Improved Cryptanalysis of Local PRGs

Shihan Lyu[1], Mo Li[1], Chuhan Ma[2], and Ximing Fu[2,3]

[1] The Chinese University of Hong Kong, Shenzhen
{225010002,220019160}@link.cuhk.edu.cn
[2] Harbin Institute of Technology, Shenzhen
mchuhan@163.com, fuximing@hit.edu.cn
[3] Pengcheng Laboratory, Shenzhen

Abstract. We present two seed-recovery attacks on a family of local PRGs, which map an n-bit random seed to an m-bit pseudorandom string by applying XOR-MAJ predicates.

For PRGs with large locality, we extend the Group-and-Solve (GAS, Eurocrypt 26) framework to Guess-Filter-Solve (GFS). GFS first guesses that a set of seed bits are all 1s and then filters for outputs whose inputs to MAJ contain a sufficiently large subset of the guessed bits. Then high-bias noisy equations can be collected for the correct guess and solved by specific solvers. Compared with GAS, GFS collects substantially more noisy equations with high bias and hence outperforms GAS significantly, especially when the output is short. For example, when the proposed asymptotic XOR-MAJ family (STOC 16) is instantiated with $n = 256$ and $m = 2^{16}$, the complexity for GFS is $2^{68.78}$ while it is $2^{157.91}$ for GAS. In addition, the asymptotic complexity of GFS is $2^{0.15n+o(n)}$.

Independently, inspired by the Guess-and-Decode (GADec, TIT 22), we propose the Filter-Guess-Propagate (FGP) attack. Compared with GADec, FGP introduces two speed-up techniques at each iteration of Belief Propagation (BP): a filter scheme to reduce the number of equations and a dynamic-programming technique to update the log-likelihood ratios of variables efficiently. Together, these improvements enable FGP to perform fast BP and outperform GADec significantly. For $\mathsf{XOR\text{-}MAJ}_{10,64}$ with $n = 256$ and $m = 2^{40}$ (Eurocrypt 24), the total complexity of FGP is $2^{52.66}$, whereas even a single BP iteration of GADec costs $2^{106.42}$. We further extend FGP to $\mathsf{XOR\text{-}MOD}_3$ predicates and experimentally evaluate the resulting attack complexity. We also apply FGP to a ZKP protocol from Eurocrypt 25 and QuietOT from Asiacrypt 24, with only a small computational overhead. For the ZKP protocol, FGP leads to an attack on witness indistinguishability, while for QuietOT, it similarly yields an attack on OT receiver privacy.

1 Introduction

Pseudorandom generators (PRGs) are a fundamental primitive in modern cryptography, expanding a truly random seed of length n into a longer m-bit string that is computationally indistinguishable from true randomness. Local PRGs realize this expansion with the additional property of *locality*, such that each output

bit is obtained by applying a fixed Boolean predicate to a small subset of the seed bits [20]. Of particular interest are local PRGs with *constant locality*, where each output bit depends on only d seed bits, independent of the seed length n. This strong locality enables highly parallel evaluation by constant-depth circuits and makes local PRGs particularly attractive for cryptographic constructions that seek to minimize computational overhead. Indeed, constant-locality PRGs underpin secure computation with constant computational overhead [21, 5, 6] and have found applications to indistinguishability obfuscation [23], cryptographic capsules [10], oblivious transfer [11, 12], garbling [30], and mixed-arithmetic zero knowledge [1].

The security of local PRGs depends critically on the underlying predicate, and a central line of work has therefore sought predicates that resist known attacks in the asymptotic regime. Statistical linear tests rule out locality 3 and 4 at sufficiently large linear stretch $m = \Theta(n)$ [15, 25], whereas the locality-5 predicate $\mathsf{XOR\text{-}AND}_{3,2}$, denoted P_5, emerged as a candidate supporting superlinear stretch [25]. For the polynomial-stretch regime $m = n^s$, weak PRGs for certain values of s were first established, with distinguishing gap at most $1/n^s$ [4]. Subsequent work strengthened this result to $s < 1.25$ for non-degenerate predicates [3] and to $s < 1.5$ for the specific predicate P_5 [27]. Motivated by larger polynomial stretch, XOR-MAJ predicates were proposed and analyzed against algebraic attacks in [7]. In particular, the $\mathsf{XOR\text{-}MAJ}_{a,b}$ challenge proposed there uses parameters satisfying $a \geq 5s$ and $b > 36s$, where a and b denote the numbers of inputs to the XOR and MAJ terms, respectively.

Asymptotic resistance, however, does not by itself determine the concrete cost of attacking practical parameter choices. This has motivated a separate line of work on concrete cryptanalysis. Guess-and-Determine (GAD) attacks P_5 by guessing selected seed bits within AND terms, thereby obtaining sufficiently many linear equations to recover the remaining seed [13]. However, it is difficult to extend to MAJ, because obtaining noise-free equations would require guessing too many seed bits. Guess and Decode (GADec) improves upon this approach by applying belief propagation (BP) to the residual system after guessing, while leveraging the bias provided by the AND terms [29]. It breaks the challenge parameters proposed by GAD and naturally extends to other XOR-AND predicates. GADec can also be extended to XOR-MAJ predicates, but remains effective only for cases with small locality [29, 11].

Group-and-Solve (GAS) exploits the property that conditioning several inputs to a majority component to be all 1s significantly biases its output toward 1, such that efficient seed recovery attack is possible by solving high-bias noisy equations [19]. However, GAS attack can only work with relatively large MAJ. When it comes to the small MAJ (but still large for GADec), GAS becomes inefficient because it is impossible to collect equations that share common inputs in the MAJ term. The imbalance of the seed has also been exploited to enable fast distinguishing attacks against MAJ functions, especially when the MAJ is large [16].

Recently, several more challenging predicates have been proposed, such as the $\mathsf{XOR\text{-}MAJ}_{4,7}$ and $\mathsf{XOR\text{-}MAJ}_{5,15}$. In these designs, the MAJ is large enough to resist GAD and GADec attacks and is small enough to resist GAS attack. These parameters have been widely adopted in recent primitives such as BitGC [22, 30], QuietOT [12], and zero knowledge proof protocol [1], with parameter choices intended to resist known concrete attacks. More recently, $\mathsf{XOR\text{-}MOD}_3$ predicates have been proposed as alternative candidates designed to resist common-bias attacks [16].

In this work, we present two complementary seed-recovery algorithms that extend the cryptanalytic power on $\mathsf{XOR\text{-}MAJ}$ predicates. The contributions are as follows.

- **Guess-Filter-Solve Attack.** For PRGs with large locality, we extend the GAS [19] attack to Guess-Filter-Solve (GFS). GFS first guesses that a set of t seed bits are all 1s, and then filters for outputs whose inputs to MAJ contain a sufficiently large subset of $r \leq t$ guessed bits. The MAJ parts of these filtered outputs take at least r 1s, thus providing high bias to 1. Then high-bias noisy equations can be derived and solved by specific solvers, just as in GAS. Note that the grouping strategy of GAS is a special case of GFS with $t = r$. The decoupled guess-and-filter scheme enables the collection of substantially more equations without sacrificing bias, especially when the output is short. As a result, GFS significantly outperforms GAS and successfully breaks many parameters designed to resist GAS, including the challenge parameters proposed by GAS. For example, with parameters $(n, a, b, s) = (256, 10, 75, 2)$, GFS reduces the complexity from $2^{157.91}$ for GAS to $2^{68.78}$. Combined with a heuristic choice of r, the attack complexity of GFS is $2^{0.15n+o(n)}$, significantly outperforming $2^{0.82n}$ for GAS.
- **Filter-Guess-Propagate Attack.** Independently, inspired by GADec [29], we propose the Filter-Guess-Propagate (FGP) attack. FGP strategically fixes a subset of indices and filters for outputs whose XOR inputs are nearly covered by this subset. It then enumerates all possible assignments to the subset, under which MAJ introduces certain biases, thereby turning the seed solving into a decoding problem that can be progressively solved using BP. The *filter* phase generates and identifies the sparse equations that are truly useful for BP, allowing BP to be performed more efficiently. We also introduce a dynamic-programming technique that reduces the cost of computing the log-likelihood ratio from exponential time in GADec to quadratic time. Together, these improvements allow FGP to substantially outperform GADec. For $\mathsf{XOR\text{-}MAJ}_{10,64}$ with $n = 256$ and $m = 2^{40}$, the estimated total complexity of FGP is $2^{52.66}$, whereas even a single BP iteration of GADec costs $2^{106.42}$. We further extend FGP to $\mathsf{XOR\text{-}MOD}_3$ predicates by slightly modifying the BP update rules to accommodate the MOD_3 predicate.
- **Applications to Real World Protocols.** We apply our attacks to the zero-knowledge proof protocol [1] and QuietOT protocol [12]. Their public correction values allow candidate witnesses or receiver choices to be converted into candidate PRG or wPRF output sequences, breaking witness indistinguishabil-

ity and OT receiver privacy, respectively. For the zero-knowledge application, we also identify a known-input recovery route: sufficiently many known wire values reveal genuine PRG outputs, after which seed recovery exposes the remaining values masked under the same seed.

The rest of the paper is organized as follows. Section 2 introduces the preliminary knowledge, gives brief introductions to GAS and GADec and analyzes the main drawbacks of these two attacks. Section 3 presents the Guess-Filter-Solve attack in detail, mainly focused on large MAJ, together with the complexity analysis and comparison with GAS. Section 4 shows the algorithm and analysis of Filter-Guess-Propagate attack, followed by the comparison with GADec. Section 5 extends the Filter-Guess-Propagate attack to $\mathsf{XOR\text{-}MOD}_3$ predicates. It also applies the Filter-Guess-Propagate attack to some real-world primitives and breaks the corresponding security arguments. Section 6 concludes this paper.

2 Preliminaries

We work over the binary field $\mathbb{F}_2 = \{0,1\}$, with addition denoted by $\oplus$. For a positive integer n, we write $[n] = \{1,\dots,n\}$. For an event $\mathcal{E}$, we write $\mathbf{1}[\mathcal{E}]$ for its indicator. For a finite set S, we write $|S|$ for its cardinality, and for a vector $\mathbf{x}$, we write $|\mathbf{x}|$ for its length. For an index vector $S = (s_1,\dots,s_k)$ and a vector $\mathbf{x}$ with $s_i \in [|\mathbf{x}|]$ for every $i \in [k]$, we write $\mathbf{x}[S] = (x_{s_1},\dots,x_{s_k})$ for the subvector of $\mathbf{x}$ indexed by S. We adopt the convention $\binom{u}{v} = 0$ whenever $v < 0$ or $v > u$, and define an empty sum and an empty XOR to be zero.

