On the Regularity of the Generalized Birthday Problem

Lili Tang[1,2], Yao Sun[1,2], and Xiaorui Gong[1,2(✉)]

[1] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[2] Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
{tanglili, sunyao, gongxiaorui}@iie.ac.cn

Abstract. The Generalized Birthday Problem (GBP), which seeks k hash values from k lists whose XOR is zero, is a fundamental problem across multiple cryptographic domains. While the k-list GBP has been extensively studied, many schemes including Equihash (NDSS'16) utilize a single-list variant (selecting hash values from a single list) without clear theoretical grounding. Our work reveals that the k-list GBP implicitly exhibits a regularity property, a block-wise structure that has been thoroughly studied by Esser and Santini (Crypto'24) in the context of the Syndrome Decoding Problem. Such structured regularity can often be leveraged to design efficient protocols and enable new functionalities. In this work, we revisit these two long-conflated GBPs and initiate a systematic study of the regularity in the realm of GBP.

- **Complexity.** In the worst-case setting, we develop a novel ISD-based framework for GBP. When $k/n > 0.188$ and $k/n > 0.11$ for the regular and non-regular cases, respectively, the proposed algorithms surpass the worst-case complexity of $2^{n/2}$. Through numerical optimization, we heuristically demonstrate that for any constant $k/n > 0$, the advanced ISD algorithms such as BJMM achieve an asymptotic complexity superior to the birthday bound when applied to density-one GBP instances. Our results disprove the average-case-to-worst-case k-XOR conjecture when k is non-constant (e.g., linear in n). In the average-case regime, we fill in theoretical gaps in solving the single-list GBP and show that the regular variant exhibits a $\sqrt{2}$-factor difference in the exponent, which extends naturally to the k-SUM problem and offers new insights into its complexity.
- **Implications for Cryptography.** We analyze the impact of regularity on incremental hash and propose a new collision attack against the ID-based incremental hash (Eurocrypt'97). Our attack achieves an asymptotic time complexity of $\mathcal{O}(\sqrt{n}\cdot 2^{\sqrt{2n}})$, significantly improving upon Wagner's previous bound of $\mathcal{O}(2^{\sqrt{4n}})$ (Crypto'02). Applying our attack to iSHAKE256, we reduce its security lower bound from 2^{256} to 2^{189}. In the realm of Equihash, the index-pointer technique has significantly weakened its ASIC-resistance. To address this, we propose Requihash, a PoW with enhanced ASIC-resistance and a smaller solution size, rigorously aligned with the regular k-list GBP.

Keywords: Generalized Birthday Problem · k-XOR · k-SUM · Equihash · Code-based Cryptography · Incremental Hash

Table of Contents

On the Regularity of the Generalized Birthday Problem 1
Lili Tang, Yao Sun, and Xiaorui Gong
1 Introduction 3
1.1 Motivation 4
1.2 Related Works 6
1.3 Contributions 7
2 Preliminaries 10
2.1 Wagner's Algorithmic Framework 10
2.2 Equihash 12
3 Revisiting the Generalized Birthday Problem 12
3.1 On the Worst-Case Hardness of GBP 13
3.2 On the Average Hardness of GBP 19
3.3 Upper Bound k for the Single-List Algorithm 22
4 New Attacks on Incremental Hash 24
4.1 Incremental Hash: iSHAKE 24
4.2 Our Collision Attack on iSHAKE 26
4.3 Insights into GBP-Related Schemes 27
5 Refining Equihash 29
5.1 Rethinking the Memory Hardness of Equihash. 29
5.2 Requihash: Regular Equihash 30
5.3 ASIC-Resistance of Requihash 31
6 Open Problems and Future Work 33
A Disclosure of AI Assistance 36
B Proof of Lemmas 36
B.1 Proof of Lemma 1 36
B.2 Proof of Lemma 2 37
C On the Asymptotic Order of $\mathcal{N}(n, k)$ 38
D Concrete Memory Analysis 39
D.1 Memory Complexity of Wagner's K-List Algorithm 40
D.2 Memory Complexity of Wagner's Single-List Algorithm 41
E Secondary Solutions 43

1 Introduction

The generalized birthday problem (GBP) [59] is fundamental in cryptography, with wide-ranging applications. In particular, Equihash [14], a memory-hard proof-of-work (PoW) used in blockchains, is based on the single-list variant of GBP and valued for its ASIC resistance. In 2002, Wagner [59] proposed a sub-exponential k-list algorithm, marking the first introduction of a general solution to GBP that surpasses the birthday paradox bound [18, 52, 60]. This breakthrough enabled forgery attacks for blind signature schemes [55] and compromised multiple incremental hashes [10, 15, 32, 33]. In other aspects, Wagner's k-list algorithm facilitates optimized solutions for problems such as the learning parity with noise (LPN) problem [41], the syndrome decoding problem [7], and the low-weight parity check problem [34, 37]. Notably, the NIST-selected post-quantum scheme Dilithium [25] relies on the SIS^∞ variant. Since both problems are formulated under the ℓ_∞-norm, SIS^∞ can be viewed as a generalized instance of GBP with constrained list sizes. Ducas et al. [24] demonstrate that Wagner's algorithm achieves sub-exponential complexity for SIS^∞, thereby offering a promising candidate algorithm for assessing the practical security of Dilithium. However, the distinction between single-list and k-list GBP has remained ambiguous for decades (even during the design of Equihash). This has biased the design and security analysis of GBP-based schemes.

Equihash. Proof-of-work (PoW) [42] has been essential for blockchain security since Bitcoin [48], but conventional schemes based on the SHA-256 collision problem are vulnerable to ASIC miners, which centralize mining power and threaten decentralization through risks like 51% attacks. To mitigate this, ASIC-resistant PoW schemes [61, §J], [14, 51] have been proposed. The proposal of Equihash [14] pioneered a new research direction in PoW design based on GBP, characterized by its memory hardness. As of February 2026, Equihash-based blockchains collectively had a market capitalization of over $4 billion. Notably, the index pointer technique [27] greatly reduces the ASIC-resistance of Equihash. Later, Bai et al. [5] designed efficient multi-chip ASIC miners for Equihash, significantly outperforming conventional CPU/GPU mining. Recently, Tang et al. [58] introduced post-retrieval and in-place merge techniques, reducing the peak memory consumption by more than 50%. These findings underscore the ongoing need to optimize Equihash for better ASIC resistance.

Incremental Cryptography. Incremental cryptography [9, 10] enables efficient updates of cryptographic computations without full recomputation. A classic example is the Merkle tree [45], widely used in Bitcoin and Ethereum [48, 61] to maintain immutable ledgers. The cost of data modification within a Merkle tree scales logarithmically with the full data size. Beyond such tree-based approaches, more advanced incremental hash schemes (IncHash) [10, 15], e.g., iSHAKE [46] based on the SHA-3 standard SHAKE [26], leverage GBP for security, with Wagner's k-list algorithm serving as the main benchmark for security estimates.

These schemes allow hash updates proportional only to the size of modified data, offering advantages for large, dynamic datasets and have been adopted in practice, e.g., by the Kadena blockchain [43]. Incremental hashes [20] can also be used for constructing efficient zk-VMs [56]. However, while preimage attacks based on Wagner's k-list algorithm are known, the absence of efficient collision analyses leaves the security of IncHash insufficiently understood, raising concerns in practice.

1.1 Motivation

There are two versions of GBP in practice: the k-list GBP originally proposed by Wagner (which selects elements from k different lists), and the single-list variant used in Equihash (which selects elements from a single list). This subtle distinction, which we call **regularity** in this work, has often been overlooked, and for a long time, these two variants were treated as the same problem in the community. As a result, this led to misapplications in cryptographic design. For clarity, we designate the regular k-list GBP as RGBP and the loose single-list GBP as LGBP in the following definitions.

Definition 1 (Regular Generalized Birthday Problem). RGBP(n, k)*: Given k lists $L_1, \ldots, L_k$ containing random n-bit values, the regular generalized birthday problem involves finding $x_1 \in L_1, \ldots, x_k \in L_k$ such that*

$$x_1 \oplus x_2 \oplus \cdots \oplus x_k = 0.$$

Definition 2 (Loose Generalized Birthday Problem). LGBP(n, k)*: Given a list L containing random n-bit values, the loose generalized birthday problem involves finding $x_1, \ldots, x_k \in L$ from k distinct list positions such that*

$$x_1 \oplus x_2 \oplus \cdots \oplus x_k = 0.$$

Practical Implications. The distinction between RGBP and LGBP lies in the constraints on solutions. Let $x_i = \mathbf{H}(m_i)$ be generated by a cryptographic hash. In RGBP, each $x_i \in L_i$ is associated with a unique tag indicating its list, and a valid solution must combine elements from different lists to form a consistent plaintext solution. By contrast, LGBP imposes no such restriction. For instance, the plaintext solution in RGBP may follow a fixed serial pattern,

$$m_i \equiv i - 1 \pmod{k}, \quad i = 1, \ldots, k,$$

which would be infeasible in LGBP. Equivalently, RGBP can be viewed as solving

$$\mathbf{H}_1(m_1) \oplus \mathbf{H}_2(m_2) \oplus \cdots \oplus \mathbf{H}_k(m_k) = 0,$$

where $\mathbf{H}_1, \ldots, \mathbf{H}_k$ are independent randomizers. LGBP corresponds to the special case $\mathbf{H}_1 = \cdots = \mathbf{H}_k$. Wagner's k-list algorithm [59] solves $\mathbf{RGBP}(n, 2^k)$ in time $\mathcal{O}(2^{\frac{n}{k+1}+k+1})$ and the single-list algorithm [21, 41] solves $\mathbf{LGBP}(n, 2^k)$ in

time $\mathcal{O}(k \cdot 2^{\frac{n}{k+1}+1})$. In practice, the k-list algorithm for RGBP has broader applicability, making it inappropriate to regard the single-list algorithm merely as an optimization of the k-list algorithm or to simply categorize RGBP and LGBP as the same problem. Notably, some problems are specific to RGBP, including ID-based incremental hash preimage security [10], the ROS problem [11,55], and regular forms of hard problems [4,19,29].

Regularity Analysis. The essential difference between LGBP and RGBP lies in a property known as *regularity* in the literature of code-based cryptography. A regular word $\mathbf{v} \in \mathbb{F}^n$ with Hamming weight w can be written as $\mathbf{v} = (v_1 \| v_2 \| \dots \| v_w)$ where each v_i is a sub-block of length n/w with Hamming weight 1. If we require the error vector in the syndrome decoding problem (SDP) to be regular, we obtain the corresponding regular variant. In essence, solving GBP can be reformulated as solving an under-determined linear system (exactly the same as SDP over $\mathbb{F}_2$). Assuming the input list size for $\mathbf{GBP}(n,k)$ is $N > n$, the n-dimensional column vectors in the list form a matrix $\mathbf{H} \in \mathbb{F}_2^{n \times N}$, and GBP can be rewritten as

$$\mathbf{H} \cdot \mathrm{x} = \mathrm{y} \tag{1}$$

where x is a short vector to be determined. Compared to LGBP, RGBP constrains the solution vector x to be regular. Intriguingly, RGBP and LGBP can be reduced to special instances of Regular SDP and SDP, respectively (see Propositions 2 and 1). The regularity perspective has recently gained traction in code-based cryptography and structured LPN, where it enables reductions in key sizes and communication costs for zero-knowledge proofs and secure multi-party computation protocols [19,29]. To our knowledge, our work is the first to investigate how *regularity* affects complexity and functionality within cryptographic constructions based on GBP (see Sections 4 and 5).

The k-SUM Problem. GBP is also known as the k-XOR problem. A closely related problem in complexity theory is the k-SUM problem: given an input list and a modulus $q \approx 2^n$, find k numbers whose sum modulo q equals zero. This can be interpreted as the formulation of GBP over $\mathbb{Z}_q$. The work of Dinur et al. [23] identified a regular variant of the k-SUM problem but concluded that the hardness of the regular and non-regular cases is equivalent, with at most an $\mathcal{O}_k(1)$ difference in complexity. Let N denote the input list size of k-SUM. Dinur's conclusion implicitly relies on $\binom{N}{k} \approx \mathbf{\Theta}(N^k)$ and $\left(\frac{N}{k}\right)^k \approx \mathbf{\Theta}(N^k)$[1], which holds only when $k = \mathcal{O}(1)$ in typical k-SUM context. However, our work shows that this presumed equivalence does not actually hold and even an $\mathcal{O}_k(1)$ difference can result in a significant complexity gap for non-constant k. Following the work of Agrawal et al. [2], the density of GBP/k-SUM with constant k is defined as

$$\delta = \frac{k \log_2 N}{n}. \tag{2}$$

[1] The hardness comparison under the same list size is not accurate since the solution density differs between the regular and non-regular cases.

$\mathsf{GBP}(n,k)$ focuses on the non-planted case, i.e., $\delta \in [1,\infty)$ where multiple solutions exist. The well-known average-case-to-worst-case conjecture is as follows.

Conjecture 1 (Average-Case k-SUM Conjecture [1]). When $k = \mathcal{O}(1)$, any algorithm with constant success probability for average-case k-SUM (or k-XOR) at density 1 has running time at least $\Omega(N^{\lceil \frac{k}{2} \rceil - o(1)})$[2].

Previous work has rarely examined how the parameter k influences the hardness of GBP/k-SUM. Our work is particularly concerned with the hardness of GBP in the non-constant k regime. When $k \neq \mathcal{O}(1)$, the density formula (2) must be refined. Define density 1 as the regime in which the given N yields one expected unique solution. When k is not a constant, $\binom{N}{k} = \mathbf{\Theta}(N^k)$ does not hold. Restricting our attention to worst-case birthday algorithms [36], the complexity in the worst case should not exceed $2^{n/2}$. We therefore propose the following strong conjecture for non-constant k (the same applies to GBP).

Conjecture 2 (Strong Average-Case k-SUM Conjecture, k Loosely Bounded). When $k = o(n)$, any algorithm with constant success probability for average-case k-SUM (or k-XOR) at density 1, i.e., when one solution exists in expectation, has running time at least $\Omega(2^{\lceil \frac{n}{2} \rceil - o(1)})$.

Discussions of this conjecture are detailed in the contribution section.

1.2 Related Works

Wagner's Algorithm and GBP. The k-list algorithm, initially proposed by Wagner, is tailored for RGBP. Since the formal proposal of Wagner's k-list algorithm, numerous studies [22, 38, 47, 50] have been dedicated to refining time and memory complexities. Minder and Sinclair [47] extended the k-list algorithm to handle the GBP with smaller list sizes. Dinur's work [22] focused on time-memory trade-off solutions for GBP and the K-list algorithm where K is not a power of 2. Inspired by Wagner's work, Levieil et al. [41] introduced the single-list algorithm for LGBP in 2006, which was initially applied in the context of the LPN problem. In 2016, Biryukov et al. [14] proposed Equihash and implemented it as the PoW scheme for the blockchain Zcash [35, §7.7.1]. Later, the index pointer technique [27] within the single-list algorithm was found to effectively reduce the memory complexity of Equihash. Alcock et al. [3] were the first to highlight the mismatch between Equihash and GBP proposed by Wagner. Recently, the single-list algorithm has been formally studied in [3, 21] following the rise of Equihash. However, the rigorous study of the single-list algorithm and Equihash remains ambiguous. Our work focuses on distinguishing the k-list and single-list algorithms for GBP through the lens of regularity, and filling in gaps.

[2] At density 1 with $k = \mathcal{O}(1)$, $2^n = \binom{N}{k} \approx N^k$. We can also use $\Omega(2^{\lceil \frac{n}{2} \rceil - o(1)})$ here.

***k*-SUM Problem.** The k-SUM problem (GBP over $\mathbb{Z}_q$) serves as a fundamental cornerstone in complexity theory, and many hardness assumptions stem from a special case of Conjecture 1: the well-known 3-SUM hypothesis [31]. The k-SUM conjecture has inspired a line of work in fine-grained cryptography [2, 6, 23, 39]. Wagner's algorithm [59] also plays a significant role in the study of the k-SUM problem. Brakerski et al. established the asymptotic optimality of Wagner's algorithm: any algorithm solving average-case k-SUM asymptotically faster than Wagner's would yield a super-polynomial speedup for hard lattice problems. Furthermore, based on Conjecture 1, Dinur et al. [23] proved that Wagner's algorithm is optimal for $k = 3, 4, 5$. Prior research [2,17,23,39] focused on the constant-k setting and thus did not strictly distinguish the regular variant. The novelty of our work lies in: (a) Our work examines the practical significance of regular variants and focuses on their complexity difference which has long been overlooked. (b) Our study is among the first to explore how non-constant k affects the hardness of k-SUM/GBP, and to quantify the resulting complexity gap introduced by regularity in this regime.

1.3 Contributions

With the perspective of regularity introduced for the first time, we provide a systematic treatment of LGBP and RGBP, highlighting their distinct applications that have remained largely unexplored for more than two decades. Our primary contributions comprise two dimensions: a rigorous hardness analysis of GBP, and its applications to practical cryptanalysis and cryptographic design.

