editors, *Selected Areas in Cryptography*, volume 4356 of *Lecture Notes in Computer Science*, pages 174–187. Springer, 2007.
6. C. Berbain and H. Gilbert. On the security of IV dependent stream ciphers. In Biryukov [9], pages 254–273.
7. C. Berbain, H. Gilbert, and J. Patarin. QUAD: A practical stream cipher with provable security. In S. Vaudenay, editor, *EUROCRYPT*, volume 4004 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2006.
8. O. Billet, M. J. B. Robshaw, and T. Peyrin. On building hash functions from multivariate quadratic equations. In J. Pieprzyk, H. Ghodosi, and E. Dawson, editors, *ACISP*, volume 4586 of *Lecture Notes in Computer Science*, pages 82–95. Springer, 2007.
9. A. Biryukov, editor. *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*. Springer, 2007.
10. L. Blum, M. Blum, and M. Shub. Comparison of two pseudo-random number generators. In R. L. Rivest, A. Sherman, and D. Chaum, editors, *CRYPTO'82*, pages 61–78, New York, 1983. Plenum Press.
11. B. Buchberger. *Ein Algorithmus zum Auffinden der Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal*. PhD thesis, Innsbruck, 1965.

12. N. T. Courtois, A. Klimov, J. Patarin, and A. Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Bart Preneel, ed., Springer, 2000. Extended Version: <http://www.minrank.org/xlfull.pdf>.
13. N. T. Courtois and J. Pieprzyk. Cryptanalysis of block ciphers with overdefined systems of equations. In *Advances in Cryptology — ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 267–287. Yuliang Zheng, ed., Springer, 2002.
14. C. Diem. The XL-algorithm and a conjecture from commutative algebra. In *Advances in Cryptology — ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 323–337. Pil Joong Lee, ed., Springer, 2004. ISBN 3-540-23975-8.
15. J. Ding and B.-Y. Yang. Multivariate polynomials for hashing. In *Inscrypt*, Lecture Notes in Computer Science. Springer, 2007. to appear, cf. <http://eprint.iacr.org/2007/137>.
16. R. R. Farashahi, B. Schoenmakers, and A. Sidorenko. Efficient pseudorandom generators based on the ddd assumption. In *Public Key Cryptography*, pages 426–441, 2007.
17. J.-C. Faugère. A new efficient algorithm for computing Gröbner bases (F_4). *Journal of Pure and Applied Algebra*, 139:61–88, June 1999.
18. J.-C. Faugère. A new efficient algorithm for computing Gröbner bases without reduction to zero (F_5). In *International Symposium on Symbolic and Algebraic Computation — ISSAC 2002*, pages 75–83. ACM Press, July 2002.
19. M. R. Garey and D. S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979. ISBN 0-7167-1044-7 or 0-7167-1045-5.
20. R. Gennaro. An improved pseudo-random generator based on the discrete logarithm problem. *Journal of Cryptology*, 18:91–110, 2000.
21. O. Goldreich, R. Rubinfeld, and M. Sudan. Learning polynomials with queries: The highly noisy case. *SIAM Journal on Discrete Mathematics*, 13(4):535–570, Nov. 2000.
22. S. Jiang. Efficient primitives from exponentiation in \mathbb{Z}_p . In L. M. Batten and R. Safavi-Naini, editors, *ACISP*, volume 4058 of *Lecture Notes in Computer Science*, pages 259–270. Springer, 2006.
23. N. Kobitz and A. Menezes. Another look at "provable security" (part 2). In R. Barua and T. Lange, editors, *INDOCRYPT*, volume 4329 of *Lecture Notes in Computer Science*, pages 148–175. Springer, 2006.
24. D. Lazard. Gröbner-bases, Gaussian elimination and resolution of systems of algebraic equations. In *EUROCAL 83*, volume 162 of *Lecture Notes in Computer Science*, pages 146–156. Springer, March 1983.
25. L. Levin and O. Goldreich. A hard-core predicate for all one-way functions. In D. S. Johnson, editor, *21th ACM Symposium on the Theory of Computing — STOC'89*, pages 25–32. ACM Press, 1989.
26. T. Matsumoto and H. Imai. Public quadratic polynomial-tuples for efficient signature verification and message-encryption. In *Advances in Cryptology — EUROCRYPT 1988*, volume 330 of *Lecture Notes in Computer Science*, pages 419–545. Christoph G. Günther, ed., Springer, 1988.
27. H. Raddum and I. Semaev. New technique for solving sparse equation systems. Cryptology ePrint Archive, Report 2006/475, 2006. <http://eprint.iacr.org/>.
28. I. Semaev. On solving sparse algebraic equations over finite fields (part ii). Cryptology ePrint Archive, Report 2007/280, 2007. <http://eprint.iacr.org/>.
29. R. Steinfeld, J. Pieprzyk, and H. Wang. On the provable security of an efficient rsa-based pseudorandom generator. In X. Lai and K. Chen, editors, *ASIACRYPT*, volume 4284 of *Lecture Notes in Computer Science*, pages 194–209. Springer, 2006. formerly ePrint 2006/206.
30. C. Wolf. *Multivariate Quadratic Polynomials in Public Key Cryptography*. PhD thesis, Katholieke Universiteit Leuven, 2005. <http://eprint.iacr.org/2005/393>.
31. B.-Y. Yang and J.-M. Chen. All in the XL family: Theory and practice. In *ICISC 2004*, volume 3506 of *Lecture Notes in Computer Science*, pages 67–86. Springer, 2004.
32. B.-Y. Yang and J.-M. Chen. Theoretical analysis of XL over small fields. In *ACISP 2004*, volume 3108 of *Lecture Notes in Computer Science*, pages 277–288. Springer, 2004.
33. B.-Y. Yang, O. C.-H. Chen, D. J. Bernstein, and J.-M. Chen. Analysis of QUAD. In Biryukov [9], pages 290–307.