2.1 Local PRGs and Seed Recovery

Let a, b be nonnegative integers and let h be an integer. Define the threshold function

$$\mathsf{THR}_{h,b}(z_1,\dots,z_b) \coloneqq \mathbf{1}\left[\sum_{j=1}^{b} z_j \geq h\right].$$

In particular, $\mathsf{THR}_{h,b}$ is the constant-one function when $h \leq 0$ and the constant-zero function when $h > b$. The corresponding $\mathsf{XOR\text{-}THR}$ predicate with parameters (a,h,b) is defined as

$$\mathsf{XOR\text{-}THR}_{a,h,b}(\mathbf{u},\mathbf{z}) \coloneqq \bigoplus_{j=1}^{a} u_j \oplus \mathsf{THR}_{h,b}(\mathbf{z}), \qquad \mathbf{u} \in \{0,1\}^a, \quad \mathbf{z} \in \{0,1\}^b.$$

We also define

$$\mathsf{MAJ}_b = \mathsf{THR}_{\lceil b/2 \rceil, b}, \qquad \mathsf{XOR\text{-}MAJ}_{a,b} = \mathsf{XOR\text{-}THR}_{a,\lceil b/2 \rceil, b}.$$

Let $\mathbf{x} \in \{0,1\}^n$ be a random seed, where $a + b \leq n$, and let $m = n^s$ denote the output length. For each $i \in [m]$, a local PRG instantiated with predicate $\mathsf{XOR\text{-}THR}_{a,h,b}$ independently chooses a uniformly random a-element subset

$\sigma_{i,1} \subseteq [n]$ and then a uniformly random b-element subset $\sigma_{i,2} \subseteq [n] \setminus \sigma_{i,1}$. The i-th output bit is

$$y_i(\mathbf{x}) \coloneqq \bigoplus_{j \in \sigma_{i,1}} x_j \oplus \mathsf{THR}_{h,b}\big(\mathbf{x}[\sigma_{i,2}]\big).$$

The resulting instance is

$$\mathcal{I}(\mathbf{x}) \coloneqq \big((\sigma_{i,1}, \sigma_{i,2}, y_i(\mathbf{x}))\big)_{i=1}^{m},$$

and we omit the argument $\mathbf{x}$ when it is clear from context.

Given $\mathcal{I}$, the seed recovery problem asks to find a vector $\widehat{\mathbf{x}} \in \{0,1\}^n$ satisfying

$$y_i(\widehat{\mathbf{x}}) = y_i \qquad \text{for every } i \in [m].$$

2.2 Group-and-Solve (GAS)

When an increasing number of inputs to THR are fixed to 1s (or 0s), the output distribution can become rapidly biased toward 1 (or 0). Group-and-Solve (GAS) exploits this property by grouping outputs that share a common set of threshold inputs [19]. In groups whose common inputs are all 1s (or 0s), replacing the THR terms by 1s (or 0s) leads to high-bias noisy equations that can then be solved using Learning Parity with Noise (LPN) solvers.

GAS requires each group to contain sufficiently many highly biased noisy equations. Assuming that we group according to r common inputs in the THR term, the expected number of equations is $m\binom{b}{r}/\binom{n}{r}$ in each group. It is intuitive that increasing r can increase the bias of each equation, but decrease the expected available equation number in each group. Hence, GAS can not perform efficiently for small b because sufficient equations cannot be collected with reasonable bias, especially when m is also small.

Gauss and SubGauss solvers were extensively evaluated against other LPN solvers [24, 8, 17] and were suggested for solving the system with a small number of high-bias noisy equations [19]. In this paper, we adopt Gauss and SubGauss, following GAS.

2.3 Guess-and-Decode (GADec)

GADec combines guessing with iterative decoding and, unlike GAS, retains the nonlinear constraints during decoding [29]. It guesses selected seed variables using a strategy designed to determine additional variables for free, and propagates all values implied by the resulting constraints. It then applies Belief Propagation (BP) to the remaining sparse nonlinear constraint system. To handle the nonlinearity, it uses predicate-specific BP update rules to iteratively update soft information about the unknown seed variables.

The main computational bottleneck of GADec lies in the BP phase. For large m, each BP iteration must process a large constraint graph. Moreover, for a nonlinear predicate involving many unknown inputs, a straightforward check-node update requires summing over exponentially many assignments to those inputs. Consequently, a direct application of GADec scales poorly when either m or the nonlinear locality b becomes large.

3 Guess-Filter-Solve Attack

This section presents Guess-Filter-Solve (GFS), an extension of GAS that relaxes the inherent trade-off between the number of equations available in a group and their reliability. GFS applies to predicates in the XOR-THR family, but for simplicity of presentation, we focus on XOR-MAJ, where $h = \lceil b/2 \rceil$. We analyze the attack conditioned on the seed having Hamming weight $\mathrm{wt}(\mathbf{x}) = w$. For simplicity, we omit rounding operations throughout this section.

3.1 Attack Algorithm

At a high level, GFS guesses that a set of t seed positions are all 1 and filters for outputs whose MAJ-input sets overlap with the guessed set in at least $r \leq t$ positions. Conditioned on the guess being correct, replacing the MAJ term of every retained output by 1 yields a noisy linear equation with a guaranteed minimum reliability. GFS then applies the solvers of [19] to the resulting noisy linear system. The illustration of the attack is shown in Fig. 1.

GFS consists of three phases, namely Guess, Filter, and Solve. We first introduce the Guess and Filter phases together, shown in Algorithm 1.

Guess and Filter Phases. Fix hyperparameters t and r satisfying $1 \leq r \leq \min\{t, b\}$ and $t \leq n$. We choose g subsets $\rho_1, \ldots, \rho_g \subseteq [n]$, each of size t, and guess that $\mathbf{x}[\rho_j] = \mathbf{1}$ for each $j \in [g]$. One may analogously guess $\mathbf{x}[\rho_j] = \mathbf{0}$. Trying both possibilities increases the total attack complexity by at most a factor of two, so we focus on the all-one case throughout.

For a guessed set ρ, GFS retains every output whose MAJ-input set overlaps with ρ in at least r positions, forming

$$\mathcal{G}_\rho \coloneqq \{i \in [m] : |\rho \cap \sigma_{i,2}| \geq r\}. \tag{1}$$

Although $\mathcal{G}_\rho$ is formally a set of output indices, each $i \in \mathcal{G}_\rho$ naturally induces the noisy linear equation

$$y_i = \bigoplus_{v \in \sigma_{i,1}} x_v \oplus 1. \tag{2}$$

Equation (2) is correct exactly when the corresponding MAJ term evaluates to 1. In the following, we use $\mathcal{G}_\rho$ interchangeably for the set of retained outputs and the corresponding set of noisy equations when there is no ambiguity.

Since $\sigma_{i,2}$ is a uniformly random b-element subset of $[n]$, the intersection size $|\rho \cap \sigma_{i,2}|$ follows a hypergeometric distribution. Hence, the expected number of equations retained for each guessed set ρ is

$$e_{t,r} \coloneqq \mathbb{E}[|\mathcal{G}_\rho|] = n^s \sum_{u=r}^{\min\{t,b\}} \frac{\binom{t}{u}\binom{n-t}{b-u}}{\binom{n}{b}}.$$

We next analyze the reliability of the retained equations conditioned on a correct guess $\mathbf{x}[\rho] = \mathbf{1}$. Suppose that an output satisfies $|\rho \cap \sigma_{i,2}| = u$. Among

its b majority inputs, u are known to be 1, while the remaining $b-u$ inputs are drawn from the $n-t$ positions outside ρ. Conditioned on $wt(\mathbf{x}) = w$ and $\mathbf{x}[\rho] = \mathbf{1}$, these remaining positions contain exactly $w-t$ ones and $n-w$ zeros. Therefore, the probability that Eq. (2) is correct is

$$\begin{aligned} q_{t,u} &\coloneqq \Pr\big[\mathsf{MAJ}_b\big(\mathbf{x}[\sigma_{i,2}]\big) = 1 \,\big|\, \mathbf{x}[\rho] = \mathbf{1},\ |\rho \cap \sigma_{i,2}| = u,\ wt(\mathbf{x}) = w\big] \\ &= \frac{\displaystyle\sum_{\ell=\max\{0,h-u\}}^{\min\{b-u,w-t\}} \binom{w-t}{\ell}\binom{n-w}{b-u-\ell}}{\binom{n-t}{b-u}}. \end{aligned}$$

The probability $q_{t,u}$ is nondecreasing in u, since replacing an unknown majority input by a fixed 1 can only make the threshold easier to reach. Consequently, every equation retained by Eq. (1) holds with probability at least

$$q_{t,r} = \frac{\displaystyle\sum_{\ell=\max\{0,h-r\}}^{\min\{b-r,w-t\}} \binom{w-t}{\ell}\binom{n-w}{b-r-\ell}}{\binom{n-t}{b-r}}. \tag{3}$$

For parameter choices satisfying $q_{t,r} > 1/2$, the resulting noisy equations therefore have bias at least $q_{t,r} - 1/2$. When there is no ambiguity, we write $e = e_{t,r}$ and $q = q_{t,r}$.

Conditioned on a seed of Hamming weight w, a uniformly sampled size-t set ρ is an all-one set with probability

$$p_{\text{guess}} = \frac{\binom{w}{t}}{\binom{n}{t}}.$$

Sampling approximately $1/p_{\text{guess}}$ guessed sets yields a constant success probability, and accounting for the expected number of repetitions until success gives

$$g = \frac{\mathrm{e}}{\mathrm{e}-1}\frac{\binom{n}{t}}{\binom{w}{t}},$$

where e denotes the base of the natural logarithm [19].

Solve Phase. We use the same Gauss and SubGauss solvers as in GAS [19].

Gauss For each group $\mathcal{G}$, the Gauss solver invokes Gaussian Elimination (GE) once using the full available equation budget. Specifically, up to $\min\{n-t, |\mathcal{G}|\}$ equations from $\mathcal{G}$ are selected and solved by GE, while any remaining free variables are recovered by enumeration. When only a small number of equations is available in each group, Gauss can be advantageous because it uses as many equations as possible in each solving attempt.

Additionally, some variables appearing in the XOR part may already have been guessed during the Guess phase, and their guessed values are naturally substituted into the system during the Solve phase.

SubGauss For each group $\mathcal{G}$, the SubGauss solver invokes GE approximately $q^{-\min\{n-t,e\cdot q\}}$ times. In each invocation, the solver selects $\min\{n-t, eq\}$ equations, treats them as correct, and applies GE. If GE succeeds, any remaining free variables are recovered by enumeration.

For simplicity, we ignore the integrality of eq in the analysis, while in the concrete experiments $\lfloor eq \rfloor$ is used.

The attack algorithm is shown in Algorithm 2.

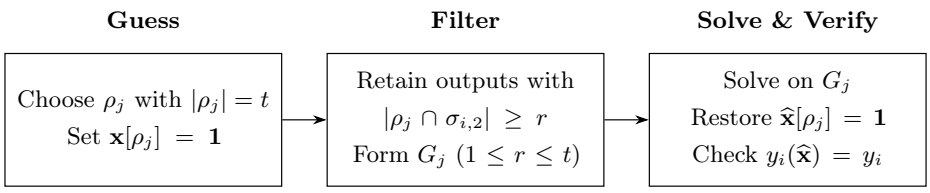


Fig. 1. Guess, Filter, and Solve phases for one guessed set ρ_j in GFS.

Algorithm 1 Guess-and-Filter phase of GFS

Require: Instance $\mathcal{I}$; integers $1 \leq r \leq \min\{t, b\}$, $t < n$; number of guessed sets g

for $j = 1, \ldots, g$ **do**
  Sample a uniform t-element set $\rho_j \subseteq [n]$ and set $G_j \leftarrow \varnothing$
  Guess $\mathbf{x}[\rho_j] = \mathbf{1}$
  for $i = 1, \ldots, m$ **do**
    if $|\rho_j \cap \sigma_{i,2}| \geq r$ **then**
      Add Eq. (2) for output i to G_j
return $(G_1, \ldots, G_g)$ and $(\rho_1, \ldots, \rho_g)$

Algorithm 2 GFS attack with SubGauss

Require: Instance $\mathcal{I}$; integers $1 \le r \le \min\{t, b\}$ and $t < n$; reliability estimate q; number of guessed sets g
Ensure: A verified seed candidate, or $\perp$

Obtain $(G_1, \ldots, G_g)$ and $(\rho_1, \ldots, \rho_g)$ using Algorithm 1
for $j = 1, \ldots, g$ **do**
  Substitute $\mathbf{x}[\rho_j] = \mathbf{1}$ in G_j
  $k_j \leftarrow \min\{n - t, \lfloor q|G_j| \rfloor\}$
  for $u = 1, \ldots, \lceil q^{-k_j} \rceil$ **do**
    Sample a fresh subset $H_{j,u} \subseteq G_j$ with $|H_{j,u}| = k_j$
    $\mathcal{X}_{j,u} \leftarrow \textsc{SubGaussTrial}(H_{j,u}, n - t)$
    for each candidate $\hat{\mathbf{x}} \in \mathcal{X}_{j,u}$ **do**
      Restore $\hat{\mathbf{x}}[\rho_j] = \mathbf{1}$
      if $\textsc{Verify}(\hat{\mathbf{x}}, \mathcal{I})$ **then**
        return $\hat{\mathbf{x}}$
return $\perp$

3.2 Complexity Analysis

In the complexity analysis, we follow the same accounting convention as GAS [19]. For the Solver phase, we measure complexity by the number of GE invocations, whereas for the Filter and Guess phases, we measure complexity by the number of equation-comparison operations.