Hardness of GBP. Our work aims to address the following two questions:

1. **For which k does Conjecture 2 break down, and is there an efficient algorithm for solving GBP at density 1 in this regime?**
2. **As k increases with fixed n, how significant is the (optimal) asymptotic complexity gap between average-case RGBP and LGBP?**

Addressing the two questions helps clarify the complexity of k-XOR and k-SUM, and has important implications for the instantiation of fine-grained cryptographic schemes, in particular for determining appropriate choices of the parameters n and k. We partially explore the first question and establish a new framework for solving **GBP** over $\mathbb{F}_2$ at density 1, faster than the worst case when $k/n > c^\star$ with some constant $c^\star$. This is achieved by reducing **GBP** at density 1 to a special instance of the (regular) Syndrome Decoding Problem (see Propositions 1 and 2), and then solving it using (regular) ISD algorithms. For the plain ISD algorithm, we prove an explicit threshold $c^\star$ such that Conjecture 2 no longer holds once $c = k/n$ exceeds this bound (see Theorem 1, 2 and poly-bounds in Theorem 3). For advanced ISD algorithms incorporating meet-in-the-middle techniques, our heuristic numerical optimization results suggest that, for any $c \to 0^+$, the resulting asymptotic time complexity can surpass the

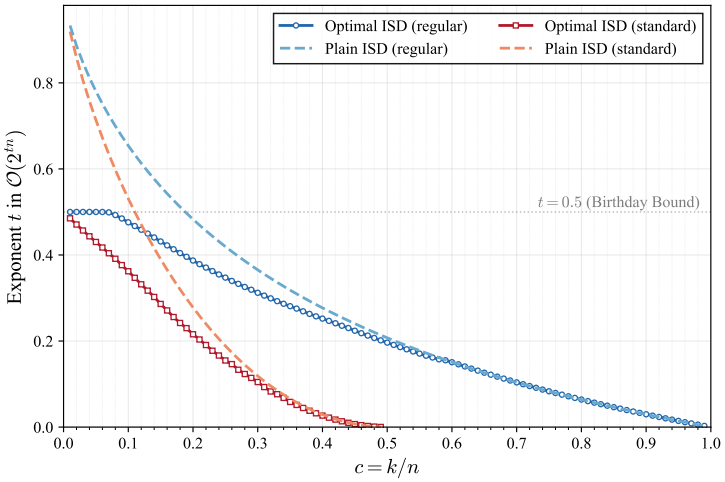


Fig. 1: Asymptotic Complexity of Worst-Case $\mathbf{GBP}(n, k)$ over $\mathbb{F}_2^n$.

birthday bound, i.e., below $\mathcal{O}(2^{0.5n})$. In Figure 1, we present the corresponding optimal asymptotic complexity estimates for these ISD variants.

Furthermore, based on the analysis of Wagner's algorithms, we show that LGBP and RGBP exhibit a $\sqrt{2}$-factor difference in the exponent in the context of the second question (Theorem 4), which naturally extends to k-SUM over $\mathbb{Z}_q$. Specifically, we fill a longstanding gap in the analysis of the single-list algorithm by establishing an admissible upper bound on k (Theorem 5). The overall complexity landscape of average-case GBP is summarized in Figure 2. This regularity-based analysis and quantification carry significant implications for Equihash and incremental hash.

Implications for Cryptographic Primitives.

- **New Attack on IncHash**. Based on regularity analysis, we propose a new collision attack on ID-based incremental hash including LtHASH and AdHASH, and present improved preimage attacks on the pair chaining incremental hash function (PCIHF). In the case of iSHAKE256, our collision attack reduces its security lower bound from 2^{256} to 2^{189}. The cryptanalysis results are summarized in Table 1. Our attack fills a gap in efficient collision attacks against IncHash.
- **Equihash PoW**. We revisit Equihash and show that its memory hardness relies on a variant of the hash collision problem, which we call the self-merge problem. As the index-pointer technique favors ASIC implementations, we propose Requihash to address this, which introduces regular constraints to rigorously align Equihash with RGBP. Requihash is immune to index-pointer optimizations and demonstrates superior resistance to ASIC

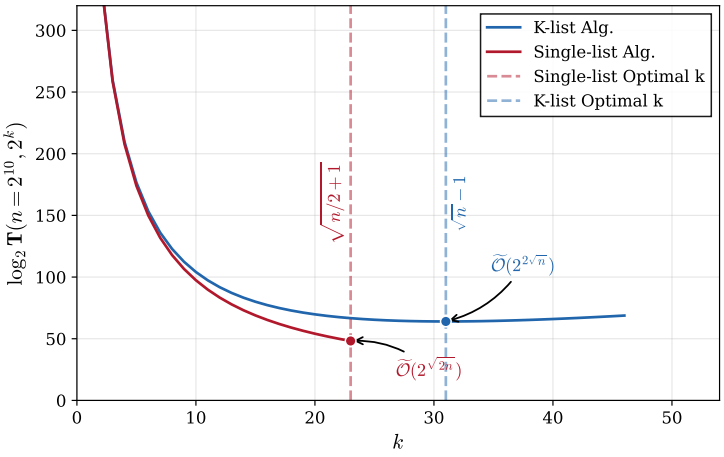


Fig. 2: Time Complexity of Average-Case $\mathbf{GBP}(n, 2^k)$ over $\mathbb{F}_2^n$ or $\mathbb{Z}_{q\approx 2^n}$.

implementations. This initiative aims to enhance the security of Equihash-based blockchains. Upgrading from Equihash$(200, 2^9)$ to Requihash$(200, 2^9)$ increases the theoretical peak memory from 49 MB to 223 MB, with a slightly smaller solution.

Table 1: Cryptanalysis Results on Incremental Hash.

Hash \ Attack	$(\mathbf{n}, \mathbf{K_{max}})$	Prev. Work		Our Work	
		Security	**Ref.**	**Security**	**Ref.**
iSHAKE	$(4160, \infty)$	$2^{128\dagger\dagger}$	[46, §4]	$\mathbf{2^{96.927\dagger}}$	Sec. 4.2
	$(16512, \infty)$	$2^{256\dagger\dagger}$	[46, §4]	$\mathbf{2^{188.942\dagger}}$	Sec. 4.2
PCIHF	(n, ∞)	$\mathcal{O}(2^{2\sqrt{n}})^{\dagger\dagger}$	[59, §4]	$\mathcal{O}(\sqrt{\mathbf{n}} \cdot \mathbf{2^{\sqrt{2n}}})^{\dagger\dagger}$	Sec. 4.3

† Collision attack †† Collision and preimage attacks

Roadmap. Section 2 introduces the syndrome decoding problem, Wagner's algorithmic framework and Equihash. In Section 3, we revisit LGBP and RGBP from multiple perspectives and present our main theoretical results. In Section 4, we conduct a cryptanalysis of the incremental hash and GBP-related schemes. The new insights into Equihash and our proposal of Requihash are detailed in Section 5. Our accompanying code is available at:

https://github.com/tl2cents/Generalized-Birthday-Problem

2 Preliminaries

Throughout, logarithms are base 2. Let $\mathcal{LSB}_\ell(x)$ denote the ℓ least significant bits of x. Let $[1, n]$ be the set of integers from 1 to n. GBP refers to both RGBP and LGBP. Let $K = 2^k, \ell = \frac{n}{k+1}, N = 2^{\ell+1}$ for $\mathbf{GBP}(n, 2^k)$ throughout the following discussions. We begin by recalling some fundamental definitions from code-based cryptography.

Definition 3 (Syndrome Decoding Problem). *SDP(n, r, w): Given a matrix $\mathbf{H} \in \mathbb{F}_q^{r \times n}$ and a syndrome vector $s \in \mathbb{F}_q^r$, the syndrome decoding problem involves finding a word e of weight w such that $\mathbf{H} \cdot e = s$.*

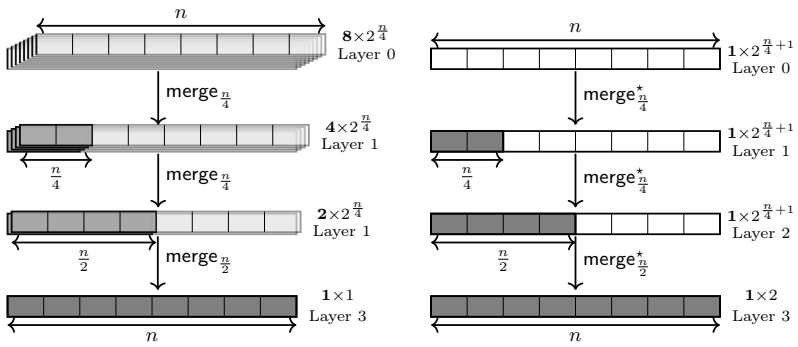
Regular Word. Let n be the dimension of a linear code and let w be a positive number that divides n. Then a (n, w)-regular word of length n and weight w is a special word of the form:

$$\mathbf{b} = (b_1||b_2|| \cdots ||b_w), \quad b_i \in \mathbb{F}_q^{\frac{n}{w}} \text{ and } \mathbf{wt}(b_i) = 1.$$

Definition 4 (Regular Syndrome Decoding Problem [4]). *RSDP(n, r, w) is analogous to SDP(n, r, w), except that the solution must be a regular word.*

2.1 Wagner's Algorithmic Framework

Wagner [59] proposed a sub-exponential k-list algorithm to solve $\mathbf{RGBP}(n, 2^k)$ in time $\mathcal{O}(2^{k+\ell+1})$. The single-list algorithm, generally known as the LF2 method in the realm of LPN [41], employs a framework analogous to Wagner's k-list algorithm and solves $\mathbf{LGBP}(n, 2^k)$ in an improved time complexity of $\mathcal{O}(k \cdot 2^{\ell+1})$.



(a) K-list Algorithm (b) Single-list Algorithm

Fig. 3: Illustration of Wagner's algorithm for $\mathbf{GBP}(n, 2^3)$ where the gray area in each layer represents the portion of the list element that is XORed to 0.

***k*-List Algorithm.** The operation $\mathsf{merge}_\ell(L_1, L_2)$ represents merging two lists L_1 and L_2 to find pairs $(x_1 \in L_1, x_2 \in L_2)$ colliding on ℓ bits, i.e., $\mathcal{LSB}_\ell(x_1 \oplus x_2) = 0$, which can be performed in time $\mathcal{O}(|L_1|+|L_2|)$ with memory $\mathcal{O}(\min(|L_1|, |L_2|))$. The k-list algorithm for $\mathbf{RGBP}(n, 2^k)$ can be visualized as a complete binary tree of height k, where the original 2^k lists serve as the leaves. In each layer, the algorithm merges two lists of size 2^ℓ to generate a new list $L = \mathsf{merge}_\ell(L_1, L_2)$ of expected size 2^ℓ. In the last layer, the merge operation is performed to collide on 2ℓ bits and is expected to generate one solution to $\mathbf{RGBP}(n, 2^k)$. Figure 3a illustrates the simplified collision layers. The k-list algorithm can be executed in post-order using a stack to reduce peak memory [59]. In this case, the k-list algorithm (see Appendix D.1) requires storing at most k lists in memory, as opposed to 2^k lists.

Single-List Algorithm. Algorithm 1 requires a single list of size $2^{\ell+1}$ to solve $\mathbf{LGBP}(n, 2^k)$. The self-merging operation denoted as $\mathsf{merge}^\star_\ell(L)$ is applied to a single list L to find distinct pairs (x_1, x_2) colliding on ℓ bits. When performing $L^{(i)} = \mathsf{merge}^\star_\ell\left(L^{(i-1)}\right)$ for $i = 1, \ldots, k-1$, the expected list size stays at $\mathcal{O}\left(\binom{2^{\ell+1}}{2}/2^\ell\right) = \mathcal{O}\left(2^{\ell+1}\right)$. At the k-th self-merging, the algorithm attempts to collide on 2ℓ bits and generate a list of expected size 2 which contains the solutions to $\mathbf{LGBP}(n, 2^k)$. Figure 3b illustrates the simplified collision layers of the single-list algorithm for $\mathbf{LGBP}(n, 8)$. The index pointer technique [3,27] can be applied in the single-list algorithm to significantly reduce peak memory. The algorithmic details and concrete memory analysis are provided in Appendix D.2.

Algorithm 1 The single-list algorithm for $\mathbf{LGBP}(n, 2^k)$

Require: The self-merge function $\mathsf{merge}^\star_\ell(L) := \{((x_1 \oplus x_2) \gg \ell, i_1 \cup i_2) \,|\mathcal{LSB}_\ell(x_1 \oplus x_2) = 0 \text{ and } i_1 \neq i_2 \; \forall (x_1, i_1), (x_2, i_2) \in L\}$.
Input: A single list L of size $N = 2^{\ell+1}$.
Output: A solution list to $\mathbf{LGBP}(n, 2^k)$.

1: Initialize $L^{(0)} \leftarrow \{(a_i, \{i\}) \mid \forall a_i \in L\}$. ▷ Attach index.
2: **for** $j = 1$ to $k-1$ **do**
3: $\quad L^{(j)} \leftarrow \mathsf{merge}^\star_\ell(L^{(j-1)})$.
4: $\quad$ Remove $L^{(j-1)}$ from memory.
5: Return $L^{(k)} \leftarrow \mathsf{merge}^\star_{2\ell}(L^{(k-1)})$.

Binary Tree Structured Solution. The solution generated from Wagner's algorithmic framework features a binary tree structure of height $k = \log K$. When constructing a binary tree connected by XOR operations, the leaves correspond to the K elements of the solution and the root value of the tree is 0. For each node at a height $h < k$, the lower $h \cdot \ell$ bits of the XOR value are zero. This binary tree structure is illustrated in Figure 4 for $\mathbf{GBP}(48, 4)$.

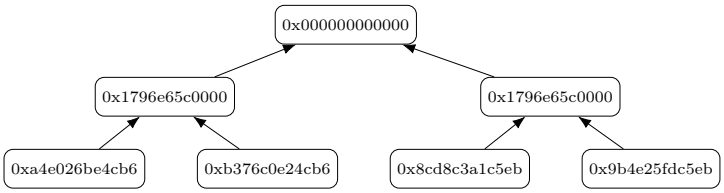


Fig. 4: The binary tree structure of Wagner's solution for $\mathbf{GBP}(48, 4)$

2.2 Equihash

Equihash [14] has gained prominence as one of the most widely implemented PoW algorithms in blockchains, including Zcash and Bitcoin Gold. Equihash is specifically designed to be memory-hard, leveraging GBP. In Definition 5, we omit the difficulty filter and duplicate check for solutions. Equihash demands a solution to LGBP, which is structured as a binary tree in Figure 4. The associated constraints are directly derived from Wagner's algorithmic framework.

Definition 5 (Equihash). *Equihash*$(n, 2^k)$*: Given a cryptographic hash function* $\mathcal{H}$*, find* $K = 2^k$ *messages, i.e.,* $x_0, x_1, \ldots, x_{K-1}$ *such that*

$$\mathcal{H}(x_0) \oplus \mathcal{H}(x_1) \oplus \cdots \oplus \mathcal{H}(x_{K-1}) = 0. \tag{3}$$

Additionally, let $\ell = \frac{n}{k+1}$ *and* $L_i^{(0)} := \mathcal{H}(x_i)$ *for* $i \in [0, 2^k)$*. The solution satisfies Wagner's constraints:*

$$\begin{cases} L_i^{(1)} := L_{2i}^{(0)} \oplus L_{2i+1}^{(0)}, & i \in [0, 2^{k-1}) \text{ where } \mathcal{LSB}_\ell(L_i^{(1)}) = 0 \\ L_i^{(2)} := L_{2i}^{(1)} \oplus L_{2i+1}^{(1)}, & i \in [0, 2^{k-2}) \text{ where } \mathcal{LSB}_{2\ell}(L_i^{(2)}) = 0 \\ \quad \cdots & \\ L_i^{(k-1)} := L_{2i}^{(k-2)} \oplus L_{2i+1}^{(k-2)}, & i \in [0, 2^1) \text{ where } \mathcal{LSB}_{(k-1)\ell}(L_i^{(k-1)}) = 0 \\ L_0^{(k)} := L_0^{(k-1)} \oplus L_1^{(k-1)}, & \text{where } \mathcal{LSB}_{(k+1)\ell}(L_0^{(k)}) = 0 \end{cases} \tag{4}$$

Tree-Binding GBP. Equihash is neither an RGBP nor an LGBP, but rather should be regarded as a new hard problem since the solution to Equihash must adhere to Eq. (4) imposed by the algorithmic framework.

3 Revisiting the Generalized Birthday Problem

We apply regularity in the analysis of GBP. Our main contributions in this section are twofold: (a) **Worst-Case Analysis**. We provide a new perspective on analyzing and solving GBP. In particular, we reduce GBP over $\mathbb{F}_2$ to a special form of SDP and discuss the complexity differences introduced by regularity in the worst case. (b) **Average-Case Analysis**. We provide an initial analysis

of the average-case complexity of GBP by studying two variants of Wagner's algorithm. In particular, for fixed n, we derive the optimal complexity of the regular and non-regular GBP, showing that they differ by a factor of $\sqrt{2}$ in the exponent. The optimal time complexities facilitate our subsequent analysis of the collision attack complexity on IncHash (see Section 4).