A Proof of Prop. 1

Proof. We introduce hybrid probability distributions $D_i(\mathbf{S})$ over K^L ($L := \lambda r$):

For $0 \leq i \leq \lambda$ respectively associate with the random variables

$$t^i(\mathbf{S}, \mathbf{x}) := (\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_i, \mathbf{P}(\mathbf{x}), \mathbf{P}(\mathbf{Q}(\mathbf{x})), \dots, \mathbf{P}(\mathbf{Q}^{\lambda-i-1}(\mathbf{x})))$$

where the \mathbf{w}_j and \mathbf{x} are random independent uniformly distributed vectors in K^n and we use the notational conventions that $(\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_i)$ is the null string if $i = 0$, and that

$$(\mathbf{P}(\mathbf{x}), \mathbf{P}(\mathbf{Q}(\mathbf{x})), \dots, \mathbf{P}(\mathbf{Q}^{\lambda-i-1}(\mathbf{x})))$$

is the null string if $i = \lambda$. Consequently $D_0(\mathbf{S})$ is the distribution of the L -unit keystream and $D_\lambda(\mathbf{S})$ is the uniform distribution over K^L . We denote by $p_i(\mathbf{S})$ the probability that A accepts a random L -long sequence distributed according to $D_i(\mathbf{S})$, and p_i the mean value of $p_i(\mathbf{S})$ over the space of sparse polynomial systems \mathbf{S} . We have supposed that algorithm A distinguishes between $D_0(\mathbf{S})$ and $D_\lambda(\mathbf{S})$ with advantage ϵ , in other words that $|p_0 - p_\lambda| \geq \epsilon$.

Algorithm B works thus: on input $(\mathbf{x}_1, \mathbf{x}_2) \in K^{n+r}$ with $\mathbf{x}_1 \in K^r$, $\mathbf{x}_2 \in K^n$, it selects randomly an i such that $0 \leq i \leq \lambda - 1$ and constructs the L -long vector

$$t(\mathbf{S}, \mathbf{x}_1, \mathbf{x}_2) := (\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_i, \mathbf{x}_1, \mathbf{P}(\mathbf{x}_2), \mathbf{P}(\mathbf{Q}(\mathbf{x}_2)), \dots, \mathbf{P}(\mathbf{Q}^{\lambda-i-1}(\mathbf{x}_2))).$$