For simplicity, we only consider the SubGauss solver here, although the analysis can be extended to the Gauss solver in the same manner as in GAS. In particular, we approximate the size of each filtered group by its expectation e and therefore set the available correct-equation budget to eq. Following the analysis of GAS, we approximate the correctness events of the equations selected in each GE trial as independent, and likewise treat different subsystem trials as independent. We also assume that the equations selected by SubGauss attain the intended rank. If a selected subsystem is rank deficient, additional free variables remain and additional equations or subsystem trials may be required; we omit this correction and assume full rank throughout the analysis.

Overall, the complexity of GFS is given by the number of guessed sets multiplied by the sum of the filtering complexity and the solving complexity for each guessed set. Thus,

$$T_{\mathrm{GFS}} = g \cdot (T_{\mathrm{filter}} + T_{\mathrm{solve}}) \,. \tag{4}$$

Under the current accounting convention, T_{filter} and T_{solve} are measured in different units and therefore should not be added numerically. Eq. (4) is intended only to illustrate the decomposition of the overall attack cost and the multiplicative effect of the number of guesses g. Both cost components depend on the choice of (t, r).

Filter Complexity For each guessed set ρ, GFS scans the m outputs to determine whether each output belongs to $\mathcal{G}_\rho$. We count each overlap test as one unit

operation. Therefore, the filtering complexity is

$$T_{\text{filter}} = m = n^s. \tag{5}$$

Solve complexity. Recall that e is the expected number of equations retained in each group and that every retained equation is correct with probability at least q. Under our assumptions, eq is the expected number of correct equations available in each group. Let

$$k \coloneqq \min\{n - t, eq\}.$$

As each selected equation is correct with probability at least q, and under our independence approximation for the selected equations, a subsystem consisting of k equations is entirely correct with probability at least q^k. SubGauss therefore requires approximately q^{-k} subsystem trials to obtain an all-correct subsystem.

Following the assumption of GAS, one trial on a full-rank subsystem of k equations incurs a GE cost of k^ω, where $\omega \leq 3$. If $n - t - k > 0$, there remain $n - t - k$ free variables, which are recovered by enumeration. This contributes a factor of

$$2^{\max\{0, n-t-k\}}.$$

The polynomial cost of verifying each resulting seed candidate is omitted from the complexity estimate, since the attacker can use an appropriate number of outputs for verification rather than necessarily evaluating all m outputs.

Therefore, the complexity of the Solve phase is

$$T_{\text{solve}} = q^{-k} \cdot k^\omega \cdot 2^{\max\{0, n-t-k\}}. \tag{6}$$

Substituting Eqs. (5) and (6) into Eq. (4), we obtain

$$T_{\text{GFS}} = g \cdot \left(n^s + q^{-k} \cdot k^\omega \cdot 2^{\max\{0, n-t-k\}} \right). \tag{7}$$

The optimal GFS complexity is obtained by enumerating all admissible choices of t and r. Notably, Eq. (7) still includes the GE cost term k^ω, although this polynomial factor may be omitted in subsequent sections.

Comparison with GAS Table 1 compares the concrete complexity estimates of GFS and GAS under the same solver and complexity model. Overall, GFS matches or improves upon GAS across all evaluated parameter sets. In particular, GFS breaks all challenge parameter sets proposed by GAS [19]: for every GAS challenge reported in Table 1, the estimated GFS complexity falls below the corresponding target security level λ. The improvement is especially significant for the larger challenge instances. For the largest instances with $b = 37$, 47, 57, and 75, the dominant cost decreases from $2^{257.77}$ to $2^{242.48}$, from $2^{261.94}$ to $2^{170.13}$, from $2^{261.11}$ to $2^{124.12}$, and from $2^{258.41}$ to $2^{93.57}$, respectively.

The concrete results also illustrate how GFS achieves this improvement. For $(n, a, b, s) = (205, 10, 75, 2)$, the GAS grouping and solving costs are $2^{21.13}$

and $2^{136.11}$, respectively. With $(t, r) = (37, 21)$, GFS increases the filtering cost to $2^{58.92}$ but reduces the solving cost to $2^{60.58}$. Similarly, for $(n, a, b, s) = (365, 10, 75, 2)$, GAS has a dominant solving cost of $2^{258.41}$, whereas GFS with $(t, r) = (59, 21)$ reduces the filtering and solving costs to $2^{85.07}$ and $2^{93.57}$, respectively. Thus, GFS can trade additional work in the Guess and Filter phases for a substantially larger reduction in the Solve cost.

These results are consistent with the design of GFS discussed above. By decoupling t from r, GFS can increase t to retain more equations while keeping r sufficiently large to maintain high equation reliability. This is particularly effective when GAS is bottlenecked by an insufficient number of sufficiently reliable equations. Consequently, GFS often changes a strongly Solve-dominated attack into one with much more balanced Filter and Solve costs.

For $\mathsf{XOR\text{-}MAJ}_{4,7}$, the optimizer instead chooses $r = 4$. In this case, four fixed ones already determine the MAJ_7 output, so the retained equations are noise-free and GFS reduces to a GAD attack. This represents a different regime from the larger-b instances, where the optimal r is typically below the MAJ threshold and GFS operates on highly reliable noisy equations.

Finally, GAS is a special case of GFS with $t = r$. Therefore, every parameter choice available to GAS is also available to GFS under the same solver and complexity model. When decoupling t and r provides no additional benefit, the GFS optimizer can simply choose $t = r$ and recover the corresponding GAS attack, as observed for several parameter sets in Table 1.

3.3 Asymptotic Analysis

Although, for any given parameter set, it is feasible to enumerate the admissible choices of t and r and substitute the best pair into T_{GFS}, deriving their joint analytical optimum is difficult. Therefore, in this subsection, we determine asymptotically suitable choices of r and t as n tends to infinity and derive an asymptotic upper bound on the complexity of GFS. Throughout the analysis, we treat a, b, and s as fixed with respect to n and assume $s > 1$. For simplicity, we suppress rounding operations when writing the asymptotic parameter choices.

Theorem 1. *Let a, b, and $s > 1$ be fixed with respect to n, and consider the balanced case $w = n/2$. Under the complexity model above, choose*

$$r^\star \approx 1.7\sqrt{b}$$

and let $t^\star$ be the smallest value satisfying

$$e_{t^\star, r^\star} q_{r^\star} \geq n - t^\star.$$

Assuming the binomial and normal approximations used below, we have $q_{r^\star} \gtrsim 0.9$ and

$$T_{\mathrm{GFS}} \lesssim 2^{0.152n + o(n)}. \tag{8}$$

Table 1. GAS, Guess-Filter-Solve, and FGP complexity estimates. λ denotes the target security level in bits. All reported costs are base-two logarithms, rounded upward to two decimals.

Parameters					GAS[19]				GFS					FGP		
λ	n	a	b	s	Solver	r	$\log_2 T_{\text{group}}$	$\log_2 T_{\text{solve}}$	Solver	t	r	$\log_2 G_{\text{filter}}$	$\log_2 G_{\text{solve}}$	t	p_t	$\log_2 T_{\text{FGP}}$
XOR-MAJ family proposed in [7]																
–	42	5	37	1	Gauss	6	23.05	**35.04**	Gauss	6	6	23.05	**35.04**	17	12.1%	40.65
–	80	5	37	1	Gauss	1	39.35	**75.03**	Gauss	17	9	44.66	**72.35**	35	6.2%	60.64
–	128	5	37	1	Gauss	1	39.34	**122.34**	Gauss	26	9	61.30	**116.45**	60	7.1%	86.20
–	54	7	47	1.25	Gauss	9	34.72	**35.12**	Gauss	9	9	34.72	**35.12**	22	18%	46.33
–	80	7	47	1.25	Gauss	3	43.88	**65.81**	Gauss	22	15	**52.73**	52.03	31	6.7%	57.29
–	128	7	47	1.25	Gauss	2	53.70	**113.76**	Gauss	34	15	**81.35**	80.30	50	6.9%	77.14
–	65	8	57	1.5	Gauss	17	**43.24**	43.03	Gauss	17	17	**43.24**	43.03	24	13.9%	49.75
–	80	8	57	1.5	Gauss	7	46.05	**50.06**	SubGauss	20	17	34.76	**39.34**	29	13.5%	55.17
–	128	8	57	1.5	Gauss	3	**98.93**	89.31	SubGauss	32	18	50.47	**60.30**	44	7.5%	71.79
–	85	10	75	2	SubGauss	23	41.23	**43.86**	SubGauss	23	23	41.23	**43.86**	28	8.1%	55.93
–	128	10	75	2	SubGauss	8	22.74	**60.56**	SubGauss	23	19	41.02	**41.19**	40	9%	68.68
–	256	10	75	2	SubGauss	4	20.45	**157.91**	SubGauss	44	21	67.09	**68.78**	73	5.1%	103.87
GAS challenges [19]																
80	85	5	37	1	Gauss	2	21.54	**82.27**	Gauss	17	9	40.76	**79.18**	37	5.8%	62.81
128	133	5	37	1	Gauss	1	41.44	**129.39**	Gauss	23	8	57.60	**123.16**	61	5.2%	87.68
256	263	5	37	1	Gauss	1	40.81	**257.77**	Gauss	51	9	109.32	**242.48**	129	6.8%	156.35
80	91	7	47	1.25	Gauss	3	40.30	**82.66**	Gauss	30	18	63.19	**63.92**	35	6%	61.65
128	141	7	47	1.25	Gauss	2	53.53	**130.49**	Gauss	37	15	89.84	**93.32**	55	7%	82.29
256	285	7	47	1.25	Gauss	1	181.08	**261.94**	Gauss	83	17	167.13	**170.13**	113	5.3%	142.04
80	105	8	57	1.5	Gauss	4	77.96	**80.05**	SubGauss	30	20	49.84	**53.16**	36	5.5%	63.77
128	155	8	57	1.5	Gauss	3	81.14	**129.49**	SubGauss	42	20	65.34	**71.30**	53	7.1%	81.27
256	303	8	57	1.5	Gauss	2	156.45	**261.11**	SubGauss	68	18	95.58	**124.12**	103	7.5%	132.65
80	146	10	75	2	SubGauss	7	22.26	**77.98**	SubGauss	22	17	39.60	**46.15**	45	10.2%	73.80
128	205	10	75	2	SubGauss	5	21.13	**136.11**	SubGauss	37	21	58.92	**60.58**	60	7%	89.95
256	365	10	75	2	SubGauss	3	19.79	**258.41**	SubGauss	59	21	85.07	**93.57**	102	7.3%	133.25
VOLE-ZK [1]																
128	1024	4	7	2	SubGauss	1	19.47	**624.36**	Gauss	78	4	**102.56**	82.59	27	19.1%	57.14
Silent OT [11]																
128	256	10	64	5	SubGauss	11	**43.03**	36.54	SubGauss	12	11	**40.83**	39.24	18	43%	53
QuietOT [12]																
128	2048	5	15	3	SubGauss	3	36.56	**642.01**	SubGauss	71	7	**106.48**	92.40	23	34%	61
HSS PRF [14]																
–	512	32	64	40/9	SubGauss	10	50.80	**76.18**	SubGauss	16	12	**53.19**	52.07	206	10%	237.13
Shallow wPRF [9]																
128	512	10	64	4.5	SubGauss	10	50.82	**68.04**	SubGauss	16	12	**53.19**	52.07	25	10%	61.16
128	894	10	64	3.4	SubGauss	6	39.93	**268.46**	SubGauss	39	14	74.20	**79.65**	74	6%	109.78
128	2048	10	64	3	SubGauss	4	36.24	**897.91**	SubGauss	118	16	156.68	**176.45**	191	7%	227.61
Common Bias [16]																
128	1536	15	49	3	SubGauss	4	35.78	**726.78**	SubGauss	94	14	130.63	**142.97**	341	13%	373.86
128	2000	21	64	4	SubGauss	6	48.06	**583.18**	SubGauss	60	15	105.85	**111.29**	451	14%	485.34
128	2048	15	49	3	Gauss	4	**997.10**	964.39	SubGauss	118	14	156.69	**180.14**	436	8%	470.16

We use $m = \lfloor n^s \rfloor$ and $h = \lceil b/2 \rceil$. For GAS and GFS, following the fixed-weight worst-case analysis, we set $w = \lfloor n/2 \rfloor$.
For GAS, T_{group} denotes the grouping cost and T_{solve} denotes the solving cost measured by the number of Gaussian-elimination calls. For Guess-Filter-Solve, G_{filter} and G_{solve} denote the corresponding filtering and solving costs.