3.1 On the Worst-Case Hardness of GBP

Assuming the input list size is N (N of each list for the regular case), let the density of LGBP and RGBP be $\delta_L = \frac{\log_2 \binom{N}{k}}{n}$ and $\delta_R = \frac{k \log_2 N}{n}$. For ease of notation, we write the density of both GBPs as δ where $2^{(\delta-1)n}$ solutions exist in expectation. Typically, $\mathbf{GBP}(n, k)$ focuses only on the non-planted case, i.e., $\delta \in [1, \infty)$ where multiple solutions exist. The known worst-case algorithms for LGBP and RGBP are very similar and do not show significant difference in complexity.

- **Birthday Attack for LGBP**(n, k): Compute sums of $\frac{k}{2}$ random distinct list elements and look for a collision. By birthday paradox, this runs in time at most $\mathcal{O}(2^{n/2})$ or $\binom{N}{k/2} \approx \mathcal{O}(N^{\frac{k}{2}})$ for constant k.
- **Meet-in-the-middle Attack for RGBP**(n, k): Enumerate all sums obtained by choosing exactly one element from each of $\frac{k}{2}$ lists, and store them in T. Compute all possible sums of the remaining $\frac{k}{2}$ lists and look for a collision in T. This runs in time at most $\mathcal{O}(2^{n/2})$ or $\mathcal{O}(N^{\frac{k}{2}})$.

Since any high-density input can be resampled into the density-1 case, we therefore regard the density-1 GBP as the worst case (the k-SUM conjecture). However, when k is non-constant, we find that the GBP at density 1 admits algorithms with better asymptotic complexity than $\mathbf{\Theta}(2^{\frac{n}{2}})$.

$\mathcal{L}_p$-norm of GBP. GBP can be extended to $\mathbb{Z}_q^m$ and can be reformulated as $\mathbf{H} \cdot \mathrm{x} = \mathrm{y}$ where $\mathbf{H} \in \mathbb{Z}_q^{m \times N}$ and x is a short vector to be found. Note that finding a short solution $\mathrm{x} = (x_1, x_2, \cdots, x_N)$ of $\mathbf{H} \cdot \mathrm{x} = \mathrm{y}$ can also be regarded as SDP in the code-based realm or Short Integer Solution (SIS) problem in the lattice-based realm. In the context of SDP, the notion of "shortness" is commonly quantified via the $\mathcal{L}_0$-norm (Hamming weight): $\|\mathrm{x}\|_0 = \sum_{i=0}^{N} [x_i \neq 0]$. For the SIS problem, the corresponding measure is usually the $\mathcal{L}_2$-norm (Euclidean distance): $\|\mathrm{x}\|_2 = \left(\sum_{i=0}^{N} x_i^2\right)^{1/2}$. Within GBP, shortness is defined with respect to the $\mathcal{L}_0$-norm and $\mathcal{L}_\infty$-norm: $\|\mathrm{x}\|_\infty = \max(x_1, x_2, \ldots, x_N)$, where the solution vector x is strictly constrained to have $\mathcal{L}_\infty(\mathrm{x}) = 1$ and $\mathcal{L}_0(\mathrm{x}) = k$ with $k \ll N$. In $\mathbb{F}_2$, the $\mathcal{L}_\infty$-norm of any vector is always 1 (except the zero vector). Hence, GBP degenerates into a special case of SDP where the code length N is not upper bounded. This connection explains why GBP admits numerous applications in the domains of SDP, SIS, and its dual LPN problem (all defined over $\mathbb{F}_2$) [7,21,24,41,50]. In the following, we restrict our attention to GBP over $\mathbb{F}_2^n$.

Information-Theoretic Lower Bound for GBP. From Eq. (1), Wagner's formulation [59] of GBP inherently exhibits a regularity property, where $\mathbf{x} \in \mathbb{F}_2^N$ is an (N, k)-regular vector. We first give the information-theoretic lower bounds of RGBP and LGBP, which to some extent capture their difference in computational hardness.

- The minimum list size of $\mathbf{LGBP}(n, k)$ is defined as follows:

$$\mathcal{N}_L(n, k) = \arg\min_{\mathcal{N}} \left\{ \binom{\mathcal{N}}{k} \geq 2^n \right\}$$

  implying that there exists at least one solution to $\mathbf{LGBP}(n, k)$ with non-negligible probability, given a list of size $\mathcal{N}_L(n, k)$.
- The minimum list size of $\mathbf{RGBP}(n, k)$ is defined as follows:

$$\mathcal{N}_R(n, k) = \begin{cases} k \cdot 2^{n/k} & \text{if } k \leq n \\ k + n & \text{if } k > n \end{cases},$$

  implying that there exists at least one solution to $\mathsf{RGBP}(n, k)$ with non-negligible probability, given k lists of total size $\mathcal{N}_R(n, k)$. For $k > n$, the minimum list size of RGBP is $k+n$ which implies each of the n lists contains 2 elements and each of the remaining $k - n$ lists contains 1 element.

One can readily verify that $\mathcal{N}_R(n, k) > \mathcal{N}_L(n, k)$, as the inequality $\binom{\mathcal{N}_R(n,k)}{k} > 2^n$ holds in all cases. In line with intuition, RGBP is harder than LGBP at density $\delta > 1$, since any solution to RGBP is necessarily also a solution to LGBP, but not vice versa.

From Collision to Decoding. Let $\mathcal{N}(n, k)$ be the minimum list size such that there exists at least one solution to $\mathbf{GBP}(n, k)$ in expectation. We show that $\mathcal{N}(n, k) = \mathbf{\Theta}\left(k \cdot 2^{n/k}\right)$ in both regular and non-regular cases (see Theorem 6 in Appendix C). The validity of the k-XOR conjecture fundamentally relies on the requirement that $\mathcal{N}(n, k)$ be exponential in n, which holds only for $k = \mathcal{O}(1)$. Once $\mathcal{N}(n, k)$ drops to the regime where it is linear in n, one can exploit random permutations of dimension $\mathcal{N}(n, k)$ to obtain a favorable algebraic structure, e.g., all errors are in one fixed information set. In this case, ISD techniques become advantageous, and the k-XOR conjecture fails. Indeed, if $\mathcal{N}(n, k)$ is exponential in n, the probability of finding a "good" permutation in ISD algorithms would decrease dramatically, which does not undermine the k-XOR conjecture. In other words, as k increases from $\mathcal{O}(1)$ to $\mathbf{\Theta}(n)$, the worst-case $\mathbf{GBP}(n, k)$ transitions from a collision problem to the classical decoding problem of coding theory. In what follows, we elaborate on how the k-XOR conjecture breaks down when k is linear in n, namely when $k = cn$. The linear modeling makes it convenient for applying Stirling-type approximations.

Code Parameters of GBP. We can derive the following two propositions, which directly reduce GBP to SDP under specific code-rate and weight-rate parameters. This reduction theoretically reveals the fundamental difference between the regular and non-regular variants of GBP or k-XOR from a coding-theoretic perspective. For consistency, we use the notations taken from SDP in the following propositions.

Proposition 1 (Code Parameters of RGBP at Density 1). *Let* $c = \frac{\hat{k}}{\hat{n}} \in (0, 1)$ *be a constant for* $\mathbf{RGBP}(\hat{n}, \hat{k})$. *Then* $\mathbf{RGBP}(\hat{n}, \hat{k})$ *at density 1 reduces to* $\mathbf{RSDP}\big(n = \mathcal{N}_R(\hat{n}, \hat{k}),\ r = \hat{n},\ w = \hat{k}\big)$. *The corresponding weight-rate* $\omega = w/n$ *and code-rate* $\kappa = k/n$ *satisfy*

$$\begin{cases} \omega = 2^{-\frac{1}{c}}, \\ \kappa = 1 - \dfrac{\omega}{c}. \end{cases}$$

Proof. In the worst-case setting, the input list size of $\mathbf{RGBP}(\hat{n}, \hat{k})$ satisfies $\mathcal{N}_R(\hat{n}, \hat{k}) \approx \hat{k} \cdot 2^{\frac{\hat{n}}{\hat{k}}}$. Thus, the code parameters satisfy:

$$n = w \cdot 2^{\frac{r}{w}} \implies \omega = 2^{-\frac{1}{c}}.$$

The code rate κ can be derived as follows:

$$\begin{cases} r = (1 - \kappa)n \\ r = \frac{w}{c} \end{cases} \implies \kappa = 1 - \frac{\omega}{c}. \tag{5}$$

□

Proposition 2 (Code Parameters of LGBP at Density 1). *Let* $c = \frac{\hat{k}}{\hat{n}} \in (0, \frac{1}{2})$ *be a constant for* $\mathbf{LGBP}(\hat{n}, \hat{k})$. *Then* $\mathbf{LGBP}(\hat{n}, \hat{k})$ *at density 1 reduces to* $\mathbf{SDP}\big(n = \mathcal{N}_L(\hat{n}, \hat{k}),\ r = \hat{n},\ w = \hat{k}\big)$. *The corresponding weight-rate* $\omega = w/n$ *and code-rate* $\kappa = k/n$ *satisfy*

$$\begin{cases} \omega = f^{-1}(c), \\ \kappa = 1 - \frac{\omega}{c}. \end{cases}$$

where $f(\omega) := \frac{\omega}{\mathrm{H}(\omega)}$ *and* $\mathrm{H}(\cdot)$ *is the binary entropy function.*

Proof. In the worst-case setting, it is known that

$$\binom{\mathcal{N}_L(\hat{n}, \hat{k})}{\hat{k}} \approx 2^{\hat{n}} \quad \implies \quad \binom{n}{w} = 2^r. \tag{6}$$

Considering asymptotic complexity and applying Stirling's approximation $\binom{n}{w} = 2^{\mathrm{H}(w/n) \cdot n} = 2^{\mathrm{H}(\omega) \cdot n}$ in Eq. (6), we have $2^{\mathrm{H}(\omega) \cdot n} = 2^r$. This further implies:

$$\mathrm{H}(\omega) = \frac{r}{n} = \frac{w}{c \cdot n} = \frac{\omega}{c}. \tag{7}$$

Denote the inverse function of $f(\omega) = \frac{\omega}{\mathrm{H}(\omega)}$ as f^{-1}. We have $\omega = f^{-1}(c)$. The code-rate parameter $\kappa = 1 - \frac{\omega}{c}$ can be derived in the same way as in Eq. (5). □

Remark 1. In practice, the code parameters must be integers, and in the regular case an additional constraint $w \mid n$ is required. Here, we focus solely on the theoretical asymptotic complexity and therefore ignore rounding issues. We will show in subsequent proofs that when $c > 0.5$ or $c > 1$, the corresponding LGBP and RGBP admit polynomial-time algorithms, respectively. This justifies restricting the parameter c to the intervals $(0, \frac{1}{2})$ for LGBP and $(0, 1)$ for RGBP. Only within these ranges does it make sense to analyze the complexity exponent of interest. These parameter regimes are of particular importance for the complexity of k-XOR (especially to Conjecture 1) and for the security of fine-grained cryptography.

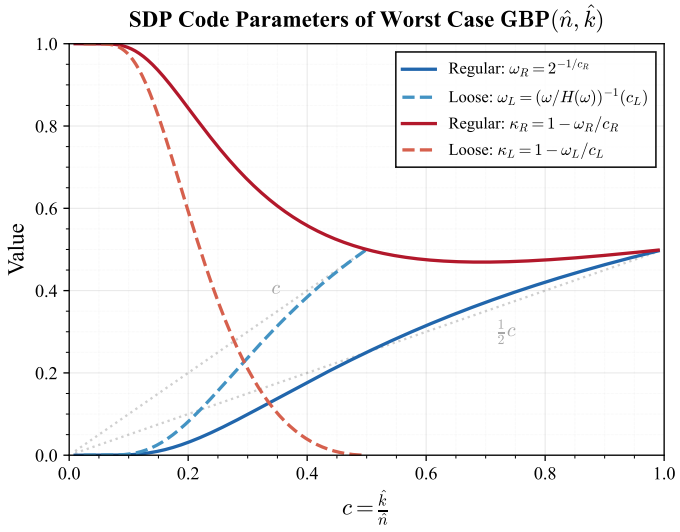


Fig. 5: The equivalent code parameters $\kappa = \frac{k}{n}$ and $\omega = \frac{w}{n}$ of $\mathbf{SDP}(n, r, w)$ (with $r = n - k$) corresponding to $\mathbf{LGBP}(\hat{n}, \hat{k})$ and $\mathbf{RGBP}(\hat{n}, \hat{k})$ at density 1.

As illustrated in Figure 5, when $c = 0$ or $c \to 0$, the weight rate is close to zero. In this regime, GBP inherently exhibits high complexity from an information-theoretic perspective. The input list size corresponding to the uniqueness bound (i.e., the code length) is exponential, and the regular and non-regular variants of GBP behave almost identically. Heuristically, we observe that as c increases, the weight rate of regular GBP gradually approaches the linear relation $y = c$, whereas in the non-regular case it approaches $y = \frac{1}{2}c$. In both cases, the code rate always satisfies the relation $\kappa = 1 - \omega/c$. Our primary interest lies in the complexity of solving SDP under these code parameters, and in particular, determining when the worst-case complexity of k-XOR becomes asymptotically smaller than $\mathcal{O}(2^{n/2})$.

Information Set Decoding for GBP. The reductions from GBP to SDP provide new approaches for studying GBP, particularly for solving GBP in the worst-case setting. Information set decoding (ISD) algorithms [7,16,40,44,49,53,57] and their regular variants [19,29] can be applied to GBP. Given $\mathbf{SDP}(n, r, w)$, let $k = n - r$. Prange's plain ISD algorithm [53] randomly permutes the columns of $\mathbf{H} \in \mathbb{F}_2^{r \times n}$ to make the first k columns error-free (i.e., the error vector e has zeros in those k positions). If correctly permuted, solving the remaining r linear equations yields a valid weight solution. The success rate is $\binom{r}{w}/\binom{n}{w}$ and the expected running time is given as:

$$\mathrm{T}_{\mathrm{isd}}(n, r, w) = \mathcal{O}\left(\frac{\binom{n}{w}}{\binom{r}{w}} \cdot r^3\right) = \widetilde{\mathcal{O}}\left(\frac{\binom{n}{w}}{\binom{r}{w}}\right)$$

For the regular case, denote the sub-block length as $b = \frac{n}{w}$. Define the following block vectors $\mathbf{c}_i \in \mathbb{F}_2^n$ for $i \in [1, w]$:

$$\mathbf{c}_i := (\underbrace{0, \ldots, 0}_{(i-1)\cdot b}, \underbrace{1, \ldots, 1}_{b}, \underbrace{0, \ldots, 0}_{(w-i)\cdot b}).$$

We can add extra w equations from regularity: $\mathbf{c}_i \cdot \mathbf{e} = 1$ for $i \in [1, w]$. Therefore, the actual code parameters after encoding regularity are $k' = k - w$ and $r' = n - k + w$. By randomly selecting $v = k'/w$ positions from each sub-block, we construct a regular information set of length k'. Assuming a favorable permutation that renders the information set error-free, solving the remaining r' equations allows for the complete recovery of the error vector with the target weight. The success rate of finding a "good" permutation is:

$$\left(\frac{\binom{b-1}{v}}{\binom{b}{v}}\right)^w = \left(\frac{b-v}{b}\right)^w = \left(1 - \frac{k-w}{n}\right)^w.$$

Finally, the permutation-based regular ISD algorithm [29, §4.1] solves $\mathbf{RSDP}(n, r, w)$ in time:

$$\mathrm{T}_{\mathrm{risd}}(n, r, w) = \widetilde{\mathcal{O}}\left(\left(1 - \frac{k-w}{n}\right)^{-w}\right) = \widetilde{\mathcal{O}}\left(\left(\frac{r+w}{n}\right)^{-w}\right)$$

Building upon the above ISD algorithms, we derive the following two theorems.