If $(\mathbf{x}_1, \mathbf{x}_2)$ is distributed accordingly to the output distribution of \mathbf{S} , i.e. $(\mathbf{x}_1, \mathbf{x}_2) = \mathbf{S}(\mathbf{x}) = (\mathbf{P}(\mathbf{x}), \mathbf{Q}(\mathbf{x}))$ for a uniformly distributed value of \mathbf{x} , then

$$t(\mathbf{S}, \mathbf{x}_1, \mathbf{x}_2) := (\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_i, \mathbf{P}(\mathbf{x}), \mathbf{P}(\mathbf{Q}(\mathbf{x})), \dots, \mathbf{P}(\mathbf{Q}^{\lambda-i-1}(\mathbf{x})))$$

is distributed according to $D_i(\mathbf{S})$. Now if $(\mathbf{x}_1, \mathbf{x}_2)$ is distributed according to the uniform distribution, then

$$t(\mathbf{S}, \mathbf{x}_1, \mathbf{x}_2) = (\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_i, \mathbf{x}_1, \mathbf{P}(\mathbf{x}_2), \mathbf{P}(\mathbf{Q}(\mathbf{x}_2)), \dots, \mathbf{P}(\mathbf{Q}^{\lambda-i-2}(\mathbf{x}_2)))$$

which is distributed according to $D_{i+1}(\mathbf{S})$. To distinguish between the output of \mathbf{S} from uniform, algorithm B calls A with inputs $(\mathbf{S}, t(\mathbf{S}, \mathbf{x}_1, \mathbf{x}_2))$ and returns that same return value. Hence

$$\begin{aligned} & \left| \Pr_{\mathbf{S}, \mathbf{x}}(B(\mathbf{S}, \mathbf{S}(\mathbf{x})) = 1) - \Pr_{\mathbf{S}, \mathbf{x}}(B(\mathbf{S}, \mathbf{S}(\mathbf{x}_1, \mathbf{x}_2)) = 1) \right| \\ &= \left| \frac{1}{\lambda} \sum_{i=0}^{\lambda-1} p_i - \frac{1}{\lambda} \sum_{i=0}^{\lambda-1} p_i \right| = \frac{1}{\lambda} |p_0 - p_\lambda| \geq \frac{\epsilon}{\lambda}. \end{aligned}$$

B XL and F₄-F₅ Families for System-Solving

The XL and F₄-F₅ families of algorithms are spiritual descendants of Lazard's idea [24]: run an elimination on an extended Macaulay matrix (i.e., extending the resultant concept to many variables) as an improvement to Buchberger's algorithm for computing Gröbner bases [11].

Since we cannot discuss these methods in detail, we try to describe them briefly along with their projected complexities. Again, suppose we have the system $P_1(\mathbf{x}) = P_2(\mathbf{x}) = \dots = P_m(\mathbf{x}) = 0$, where P_i is a degree- d_i polynomial in $\mathbf{x} = (x_1, \dots, x_n)$, coefficients and variables in $K = \mathbb{F}_q$.

Method XL [12]: Fix a degree $D (\geq \max P_i)$. The set of degree- D -or-lower monomials is denoted $\mathcal{T} = \mathcal{T}^{(D)}$. $|\mathcal{T}^{(D)}|$ is the number of degree $\leq D$ monomials and will be denoted $T = T^{(D)}$. We now take each equation $P_i = 0$ and multiply it by every monomial up to $D - d_i$ to get an equation that is at most degree D . Collect all such equations in the set $\mathcal{R} = \mathcal{R}^{(D)} := \bigcup_{i=1}^m \{uP_i = 0 : u \in \mathcal{T}^{(D-d_i)}\}$. We treat every monomial in \mathcal{T} as independent and try to solve \mathcal{R} as a linear system of equations.

The critical parameter is the difference between $I = \dim(\text{span}\mathcal{R})$, the rank of the space of equations \mathcal{R} , and T . If $T - I = 0$, the original system cannot be satisfied; if $T - I = 1$, then we should find a unique solution (with very high probability). Also, if $T - I < \min(D, q - 1)$, we can reduce to a univariate equation [12]. We would like to predict D_0 , the smallest D enabling resolution.

Note: For any pair of indices $i, j \leq m$, among linear combinations of the multiples of $P_j = 0$ will be $P_i P_j = 0$, and among linear combinations of the multiples of $P_i = 0$ will be $P_i P_j = 0$ — i.e., one dependency in $\text{span}\mathcal{R}$. In \mathbb{F}_q , $(P_i)^q = P_i$ which generates a similar type of dependency.