Proof. Based on the numerical results in Table 1, for relatively large b, a least-squares fit suggests the heuristic choice

$$r^\star \approx 1.7\sqrt{b}. \tag{9}$$

We first estimate the reliability under this choice. From Eq. (3), when $t = o(n)$,

$$q \approx \sum_{i=h-r}^{b-r} \binom{b-r}{i} \left(\frac{w}{n}\right)^i \left(1-\frac{w}{n}\right)^{b-r-i}. \tag{10}$$

For $w = n/2$, this becomes

$$q \approx 2^{-(b-r)} \sum_{i=h-r}^{b-r} \binom{b-r}{i}. \tag{11}$$

Equivalently, $q \approx \Pr[X \geq h-r]$, where $X \sim \mathrm{Bin}(b-r, 1/2)$.

Ignoring rounding and taking $h = b/2$, the normal approximation gives

$$q \approx \Phi\left(\frac{r}{\sqrt{b-r}}\right), \tag{12}$$

where Φ denotes the standard normal cumulative distribution function. For $r = r^\star \approx 1.7\sqrt{b}$,

$$\frac{r^\star}{\sqrt{b-r^\star}} \approx 1.7,$$

and hence

$$q_{r^\star} \approx \Phi(1.7) \gtrsim 0.9. \tag{13}$$

Thus, $r^\star$ maintains a constant equation reliability independent of n.

We next estimate $t^\star$. Under the assumption $t = o(n)$, the hypergeometric distribution of $|\rho \cap \sigma_{i,2}|$ can be approximated by $\mathrm{Bin}(b, t/n)$. Since $t/n = o(1)$, its upper tail is dominated by the first term, giving

$$\Pr[|\rho \cap \sigma_{i,2}| \geq r] \approx \binom{b}{r} \left(\frac{t}{n}\right)^r. \tag{14}$$

Substituting this approximation into $e_{t,r} q_r \geq n - t$ and using $n - t \approx n$ gives

$$n^s \binom{b}{r} \left(\frac{t}{n}\right)^r q_r \gtrsim n. \tag{15}$$

Setting $r = r^\star$ and rearranging yields

$$t^\star \approx \left(\frac{1}{\binom{b}{r^\star} q_{r^\star}}\right)^{1/r^\star} n^{1-\frac{s-1}{r^\star}}. \tag{16}$$

Since $s > 1$ and $r^\star$ is independent of n, this gives $t^\star = o(n)$ in the parameter regime considered here, confirming the self-consistency of the assumption used above.

For the balanced case $w = n/2$, the number of guessed sets satisfies

$$g = \frac{\mathrm{e}}{\mathrm{e}-1} \frac{\binom{n}{t^\star}}{\binom{n/2}{t^\star}}.$$

Since $t^\star = o(n)$, we have

$$\begin{aligned}\log \frac{\binom{n}{t^\star}}{\binom{n/2}{t^\star}} &= \sum_{i=0}^{t^\star-1} \log \frac{n-i}{n/2-i} \\ &= t^\star \log 2 + O\left(\sum_{i=0}^{t^\star-1} \frac{i}{n}\right) \\ &= t^\star \log 2 + O\left(\frac{(t^\star)^2}{n}\right).\end{aligned}$$

Therefore,

$$g = 2^{t^\star + o(t^\star)} = 2^{O(t^\star)} = 2^{o(n)}.$$

Since $T_{\text{filter}} = n^s$ is polynomial in n, the total Guess and Filter cost satisfies

$$gT_{\text{filter}} = 2^{o(n)} n^s = 2^{o(n)}.$$

Moreover, by the definition of $t^\star$,

$$k = n - t^\star = n - o(n) = (1 - o(1))n,$$

the GE cost k^ω is polynomial in n and can be absorbed into the $2^{o(n)}$ factor. Substituting these relations into Eq. (7) gives

$$T_{\text{GFS}} \lesssim 2^{o(n)} q_{r^\star}^{-n+o(n)}. \tag{17}$$

Equivalently,

$$T_{\text{GFS}} \lesssim 2^{n \log_2(1/q_{r^\star}) + o(n)}. \tag{18}$$

Finally, using the conservative estimate $q_{r^\star} \gtrsim 0.9$ gives

$$\log_2(1/q_{r^\star}) \lesssim \log_2(1/0.9) \approx 0.152,$$

and therefore

$$T_{\text{GFS}} \lesssim 2^{0.152n + o(n)}.$$

For comparison, we consider the XOR-MAJ family proposed in [7], with parameters chosen as follows:

$$a = 5s, \qquad b = \text{the least odd integer satisfying } b \geq 37s, \qquad m = n^s,$$

GAS has asymptotic complexity

$$T_{\text{GAS}} \approx 2^{n\left(1 - \log_2\left(1 + \sqrt{\frac{s}{18\pi}}\right)\right)}, \tag{19}$$

whose leading exponent coefficient is at most 0.82 for $s > 1$ [19]. Thus, GFS reduces the leading linear coefficient in the exponent from at most 0.82 to approximately 0.152. More detailed comparison results are shown in Fig. 2.

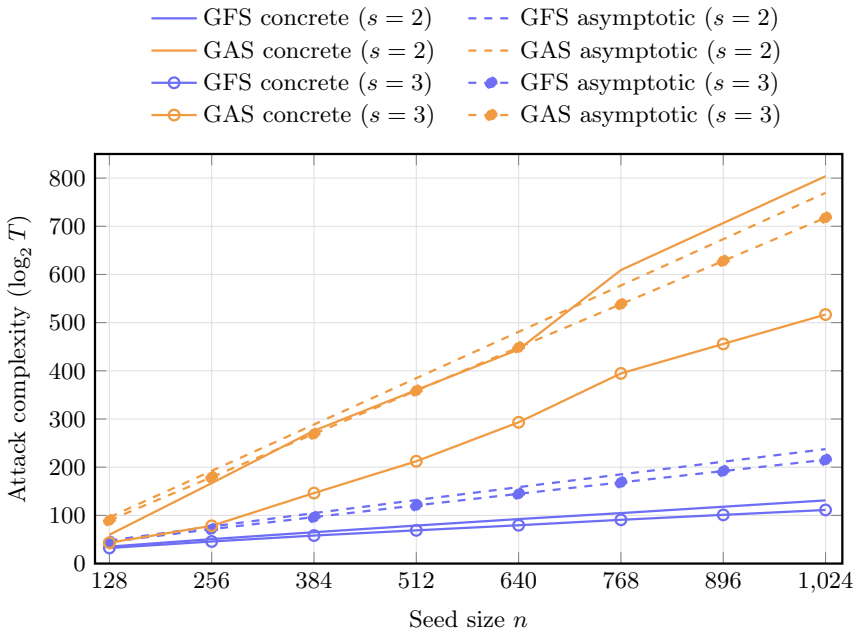


Fig. 2. Concrete and asymptotic complexities of GFS and GAS for the XOR-MAJ family proposed in [7]. Here, we set $a = 5s$ and $m = n^s$, and choose b to be the least odd integer satisfying $b \geq 37s$. For GFS, the asymptotic curves use $r^\star$ equal to the nearest integer to $1.7\sqrt{b}$, $q_{r^\star} = 0.9$, and the corresponding value of $t^\star$, while retaining the polynomial factor. The GAS asymptotic curves are obtained directly from Eq. (19). The concrete curves use the corresponding finite-parameter complexity estimates.

4 Filter-Guess-Propagate Attack

GFS is effective when b is sufficiently large, since enough high-bias equations can be collected at moderate cost. Its complexity can increase substantially, however, when the seed length n is large while b remains relatively small. This regime includes several practically relevant parameter sets. For $\mathsf{XOR\text{-}MAJ}_{4,7}$ with $n = 1024$ and $s = 2$ [1], GFS has an estimated complexity of $2^{102.56}$. The complexity remains prohibitively high, motivating us to seek a more efficient attack for such parameter sets.

GADec is effective for predicates with small locality, but a direct extension to these parameter sets faces two computational bottlenecks. First, computing the threshold Log-Likelihood Ratio (LLR) by explicitly summing over assignments grows exponentially with the threshold arity. Second, BP over the full system becomes expensive when the number of outputs is large. This also makes the experimental estimation of BP success probabilities impractical. Therefore, we propose the FGP attack to address both challenges simultaneously.

4.1 Belief Propagation Updates

We first briefly introduce the BP updates used in GADec [29].

Definitions Fix a guessed set $G \subseteq [n]$ and an assignment $\mathbf{g} \in \{0,1\}^G$. For each output i, define

$$S_i := \sigma_{i,1} \setminus G, \qquad T_i := \sigma_{i,2} \setminus G.$$

After substituting the guessed values, we define

$$y'_i := y_i \oplus \bigoplus_{u \in \sigma_{i,1} \cap G} g_u, \qquad \tau_i := h - \sum_{u \in \sigma_{i,2} \cap G} g_u.$$

The residual equation is therefore

$$\bigoplus_{u \in S_i} x_u \ \oplus \ \mathbf{1}\left[\sum_{u \in T_i} x_u \geq \tau_i\right] = y'_i.$$

The threshold term is constant one when $\tau_i \leq 0$ and constant zero when $\tau_i > |T_i|$.

Let $\mathcal{A} \subseteq [m]$ denote the set of indices of the residual equations used in BP. For each unknown variable x_v, let

$$N(v) := \{\, i \in \mathcal{A} : v \in S_i \cup T_i \,\}$$

denote the equations in $\mathcal{A}$ that contain x_v.

For each unknown variable x_v, let

$$p_v^z := \Pr[x_v = z], \qquad z \in \{0,1\},$$

denote its current belief. The log-likelihood ratio (LLR) of x_v is defined as

$$L_{x_v} := \log \frac{p_v^0}{p_v^1}.$$

Thus, $L_{x_v} > 0$ indicates that $x_v = 0$ is more likely, whereas $L_{x_v} < 0$ indicates that $x_v = 1$ is more likely.

For each equation $i \in N(v)$, let $L^{(\ell)}_{x_v \to i}$ denote the LLR share from x_v to equation i at iteration ℓ, and let $L^{(\ell)}_{i \to x_v}$ denote the LLR share from equation i to x_v. We write $L^{(\ell)}_{x_v}$ for the LLR of x_v obtained by combining the shares from all equations in $N(v)$ at iteration ℓ.