Theorem 1 (Worst-Case LGBP). *Let $k = c \cdot n$ with constant $c \in (0, 0.5)$ and let $H(p) = -p\log_2(p) - (1-p)\log_2(1-p)$ be the binary entropy function. The plain ISD algorithm solves $\mathbf{LGBP}(n, k)$ at density 1 in time $\widetilde{\mathcal{O}}\left(2^{(1-H(c))\cdot n}\right)$, which is asymptotically faster than the worst case when $c > 0.11$.*

Proof. By Proposition 2, LGBP at density 1 can be transformed into an instance of $\mathsf{SDP}(n = N, r = n, w = k)$ where $\binom{N}{k} \approx 2^n$. The complexity of solving LGBP with the plain ISD algorithm is as follows.

$$\mathrm{T}_{\mathrm{isd}} = \widetilde{\mathcal{O}}\left(\frac{\binom{N}{k}}{\binom{n}{k}}\right) = \widetilde{\mathcal{O}}\left(\frac{2^n}{\binom{n}{k}}\right) = \widetilde{\mathcal{O}}\left(\frac{2^n}{\binom{n}{k}}\right) = \widetilde{\mathcal{O}}\left(2^{(1-H(c))n}\right).$$

The last equality follows from Stirling's approximation $\binom{n}{k} = \widetilde{\mathcal{O}}(2^{nH(k/n)})$. When $c > H^{-1}(0.5) \approx 0.11$, the plain ISD algorithm solves LGBP at density 1 asymptotically faster than $\mathbf{\Theta}(2^{\frac{n}{2}})$. □

Theorem 2 (Worst-Case RGBP). *Let $k = c \cdot n$ with constant $c \in (0,1)$. The permutation-based regular ISD algorithm solves $\mathbf{RGBP}(n,k)$ at density 1 in time $\widetilde{\mathcal{O}}\left(2^{\left(1+c\log_2(\frac{c}{1+c})\right)\cdot n}\right)$, which is asymptotically faster than the worst case when $c > 0.188$.*

Proof. Let each list contain N elements, so the total input size for $\mathbf{RGBP}(n,k)$ is kN. By Proposition 1, RGBP at density 1 can be transformed into an instance of $\mathsf{RSDP}(n = kN, r = n, w = k)$ where $N^k \approx 2^n$. The complexity of solving RGBP with the plain regular ISD algorithm is as follows.

$$\mathrm{T}_{\mathrm{risd}} = \widetilde{\mathcal{O}}\left(\left(\frac{n+k}{kN}\right)^{-k}\right) = \widetilde{\mathcal{O}}\left(\left(\frac{c}{1+c}\right)^{cn} 2^n\right) = \widetilde{\mathcal{O}}\left(2^{n\cdot\left(1+c\log_2(\frac{c}{1+c})\right)}\right).$$

When $c > 0.188$, the plain regular ISD algorithm solves RGBP at density 1 asymptotically faster than $\mathbf{\Theta}(2^{\frac{n}{2}})$. □

Advanced ISD Algorithms for $\mathbf{GBP}(\hat{n}, \hat{k})$. The above theorems are based on the plain permutation-based ISD algorithms (denoted as PERM). Advanced ISD algorithms including generalized Stern's algorithm [30, 57] (enumeration-based, denoted as ENUM), BJMM algorithm [7] (representation-based, denoted as REP), and nearest-neighbor BJMM algorithm [44] (denoted as $\mathsf{REP\text{-}MO}$), can be applied to solve LGBP with better asymptotic complexity. Their regular variants are devised in [29]. The asymptotic complexity of these algorithms does not admit a clean closed-form analysis, as it depends on the optimization of extra algorithmic parameters. Following the approach of [29], we derive the optimal asymptotic complexity exponents of these ISD algorithms by modeling the free parameters as linear functions of $\hat{n}$, namely $p = c_p \cdot \hat{n}$, and then minimizing the resulting complexity exponent subject to the algorithmic constraints.

Heuristic Observations. When applied to worst-case GBP, the time complexity of advanced ISD algorithms is bounded by $\mathcal{O}(2^{\hat{n}/2})$ as shown in Figure 6. This behavior stems from the fact that both ENUM and REP algorithms inherently incorporate a meet-in-the-middle (MITM) search subroutine. From an algorithmic perspective, the enumeration-based and representation-based ISD algorithms can be viewed as hybrids of the permutation-based ISD algorithm (i.e., permutation-based Gaussian elimination) and MITM. By tuning the algorithmic parameters, these methods trade off the cost of the MITM step against that of permutation-based Gaussian elimination, thereby optimizing the overall complexity exponent. Consequently, when $c \to 0$, all advanced ISD algorithms degenerate to naive MITM, which provides supporting evidence for Conjecture 1. However, as c increases, the asymptotic complexity of solving GBP at density 1

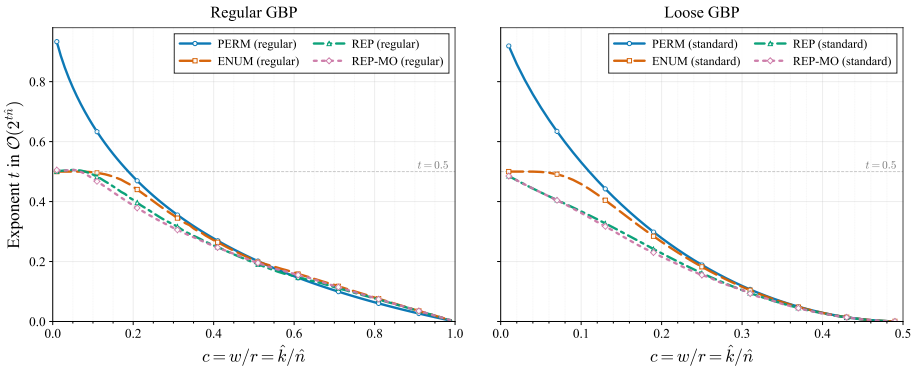


Fig. 6: The asymptotic time complexity exponents of ISD algorithms for solving $\mathbf{GBP}(\hat{n}, \hat{k})$ at density 1.

via ISD begins to deviate from $2^{0.5\hat{n}}$ in Figure 6. This implies that parameter selection in fine-grained cryptography [2] based on k-XOR must carefully constrain the relationship between $\hat{k}$ and $\hat{n}$, in order to prevent the actual security level from falling below expectations.

Concrete Cases. When $k/n = 0.5$ and $k/n = 1$, the asymptotic exponents of LGBP and RGBP vanish, leading to polynomial-time solvability. See Theorem 3 for the explicit linearization attacks. At density near 1, the complexity of the state-of-the-art ISD algorithm is close to that of average-case Wagner's algorithm for certain parameters. According to the SD estimator [28], the Both-May ISD algorithm [16] solves the special instance $\mathsf{SDP}(126, 96, 32)$, derived from $\mathbf{LGBP}(96, 2^5)$ with input list size 126, in time $2^{19.59}$. In comparison, Wagner's single-list algorithm runs in time $2^{19.32}$ but requires a much larger input list size of 2^{17}. For given parameters n and k, identifying the optimal code length N to solve GBP with density slightly greater than 1, together with the development of efficient ISD algorithms tailored to GBP, constitute significant open problems. Our proof-of-concept implementation for solving $\mathbf{LGBP}(96, 2^5)$ is available at https://github.com/tl2cents/Generalized-Birthday-Problem.

3.2 On the Average Hardness of GBP

Average-Case Hardness Comparison. For the average case, the time complexity of the k-list algorithm [59] for $\mathsf{RGBP}(n, k)$ compared to the single-list algorithm [14] for $\mathsf{LGBP}(n, k)$ scales as $\mathcal{O}(k/\log_2 k)$. This corresponds to an $\mathcal{O}_k(1)$ difference in complexity as claimed in Dinur et al.'s work [23]. Our work shows that even an $\mathcal{O}_k(1)$ difference can result in a significant complexity gap in certain k-sensitive scenarios. To illustrate this, we first present the linearization bounds for $\mathbf{GBP}(n, k)$ over $\mathbb{F}_2$. The following lemma shows that the average-case

$\mathbf{GBP}(n,k)$ can always be polynomially reduced to an instance with a smaller parameter $k' < k$.

Lemma 1 (D. Wagner [59]). *Let algorithm $\mathcal{A}$ solve $\mathbf{GBP}(n,k)$ in time $\mathcal{O}(T)$. For any $k' > k$, one can solve $\mathbf{GBP}(n,k')$ using a constant number of calls to $\mathcal{A}$ together with $\mathcal{O}(k)$ preprocessing time, which also runs in time $\mathcal{O}(T)$.*

Theorem 3 (Linearization Bound over $\mathbb{F}_2$).
When $k \geq \frac{n}{2}$, the average-case $\mathbf{LGBP}(n,k)$ can be solved in time $\mathcal{O}(n^3)$.
When $k \geq n$, the average-case $\mathbf{RGBP}(n,k)$ can be solved in time $\mathcal{O}(n^3)$.

Proof. Let $t = n+\delta$, with $\delta = \mathcal{O}(1)$. Sample t random values: $x_1, x_2, \ldots, x_t$. Let

$$\mathbf{M} := \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{t,1} & x_{t,2} & \cdots & x_{t,n} \end{bmatrix} \in \mathbb{F}_2^{t\times n},$$

where the i-th row of $\mathbf{M}$ is the binary vector of x_i. Let $\mathcal{X} := (\mathcal{X}_1, \ldots, \mathcal{X}_t)$ be the solution to $\mathcal{X} \cdot \mathbf{M} = 0$ in $\mathbb{F}_2$. Note that each $\mathcal{X}$ is the solution to $\mathbf{LGBP}(n, \mathbf{wt}(\mathcal{X}))$. We can make the matrix M be full-rank by adding new random vectors or resampling. The left kernel dimension of M is given by $t - \mathrm{rank}(\mathbf{M}) = \delta$, which implies that there are δ free variables in $\mathcal{X}$. By setting these free variables to zero, the remaining n variables are fixed, with approximately half assigned the value 1 and the other half 0. Consequently, this gives $\mathbf{wt}(\mathcal{X}) = \frac{n}{2} + \epsilon$ for some small integer ϵ, with expectation $\mathbf{E}[\epsilon] = 0$. Hence, $\mathcal{X}$ is a solution to $\mathbf{LGBP}(n, \frac{n}{2} + \epsilon)$. This shows that $\mathbf{LGBP}(n,k)$ is solvable in polynomial time $\mathcal{O}(n^3)$ when $k \geq \frac{n}{2}$.

The linearization attack against RGBP was discovered in XHASH [10] and the initial version of FSB [54]. Let $k = n$ and generate a pair of hash values: $x_i^0, x_i^1 \in L_i$ for $i \in [1,n]$. Denote $x_i' = x_i^0 \oplus x_i^1$ and $C = \oplus_{i=1}^n x_i^0$ and consider the linear system $\mathcal{X}'\mathbf{M}' = C$ where $\mathbf{M}' \in \mathbb{F}_2^{n\times n}$ is constructed by the binary vectors of x_i'. Note that each $\mathcal{X}' = (b_1, \ldots, b_n)$ corresponds to a solution $(x_1^{b_1}, x_2^{b_2}, \ldots, x_n^{b_n})$ to $\mathbf{RGBP}(n,n)$. With a small amount of brute force, we can ensure that $\mathbf{M}$ is invertible and obtain the solution $\mathcal{X}'$ to $\mathbf{RGBP}(n,n)$ in time $\mathcal{O}(n^3)$. By Lemma 1, when $k \geq n$, $\mathbf{GBP}(n,k)$ can be solved in polynomial time $\mathcal{O}(n^3)$. □

Remark 2. The linearization bounds are applicable only to GBP in $\mathbb{F}_2^n$. In the context of finite field $\mathbb{F}_p^n$, the linearization attacks are not feasible since solving the linear equation system in $\mathbb{F}_p$ yields a solution in $\mathbb{F}_p^N$ instead of $\mathbb{F}_2^N$ required by GBP (i.e., the $\mathcal{L}_\infty$-norm constraint). However, the single-list and k-list algorithms also work in the modular addition group $(\mathbb{Z}_q^n, +)$. Prior works [10,17] show that the GBP over $\mathbb{F}_m$ is closely related to hard lattice problems, with Wagner's algorithms being asymptotically optimal for the given input list size N.

Optimal Complexity within Wagner's Framework. The following theorem demonstrates that, for fixed n, the optimal asymptotic complexities for solving average-case $\mathsf{GBP}(n, 2^k)$ differ by a factor of $\sqrt{2}$ in the exponent.

Theorem 4 (Optimal Complexity). [3] *Let n be a fixed large number and k vary for $\mathsf{GBP}(n, 2^k)$. When $k = \sqrt{n} - 1$, the k-list algorithm achieves an optimal time complexity of $\mathcal{O}(2^{2\sqrt{n}})$. When $k = \sqrt{\frac{n}{2} + 1}$, the single-list algorithm achieves an optimal time complexity of $\mathcal{O}(\sqrt{\frac{n}{2} + 1} \cdot 2^{\sqrt{2n+4}-1})$ which simplifies to $\mathcal{O}(\sqrt{n} \cdot 2^{\sqrt{2n}})$.*

Proof. The k-list algorithm runs in time complexity $\mathcal{O}(2^{k+\ell+1})$ where $\ell = \frac{n}{k+1}$ is the number of bits for collision in each merge. Note that

$$2^{k+\ell+1} = 2^{k+1+\frac{n}{k+1}} \geq 2^{2\sqrt{n}}.$$

The optimal complexity $\mathcal{O}(2^{2\sqrt{n}})$ is achieved when $k = \sqrt{n} - 1$.

The single-list algorithm runs in time complexity $\mathcal{O}(k \cdot 2^{\ell+1})$, but the upper bound of k with constant success rate is $\sqrt{\frac{n}{2} + 1}$. The proof of this bound is given in Section 3.3. Let $T(k) = k \cdot 2^{\ell+1} = k \cdot 2^{\frac{n}{k+1}+1}$. Over this interval, we prove that $T(k)$ is decreasing in k for large n, i.e., $T'(k) < 0$. We have

$$\begin{aligned} T'(k) &= 2^{\frac{n}{k+1}+1} + k \cdot 2^{\frac{n}{k+1}+1} \cdot \ln 2 \cdot \left(-\frac{n}{(k+1)^2}\right) \\ &= 2^{\frac{n}{k+1}+1} \left(1 - \frac{kn \ln 2}{(k+1)^2}\right). \end{aligned} \tag{8}$$

It suffices to show $1 - \frac{kn \ln 2}{(k+1)^2} < 0$ to make Eq. (8) negative. This is equivalent to

$$1 - \frac{kn \ln 2}{(k+1)^2} < 0 \iff n \ln 2 > k + 2 + \frac{1}{k} \tag{9}$$

Since $1 \leq k \leq \sqrt{\frac{n}{2} + 1}$, we have $k + 2 + \frac{1}{k} \leq \sqrt{\frac{n}{2} + 1} + 3$. Note that $n \ln 2 > \sqrt{\frac{n}{2} + 1} + 3$ holds for $n \geq 8$. Therefore, Eq. (9) holds, and $T'(k) < 0$ for large n. Since $T(k)$ is decreasing over the interval $1 \leq k \leq \sqrt{\frac{n}{2} + 1}$, the optimal time complexity $\mathcal{O}(\sqrt{\frac{n}{2} + 1} \cdot 2^{\sqrt{2n+4}-1})$ is achieved when $k = \sqrt{\frac{n}{2} + 1}$. □

Wagner's Algorithmic Framework over $\left(\mathbb{Z}_q^m, +\right)$. Wagner [59] extended the k-list algorithm to solve RGBP in the finite field $\mathbb{Z}_q^m$. This extension can be directly applied to the single-list algorithm as well. We merely need to modify the collision logic within the self-merging operation. For $\mathbb{Z}_{2^m}$, the collision logic is defined as $(x_1 + x_2) \bmod 2^\ell = 0$. In the general case of $\mathbb{Z}_q$, the collision logic is defined as $(x_1 + x_2) \bmod q \in [-\frac{q}{2^{\ell+1}}, \frac{q}{2^{\ell+1}} - 1]$. These observations can easily extend Wagner's algorithmic framework to $\mathbb{Z}_q^m$ and Theorem 4 also holds true in

[3] In practical implementations, it is required that both k and ℓ are integers.

$\mathbb{Z}_q^m$ if we replace n with $m \cdot \log q$. Theorem 4 thus admits broader applications. In particular, it can be directly extended to the classical k-SUM problem over $\mathbb{Z}_m$. Moreover, in the context of cryptanalysis, the complexity gap revealed by this theorem corresponds to the complexities of preimage attacks $\mathcal{O}(2^{2\sqrt{n}})$ and collision attacks $\mathcal{O}(2^{\sqrt{2n}})$ on incremental hash functions (see Section 4).

3.3 Upper Bound k for the Single-List Algorithm

The single-list algorithm fails to produce two valid solutions when k becomes large. However, prior works [3, 21] do not explicitly quantify this upper bound. This issue stems from invalid collisions within self-merging, where we encounter index overlaps, i.e., multiple inclusions of the same index. Denote the index vectors involved in the self-merge operation as i_1, i_2.

1. **Trivial Collisions:** Every index occurring in the index vectors i_1 and i_2 is counted an even number of times. The most trivial case is choosing the same item twice: $i_1 = i_2$.
2. **Non-trivial Collisions:** The index vectors i_1, i_2 are not trivial collisions but share at least one common index, i.e., $i_1 \cap i_2 \neq \emptyset$.

To avoid an exponential explosion in the list size, trivial collisions (at least the most trivial case: $i_1 = i_2$) must be discarded. After self-merging, the size of the index vector doubles and the probability of index collisions increases. Consequently, the expected number of solutions for Algorithm 1 decreases with increasing k, which is detailed by the following lemma.

Lemma 2 (Expected Number of Solutions). (cf. [21, §2.3]) *The expected number of valid solutions produced by Algorithm 1 is*

$$\mathbf{E}\left[\left|L^{(k)}\right|\right] = \frac{2\binom{N}{2^k}(2^k)!}{N^{2^k}}. \tag{N}$$

Proof. We present a simplified proof in Appendix B.2.