Proposition 7 ([32]) Denote by $[u]_s$ the coefficient of the monomial u in the expansion of s , then:

1. $T = [t^D] \frac{(1-t^q)^n}{(1-t)^{n+1}}$ which reduces to $\binom{n+D}{D}$ when $q > D$, and $\sum_{j=0}^D \binom{n}{j}$ when $q = 2$.
2. If the system is regular up to degree D , i.e., if the relations $\mathcal{R}^{(D)}$ has no other dependencies than the obvious ones generated by $P_i P_j = P_j P_i$ and $P_i^q = P_i$, then

$$T - I = [t^D] G(t), \text{ where } G(t) := G(t; n; d_1, d_2, \dots, d_m) = \frac{(1-t^q)^n}{(1-t)^{n+1}} \prod_{j=1}^m \left(\frac{1-t^{d_j}}{1-t^{q d_j}} \right). \quad (3)$$

3. For overdefined systems, Eq. 3 cannot hold when $D > D_{XL} = \min\{D : [t^D]G(t) \leq 0\}$. If Eq. 3 holds up for every $D < D_{XL}$ and resolves at D_{XL} , we say that the system is q -semiregular. It is generally believed [3, 14] that **for random systems it is overwhelmingly likely that $D_0 = D_{XL}$, and indeed the system is not q -semiregular with very small probability.**
4. When it resolves, XL takes $C_{XL} \lesssim (c_0 + c_1 \lg T) \tau T^2$ multiplications in \mathbb{F}_q , using a sparse solver like Wiedemann [31]. Here τ is the average number of terms per equation.

We cannot describe methods \mathbf{F}_4 - \mathbf{F}_5 [17, 18], which are just too sophisticated and complex to present here. Instead, we simply sketch a result that yields their complexities:

Proposition 8 [3] For q -semiregular systems, \mathbf{F}_4 or \mathbf{F}_5 operates at the degree

$$D = D_{reg} := \min \left\{ D : [t^D] \left(\frac{(1-t^q)^n}{(1-t)^n} \prod_{j=1}^m \left(\frac{1-t^{d_j}}{1-t^{q d_j}} \right) \right) < 0 \right\},$$

and take $\lesssim (c'_0 + c'_1 \lg \bar{T}) \bar{T}^\omega$ multiplications, where $\bar{T} = [t^{D_{reg}}] ((1-t^q)^n (1-t)^{-n})$ counts the monomials of degree exactly D_{reg} , and $2 < \omega \leq 3$ is the order of matrix multiplication used.

We do not know what works best under various resource limitations. We take the position of [33], e.g., XL with a sparse solver represents the best way to solve large and more or less random overdetermined systems when *the size of main memory space* is the critical restraint.

C SRQ, a potential candidate for one way function

An SRQ (Sparse Rotated Quadratics) instance is an \mathcal{MQ} system specialized so that it is non-sparse but can be computed with fewer computations than normal quadratics.

Problem SRQ(q, n, m, h): In \mathbb{F}_q , solve $P_1(\mathbf{x}) = P_2(\mathbf{x}) = \dots = P_m(\mathbf{x}) = 0$. The P_i are quadratics formed from “sequence of rotations”, that is Start with $P_0 = x_1 x_2 + x_3 x_4 + \dots + x_{n-1} x_n$ (where n is even), and obtain successive P_j by performing sparse affine maps on \mathbf{x} . I.e., $\mathbf{x}^{(0)} := \mathbf{x}$, $\mathbf{x}^{(i)} := M^{(i)} \mathbf{x}^{(i-1)} + \mathbf{b}^{(i)}$, $y_i := P_i(\mathbf{x}) := P_0(\mathbf{x}^{(i)}) + c_i$, $\forall i$. Matrices $M^{(i)}$ are randomly chosen, invertible and sparse with h entries per row.

The idea behind SRQ is that any quadratic map can be written as $f \circ L$, where f is a standard form and L is an invertible linear map. Now we will choose L to be sparse. A standard form for characteristic 2 fields is the “rank form” which for full-rank quadratics is

$$P_0(\mathbf{x}) = x_1x_2 + x_3x_4 + \cdots x_{n-1}x_n.$$

Clearly, by taking a random \mathbf{c} and b , we can give $P_0(\mathbf{x} + \mathbf{c}) + b$ *any* random affine part. Since each $\mathbf{x}^{(i)}$ is related to $\mathbf{x} = \mathbf{x}^{(0)}$ by an invertible affine map, this holds for every component P_i . This means that results pertaining to sparsity of the linear terms such as [1] (cf. Sec. 4.3) never apply, and hence it is plausible for SRQ to form a one way function class.

In Fig. 1, the samples labelled “sparse non-random” are SRQ tests. It seem as if their behavior under MAGMA’s \mathbf{F}_4 is no different than normal quadratics.