One BP Iteration At iteration $\ell \geq 1$, each equation i first takes the LLR shares $L^{(\ell-1)}_{x_v \to i}$ from its variables as input and computes the LLR share $L^{(\ell)}_{i \to x_v}$ for each variable x_v in the equation. After all equations have been updated, each variable x_v combines the LLR shares from the equations in $N(v)$ to obtain $L^{(\ell)}_{x_v}$ and the LLR shares $L^{(\ell)}_{x_v \to i}$ used in the next iteration. The two steps of one BP iteration are described below.

Step 1: Compute the LLR shares from each equation. For each equation $i \in \mathcal{A}$, we first collect the LLR shares

$$L^{(\ell-1)}_{x_u \to i}, \qquad u \in S_i \cup T_i,$$

from its variables in the previous iteration. For each variable x_u, the LLR share $L^{(\ell-1)}_{x_u \to i}$ corresponds to the probabilities

$$\Pr[x_u = 0] = \frac{\exp\left(L^{(\ell-1)}_{x_u \to i}\right)}{1 + \exp\left(L^{(\ell-1)}_{x_u \to i}\right)}, \qquad \Pr[x_u = 1] = \frac{1}{1 + \exp\left(L^{(\ell-1)}_{x_u \to i}\right)}.$$

As in standard BP, the variables in equation i are treated as independent according to these probabilities when computing its LLR shares.

Let E_i denote the event that the residual equation i is satisfied. For a variable x_v in equation i, define

$$P_{i \to x_v}(z) := \Pr[E_i \mid x_v = z], \qquad z \in \{0, 1\}.$$

The LLR share from equation i to x_v is then

$$L^{(\ell)}_{i \to x_v} = \log \frac{P_{i \to x_v}(0)}{P_{i \to x_v}(1)}. \tag{20}$$

The computation of $P_{i \to x_v}(z)$ differs depending on whether x_v belongs to the XOR part or the threshold part of equation i. We consider the two cases separately below.

XOR inputs. Suppose that $v \in S_i$. For each $u \in S_i \setminus \{v\}$, define the signed bias

$$\delta_u := \Pr[x_u = 0] - \Pr[x_u = 1] = \tanh\left(\frac{L^{(\ell-1)}_{x_u \to i}}{2}\right).$$

Let

$$p_{M,i} := \Pr\left[\sum_{u \in T_i} x_u \geq \tau_i\right]$$

be the probability that the threshold term equals one. Its signed bias is $1 - 2p_{M,i}$. By the piling-up lemma,

$$P_{i \to x_v}(z) = \frac{1 + (-1)^{y'_i \oplus z}(1 - 2p_{M,i}) \prod_{u \in S_i \setminus \{v\}} \delta_u}{2}.$$

Therefore,

$$\begin{aligned} L_{i\to x_v}^{(\ell)} &= \log \frac{P_{i\to x_v}(0)}{P_{i\to x_v}(1)} \\ &= \log \frac{1+(-1)^{y_i'}(1-2p_{M,i}) \prod\limits_{u\in S_i\setminus\{v\}} \delta_u}{1-(-1)^{y_i'}(1-2p_{M,i}) \prod\limits_{u\in S_i\setminus\{v\}} \delta_u} \\ &= 2\,\mathrm{atanh}\left[(-1)^{y_i'}(1-2p_{M,i}) \prod_{u\in S_i\setminus\{v\}} \delta_u\right]. \end{aligned} \tag{21}$$

MAJ inputs. Suppose that $v \in T_i$. Let

$$x_\oplus := \bigoplus_{u\in S_i} x_u.$$

By the piling-up lemma,

$$\Pr[x_\oplus = z] = \frac{1+(-1)^z \prod\limits_{u\in S_i} \delta_u}{2}, \qquad z\in\{0,1\}.$$

Conditioned on $x_v = z$, the threshold term equals one with probability

$$p_{M,i}^{(v)}(z) := \Pr\left[\sum_{u\in T_i\setminus\{v\}} x_u \geq \tau_i - z\right].$$

Hence,

$$P_{i\to x_v}(z) = \Pr[x_\oplus = y_i']\big(1-p_{M,i}^{(v)}(z)\big) + \Pr[x_\oplus = y_i'\oplus 1]p_{M,i}^{(v)}(z).$$

Therefore,

$$\begin{aligned} L_{i\to x_v}^{(\ell)} &= \log \frac{P_{i\to x_v}(0)}{P_{i\to x_v}(1)} \\ &= \log \frac{\Pr[x_\oplus = y_i']\big(1-p_{M,i}^{(v)}(0)\big) + \Pr[x_\oplus = y_i'\oplus 1]p_{M,i}^{(v)}(0)}{\Pr[x_\oplus = y_i']\big(1-p_{M,i}^{(v)}(1)\big) + \Pr[x_\oplus = y_i'\oplus 1]p_{M,i}^{(v)}(1)}. \end{aligned} \tag{22}$$

Step 2: Update the variable LLRs and LLR shares. After all equations have computed their LLR shares, each unknown variable x_v combines the shares received to obtain

$$L_{x_v}^{(\ell)} = \sum_{i\in N(v)} L_{i\to x_v}^{(\ell)}. \tag{23}$$

The variable x_v then computes the LLR share to be sent to each equation $i \in N(v)$. When computing the share for equation i, the LLR share received from i itself is excluded:

$$\begin{aligned} L^{(\ell)}_{x_v \to i} &= \sum_{j \in N(v) \setminus \{i\}} L^{(\ell)}_{j \to x_v} \\ &= L^{(\ell)}_{x_v} - L^{(\ell)}_{i \to x_v}. \end{aligned} \tag{24}$$

The resulting $L^{(\ell)}_{x_v \to i}$ is then used by equation i in iteration $\ell + 1$.

Initialization and Decision Rule All unknown variables have uniform initial beliefs. We therefore initialize

$$L^{(0)}_{x_v} = 0, \qquad L^{(0)}_{x_v \to i} = 0, \qquad v \notin G, \quad i \in N(v).$$

The two steps above are then repeated for I_{BP} iterations. After the final iteration, the candidate value of each unknown variable is

$$\widehat{x}_v = \mathbf{1}\left[L^{(I_{\mathrm{BP}})}_{x_v} < 0\right], \qquad v \notin G.$$

Algorithm 3 summarizes the complete BP procedure.

Algorithm 3 Belief propagation Updates

Require: Equations $(S_i, T_i, \tau_i, y'_i)_{i \in \mathcal{A}}$; iteration rounds $I_{\mathrm{BP}} \geq 1$
Ensure: Candidate values $\widehat{x}_v$ for all $v \notin G$
Set $L^{(0)}_{x_v \to i} \leftarrow 0$ for all $v \notin G$ and $i \in N(v)$
for $\ell = 1, \ldots, I_{\mathrm{BP}}$ **do**
Equation update:
  for all $i \in \mathcal{A}$ **do**
    Collect $L^{(\ell-1)}_{x_u \to i}$ for all $u \in S_i \cup T_i$
    for all $v \in S_i$ **do**
      Compute $L^{(\ell)}_{i \to x_v}$ using Eq. (21)
    for all $v \in T_i$ **do**
      Compute $L^{(\ell)}_{i \to x_v}$ using Eq. (22)
Variable update:
  for all $v \in [n] \setminus G$ **do**
    Compute $L^{(\ell)}_{x_v}$ using Eq. (23)
    for all $i \in N(v)$ **do**
      Compute $L^{(\ell)}_{x_v \to i}$ using Eq. (24)
Set $\widehat{x}_v \leftarrow \mathbf{1}[L^{(I_{\mathrm{BP}})}_{x_v} < 0]$ for all $v \notin G$
return $(\widehat{x}_v)_{v \notin G}$

4.2 Fast LLR Updating with Dynamic Programming

In the equation update of Section 4.1, computing the threshold probabilities $p_{M,i}$ and $p^{(v)}_{M,i}(z)$ is the main computational bottleneck.

GADec [29] computes these threshold probabilities by enumerating all assignments to the threshold inputs and summing the corresponding terms, which requires $O(b2^b)$ operations per equation.

We instead compute the same probabilities exactly using dynamic programming.

Consider an equation i at iteration ℓ, and write

$$T_i = \{w_1, \ldots, w_d\}, \qquad d = |T_i|.$$

For each threshold input x_{w_j}, let

$$p_j := \Pr[x_{w_j} = 1] = \frac{1}{1 + \exp\left(L^{(\ell-1)}_{x_{w_j} \to i}\right)}.$$

If $\tau_i \leq 0$, the threshold term is constant one, while if $\tau_i > d$, it is constant zero. We therefore consider $1 \leq \tau_i \leq d$ below.

For $r \in \{0, \ldots, \tau_i\}$, define the forward and backward distributions by

$$F_j(r) = \Pr\left[\min\left\{\tau_i, \sum_{k=1}^{j} x_{w_k}\right\} = r\right], \qquad 0 \leq j \leq d,$$

$$R_j(r) = \Pr\left[\min\left\{\tau_i, \sum_{k=j}^{d} x_{w_k}\right\} = r\right], \qquad 1 \leq j \leq d+1.$$

The state $r = \tau_i$ collects all cases in which the corresponding sum is at least τ_i. Initially,

$$F_0(0) = R_{d+1}(0) = 1,$$

with all other initial states equal to zero.

For $j \in \{1, \ldots, d\}$ and $0 \leq r < \tau_i$,

$$F_j(r) = (1 - p_j)F_{j-1}(r) + p_j F_{j-1}(r-1),$$
$$R_j(r) = (1 - p_j)R_{j+1}(r) + p_j R_{j+1}(r-1),$$

where terms with a negative state index are defined to be zero. For $r = \tau_i$,

$$F_j(\tau_i) = F_{j-1}(\tau_i) + p_j F_{j-1}(\tau_i - 1),$$
$$R_j(\tau_i) = R_{j+1}(\tau_i) + p_j R_{j+1}(\tau_i - 1).$$

The probability that the threshold term equals one is then

$$p_{M,i} = F_d(\tau_i).$$

To compute the LLR share for a threshold input x_{w_j}, we condition on $x_{w_j} = z$ and evaluate the threshold probability over the remaining threshold inputs. Then, the remaining threshold variables are divided into $x_{w_1}, \ldots, x_{w_{j-1}}$

and $x_{w_{j+1}}, \ldots, x_{w_d}$. The corresponding forward and backward distributions are already available as F_{j-1} and R_{j+1}, respectively.

For $1 \le j \le d+1$ and $q \le \tau_i$, define

$$R_j^{\ge}(q) = \Pr\left[\sum_{k=j}^{d} x_{w_k} \ge q\right] = \begin{cases} 1, & q \le 0, \\ \sum_{r=q}^{\tau_i} R_j(r), & 1 \le q \le \tau_i. \end{cases}$$

For a target input w_j, the probability that all other threshold inputs contain at least q ones is

$$\begin{aligned} Q_j(q) &= \Pr\left[\sum_{\substack{1 \le k \le d \\ k \ne j}} x_{w_k} \ge q\right] \\ &= \sum_{r=0}^{\tau_i} F_{j-1}(r)\, R_{j+1}^{\ge}(q-r), \qquad q \le \tau_i. \end{aligned}$$

For $r < \tau_i$, the prefix contains exactly r ones and the suffix must therefore contain at least $q-r$ additional ones. For $r = \tau_i$, the prefix already contains at least τ_i ones and hence already satisfies every query with $q \le \tau_i$.

The two conditional probabilities required in Eq. (22) are therefore

$$p_{M,i}^{(w_j)}(z) = Q_j(\tau_i - z), \qquad z \in \{0,1\}.$$

The forward and backward distributions are computed once for each equation and reused for all of its threshold inputs. Since $d \le b$ and each distribution contains at most $b+1$ states, constructing all forward and backward distributions costs $O(b^2)$.