Lemma 3 (Distribution of Solutions). (cf. [21, §2.5]) *Let $p = 2^{-\frac{n}{k+1}}$ be the collision probability and $\lambda = \mathbf{E}\left[\left|L^{(k)}\right|\right]$ be the expected number of solutions. If $\lambda \ll \frac{1}{p}$, the solution distribution is approximately Poisson with parameter λ.*

Lemma 4 (Approximate Number of Solutions). *When $2^k \ll N$, a precise approximation of the expected number of solutions for Algorithm 1 is*

$$\mathbf{E}\left[\left|L^{(k)}\right|\right] \approx 2\exp\left(-\frac{2^{2k}-2^k}{2N}\right).$$

Proof. Expanding $\binom{N}{2^k}$ in Eq. (N), we derive:

$$\mathbf{E}\left[\left|L^{(k)}\right|\right] = 2\frac{\prod_{i=0}^{2^k-1}(N-i)}{N^{2^k}} = 2\prod_{i=0}^{2^k-1}(1-\frac{i}{N}).$$

Take the natural logarithm of the expression:

$$\ln\left(\mathbf{E}\left[\left|L^{(k)}\right|\right]\right) = \ln 2 + \sum_{i=1}^{2^k-1} \ln\left(1 - \frac{i}{N}\right).$$

Apply the Taylor expansion $\ln(1-x) = -x - \frac{x^2}{2} - \frac{x^3}{3} - \cdots$:

$$\begin{aligned}\ln\left(\mathbf{E}\left[\left|L^{(k)}\right|\right]\right) &= \ln 2 - \sum_{i=0}^{2^k-1} \frac{i}{N} - \frac{1}{2}\sum_{i=0}^{2^k-1} (\frac{i}{N})^2 - \cdots \\ &= \ln 2 - \frac{1}{N}\cdot\frac{(2^k-1)2^k}{2} - \frac{1}{2N^2}\cdot\frac{(2^k-1)2^k(2^{k+1}-1)}{6} - \cdots\end{aligned}$$

The terms decrease as follows: (1) The first term is $\mathcal{O}(\frac{2^{2k}}{N})$; (2) The second term is $\mathcal{O}(\frac{2^{3k}}{N^2}) = \mathcal{O}(\frac{2^{2k}}{N}\cdot\frac{2^k}{N})$. Since $\frac{2^k}{N} \to 0$, the higher-order terms, including $\mathcal{O}(\frac{2^{3k}}{N^2})$ and beyond, can be safely neglected.

$$\ln\left(\mathbf{E}\left[\left|L^{(k)}\right|\right]\right) \approx \ln 2 - \frac{(2^k-1)2^k}{2N} \implies \mathbf{E}\left[\left|L^{(k)}\right|\right] \approx 2\exp\left(-\frac{2^{2k}-2^k}{2N}\right).$$

□

The next step is to establish the upper bound on k for the single-list algorithm.

Theorem 5 (Upper Bounds of k). *When $k \geq \sqrt{n+1}$, Algorithm 1 generates no solutions. When $k \leq \sqrt{\frac{n}{2}+1}$, Algorithm 1 generates at least one solution to* $\mathbf{LGBP}(n, 2^k)$ *with non-negligible probability (greater than $1 - e^{-\frac{2}{e}}$).*

Proof. If $k \geq \sqrt{n+1}$, the initial list size N of $L^{(0)}$ satisfies:

$$N = 2^{\ell+1} = 2^{\frac{n}{k+1}+1} \leq 2^{\frac{n}{\sqrt{n+1}+1}+1} = 2^{\sqrt{n+1}} \leq 2^k.$$

There is no valid solution to $\mathbf{LGBP}(n, 2^k)$ since the list size is less than 2^k. If $k \leq \sqrt{\frac{n}{2}+1}$, the initial size $N = 2^{\ell+1} \geq 2^{\sqrt{2n+4}-1} \gg 2^k$. By Lemma 4, we have:

$$\begin{aligned}\mathbf{E}\left[\left|L^{(k)}\right|\right] &\approx 2\exp\left(-\frac{2^{2k}-2^k}{2N}\right) \\ &\geq 2\exp\left(-\frac{2^{2k}}{2N}\right) = 2\exp\left(-2^{2k-2-\frac{n}{k+1}}\right) \\ &\geq 2\exp\left(-2^{2\sqrt{\frac{n}{2}+1}-2-\frac{n}{\sqrt{\frac{n}{2}+1}+1}}\right) = 2e^{-2^0} = \frac{2}{e}.\end{aligned}$$

By Lemma 3, the solution distribution is approximately Poisson with parameter $\lambda = \mathbf{E}\left[\left|L^{(k)}\right|\right]$. Thus, the probability of generating at least one solution is:

$$\Pr\left[\left|L^{(k)}\right| \geq 1\right] \approx \sum_{k=1}^{\infty} \frac{\lambda^k e^{-\lambda}}{k!} = 1 - e^{-\lambda} \geq 1 - e^{-\frac{2}{e}}.$$

□

The implied optimal complexity in Theorem 4 provides new benchmarks for the security lower bound of incremental hash: single-list algorithm for collision attacks and k-list algorithm for preimage attacks (see Section 4). The above lemmas and theorems have been carefully verified in our proof-of-concept implementation. Some experimental results are presented in Appendix E.

Remark 3. A pending question is whether the check of $i_1 \cap i_2 = \emptyset$ is necessary in Algorithm 1. The algorithm of [21] explicitly checks whether $i_1 \cap i_2 = \emptyset$. However, the authors do not explain how this disjointness test can be performed in constant time. To the best of our knowledge, this check costs $\mathcal{O}(2^k)$ time in the worst case, thereby increasing the running time to $\mathcal{O}(2^k N)$. The practical single-list algorithm in Equihash [14] does not verify the condition $i_1 \cap i_2 = \emptyset$, but ensures that two distinct list items are selected, i.e., $i_1 \neq i_2$ in our given Algorithm 1. However, it does not work well when $k = \sqrt{n/2+1}$. Our work shows trivial collisions must be filtered to avoid list size explosion, especially when $k \approx \sqrt{\frac{n}{2}+1}$. Non-trivial collisions can be maintained to generate secondary solutions. Small optimizations are proposed to handle index collisions. This is further explored in Appendix E.

4 New Attacks on Incremental Hash

In this section, our work clarifies how *regularity* affects the security and functionality of the ID-based IncHash [10] (specifically, AdHASH and LtHASH in the unlimited block setting). For simplicity, we utilize iSHAKE (an instantiation of LtHASH) as a concrete example to elucidate our attack that results in a theoretical gap of about 70 bits regarding the lower bound (see Table 2). We find that regularity, when instantiated through limited sequential IDs, can withstand LGBP-based attacks and yield higher security bounds. However, this regularity also constrains the available types of incremental operations.

4.1 Incremental Hash: iSHAKE

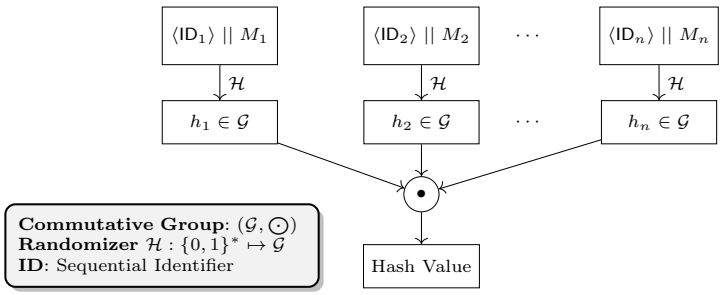


Fig. 7: The ID-based IncHash paradigm [10] in a commutative group.

Figure 7 shows the ID-based incremental hash paradigm based on a commutative group $(\mathcal{G}, \bigodot)$ where the computational effort for updating is proportional only to the size of the modified data. Let the old hash be $Y = h_1 \bigodot h_2 \cdots \bigodot h_n$. When we update the i-th block from M_i to M_i', we can simply compute $Y' = Y \bigodot h_i^{-1} \bigodot h_i'$ as the new hash where $h_i \bigodot h_i^{-1} = \mathbf{1}_{\mathcal{G}}$. The collision and preimage attacks on this incremental hash can be easily transformed into solving the generalized birthday problem over the group $(\mathcal{G}, \bigodot)$. The k-list algorithm, which can be extended to $\mathbb{Z}_q^m$, is an important lower bound estimator for the following two cases:

1. AdHASH: $\mathcal{G} := \mathbb{Z}_q$ and $\bigodot$ is defined as the modular addition in $\mathbb{Z}_q$.
2. LtHASH: $\mathcal{G} := \mathbb{Z}_q^m$ and $\bigodot$ is defined as the vector modular addition in $\mathbb{Z}_q^m$.

iSHAKE. Mihajloska et al. [46] proposed the incremental hash iSHAKE based on the generalized birthday problem and SHA-3. Let the output bit length n be a multiple of 64. Let $\mathcal{H} : \{0,1\}^* \mapsto \{0,1\}^n$ be the underlying hash function which is SHAKE from the NIST Draft FIPS-202 [26]. iSHAKE uses efficient word-wise addition in the commutative group $\left((\mathbb{Z}_{2^{64}})^{n/64}, \boxplus_{64}\right)$ to combine hashes of different blocks. Denote SHAKE hash function by $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$. Given a sequence of blocks $M_1, M_2, \ldots, M_K$ with $K \leq K_{max}$, the iSHAKE(n, K_{max}) hash is computed as follows:

$$y = \mathcal{H}(M_1||ID_1) \boxplus_{64} \mathcal{H}(M_2||ID_2) \boxplus_{64} \cdots \boxplus_{64} \mathcal{H}(M_K||ID_K).$$

Denote the augmented message as $\overline{M_i} = M_i||ID_i$. There are two practical settings for iSHAKE: fixed-size data and variable-size data. In the fixed-size setting, the ID_i is a fixed-length string $\langle i \rangle$. The input message blocks have the same size, i.e., b bit. In the variable-size setting, the message blocks may differ in length and the ID_i consists of two parts: $(BN_i, \mathrm{ptr}_{BN_{i+1}})$, where BN_i is the current block nonce and $\mathrm{ptr}_{BN_{i+1}}$ is the pointer to the next block.

Considering the combination process of K blocks: $y = y_1 \boxplus_{64} y_2 \boxplus_{64} \cdots \boxplus_{64} y_K$, the preimage attack on iSHAKE is to solve RGBP in $\left(\mathbb{Z}_{2^{64}}^{n/64}, \boxplus_{64}\right)$ with $K \leq K_{max}$ whose hardness is equivalent to $\mathbf{RGBP}(n, K)$ in $(\mathbb{Z}_2^n, +)$ under Wagner's algorithmic framework. Mihajloska et al. gave two sets of parameters for iSHAKE, where the security lower bound is estimated under $(n, K_{max}) = (n_{max}, \infty)$:

- iSHAKE128: The output hash length is in the range $(n_{min} = 2688, n_{max} = 4160)$; the maximum allowed number of blocks for n_{min} is $K_{max} = 2^{25}$.
- iSHAKE256: The output hash length is in the range $(n_{min} = 6528, n_{max} = 16512)$; the maximum allowed number of blocks for n_{min} is $K_{max} = 2^{28}$.

The hard problem of fixed-size iSHAKE is RGBP provided it is implemented correctly, as this scheme necessitates the K messages from K different lists, meaning the IDs must be $\{1, 2, \ldots, K\}$. For the variable-size iSHAKE, however, the underlying problem is typically LGBP. Mihajloska et al. [46] utilize the time complexity of the k-list algorithm to estimate the security bound of iSHAKE as

shown in Table 2. We propose a non-trivial technique to transform the collision problem of iSHAKE into LGBP, particularly efficient in the unlimited block setting ($K_{max} = \infty$). By Theorem 4, the security strength of iSHAKE will be reduced by approximately 70 bits as shown in Table 2.

Table 2: Security bounds of iSHAKE

iSHAKE	Mode	(n, K_{max})	Security Bound†		Optimal K	
			In [46]	Our Work	In [46]	Our Work
128	Limited Block*	$(2688, 2^{25})$	$2^{128.384}$	$\mathbf{2^{113.105}}$	2^{25}	$\mathbf{2^{24}}$
	Unlimited Block	$(4160, \infty)$	2^{128}	$\mathbf{2^{96.927}}$	2^{63}	$\mathbf{2^{45}}$
256	Limited Block*	$(6528, 2^{28})$	$2^{253.103}$	$\mathbf{2^{238.898}}$	2^{28}	$\mathbf{2^{27}}$
	Unlimited Block	$(16512, \infty)$	2^{256}	$\mathbf{2^{188.942}}$	2^{127}	$\mathbf{2^{90}}$

* Variable-size setting † Collision attack

4.2 Our Collision Attack on iSHAKE

We propose a novel *pre-combining* technique that reduces the collision attack on iSHAKE to solving LGBP. Given iSHAKE(n, ∞), we choose the number of blocks $K = 2^k$ where $k \leq \sqrt{\frac{n}{2} + 1}$. Let $N = 2^{\frac{n}{k+1}+1}$. Denote the word-wise minus operation in $\mathbb{Z}_{2^{64}}^{n/64}$ as $\boxminus_{64}$. Our collision attack on iSHAKE proceeds as follows:

1. **Pre-Combining.** Generate block pairs $\left(\overline{M_i^0} = M_i^0||ID_i, \overline{M_i^1} = M_i^1||ID_i\right)$ for $i \in [1, N]$ and compute the hash value list $L = \{y_i \mid y_i = \mathcal{H}(\overline{M_i^0}) \boxminus_{64} \mathcal{H}(\overline{M_i^1}), i \in [1, N]\}$.
2. **LGBP Phase.** Apply the single-list algorithm to list L until finding a solution such that
$$y_{i_1} \boxplus_{64} y_{i_2} \boxplus_{64} \cdots \boxplus_{64} y_{i_K} = 0.$$
Without loss of generality, let $i_1, i_2, \ldots, i_K$ be in increasing order. If no feasible solution is found, return to step 1 to generate a new list.
3. **Reconstruction.** Make all blocks identical except for $i_1, \ldots, i_K$ in the following two sets of messages:
$$\hat{M}_0 = \left(M_1, M_2, \ldots, M_{i_j}^0, \ldots, M_{i_K}^0\right),$$
$$\hat{M}_1 = \left(M_1, M_2, \ldots, M_{i_j}^1, \ldots, M_{i_K}^1\right).$$
These two messages satisfy iSHAKE$(\hat{M}_0) =$ iSHAKE$(\hat{M}_1)$.

By Theorem 5, since $k \leq \sqrt{\frac{n}{2} + 1}$, step 2 finishes within a finite number of iterations, i.e., $\mathcal{O}(1)$. The overall time complexity of this attack is $\mathcal{O}(kN)$ with

the solution messages extending to a length of around N. When $k = \sqrt{\frac{n}{2}+1}$, our attack achieves the optimal time complexity $\mathcal{O}(\sqrt{\frac{n}{2}+1} \cdot 2^{\sqrt{2n+4}-1})$. The security lower bounds of iSHAKE128 and iSHAKE256 decrease from 2^{128} to $2^{96.9}$ and from 2^{256} to $2^{188.9}$ respectively as detailed in Table 2.

Attack with Fewer Blocks. In the fixed-size case, when $K \leq 2^{\sqrt{\frac{n}{2}+1}}$, $K^2 \lesssim N$ holds true. We generate IDs from $[1, B]$ where $B \geq K^2$. By the birthday paradox, the K messages generated from the single-list algorithm contain IDs randomly sampled from $[1, B]$ and will not collide in the ID field with non-negligible probability. The number of blocks needed in our attack is reduced to $\mathcal{O}(K^2)$. In the variable-size iSHAKE, the ID field contains: the current block nonce denoted as BN_i and a pointer to the next block nonce denoted as $\mathrm{ptr}_{BN_{i+1}}$. Given the solution $\overline{M^b_{i_1}}, \overline{M^b_{i_2}}, \ldots, \overline{M^b_{i_K}}$ for $b \in \{0,1\}$, we insert a new block between i_j and i_{j+1} with its ID field being $\left(^*\mathrm{ptr}_{BN_{i_{j}+1}}, \mathrm{ptr}_{BN_{i_{j+1}}}\right)$, which bridges $\overline{M^b_{i_j}}$ and $\overline{M^b_{i_{j+1}}}$. In this case, our attack generates collision messages with a block number of $2K$.

Considering fixed-size iSHAKE in the limited block case, the LGBP-based collision attack can generate valid collision messages with $K \leq \sqrt{K_{max}}$ which is generally not superior to the k-list algorithm. However, regarding the variable-size iSHAKE, our attack can generate valid collision messages with $K \leq K_{max}/2$ and outperforms the k-list algorithm. This indicates that security lower bounds of variable-size iSHAKE128 and iSHAKE256 in the limited block case decrease from $2^{128.4}$ to $2^{113.1}$ and from $2^{253.1}$ to $2^{238.9}$ respectively as detailed in Table 2.