In this SRQ construction, even if $h = 3$ (the rotation matrices $M^{(i)}$ have only three entries per row), the number of cross-terms in each equations still quickly increases to have as many terms as totally random ones, so point 2 in Sec. 4.3 does not apply here. Indeed, since a quadratic over \mathbb{F}_2 of rank $2k$ has bias 2^{-k-1} , the SRQ form acts exactly like a random quadratic under point 3 in Sec. 4.3.

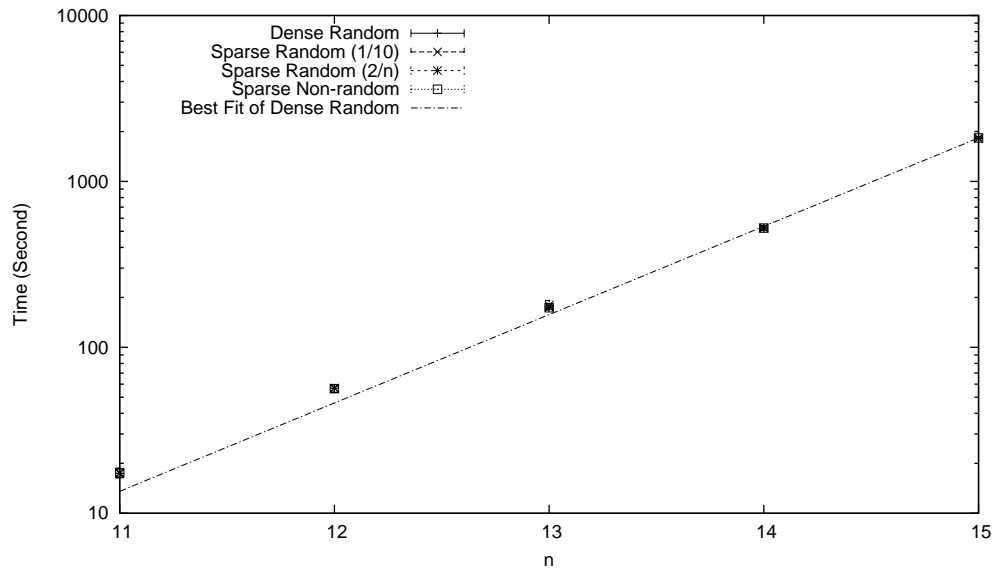


Fig. 1. “Sparsely $2n \rightarrow 3n \mathbb{F}_2$ quadratics” in MAGMA

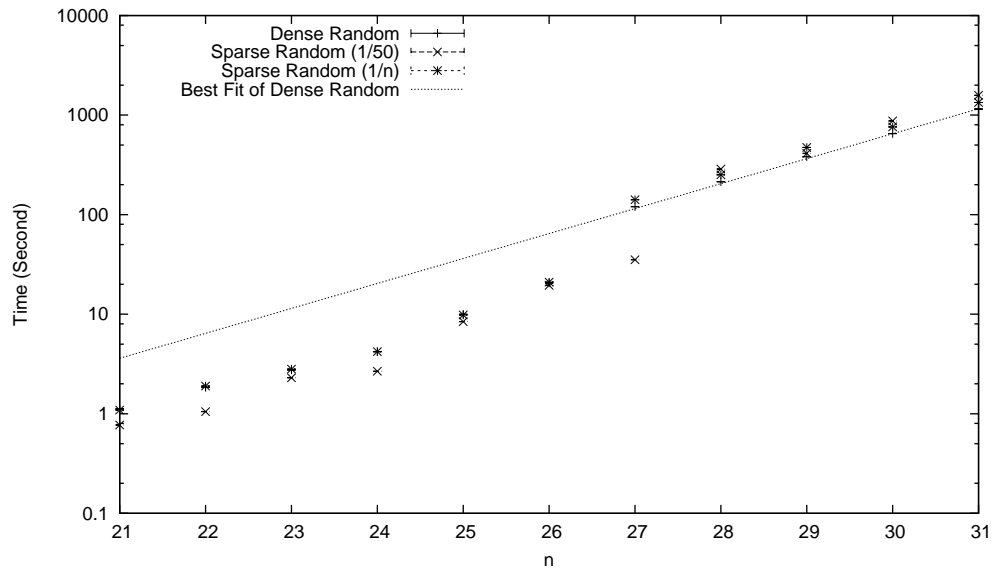


Fig. 2. “Sparsely $n \rightarrow 2n \mathbb{F}_2$ quadratics” in MAGMA