For each threshold input w_j, computing $Q_j(\tau_i)$ and $Q_j(\tau_i - 1)$ from the precomputed distributions costs $O(b)$. Repeating this for all threshold inputs therefore costs another $O(b^2)$.

For the XOR inputs, prefix and suffix products of the signed biases δ_u allow the product $\prod_{u \in S_i \setminus \{v\}} \delta_u$ to be computed for all $v \in S_i$ in $O(a)$ time. Hence, all LLR shares from one equation can be updated in $O(a + b^2)$ time.

4.3 Attack Algorithm

We now present the *Filter-Guess-Propagate* (FGP) attack. FGP first chooses a set of t seed positions to guess and filters the equations according to their residual XOR arities, and then applies BP to all retained equations. The overview of FGP is shown in Fig. 3.

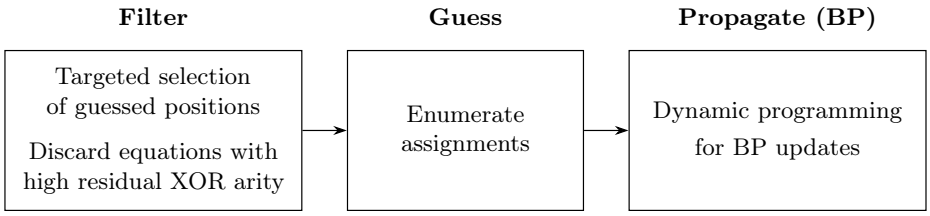


Fig. 3. Overview of FGP.

Filter Phase GADec selects variables to guess from equations whose XOR part has relatively large Hamming weight, hoping that guessing and substitution will reduce some equations to XOR-part Hamming weight one.

In contrast, we employ a more targeted guessing strategy. We greedily search over candidate variables and assign each variable a score. For an unguessed variable v, let $d_r(v)$ denote the number of residual equations whose XOR part has Hamming weight r and contains v. We define

$$\mathrm{score}(v) = d_2(v) - d_1(v).$$

We then preferentially guess variables with higher scores.

Our strategy creates equations with XOR-part Hamming weight one more efficiently while avoiding further reduction of existing equations with XOR-part Hamming weight one. With fewer guessed positions, our strategy can generate more equations with low XOR-part Hamming weight, thereby accelerating the subsequent BP procedure, as confirmed by our experimental results in Fig. 4. The complexity of this selection procedure is approximately $O(mt)$, where t is the number of guessed positions. Since this cost is additive to the complexity of the subsequent phases, it is typically negligible.

Fix a guessed set $G \subseteq [n]$ with $|G| = t$. Partition all equations into

$$\mathcal{C}_{k,d} = \{\, i : |S_i| = k,\ |T_i| = d \,\}, \qquad 0 \le k \le a, \quad 0 \le d \le b.$$

These classes depend only on G and are therefore constructed once before assignment enumeration.

FGP retains all equations with residual XOR arity at most three. Writing

$$\mathcal{C}_k = \bigcup_{d=0}^{b} \mathcal{C}_{k,d}, \qquad k \in \{0,1,2,3\},$$

the retained collection is

$$\mathcal{A} = (\mathcal{C}_0 \setminus \mathcal{C}_{0,0}) \cup \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3.$$

The fully determined class $\mathcal{C}_{0,0}$ is stored separately and used only to reject inconsistent guessed assignments. The complexity is $O(m(a+b))$, which is also an additive cost and can be ignored.

Guess and Propagation phases For each assignment $\mathbf{g} \in \{0,1\}^G$, FGP updates the residual parameters

$$y_i' = y_i \oplus \bigoplus_{u \in \sigma_{i,1} \cap G} g_u, \qquad \tau_i = h - \sum_{u \in \sigma_{i,2} \cap G} g_u.$$

After substitution, fully determined equations are checked directly.

FGP initializes all unknown-variable LLRs to zero and performs I_{BP} BP iterations on the retained equations using the updates of Section 4.2. The threshold probabilities are recomputed from the current incoming beliefs in every iteration.

The low residual XOR arities provide a natural initialization. With zero initial LLRs, an equation with at least two residual XOR variables sends zero LLR to its XOR variables in the first iteration, since each update contains at least one factor $\tanh(0) = 0$. In contrast, for $i \in \mathcal{C}_1$ with $S_i = \{v\}$, Eq. (21) gives

$$L_{i \to x_v}^{(1)} = 2 \operatorname{atanh}\Big[(-1)^{y_i'}\big(1 - 2p_{\mathrm{M},i}^{(0)}\big)\Big].$$

Thus, a biased residual threshold term can provide a nonzero initial LLR. Equations in $\mathcal{C}_0$ can similarly provide initial LLRs through their threshold inputs. Once nonzero LLRs have been established, the retained higher-arity equations can further propagate these LLRs.

After completing the I_{BP} iterations, FGP forms the final hard-decision candidate

$$\widehat{\mathbf{x}}[G] = \mathbf{g}, \qquad \widehat{x}_v = \mathbf{1}\Big[L_{x_v}^{(I_{\mathrm{BP}})} < 0\Big], \quad v \notin G.$$

The resulting candidate is then verified against the original outputs. If verification succeeds, FGP returns the candidate; otherwise it proceeds to the next guessed assignment.

Algorithm 4 summarizes the FGP procedure.

4.4 Complexity Analysis

Let $W_{\mathrm{FGP}}(G)$ denote the cost of one BP iteration on the equations retained for the guessed set G. By Section 4.2,

$$W_{\mathrm{FGP}}(G) = O\big(|\mathcal{A}|(a + b^2)\big),$$

where $\mathcal{A}$ is the set of retained equations.

The selection of G and the filtering step depend only on the equations and are performed once before assignment enumeration. Selecting the t guessed positions requires scanning the m equations during each greedy selection step, giving a cost of $O(mt)$. Once G is fixed, computing the residual arities and partitioning the equations costs $O(m(a+b))$.

We therefore charge

$$C_{\mathrm{pre}} = mt + m(a+b) = n^s(t + a + b)$$

Algorithm 4 Filter-Guess-Propagate (FGP)

Input: Public equations; guess size t; number of BP iterations $I_{\mathrm{BP}} \geq 1$
Output: A seed satisfying all public equations, or $\perp$

$G \leftarrow \mathsf{Select}(t)$
$(\mathcal{A}, \mathcal{D}) \leftarrow \mathsf{Filter}(G)$ ▷ $\mathcal{D} = C_{0,0}$
for each assignment $\mathbf{g} : G \to \{0,1\}$ **do**
  if an equation in $\mathcal{D}$ is violated by $\mathbf{g}$ **then**
    continue
  Substitute $\mathbf{g}$ into $\mathcal{A}$
  Initialize all unknown-variable LLRs to zero
  for $\ell = 1, \ldots, I_{\mathrm{BP}}$ **do**
    Perform one BP iteration
  $\widehat{\mathbf{x}}[G] \leftarrow \mathbf{g}$
  $\widehat{x}_v \leftarrow \mathbf{1}[L_v^{(I_{\mathrm{BP}})} < 0]$ for all $v \notin G$
  if $\widehat{\mathbf{x}}$ satisfies all outputs **then**
    return $\widehat{\mathbf{x}}$
return $\perp$

for the preprocessing cost.

For I_{BP} BP iterations, we estimate the total cost by

$$T_{\mathrm{FGP}}(t) = \frac{C_{\mathrm{pre}} + 2^t I_{\mathrm{BP}} W_{\mathrm{FGP}}(G)}{p_t^{\mathrm{FGP}}}. \tag{25}$$

Here p_t^{FGP} is the measured probability of complete recovery conditioned on the guessed assignment being correct. The factor 2^t accounts for enumerating all assignments to the selected positions. Incorrect candidates are typically rejected after only a few output checks, so the verification cost is negligible compared with the repeated BP cost. We therefore omit it, following the same convention as GAS and GADec.

For each instance, we report the tested value of t that minimizes Eq. (25). Table 2 gives the concrete estimates for the current parameter points.

4.5 Experimental Evaluation

Experimental setup For each tested parameter point, we perform 2000 independent trials. In each trial, the seed is sampled uniformly from $\{0,1\}^n$, and the index sets of all output equations are generated independently and uniformly .All recovery experiments are performed conditioned on the correct assignment of the t guessed seed bits. To avoid introducing an additional tuning parameter, we fix the number of BP iterations to $I_{\mathrm{BP}} = 40$ throughout all experiments.

All experiments are performed on a consumer desktop equipped with a 14-core Intel CPU, consisting of 6 performance cores and 8 efficiency cores, with 20 hardware threads in total. The maximum turbo frequency is 5.0 GHz, and the processor has 24 MB of Intel Smart Cache. The implementation is available at `https://github.com/Analytic233/FGP`.

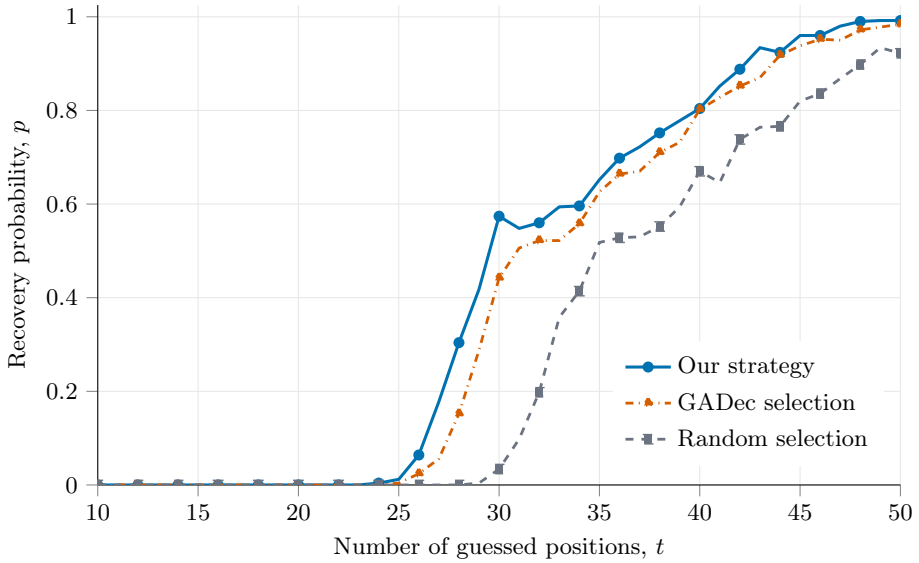


Fig. 4. Effect of guess selection on FGP for $\mathsf{XOR\text{-}MAJ}_{4,7}$ with $n = 1024$ and $m = 2^{20}$.

Guess Selection We first evaluate the effectiveness of our guessing strategy. Figure 4 compares our strategy with the GADec selection rule and uniformly random selection. To isolate the effect of guess selection, all three strategies use the same FGP BP algorithm. Our strategy reaches a given recovery probability with fewer guessed positions than either baseline. For example, the recovery probability first exceeds 50% at $t = 30$ for our strategy, compared with $t = 31$ for GADec selection and $t = 35$ for random selection. The corresponding values for 90% recovery probability are $t = 43$, $t = 44$, and $t = 49$, respectively. These results support directly targeting residual low-arity equations to improve the subsequent BP procedure.

Comparison with GADec Table 2 summarizes the $\mathsf{XOR\text{-}MAJ}_{4,7}$, QuietOT, and $\mathsf{XOR\text{-}MAJ}_{10,64}$ cost estimates. For QuietOT and $\mathsf{XOR\text{-}MAJ}_{10,64}$, we use reduced output lengths because, under the full output lengths, the cost of a single GADec BP iteration is too large for practical experimental evaluation.