Remark 4 (Pointer Size.). We reasonably assume that, in the variable-size scenario, the space for pointers is sufficiently large, as the nonce is typically randomly generated. To ensure that ID collisions do not occur in common usage, the space for IDs must be larger than K^2_{max}. By examining an open-source C implementation[4], we find that the data structure of $(BN_i, \mathrm{ptr}_{BN_i})$ is defined as (uint64, uint64). This provides about 2^{64} possible IDs, thus ensuring the feasibility of our attack. In the fixed-size setting, the data type of $\langle i \rangle$ is uint64 and a lack of length check may also result in a diminishment of security strength.

4.3 Insights into GBP-Related Schemes

Pair Chaining Incremental Hash. Goi et al. [15] proposed the pair chaining incremental hash function (PCIHF). Let q be an n-bit modulus and $\mathbf{H} : \{0,1\}^* \mapsto \{0,1\}^n$ be a cryptographic hash. Let $x = (x_1, x_2, \ldots, x_k)$ where x_i is a fixed-length block. The PCIHF is defined as follows:

$$\mathcal{H}(x) \stackrel{\text{def}}{=} \sum_{i=1}^{k-1} \mathbf{H}(x_i, x_{i+1}) \bmod q.$$

[4] https://github.com/jaimeperez/iSHAKE

The advantages of PCIHF include: (a) No index counter payload and convenient for updating; (b) The updating process is oblivious and completely tamper-proof secure. Wagner's attack [59] on PCIHF transforms it into the generalized birthday problem by setting $x_{2i} = 0$ for $i \in [1, k/2]$. Let $k = 2m$. The preimage attack on PCIHF involves solving $h = \mathbf{H}(x_1, 0) + \sum_{j=1}^{m-1} (\mathbf{H}(0, x_{2j+1}) + \mathbf{H}(x_{2j+1}, 0)) \bmod q$. We can fix h, $\mathbf{H}(x_1, 0)$ and generate a hash list by $\bar{y}_i = \mathbf{H}(0, y_i) + \mathbf{H}(y_i, 0)$. Then we have to solve an instance of LGBP. Wagner's k-list attack has an optimal time complexity of $\mathcal{O}(2^{2\sqrt{n}})$ with $k = 2^{\sqrt{n}-1}$. Our work indicates that the single-list attack has a concrete time complexity of $\mathcal{O}(\sqrt{\frac{n}{2}+1} \cdot 2^{\sqrt{2n+4}-1})$ (asymptotically $\mathcal{O}(\sqrt{n} \cdot 2^{\sqrt{2n}})$) with $k = 2^{\sqrt{\frac{n}{2}+1}}$, which is more efficient in both memory and time than Wagner's attack.

Incremental Multiset Hash. Clarke et al. [20] proposed the incremental multi-set hash on multi-sets (or sets) which can be used to make an offline memory integrity checker secure against active adversaries. Let M be a multi-set of elements in $B = \{0,1\}^m$ and denote the multiplicity of b in M as M_b, i.e., the number of occurrences of b in M. Let n and L be two positive moduli. Let $\mathbf{H}_K : \{0,1\}^{m+1} \mapsto \mathbb{Z}_n^\ell$ be a keyed cryptographic hash. Then the MSet-Add-Hash is defined as follows:

$$\mathcal{H}_K(M) = \left[\mathbf{H}_K(0, r) + \sum_{b \in B} M_b \mathbf{H}_K(1, b) \mod n; \sum_{b \in B} M_b \bmod L; r \right] \Bigg|_{r \leftarrow B}$$

where r is a random nonce and adversaries can only query the oracle $\mathcal{H}_K(\cdot)$. We assert that $\mathcal{H}_K(M)$ resembles an HMAC [8] rather than a hash since the key K must be kept secret. When given access to the oracle $\mathbf{H}_K(\cdot)$, the attack on MSet-Add-Hash is closely related to LGBP, i.e., solving equation $\mathbf{H}_K(0, r) + \sum_{b \in B} M_b \mathbf{H}_K(1, b) \equiv 0 \bmod n$. The secret key K and random nonce r prevent adversaries from directly querying the oracle of $\mathbf{H}_K(\cdot)$ and thus mitigate GBP-based attacks. This offers invaluable insight for the design of an incremental HMAC. Nevertheless, the key holder (or anyone who obtains the oracle of $\mathbf{H}_K(\cdot)$) can find collisions/preimages by solving LGBP. Due to the introduction of M_b, this will result in a general linear attack over $\mathbb{Z}_q^m$. For the most efficient scheme MSet-XOR-Hash, the linearization attack can generate collision messages of rather short length, about $\frac{\ell}{2}$ elements in the multi-set. In the keyed scenario, the GBP-based incremental cryptography may exhibit superior performance.

From a cryptographic design standpoint, our theoretical insights help clarify how *regularity* affects security and functionality. For instance, PCIHF [15] (based on LGBP) and iSHAKE [46] (based on RGBP) exhibit differing trade-offs. While RGBP offers stronger security under identical parameters, it supports fewer incremental operations (only replacement) due to the continuous ID requirement, whereas PCIHF allows for incremental replacement, deletion, and insertion. These distinctions underscore the broader significance of our regularity-based analysis for designing future cryptographic protocols.

5 Refining Equihash

Equihash has become a popular PoW algorithm in the blockchain realm since its release in 2017. However, the index pointer technique [27] and off-chip technique [5] are introduced to greatly weaken its ASIC-resistance. Based on our previous analysis, we first clarify two misconceptions in Equihash as follows:

1. Equihash relies on a newly derived tree-binding non-regular GBP, rather than the original regular GBP. Its memory hardness relies on the self-merge problem.
2. The parameter k of Equihash$(n, 2^k)$ must be bounded by $\sqrt{n/2+1}$. Thus, parameters like $(n, k) = (192, 11)$ should not be considered.

In this section, we first address the open problem of the memory hardness of Equihash. We then propose refinements to Equihash and analyze the resulting improvements in ASIC resistance.

5.1 Rethinking the Memory Hardness of Equihash.

Though the index pointer technique [27] and off-chip technique [5] are introduced to design multi-chip ASIC solvers, Equihash still provides considerable memory hardness on both capacity and bandwidth and completely resists single-chip ASIC implementations. In most contexts, memory hardness refers specifically to memory capacity rather than bandwidth or latency. We adopt this convention in the following discussion. Our concrete memory analysis is given in Appendix D. The state-of-the-art memory complexity of Wagner's algorithms tailored for tree-binding GBP is summarized as follows.

Proposition 3 (Complexity of k-list Algorithm). *The k-list algorithm with index-trimming for* $\mathbf{RGBP}(n, 2^k)$ *runs in time* $\mathcal{O}(2^k \cdot N)$ *and requires a peak memory of at least* $\mathcal{O}\left(\left(\frac{k^2+5k+2}{4}\ell + 2^{k-1}\right) N\right)$ *bits.*

Proposition 4 (Complexity of Single-list Algorithm [58]). *The single-list algorithm implemented with index pointer for* $\mathbf{LGBP}(n, 2^k)$ *runs in time* $\mathcal{O}(kN)$ *and requires a peak memory of at least* $\mathcal{O}(n \cdot N)$ *bits.*

The solution generated from Wagner's algorithmic framework adheres to constraints Eq. (4) which memory-efficient linearization and ISD algorithms cannot exploit. We pinpoint that the memory-hardness of Equihash relies on the tree-binding constraints Eq. (4). The tree-binding LGBP is a newly derived problem which is related to a variant of hash collision problem.

Memory Capacity The memory capacity hardness of Equihash is rooted in the self-merge operation at each layer. Notably, the hash collision problem is not memory-hard considering the rho-collision algorithm from [18,52] which can find hash collisions in time $\mathcal{O}(2^{n/2})$ and memory $\mathcal{O}(1)$. Moreover, a parallelizable and

memory-efficient alternative exists: the lambda method (or distinguished points method) [60], which also runs in $\mathcal{O}(2^{n/2})$ time. We formally define the self-merge problem in the single-list algorithm as follows.

Definition 6 (Self-Merge Problem). *Given a list L of size $N = 2^{\ell+1}$, find **all** distinct pairs of $x \in L, y \in L$ such that x, y collide on ℓ bits.*

To the best of our knowledge, no memory-efficient algorithm currently exists that solves the self-merge problem in linear time. A memory-efficient algorithm based on the rho collision method [52] can find a single solution to the self-merge problem. We simulate a pseudorandom function $f : L \mapsto L$, initialized with a seed s. According to the birthday paradox, a collision is expected to occur within $k = 2^{\ell/2}$ iterations of $f(s), f^2(s), \ldots, f^k(s)$ with high probability. By employing a memory-efficient cycle detection algorithm such as Floyd's cycle-finding algorithm, a single solution to the self-merge problem can be found in $\mathcal{O}(1)$ space and $\mathcal{O}(2^{\ell/2})$ time. Nevertheless, this approach is unsuitable for finding all solutions due to its probabilistic nature. The memory capacity hardness of Equihash primarily relies on the sequential self-merge problem.

5.2 Requihash: Regular Equihash

We elucidate how minimal modifications within the existing Equihash framework can align it with RGBP. The difficulty filter is omitted in the following proposal.

Proposal 1 (Regular Equihash) *Requihash$(n, K = 2^k)$: Let $\ell = \frac{n}{k+1}, N = 2^\ell$ and $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$ be a cryptographic hash function. The prover (miner) selects a seed I, e.g. the hash of the working block, and finds a nonce V and $(\ell + k)$-bit $x_1, x_2, \ldots, x_K$ such that:*

$$\begin{aligned} \mathcal{H}(I||V||x_1) \oplus \mathcal{H}(I||V||x_2) \oplus \cdots \oplus \mathcal{H}(I||V||x_K) = 0, \\ x_i \equiv i - 1 \mod K, \quad \forall i \in [1, K], \end{aligned} \tag{S}$$

with the hash solution following the algorithm-binding constraints defined in Eq. (4).

We estimate the complexities of Equihash and Requihash based on Propositions 3 and 4 and present specific parameters in Table 3. The Requihash proposal is largely consistent with Equihash; the only difference lies in the addition of sequential constraints (S), which will render the single-list algorithm ineffective for Requihash. It is noteworthy that this modification only impacts the client-side logic and the increase in verification workload is negligible, thereby ensuring that the upgrade can be implemented with minimal disruption to the existing blockchain infrastructure. Since the index field can be omitted during network transmission by utilizing a predefined packet structure, the solution size of Requihash is even smaller in the network, i.e., $2^k \cdot \ell$ bits compared to $2^k \cdot (\ell + 1)$ bits in Equihash. A small solution size reduces network bandwidth consumption and mitigates the risk of client-side DoS attacks. In the following discussion

and Table 3, the term "solution size" refers to the number of bytes required for network transmission, e.g., excluding the k-bit index information of Requihash. Additionally, the nonce size is disregarded, as it is a customizable parameter.

For all parameters (n, k) of Equihash with $k \geq 5$, our new scheme can increase the peak memory by at least 100%, without incurring drawbacks. Choosing larger values of k further amplifies the advantage of Requihash. Upgrading from the widely adopted Equihash$(200, 2^9)$ to Requihash$(200, 2^9)$ slightly decreases the solution size from 1344 bytes to 1280 bytes, while raising the time complexity by a factor of 84.4 and memory complexity by a factor of 4.6 (**increasing the peak memory usage from 49 MB to 223 MB**). The increase in time complexity can be offset by adjusting the difficulty parameter, thus enabling a substantial improvement in memory hardness especially for large k.

Table 3: Parameters of Equihash and Requihash.

(n, K)	Equihash			Requihash		
	Time	**Mem.**	**Sol-Size**	**Time**	**Mem.**	**Sol-Size**
$(96, 2^5)$	$2^{19.3}$	$2^{23.6}$	68 B	$2^{22.6}$	$2^{24.8}$	64 B
$(128, 2^7)$	$2^{19.8}$	$2^{24.0}$	272 B	$2^{24.6}$	$2^{25.7}$	256 B
$(160, 2^9)$	$2^{20.2}$	$2^{24.3}$	1088 B	$2^{26.6}$	$2^{26.6}$	1024 B
$(144, 2^5)$	$2^{27.3}$	$2^{32.2}$	100 B	$2^{30.6}$	$2^{33.4}$	96 B
$(150, 2^5)$	$2^{28.3}$	$2^{33.2}$	104 B	$2^{31.6}$	$2^{34.4}$	100 B
$(200, 2^9)$	$2^{24.2}$	$2^{28.6}$	1344 B	$2^{30.6}$	$2^{30.8}$	1280 B
$(288, 2^8)$	$2^{36.0}$	$2^{41.2}$	1056 B	$2^{41.6}$	$2^{42.9}$	1024 B

Equihash and Requihash. As a straight extension of Equihash, Requihash is more memory/time-hard than Equihash under the same parameters or solution size. Notably, the parameter k in Requihash is not constrained by the bound $\sqrt{n/2+1}$, allowing for more flexible selection of the parameters (n, k). As k increases, the improvement in memory hardness offered by Requihash over Equihash becomes more pronounced. In practical implementations, the index field of Requihash can effectively eliminate duplicate solutions. Redundant solutions resulting from swapping two child nodes in the binary tree can be avoided through a canonical ordering without additional checks. Most importantly, Requihash is immune to the index-pointer optimization that favors ASIC implementations.

5.3 ASIC-Resistance of Requihash

Our analysis of ASIC resistance is based on the ASIC solver for Equihash proposed by Bai et al. [5]. Additionally, since the core subroutines of Requihash are

nearly identical to those of Equihash, and the generic time-memory trade-off and parallelism techniques [12,13,38] mentioned in Equihash are originally proposed for the k-list algorithm, we do not repeat them here. Our proposal adopts the algorithm-binding technique and ensures that, for the given list size, there exists one solution in expectation. Consequently, Requihash is amortization-free.

Memory Hardness. ASIC resistance largely relies on memory hardness. Requihash provides better memory hardness than Equihash in terms of both capacity (memory complexity in Table 3) and bandwidth. Requihash performs $2^k - 1$ merge operations, each involving $\mathcal{O}(N)$ memory accesses, whereas Equihash requires only k self-merge operations with $\mathcal{O}(N)$ memory accesses each. For large k, this leads to a substantial increase in bandwidth consumption for Requihash.

One major factor for the weakened ASIC resistance of Equihash lies in the emergence of index-pointer technique. The ASIC solver for general Equihash by Bai et al. [5] heavily depends on this technique. On one hand, it significantly reduces peak memory. On the other hand, the index-pointer technique fixes both the memory access pattern and the memory bound for each self-merge subroutine. This reduces logic and memory access, thereby increasing the advantage for ASIC adversaries. Below, we analyze the implications of index-pointer techniques for Requihash.

Index Pointer for k-List Algorithm. The index-pointer technique can also be applied to the k-list algorithm. However, this will not reduce the memory complexity due to the complete binary tree structure of the k-list algorithm. Consider the list in the i-th layer of the k-list algorithm, the memory cost of storing indices is $2^i \ell N/2 = 2^{i-1}\ell N$ bits, whereas the memory cost of storing index pointers is $\sum_{j=1}^{i} 2^{i-j}(2\ell)N/2 = (2^i - 1)\ell N$ bits. Thus, index pointers at least double the memory cost of storing indices! When using index pointers, peak memory occurs during the merge of the last two leaves. For layer i where $i \in [1, k-1]$, $(2^{k-i} - 1)$ lists of size $\mathcal{O}(N/2)$ store index pointers (each of size 2ℓ bits). The total memory cost of index pointers is $M_{ip} = \sum_{i=1}^{k-1}(2^{k-i} - 1) \cdot 2\ell \cdot \frac{N}{2} = (2^k - k - 1)\ell N$ bits. The overall memory complexity of the k-list algorithm with index pointer is given by:

$$\begin{aligned} M &= \underbrace{n \cdot \frac{N}{2} \cdot 2 + \sum_{i=1}^{k-1} (n - i\ell)\frac{N}{2}}_{\text{Memory for XOR values}} + M_{ip} \\ &= \left(\frac{k^2 + k - 2}{4} + 2^k\right) \ell N. \end{aligned}$$

The k-list algorithm significantly mitigates the index-pointer technique. Even if an ASIC solver employs index pointers to reduce logic complexity and memory access cost, the increase in memory complexity still provides considerable ASIC resistance, e.g. 12 times memory penalty for Requihash$(200, 2^9)$.

6 Open Problems and Future Work

GBP is a problem that intersects multiple areas of cryptography. Our regularity-based analysis may also open new avenues for optimization in domains such as the ROS problem [11], the low-weight parity-check problem [34] and fine-grained cryptography [23, 39]. Beyond these, we outline several open problems that we believe will play a central role in advancing the study of k-SUM and GBP.

1. Is the bound $k = o(n)$ tight in the average-case-to-worst-case Conjecture 2?
2. Is it possible to design a lattice-based algorithm that achieves asymptotically better complexity for solving k-XOR (GBP over $\mathbb{F}_2$) at density 1?
3. When k is non-constant, does there exist an algorithm that solves k-SUM (GBP over $\mathbb{Z}_m$) asymptotically better than $\widetilde{\mathcal{O}}(2^{n/2})$ or in polynomial time?
4. For $\mathsf{LGBP}(n, 2^k)$, when k lies in $[\sqrt{n/2+1}, \sqrt{n+1}]$, is it possible to further optimize the single-list algorithm?

Acknowledgments. This research was supported by the National Key R&D Program of China under Grant No. 2023YFA1009500. This work underwent a long revision process, during which we received insightful comments from anonymous reviewers of Asiacrypt, Eurocrypt and Crypto. Their feedback greatly helped refine this work, and we are sincerely grateful for their efforts. We also thank Chenyu Li and Bing Shi for their help at an early stage of this work.

References

1. Abboud, A., Lewi, K.: Exact weight subgraphs and the k-sum conjecture. In: International Colloquium on Automata, Languages, and Programming. pp. 1–12. Springer (2013)
2. Agrawal, S., Saha, S., Schwartzbach, N.I., Vanukuri, A., Vasudevan, P.N.: k-sum in the sparse regime: Complexity and applications. In: Annual International Cryptology Conference. pp. 315–351. Springer (2024)
3. Alcock, L., Ren, L.: A note on the security of Equihash. In: Proceedings of the 2017 on cloud computing security workshop. pp. 51–55 (2017)
4. Augot, D., Finiasz, M., Sendrier, N.: A family of fast syndrome based cryptographic hash functions. In: Dawson, E., Vaudenay, S. (eds.) Progress in Cryptology – Mycrypt 2005. pp. 64–83. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
5. Bai, X., Gao, J., Hu, C., Zhang, L.: Constructing an adversary solver for equihash. In: NDSS (2019)
6. Ball, M., Rosen, A., Sabin, M., Vasudevan, P.N.: Proofs of work from worst-case assumptions. In: Annual International Cryptology Conference. pp. 789–819. Springer (2018)
7. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in 2n/20: How $1+1=0$ improves information set decoding. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. pp. 520–536. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
8. Bellare, M., Canetti, R., Krawczyk, H.: Keying hash functions for message authentication. In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO '96. pp. 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)

9. Bellare, M., Goldreich, O., Goldwasser, S.: Incremental cryptography: The case of hashing and signing. In: Desmedt, Y.G. (ed.) Advances in Cryptology — CRYPTO '94. pp. 216–233. Springer Berlin Heidelberg, Berlin, Heidelberg (1994)
10. Bellare, M., Micciancio, D.: A new paradigm for collision-free hashing: Incrementality at reduced cost. In: Fumy, W. (ed.) Advances in Cryptology—EUROCRYPT 97. Lecture Notes in Computer Science, vol. 1233, pp. 163–192. Springer-Verlag (11–15 May 1997)
11. Benhamouda, F., Lepoint, T., Loss, J., Orrù, M., Raykova, M.: On the (in)security of ROS. In: Canteaut, A., Standaert, F.X. (eds.) Advances in Cryptology – EUROCRYPT 2021. pp. 33–53. Springer International Publishing, Cham (2021)
12. Bernstein, D.J.: Better price-performance ratios for generalized birthday attacks. In: Workshop Record of SHARCS. vol. 7, p. 160 (2007)
13. Bernstein, D.J., Lange, T., Niederhagen, R., Peters, C., Schwabe, P.: Fsbday: Implementing wagner's generalized birthday attack against the sha-3 round-1 candidate fsb. In: Progress in Cryptology-INDOCRYPT 2009: 10th International Conference on Cryptology in India, New Delhi, India, December 13-16, 2009. Proceedings 10. pp. 18–38. Springer (2009)
14. Biryukov, A., Khovratovich, D.: Equihash: Asymmetric proof-of-work based on the generalized birthday problem. Ledger **2**, 1–30 (2017)
15. Bok-Min, G., Siddiqi, M.U., Hean-Teik, C.: Incremental hash function based on pair chaining & modular arithmetic combining. In: Rangan, C.P., Ding, C. (eds.) Progress in Cryptology — INDOCRYPT 2001. pp. 50–61. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
16. Both, L., May, A.: Optimizing BJMM with nearest neighbors: full decoding in 22/21n and McEliece security. In: WCC workshop on coding and cryptography. vol. 214 (2017)
17. Brakerski, Z., Stephens-Davidowitz, N., Vaikuntanathan, V.: On the hardness of average-case k-sum. arXiv preprint arXiv:2010.08821 (2020)
18. Brent, R.P.: An improved Monte Carlo factorization algorithm. BIT Numerical Mathematics **20**(2), 176–184 (1980)
19. Carozza, E., Couteau, G., Joux, A.: Short signatures from regular syndrome decoding in the head. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 532–563. Springer (2023)
20. Clarke, D., Devadas, S., van Dijk, M., Gassend, B., Suh, G.E.: Incremental multiset hash functions and their application to memory integrity checking. In: Laih, C.S. (ed.) Advances in Cryptology - ASIACRYPT 2003. pp. 188–207. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
21. Devadas, S., Ren, L., Xiao, H.: On iterative collision search for LPN and subset sum. In: Theory of Cryptography Conference. pp. 729–746. Springer (2017)
22. Dinur, I.: An algorithmic framework for the generalized birthday problem. Designs, Codes and Cryptography **87**, 1897–1926 (2019)
23. Dinur, I., Keller, N., Klein, O.: Fine-grained cryptanalysis: Tight conditional bounds for dense k-sum and k-xor. Journal of the ACM **71**(3), 1–41 (2024)
24. Ducas, L., Engelberts, L., Loyer, J.: Wagner's algorithm provably runs in subexponential time for SIS^{∞}. In: Annual International Cryptology Conference. pp. 353–384. Springer (2025)
25. Ducas, L., Kiltz, E., Lepoint, T., Lyubashevsky, V., Schwabe, P., Seiler, G., Stehlé, D.: Crystals-dilithium: A lattice-based digital signature scheme. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 238–268 (2018)
26. Dworkin, M.J.: SHA-3 standard: Permutation-based hash and extendable-output functions (2015)

27. Index pointer for Equihash, https://github.com/xenoncat/equihash-xenon
28. Esser, A., Bellini, E.: Syndrome decoding estimator. In: IACR International Conference on Public-Key Cryptography. pp. 112–141. Springer (2022)
29. Esser, A., Santini, P.: Not just regular decoding: asymptotics and improvements of regular syndrome decoding attacks. In: Annual International Cryptology Conference. pp. 183–217. Springer (2024)
30. Finiasz, M., Sendrier, N.: Security bounds for the design of code-based cryptosystems. In: International conference on the theory and application of cryptology and information security. pp. 88–105. Springer (2009)
31. Gajentaan, A., Overmars, M.H.: On a class of o (n2) problems in computational geometry. Computational geometry **5**(3), 165–185 (1995)
32. Gobioff, H., Nagle, D., Gibson, G.: Embedded security for network-attached storage. Tech. rep., Technical Report CMU-CS-99-154, CMU (1999)
33. Gobioff, H.B.: Security for a high performance commodity storage subsystem. Carnegie Mellon University (1999)
34. Golić, J.D.: Computation of low-weight parity-check polynomials. Electronics Letters **32**(21), 1981–1982 (1996)
35. Hopwood, D.E., Bowe, S., Hornby, T., Wilcox, N.: Zcash protocol specification (2017), https://zips.z.cash/protocol/protocol.pdf
36. Horowitz, E., Sahni, S.: Computing partitions with applications to the knapsack problem. Journal of the ACM (JACM) **21**(2), 277–292 (1974)
37. Johansson, T., Jönsson, F.: Fast correlation attacks through reconstruction of linear polynomials. In: Advances in Cryptology—CRYPTO 2000: 20th Annual International Cryptology Conference Santa Barbara, California, USA, August 20–24, 2000 Proceedings 20. pp. 300–315. Springer (2000)
38. Kirchner, P.: Improved generalized birthday attack. Cryptology ePrint Archive (2011)
39. LaVigne, R., Lincoln, A., Vassilevska Williams, V.: Public-key cryptography in the fine-grained setting. In: Annual International Cryptology Conference. pp. 605–635. Springer (2019)
40. Lee, P.J., Brickell, E.F.: An observation on the security of mceliece's public-key cryptosystem. In: Workshop on the Theory and Application of of Cryptographic Techniques. pp. 275–280. Springer (1988)
41. Levieil, É., Fouque, P.A.: An improved LPN algorithm. In: De Prisco, R., Yung, M. (eds.) Security and Cryptography for Networks. pp. 348–359. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
42. Mail, C.J.: Pricing via processing. In: Advances in Cryptology—CRYPTO'92: 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16–20, 1992. Proceedings. vol. 740, p. 139 (1993)
43. Martino, W.: The first scalable, high performance private blockchain. Revision v1. 0 pp. 1–9 (2016)
44. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 203–228. Springer (2015)
45. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) Advances in Cryptology — CRYPTO '87. pp. 369–378. Springer Berlin Heidelberg, Berlin, Heidelberg (1988)
46. Mihajloska, H., Gligoroski, D., Samardjiska, S.: Reviving the idea of incremental cryptography for the zettabyte era use case: Incremental hash functions based on SHA-3. In: Camenisch, J., Kesdoğan, D. (eds.) Open Problems in Network Security. pp. 97–111. Springer International Publishing, Cham (2016)

47. Minder, L., Sinclair, A.: The extended k-tree algorithm. Journal of cryptology **25**, 349–382 (2012)
48. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
49. Narisada, S., Uemura, S., Okada, H., Furue, H., Aikawa, Y., Fukushima, K.: Solving McEliece-1409 in one day—cryptanalysis with the improved BJMM algorithm. In: International Conference on Information Security. pp. 3–23. Springer (2024)
50. Niebuhr, R., Cayrel, P.L., Buchmann, J.: Improving the efficiency of generalized birthday attacks against certain structured cryptosystems. In: WCC 2011-Workshop on coding and cryptography. pp. 163–172 (2011)
51. Percival, C.: Stronger key derivation via sequential memory-hard functions (2009)
52. Pollard, J.M.: A Monte Carlo method for factorization. BIT Numerical Mathematics **15**(3), 331–334 (1975). https://doi.org/10.1007/BF01933667
53. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962)
54. Saarinen, M.J.O.: Linearization attacks against syndrome based hashes. Cryptology ePrint Archive, Paper 2007/295 (2007), https://eprint.iacr.org/2007/295
55. Schnorr, C.P.: Security of blind discrete log signatures against interactive attacks. In: International Conference on Information and Communications Security. pp. 1–12. Springer (2001)
56. Setty, S., Angel, S., Gupta, T., Lee, J.: Proving the correct execution of concurrent services in zero-knowledge. In: 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18). pp. 339–356 (2018)
57. Stern, J.: A method for finding codewords of small weight. In: International colloquium on coding theory and applications. pp. 106–113. Springer (1988)
58. Tang, L., Ding, R., Sun, Y., Gong, X.: Memory optimizations of wagner's algorithm with applications to equihash. IACR Transactions on Cryptographic Hardware and Embedded Systems **2026**(2), 218–239 (2026)
59. Wagner, D.: A generalized birthday problem. In: Annual International Cryptology Conference. pp. 288–304. Springer (2002)
60. Weber, B., Zhang, X.: Parallel hash collision search by rho method with distinguished points. In: 2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT). pp. 1–7. IEEE (2018)
61. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**(2014), 1–32 (2014)

A Disclosure of AI Assistance

We used ChatGPT and DeepSeek for grammar checking and basic editing, and Codex to assist with the implementation of auxiliary scripts for producing figures. All such assistance was reviewed and verified by the authors.

B Proof of Lemmas

B.1 Proof of Lemma 1

Proof. We give a brief heuristic proof here. Let $t = K - K_0$. In the case of $\mathbf{RGBP}(n, K_0)$, we first select t values $(x_1, x_2, \ldots, x_t)$ from the first t lists and compute $Y = \bigoplus_{i=1}^{t} x_i$. We XOR all elements with Y in the $(t+1)$-th list

denoted as $L_{t+1} \oplus Y$, and use algorithm $\mathcal{A}$ to solve $\mathbf{RGBP}(n, K_0)$ among lists $L_{t+1} \oplus Y, L_{t+2}, \ldots, L_K$. This produces a solution of $x_{t+1} \oplus x_{t+2} \oplus \ldots \oplus x_K = Y$ which leads to $x_1 \oplus x_2 \oplus \cdots \oplus x_K = 0$.

In the case of $\mathbf{LGBP}(n, K_0)$, we select t values $(x_1, x_2, \ldots, x_t)$ from the initial list $L^{(0)}$ and compute $Y = \bigoplus_{i=1}^{t} x_i$. If K_0 is odd, by XORing all elements with Y in $\hat{L}^{(0)} = L^{(0)}/\{x_1, \ldots, x_t\}$, we can apply algorithm $\mathcal{A}$ to solve $\mathbf{LGBP}(n, K_0)$ in the list $\hat{L}^{(0)} \oplus Y$ and thus obtain a solution to $\mathbf{LGBP}(n, K)$. If K_0 is even, we XOR half of the elements with Y in the initial list $\hat{L}^{(0)}$ and then apply algorithm $\mathcal{A}$. There exists an approximate probability of $\frac{1}{2}$ that the resultant solution will contain an odd number of values XORed with Y. This implies a $\frac{1}{2}$ chance of deriving a valid solution for $\mathbf{LGBP}(n, K)$. By repeating this process, it is expected to solve $\mathbf{LGBP}(n, K)$ within two iterations. □

B.2 Proof of Lemma 2

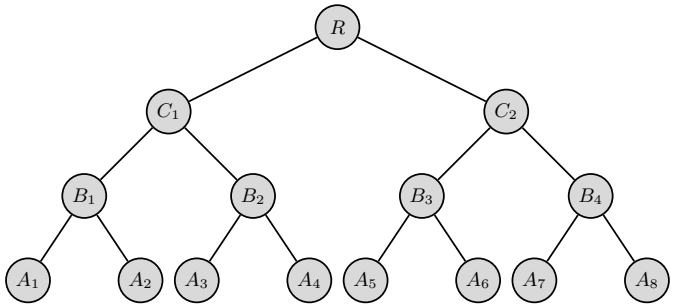


Fig. 8: The visualization of index binary tree.

Proof. This is a simplified proof of [21, §2.3]. In the single-list algorithm, there are 2^i indices in the i-th layer $L^{(i)}$. The index vector of size 2^i forms a binary tree and exchanging any node within the binary tree will result in an equivalent solution. For example, let the index vector be $\boldsymbol{A} = (A_1, \ldots, A_8)$. Swapping (A_{2i-1}, A_{2i}), (B_{2j-1}, B_{2j}), and (C_1, C_2) in Figure 8 will result in the same valid index vector (solution). Therefore, the number of candidates of index vector of size 2^i is:

$$|I(2^i)| = \binom{N}{2^i} \frac{(2^i)!}{\prod_{j=1}^{i} 2^{2^{i-j}}} = \binom{N}{2^i} \frac{(2^i)!}{2^{2^i-1}}.$$

Taking Figure 8 as an example, we need to cancel out ℓ bits in the middle nodes, i.e., B_i, C_j, R. Therefore, in the i-th layer $L^{(i)}$, we have to cancel out $\sum_{j=1}^{i} 2^{i-j} \cdot \ell = (2^i - 1)\ell$ bits. The expected number of valid index vectors in the

i-th layer is:

$$\mathbf{E}\left[\left|L^{(i)}\right|\right] = \frac{|I(2^i)|}{2^{(2^i-1)\ell}} = \binom{N}{2^i}\frac{(2^i)!}{2^{2^i-1}} \cdot \left(\frac{2}{N}\right)^{2^i-1} = \binom{N}{2^i}\frac{(2^i)!}{N^{2^i-1}}.$$

In the last $\mathsf{merge}^\star$ operation, we need to cancel out 2ℓ bits instead of ℓ bits, i.e., canceling $2^k - 1 + 1 = 2^k$ bits in total. The expected number of valid index vectors in the last layer is:

$$\mathbf{E}\left[\left|L^{(k)}\right|\right] = \binom{N}{2^k}\frac{(2^k)!}{2^{2^k-1}} \cdot \left(\frac{2}{N}\right)^{2^k} = \frac{2\binom{N}{2^k}(2^k)!}{N^{2^k}}$$

□

C On the Asymptotic Order of $\mathcal{N}(n, k)$

Recall that $\mathcal{N}(n, k)$ is defined as the minimum list size such that there exists at least one solution to $\mathbf{GBP}(n, k)$ in expectation, which can be interpreted as the information-theoretic lower bound for $\mathbf{GBP}(n, k)$. In this section, we provide a rigorous asymptotic analysis of $\mathcal{N}(n, k)$. The main result is stated in the following theorem, which holds consistently for both regular and standard cases.