Table 2. Concrete FGP and GADec cost estimates. ρ_{FGP} is the retained-equation fraction and W the per-iteration BP cost. Superscript DP denotes the dynamic-programming GADec update; *Reduced* rows use smaller output lengths.

				FGP				GADec					
Instance	(n, s)	(a, b)	ρ_{FGP}	t_{F}	p_t^{F}	W_{F}	T_{F}	t_{G}	p_t^{G}	W_{G}	T_{G}	$W_{\mathrm{G}}^{\mathrm{DP}}$	$T_{\mathrm{G}}^{\mathrm{DP}}$
XOR-MAJ$_{4,7}$	$(1024, 2)$	$(4, 7)$	1.015×10^{-1}	27	19.1%	2^{22}	2^{57}	28	19.5%	2^{30}	2^{65}	2^{26}	2^{61}
QuietOT: XOR-MAJ$_{5,15}$	$(2048, 3)$	$(5, 15)$	1.2×10^{-3}	23	34%	2^{31}	2^{61}	23	–	2^{52}	–	2^{41}	–
Reduced	$(2048, 20/11)$	$(5, 15)$	5.4×10^{-2}	163	16%	2^{24}	2^{195}	167	25%	2^{38}	2^{212}	2^{28}	2^{202}
XOR-MAJ$_{10,64}$	$(256, 5)$	$(10, 64)$	2.577×10^{-7}	18	43%	2^{30}	2^{53}	18	–	2^{106}	–	2^{52}	–
Reduced	$(256, 2)$	$(10, 64)$	5.15×10^{-3}	73	8%	2^{20}	2^{102}	76	9%	2^{70}	2^{154}	2^{28}	2^{112}

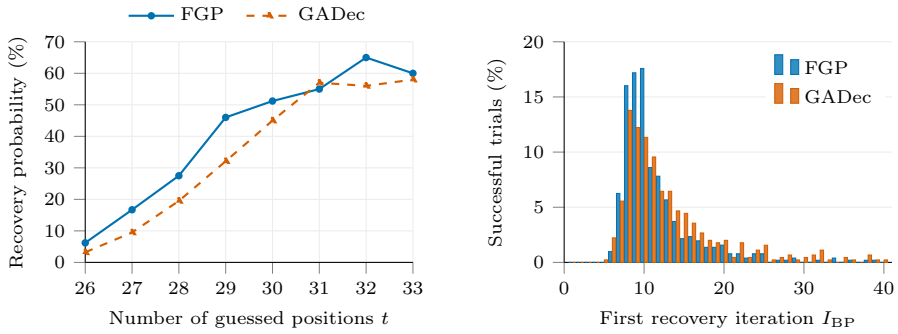
For the complexity comparison, we select the tested value of t that minimizes the total estimated cost rather than the recovery probability alone. The contribution of t and the recovery probability p to the expected cost is proportional to $2^t/p$. Thus, reducing t by one halves the guessing cost and can compensate for a corresponding factor-two decrease in p.

For XOR-MAJ$_{4,7}$, FGP selects $t = 27$ with a 19.1% recovery rate, whereas GADec selects $t = 28$ with a 19.5% recovery rate. FGP retains approximately 10.15% of the equations, reducing the estimated per-iteration work from 2^{30} to 2^{22} and the total estimated work from 2^{65} to 2^{57}.

For the QuietOT parameters and XOR-MAJ$_{10,64}$, the per-iteration cost of GADec on the full equation set is too high to complete the experiments. We therefore keep the predicate and seed length unchanged, but reduce the number of available equations for both methods so that they are evaluated on exactly the same instances. On the reduced QuietOT instance, FGP retains only 5.4% of the available equations and achieves a 16% recovery rate at $t = 163$, compared with a 25% recovery rate at $t = 167$ for GADec. At the same time, the per-iteration cost decreases from 2^{38} to 2^{24}. On the reduced XOR-MAJ$_{10,64}$ instance, FGP retains only 0.52% of the equations and achieves an 8% recovery rate at $t = 73$, compared with a 9% recovery rate at $t = 76$ for GADec. The per-iteration cost in this case decreases from 2^{70} to 2^{20}.

These results are consistent with the role of low-arity residual equations in BP: equations with residual XOR arity zero or one provide initial information, while those of arity two or three help propagate it. By targeting and retaining such equations, FGP reduces the BP cost while maintaining effective recovery.

Figure 5 illustrates the comparison on XOR-MAJ$_{4,7}$ with $n = 1024$ and $m = 2^{20}$. Although FGP retains substantially fewer equations, it achieves better recovery performance than GADec on this instance. Among the successful trials at $t = 30$, the mean first recovery iterations are 11.51 for FGP and 13.08 for GADec, with medians 10 and 11, respectively.



(a) Recovery probability after 40 rounds. (b) First recovery iteration at $t = 30$.

Fig. 5. FGP versus GADec on $\mathsf{XOR\text{-}MAJ}_{4,7}$.

Recovery Probability and Multiple Attempts Recall that p_t denotes the recovery probability of FGP when guessing t variables, conditioned on the guessed assignment being correct. Following GADec [29], our complexity estimate accounts for this recovery probability by a factor of $1/p_t$. For a fixed instance, this factor can be interpreted by trying different sets of t guessed positions.

We experimentally examine this interpretation by fixing the seed and all output equations for each instance and repeatedly running FGP with different guessed sets. For different attempts, we select different variables among those with the same score under our guessing strategy.

Table 3 reports the results for $\mathsf{XOR\text{-}MAJ}_{4,7}$ and the reduced QuietOT instance $\mathsf{XOR\text{-}MAJ}_{5,15}$. We use $t = 27$ and $t = 139$, respectively. The measured recovery probabilities are $p_t = 18.3\%$ and $p_t = 8.2\%$, giving $1/p_t = 5.464$ and 12.195.

Allowing $R_0 = \lceil 1/p_t \rceil$ attempts recovers 64.8% of the instances for $\mathsf{XOR\text{-}MAJ}_{4,7}$ and 63.6% for the reduced $\mathsf{XOR\text{-}MAJ}_{5,15}$ instance. Continuing the unsuccessful instances for up to 100 attempts recovers all tested instances. The average numbers of attempts until recovery are 6.269 and 13.318, respectively, close to the corresponding $1/p_t$ estimates. These results support using $1/p_t$ as an estimate of the average number of guessing attempts required for recovery.

Table 3. Multiple-attempt FGP experiments. The column #attempts gives the average number of attempts until successful recovery.

Instance	(n, s)	t	$1/p_t$	#attempts	success
$\mathsf{XOR\text{-}MAJ}_{4,7}$	$(1024, 2)$	27	5.464	6.269	100%
$\mathsf{XOR\text{-}MAJ}_{5,15}$ *Reduced*	$(2048, 20/11)$	139	12.195	13.318	100%

Comparison with GFS Table 1 compares FGP and GFS across 34 parameter sets.

Under these respective cost models, FGP achieves lower estimated costs for several predicates with small XOR arity. For example, FGP has estimated costs of 2^{57} for $\mathsf{XOR\text{-}MAJ}_{4,7}$ and 2^{61} for $\mathsf{XOR\text{-}MAJ}_{5,15}$. For the same parameter sets, the GFS filter/solve estimates are $2^{102.56}/2^{82.59}$ and $2^{106.48}/2^{92.40}$, respectively. This behavior is consistent with the design of FGP: when the XOR arity is small, guessing a relatively small number of seed positions can produce sufficiently many equations with low residual XOR arity, allowing BP to operate effectively at moderate guessing cost.

5 Extensions and Applications

In this section, we first extend FGP to $\mathsf{XOR\text{-}MOD}_3$ predicates by slightly modifying the BP update rules . We then apply FGP to two recent protocols [1, 12] based on local PRGs with XOR-MAJ predicates. We show that seed recovery can be translated into attacks on witness indistinguishability and receiver-choice privacy with only a small multiplicative overhead. Both attacks follow the same principle: a candidate secret value determines a candidate sequence of PRG outputs, which can then be tested by running FGP and verifying the recovered seed.

5.1 Extending FGP to $\mathsf{XOR\text{-}MOD}_3$

$\mathsf{XOR\text{-}MOD}_3$ predicates have been proposed as alternatives to XOR–MAJ predicates, with the aim of resisting GAS, common-bias attacks, and related attacks [16]. Here,

$$\mathrm{MOD}_3(z) = \mathbf{1}\left[\sum_j z_j \equiv 0 \pmod 3\right].$$

Unlike majority, MOD_3 does not have the same property exploited by GAS and GFS: fixing more inputs does not systematically increase the bias of its output. When many independent uniform inputs remain unknown, the sum of the MOD_3 inputs modulo three is approximately uniform over $\{0, 1, 2\}$. Consequently, replacing the MOD_3 term by zero yields noisy linear equations with correctness probability close to $2/3$. Despite this difference, FGP can still be applied to $\mathsf{XOR\text{-}MOD}_3$ with only a minor modification to the BP update rule for the MOD_3 term. The guessing strategy and filtering rule remain unchanged.

Suppose that the guessed MOD_3 inputs contain c ones, and let d denote the number of remaining MOD_3 inputs. For the MOD_3 term to equal one, the remaining inputs must satisfy

$$\sum_{j=1}^{d} x_j \equiv -c \pmod 3.$$

Let q_j denote the probability that the j-th remaining input equals one according to its LLR.

We compute the distribution of the partial sum modulo three by

$$F_j(r) = (1 - q_j)F_{j-1}(r) + q_j F_{j-1}\big((r-1) \bmod 3\big), \qquad r \in \{0, 1, 2\},$$

with

$$F_0(0) = 1, \qquad F_0(1) = F_0(2) = 0.$$

The probability that the MOD_3 term equals one is therefore $F_d((-c) \bmod 3)$.

As in the dynamic-programming update for majority, forward and backward distributions can be used to exclude the target variable when computing its LLR share. Since each distribution has only three states, all LLR shares of a retained equation can be computed in $O(k + d)$ time, where k and d are its remaining XOR and MOD_3 arities, respectively.

Table 4 reports results for the seven $\mathsf{XOR\text{-}MOD}_3$ parameter sets proposed in [16]. We use uniformly random seeds and 40 BP iterations.

Table 4. FGP results for $\mathsf{XOR\text{-}MOD}_3$.

(n, s)	(a, b)	t	$\hat{p}_t$	T_{FGP}
$(256, 2)$	$(11, 51)$	100	9.0%	2^{125}
$(512, 4.5)$	$(24, 111)$	157	20.8%	2^{185}
$(894, 3.4)$	$(18, 87)$	248	24.0%	2^{276}
$(1024, 2)$	$(11, 51)$	332	6.75%	2^{360}
$(1536, 3)$	$(16, 75)$	400	17.55%	2^{429}
$(2000, 4)$	$(22, 99)$	502	9.3%	2^{533}
$(2048, 3)$	$(16, 75)$	508	6.6%	2^{538}

Figure 6 illustrates the recovery curve for $(n, s) = (256, 2)$ and $(a, b) = (11, 51)$.

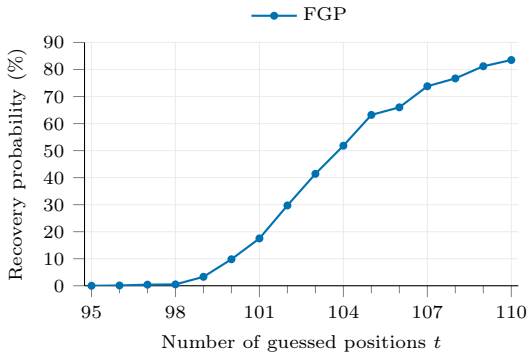


Fig. 6. Recovery probability of FGP on $\mathsf{XOR\text{-}MOD}_3$ with $(n, s) = (256, 2)$ and $(a, b) = (11, 51)$.

5.2 Attack on the ZKP Protocol

In the ZKP protocol at Eurocrypt 25 [1], the prover first commits to a seed $\mathbf{x} \in \{0,1\}^n$ over both $\mathbb{F}_2$ and $\mathbb{F}_p$, and then homomorphically expands the seed into PRG outputs

$$r_i = \mathsf{PRG}(\mathbf{x})_{\sigma_i}$$

under the $\mathsf{XOR\text{-}MAJ}_{4,7}$ predicate. For each mixed-input wire value w_i, the prover sends the verifier a correction value

$$d_i = w_i \oplus r_i.$$

The correction converts the commitment to r_i into a commitment to w_i, while the verifier uses d_i together with the remaining proof messages to check consistency with the committed seed and the public constraints. In particular, the verifier learns the correction d_i, but not w_i or r_i individually. This structure provides the verifier with the information required by the following two attacks.

Attack 1: Recovering the seed from known wire values For specific proof circuits, some wire values w_i may be publicly known or fixed by the circuit [28, 2]. For each such circuit, the verifier can directly reconstruct the corresponding genuine PRG output as

$$r_i = d_i \oplus w_i.$$

If sufficiently many such outputs derived from the same seed are available, the verifier can apply FGP to recover $\mathbf{x}$. Once the seed is recovered, any remaining wire value protected by the same seed can be obtained as

$$w_j = d_j \oplus \mathsf{PRG}(\mathbf{x})_{\sigma_j}.$$

For a seed length of 1024 bits and a PRG stretch factor of $s = 2$, the computational complexity of the attack is approximately 2^{57}, up to polynomial-time overhead.

Attack 2: Breaking witness indistinguishability More generally, we consider an attack on witness indistinguishability. This security notion requires that the verifier cannot distinguish which valid witness was used by the prover with non-negligible advantage [18]. Given two valid witnesses $\mathbf{w}^{(0)}$ and $\mathbf{w}^{(1)}$ for the same public statement, the verifier reconstructs two candidate PRG-output sequences from the public corrections:

$$r_i^{(\beta)} = d_i \oplus w_i^{(\beta)}, \qquad \beta \in \{0,1\}.$$

The verifier then applies FGP to both candidate sequences and verifies the recovered seeds against the corresponding PRG outputs. The candidate corresponding to the actual witness yields PRG outputs consistent with the seed used in the transcript, allowing the verifier to identify which witness was used and thereby break witness indistinguishability. The total computational cost is approximately twice that of a single FGP seed-recovery attack, which is about 2^{58}.

5.3 Attack on QuietOT

At Asiacrypt 24, QuietOT [12] realizes chosen-bit OT from pseudorandom correlations. For each QuietOT instance, the sender holds two messages (m_0, m_1), while the receiver holds a choice bit b. After local pre-processing, the receiver obtains a pseudorandom choice bit b' and sends the correction

$$c = b \oplus b'$$

to the sender. For multiple OT instances, the receiver analogously sends a correction string

$$C = (c_1, \ldots, c_q),$$

where the receiver's choice string is denoted by

$$B = (b_1, \ldots, b_q).$$

The sender knows the public correction string C, while the receiver's secret choice string B is masked by the PRG outputs.

The PRG instantiation with the $\mathsf{XOR\text{-}MAJ}_{5,15}$ predicate is considered, where the pseudorandom choice bits are generated as

$$b'_i = \mathsf{PRG}(\mathbf{z})_{\mathbf{x}_i},$$

where secret PRG seed, also called PRG key, is denoted by $\mathbf{z}$. In the QuietOT instance, the PRG key $\mathbf{z}$ has a length of 2048 bits, and the PRG stretches the key by a factor of $s = 3$.

Receiver-choice privacy requires that, given two admissible receiver choice strings $B^{(0)}$ and $B^{(1)}$, a malicious sender cannot determine which one was used with non-negligible advantage [26]. Our attack exploits the fact that each candidate choice string directly determines a candidate sequence of PRG outputs. Given $B^{(0)}$ and $B^{(1)}$, the sender reconstructs

$$R^{(\beta)} = C \oplus B^{(\beta)}, \qquad \beta \in \{0,1\}.$$

The sender then applies FGP to both candidate PRG output sequences and verifies the recovered PRG keys against the corresponding candidate outputs. The candidate corresponding to the actual choice string yields PRG outputs consistent with the receiver's PRG key, allowing the sender to identify which choice string was used and thereby break receiver-choice privacy. The total computational cost is approximately twice that of a single FGP seed-recovery attack, which is about 2^{62}.

6 Conclusion

We presented GFS and FGP, two seed-recovery attacks on XOR-THR local PRGs. GFS collects more sufficiently biased equations by allowing their majority inputs to overlap different subsets of a larger guessed set. FGP concentrates BP on equations with few remaining XOR inputs, which are more informative, and uses dynamic programming to accelerate their updates. These improvements reduce the recovery complexity and translate into attacks on witness indistinguishability and receiver-choice privacy in the studied zero-knowledge and QuietOT constructions.

Our results further narrow the applications of XOR-MAJ based local PRGs: instances with a large MAJ term are vulnerable to the GFS attack, while ones with a small XOR or MAJ term are vulnerable to the FGP attack. These findings should motivate the search for new predicate designs.

References

1. Agarwal, A., Baum, C., Braun, L., Scholl, P.: Low-bandwidth mixed arithmetic in VOLE-based ZK from low-degree PRGs. In: Advances in Cryptology – EUROCRYPT 2025, Part IV. LNCS, vol. 15604, pp. 396–426. Springer (2025)
2. Agrawal, S., Ganesh, C., Mohassel, P.: Non-interactive zero-knowledge proofs for composite statements. In: Annual International Cryptology Conference. pp. 643–673. Springer (2018)
3. Applebaum, B.: Pseudorandom generators with long stretch and low locality from random local one-way functions. SIAM Journal on Computing **42**(5), 2008–2037 (2013)
4. Applebaum, B., Bogdanov, A., Rosen, A.: A dichotomy for local small-bias generators. In: Theory of Cryptography Conference. LNCS, vol. 7194, pp. 600–617. Springer (2012)
5. Applebaum, B., Damgård, I., Ishai, Y., Nielsen, M., Zichron, L.: Secure arithmetic computation with constant computational overhead. In: Advances in Cryptology – CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 223–254. Springer (2017)
6. Applebaum, B., Konstantini, N.: Actively secure arithmetic computation and VOLE with constant computational overhead. In: Advances in Cryptology – EUROCRYPT 2023, Part II. LNCS, vol. 14005, pp. 190–219. Springer (2023)
7. Applebaum, B., Lovett, S.: Algebraic attacks against random local functions and their countermeasures. In: 48th Annual ACM Symposium on Theory of Computing. pp. 1087–1100. ACM (2016)

8. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How 1+1 = 0 improves information set decoding. In: Advances in Cryptology – EUROCRYPT 2012. LNCS, vol. 7237, pp. 520–536. Springer (2012)
9. Boura, C., Couteau, G., Perrin, L., Rotella, Y.: SoK: On shallow weak PRFs: A common symmetric building block for MPC protocols. IACR Transactions on Symmetric Cryptology **2025**(3), 289–336 (2025)
10. Boyle, E., Couteau, G., Gilboa, N., Ishai, Y., Orrù, M.: Homomorphic secret sharing: Optimizations and applications. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 2105–2122. ACM (2017)
11. Bui, D., Couteau, G., Meyer, P., Passelègue, A., Riahinia, M.: Fast public-key silent OT and more from constrained naor–reingold. In: Advances in Cryptology – EUROCRYPT 2024. LNCS, vol. 14656, pp. 88–118. Springer (2024)
12. Couteau, G., Devadas, L., Devadas, S., Koch, A., Servan-Schreiber, S.: QuietOT: Lightweight oblivious transfer with a public-key setup. In: Advances in Cryptology – ASIACRYPT 2024, Part II. LNCS, vol. 15485, pp. 197–231. Springer (2025)
13. Couteau, G., Dupin, A., Méaux, P., Rossi, M., Rotella, Y.: On the concrete security of goldreich's pseudorandom generator. In: Advances in Cryptology – ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 96–124. Springer (2018)
14. Couteau, G., Koch, A., Melissaris, N., Scholl, P., Servan-Schreiber, S., Ye, X.: On compressing linearly shared correlations. Cryptology ePrint Archive, Paper 2026/255 (2026), `https://eprint.iacr.org/2026/255`
15. Cryan, M., Miltersen, P.B.: On pseudorandom generators in NC^0. In: Mathematical Foundations of Computer Science 2001. LNCS, vol. 2136, pp. 272–284. Springer (2001)
16. D'Antona, A.G., Méaux, P., Ünal, A.: On the common bias of majorities: Poly-time attacks on THR-XOR PRGs. Cryptology ePrint Archive, Paper 2026/880 (2026), `https://eprint.iacr.org/2026/880`
17. Esser, A., Kübler, R., May, A.: LPN decoded. In: Advances in Cryptology – CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 486–514. Springer (2017)
18. Feige, U., Shamir, A.: Witness indistinguishable and witness hiding protocols. In: Proceedings of the twenty-second annual ACM symposium on Theory of computing. pp. 416–426 (1990)
19. Fu, X., Li, M., Lyu, S., Liu, C.: Attacks on goldreich's pseudorandom generators by grouping and solving. In: Advances in Cryptology – EUROCRYPT 2026, Part VI. LNCS, vol. 16546, pp. 3–32. Springer (2026). https://doi.org/10.1007/978-3-032-25333-0_1, earlier version: IACR ePrint 2024/1594
20. Goldreich, O.: Candidate one-way functions based on expander graphs. Electronic Colloquium on Computational Complexity, Report TR00-090 (2000), `https://eccc.weizmann.ac.il/report/2000/090/`
21. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Cryptography with constant computational overhead. In: 40th Annual ACM Symposium on Theory of Computing. pp. 433–442. ACM (2008)
22. Liu, H., Wang, X., Yang, K., Yu, Y.: BitGC: Garbled circuits with 1 bit per gate. In: Advances in Cryptology – EUROCRYPT 2025. LNCS, vol. 15607, pp. 437–466. Springer (2025)
23. Lombardi, A., Vaikuntanathan, V.: Limits on the locality of pseudorandom generators and applications to indistinguishability obfuscation. In: Theory of Cryptography Conference. pp. 119–137. Springer (2017), `https://eprint.iacr.org/2017/301`

24. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\widetilde{O}(2^{0.054n})$. In: Advances in Cryptology – ASIACRYPT 2011. LNCS, vol. 7073, pp. 107–124. Springer (2011)
25. Mossel, E., Shpilka, A., Trevisan, L.: On ε-biased generators in NC^0. In: 44th Annual IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE Computer Society (2003)
26. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: SODA. vol. 1, pp. 448–457 (2001)
27. O'Donnell, R., Witmer, D.: Goldreich's PRG: Evidence for near-optimal polynomial stretch. In: 29th Conference on Computational Complexity. pp. 1–12. IEEE Computer Society (2014)
28. Rosenberg, M., Maller, M., Miers, I.: Snarkblock: Federated anonymous blocklisting from hidden common input aggregate proofs. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 948–965. IEEE (2022)
29. Yang, J., Guo, Q., Johansson, T., Lentmaier, M.: Revisiting the concrete security of goldreich's pseudorandom generator. IEEE Transactions on Information Theory **68**(2), 1329–1354 (2022)
30. Zhang, W., Liu, H., Yang, K., jie Lu, W., Yu, Y., Wang, X., Weng, C.: BitGC made (more) efficient. In: Advances in Cryptology – EUROCRYPT 2026, Part III. LNCS, vol. 16543, pp. 214–245. Springer (2026)