Theorem 6. *Let $n > k > 0$. The minimum list size $\mathcal{N}(n, k)$ for $\mathbf{GBP}(n, k)$ satisfies: $N = \mathbf{\Theta}\left(k \cdot 2^{n/k}\right)$, and thus has the following asymptotic behaviors:*

1. *When $k = \mathcal{O}(1)$, $\mathcal{N}(n, k)$ is exponential in n; more precisely, for fixed k, $\mathcal{N}(n, k) = \mathbf{\Theta}\left(2^{n/k}\right)$.*
2. *When $k = \mathbf{\Theta}(n)$, $\mathcal{N}(n, k)$ is linear in n, i.e., $\mathcal{N}(n, k) = \mathbf{\Theta}(n)$.*

Proof. For simplicity, we denote $\mathcal{N}(n, k)$ as N in the following analysis. The regular case follows immediately from the definition via the following reduction:

$$\left(\frac{N}{k}\right)^k = 2^n \implies N = k \cdot 2^{\frac{n}{k}} = \mathbf{\Theta}\left(k \cdot 2^{n/k}\right).$$

For the standard case, we have the following expression for N:

$$\binom{N}{k} = 2^n. \tag{10}$$

Since $\binom{N}{k} = \binom{N}{N-k}$, we can assume without loss of generality that $k \le N/2$ or replace k with $\hat{k} := \min\{N-k, k\} \le N/2$ in the following analysis. On one hand, by Stirling lower bound for factorials: $k! \ge \left(\frac{k}{e}\right)^k$, we have

$$2^n = \binom{N}{k} = \frac{N!}{k!(N-k)!} \le \frac{N^k}{k!} \le \left(\frac{eN}{k}\right)^k. \tag{11}$$

Taking the k-th root on both sides of Eq. (11) and rearranging the terms yields

$$N \geq \frac{k}{e} \cdot 2^{n/k}. \tag{12}$$

On the other hand, since $k! < k^k$ and $N - k + 1 > \frac{N}{2}$, we have

$$2^n = \binom{N}{k} = \frac{N!}{k!(N-k)!} \geq \frac{(N-k+1)^k}{k!} \geq \left(\frac{N-k+1}{k}\right)^k > \left(\frac{N}{2k}\right)^k. \tag{13}$$

Take the k-th root on both sides of Eq. (13):

$$N < 2k \cdot 2^{n/k}. \tag{14}$$

Combining the bounds in Eq. (12) and Eq. (14), we have

$$\frac{k}{e} \cdot 2^{n/k} \leq N < 2k \cdot 2^{n/k} \tag{15}$$

Eq. (15) implies that $N = \mathbf{\Theta}\left(k \cdot 2^{n/k}\right)$. Therefore, when $k = \mathcal{O}(1)$, N is exponential in n, and when $k = \mathbf{\Theta}(n)$, N is linear in n. This completes the proof. □

Problem Classification. The asymptotic analysis of $\mathcal{N}(n,k)$ reveals a clear classification of the worst-case $\mathbf{GBP}(n,k)$. Specifically, we can categorize the problem into three regimes based on the relationship between k and n:

1. **Collision Problem**. When $k = \mathcal{O}(1)$, $\mathcal{N}(n,k)$ is exponential in n. In this regime, the problem is essentially a collision problem and no efficient algorithm is known to surpass the birthday bound.
2. **Decoding Problem**: when k is linear in n, $\mathcal{N}(n,k)$ is linear in n. In this regime, we prove that the worst-case $\mathbf{GBP}(n,k)$ degenerates to a special decoding problem.
3. When $k = \omega(1)$ and $k = o(n)$, $\mathcal{N}(n,k)$ is sub-exponential in n.

The first regime corresponds to the classical setting of the k-XOR conjecture. The second regime is the focus of our work, where we show ISD algorithms can be used to solve GBP efficiently. The third regime remains unexplored, and it's crucial for further sharpening the understanding of the k-XOR conjecture.

D Concrete Memory Analysis

In this section, we present the concrete memory complexity of state-of-the-art Wagner's algorithmic framework for RGBP and LGBP.

Algorithm 2 The K-list Algorithm for $\mathbf{RGBP}(n, K = 2^k)$

Require: Let $\ell = \frac{n}{k+1}$ and $\mathsf{merge}_\ell(L_1, L_2) := \{((x_1 \oplus x_2) \gg \ell, (i_1, i_2)) \mid \mathcal{LSB}_\ell(x_1 \oplus x_2) = 0, (x_1, i_1) \in L_1, (x_2, i_2) \in L_2\}$.
Input: K lists $L_1, L_2, \ldots, L_K$, each of size $\frac{N}{2} = 2^\ell$.
Output: A solution list to $\mathbf{RGBP}(n, K = 2^k)$.

Load the first list $L_1' \leftarrow \{(x_j, j) \mid x_j \in L_1\}$.
Initialize the stack $S \leftarrow [(L_1, 0)]$. ▷ The stack contains the hash list and its depth.
for $i \leftarrow 2$ to K **do**
    $d_{cur} \leftarrow 0$
    Load the i-th list $L_{cur} \leftarrow \{(x_j, j) \mid x_j \in L_i\}$.
    $d_{top} \leftarrow$ depth of top list of stack S.
    while $d_{cur} = d_{top}$ **do**
        $L_{top}, d_{top} \leftarrow S.pop()$.
        if $d_{cur} \neq k - 1$ **then**
            $L_{cur} \leftarrow \mathsf{merge}_\ell(L_{top}, L_{cur})$.
        else
            $L_{cur} \leftarrow \mathsf{merge}_{2\ell}(L_{top}, L_{cur})$.
        $d_{cur} \leftarrow d_{cur} + 1$.
        $d_{top} \leftarrow$ depth of top list of stack S.
    $S.push((L_{cur}, d_{cur}))$.
Return the top list of stack S.

D.1 Memory Complexity of Wagner's K-List Algorithm

Proposition 5 (Complexity of k-list Algorithm). *The k-list algorithm for* $\mathbf{RGBP}(n, 2^k)$ *runs in time* $\mathcal{O}(2^k \cdot N)$ *and requires a peak memory of at least* $\mathcal{O}\left(\left(\frac{k^2+5k+2}{4} + 2^{k-1}\right)\ell N\right)$ *bits.*

Proof. The K-list Algorithm 2 performs $2^{k-1} + 2^{k-2} + \cdots + 2^0 = 2^k - 1$ merge operations on two lists of size $N/2 = 2^\ell$. The time complexity of merging is $(2^k - 1)N$ and $2^k \cdot N/2$ hashing operations are needed to generate the leaf lists. Thus, the time complexity is $\mathcal{O}(2^k \cdot N)$. Since all the lists form a complete binary tree, the K-list algorithm can be performed in post-order using a stack to reduce peak memory. Therefore, the peak memory consumption occurs during the merging of the last two leaves, wherein exactly one list exists at each layer except for the root (0 list) and layer 0 (2 lists). In layer i, the list element consists of $2^i \times \ell$-bit indices and a temporary XOR result of $n - i\ell = (k + 1 - i)\ell$ bit. The peak memory measured in bits is given by:

$$M = \underbrace{(k+2)\,\ell\frac{N}{2}}_{\text{Layer 0}} + \underbrace{\sum_{i=0}^{k-1}\left(2^i + k + 1 - i\right)\ell\frac{N}{2}}_{\text{Layer 0, 1, \ldots, k -1}}$$
$$= \left(\frac{k^2+5k+2}{4} + 2^{k-1}\right)\ell N$$

The overall storage consists of $\frac{k^2+5k+2}{4}\ell N$ bits for hash values and $2^{k-1}\ell N$ bits for index vectors. □

We find that the index-trimming technique proposed in Equihash [14] can be applied to the k-list algorithm with small runtime overhead.

Proposition 6 (Complexity of Optimized k-list Algorithm). *The k-list algorithm with index-trimming for* $\mathbf{RGBP}(n, 2^k)$ *runs in time* $\mathcal{O}(2^k \cdot N)$ *and requires a peak memory of at least* $\mathcal{O}\left(\left(\frac{k^2+5k+2}{4}\ell + 2^{k-1}\right)N\right)$ *bits.*

Proof. If we store single-bit index in the k-list algorithm, we can recover 1-bit partial solution. In the subsequent run with the input list size reduced to $2^{\ell-1}$, the time and memory complexities of the second run are negligible compared to the first run. Therefore, the peak memory is reduced to $\mathcal{O}\left(\left(\frac{k^2+5k+2}{4}\ell + 2^{k-1}\right)N\right)$. □

D.2 Memory Complexity of Wagner's Single-List Algorithm

Algorithm 3 The merge* Function for $\mathbf{LGBP}(n, K = 2^k)$

Input: A list L of size N. The number of bits ℓ to be canceled out. A boolean value dis_0 indicating whether to discard zero XOR values.
Output: A new list L' after self-merging.

1: Initialize an empty hash dictionary $D \leftarrow \{\}$.
2: Initialize an empty list $L' \leftarrow \{\}$.
3: **for** each $i, x_i \in L$ **do**
4: $\quad x_\ell \leftarrow x_i \mod 2^\ell$.
5: $\quad x_h \leftarrow x_i \gg \ell$.
6: $\quad$ **if** $x_\ell \in D$ **then**
7: $\quad\quad$ **if** dis_0 and $x_h \in D[x_\ell]$ **then**
8: $\quad\quad\quad$ Continue.
9: $\quad\quad$ **for** each $(y_h, j) \in D[x_\ell]$ **do**
10: $\quad\quad\quad L' \leftarrow L' \cup \{(x_h \oplus y_h, (i, j))\}$.
11: $\quad D[x_\ell] \leftarrow D[x_\ell] \cup \{(x_h, i)\}$.
12: Return L'.

In single-list Algorithm 1, after each self-merging, the number of indices doubles in the new list. Note that the index vector of size 2^k also forms a binary tree in the single-list algorithm. The index pointer [3, 27] can be applied in the single-list algorithm to reduce peak memory as follows:

- Instead of storing index tuples in each list, two index pointers are stored, which point to the left and right child, i.e., the indices of the two elements that are XORed in the previous list.

– After merging, a new list $L^{(i+1)}$ is generated. The XOR values in $L^{(i)}$ are discarded, while the index pointers are retained.
– After obtaining the final result, each pair of index pointers is traced back to retrieve the full indices of the solution.

Algorithm 4 Practical Single-list Algorithm for $\mathbf{LGBP}(n, K = 2^k)$

Require: Let $\ell = \frac{n}{k+1}$ and $N = 2^{\ell+1}$. Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^n$ be a random oracle.
Input: A message oracle $\mathcal{M}(j)$ that returns the j-th message for $j \in [1, N]$.
Output: A solution list to $\mathbf{LGBP}(n, K = 2^k)$.

1: Compute the initial list $L^{(0)} = \{(\mathcal{H}(M(j)), j) \mid j \in [1, N]\}$.
2: **for** $i \leftarrow 1$ to $k-1$ **do**
3: $\quad L^{(i)} \leftarrow \mathsf{merge}^{\star}_{\ell}(L^{(i-1)}, \mathsf{dis}_0 = 1)$. ▷ Discard early zero XOR values.
4: $\quad$ Remove all the XOR values in $L^{(i-1)}$.
5: $L^{(k)} \leftarrow \mathsf{merge}^{\star}_{2\ell}(L^{(k-1)}, \mathsf{dis}_0 = 0)$. ▷ Save final zero XOR values.
6: Remove all the XOR values in $L^{(k-1)}$.
7: Return lists of 2^k indices retrieved from index pointers of $L^{(0)}, L^{(1)}, \cdots, L^{(k)}$.

Proposition 7 (Complexity of Single-list Algorithm). *The single-list algorithm for* $\mathbf{LGBP}(n, 2^k)$ *runs in time* $\mathcal{O}(kN)$ *and requires a peak memory of at least* $\mathcal{O}\left(\left(2^{k-1} \cdot (\ell+1) + 2\ell\right) \cdot N\right)$ *bits. Using the index pointer technique, the peak memory can be reduced to* $\mathcal{O}\left(2 \cdot (n + k - \ell - 1) \cdot N\right)$ *bits.*

Proof. The $\mathsf{merge}^{\star}$ operation has a time and memory complexity of $\mathcal{O}(N)$ when implemented with a hash dictionary or buckets in [27]. The time complexity of the single-list algorithm 1 is $\mathcal{O}(kN)$, i.e., k times $\mathsf{merge}^{\star}$ operations on a single list of size N. In the naive implementation of single-list algorithm, upon completion of self-merging, we can remove the previous list from memory. In the new list of size $\mathcal{O}(N)$, the number of indices doubles (at least $2(\ell+1)$ bits added in per element) and the temporary XOR value decreases by ℓ bits. Therefore, the peak memory occurs in the last step where we need to store 2^{k-1} indices and 2ℓ bits for each list element. The peak memory measured in bits is given by:

$$\mathcal{O}\left(\left(2^{k-1} \cdot (\ell+1) + 2\ell\right) \cdot N\right).$$

When utilizing the index pointer technique during the self-merging operation, two index pointers are added. Consequently, at least $2(\ell+1)$ bits are added in list element and the temporary XOR value decreases by ℓ bits. After one iteration, the memory usage increases by $(2(\ell+1) - \ell)N = (\ell+2)N$ bits. Given that the initial list size is nN (as index is not necessary here), the memory usage reaches its peak before the final self-merging operation:

$$\mathcal{O}\left((n + (\ell+2) \cdot (k-1))N\right) = \mathcal{O}\left(2 \cdot (n + k - \ell - 1) \cdot N\right).$$

□

Post-Retrieval and In-Place Merge Framework. In [58], the authors proposed post-retrieval and in-place merge framework that reduces the memory usage to $\mathcal{O}(n \cdot N)$ bits, i.e., the memory usage of the initial list. All the intermediate lists are not necessary to be stored in memory and can be retrieved after the final merging operation. The core idea is that index pointers are only required to reconstruct the final solution, but are not essential for advancing the self-merge process in the single-list algorithm. Therefore, the computation order of the entire index-pointer list can be reversed, allowing the complete solution to be reconstructed incrementally. If the merging operation can be performed in-place, the memory usage can be reduced to $\mathcal{O}(n \cdot N)$ bits.

E Secondary Solutions

When $k \approx \sqrt{\frac{n}{2}+1}$, the single-list algorithm in Equihash [14] encounters numerous trivial and non-trivial collisions. Trivial collisions, in particular, cause an exponential explosion in the list size, significantly increasing the time complexity. In our optimized single-list Algorithm 4, we filter out all trivial collisions by a duplicate check (this technique was also used in [27]). For instance, if the XOR value becomes entirely zero during the $(k-1)$-th merge, we consider it a trivial solution with overwhelming probability, meaning that every index appears an even number of times in the concatenated index vector $i_1|i_2$. Otherwise, the solution is valid for $\mathsf{LGBP}(n, 2^{k-1})$ which is almost impossible given our limited data size.

Secondary Solutions. The single-list algorithm frequently encounters index overlaps when k approaches $\sqrt{n/2+1}$. Consequently, around this critical value, secondary solutions may arise. These solutions are not the expected solutions to $\mathsf{LGBP}(n, 2^k)$ but instead correspond to $\mathsf{LGBP}(n, 2^k - 2t)$, where $t = 1, 2, \ldots, 2^{k-1} - 1$. The total search space for Algorithm 4 is N^{2^k} if we model it as selection of 2^k items with replacement from a set of N items. Nevertheless, we filter out all trivial solutions to prevent an exponential explosion in the number of trivial solutions, which would otherwise cause a dramatic increase in the list size. Considering the overall number of bits to be canceled out, the single-list algorithm is expected to generate at most:

$$\mathbf{E}\left[\left|L^{(k)}\right|\right] = \frac{N^{2^k}}{2^{2^k-1}} \cdot \left(\frac{2}{N}\right)^{2^k} = 2$$

solutions to $\mathsf{LGBP}(n, 2^k - 2t)$ for $t = 0, 1, \ldots, 2^{k-1}$. Denote the expected number of solutions to $\mathsf{LGBP}(n, 2^k - 2t)$ as $\mathbf{E}\left[\left|L^{(k)}(t)\right|\right]$. When k is small, the algorithm's search space is primarily concentrated on overlapping-free cases: $\mathbf{E}\left[\left|L^{(k)}(t=0)\right|\right]$, where $\binom{N}{K} \approx N^K$. However, as k approaches the threshold $\sqrt{n/2+1}$, we must account for selection with replacement. Let $K = 2^k$. We consider the most likely secondary solution to $\mathsf{LGBP}(n, 2^k - 2)$ such that there

is exactly one element occurring twice or thrice in the complete binary tree. The expected number of solutions to $\mathsf{LGBP}(n, 2^k - 2)$ is given by:

$$\mathbf{E}\left[\left|L^{(k)}(t=1)\right|\right] = \binom{2^k-1}{1}\binom{N}{2^k-1}\frac{(2^k)!}{2!\cdot 2^{2^k-1}}\left(\frac{2}{N}\right)^{2^k} + \binom{2^k-2}{1}\binom{N}{2^k-2}\frac{(2^k)!}{3!\cdot 2^{2^k-1}}\left(\frac{2}{N}\right)^{2^k}$$

The second term can be ignored which is negligible compared to the first term. By our observation (no formal proof), the expected number of solutions to $\mathsf{LGBP}(n, 2^k - 2t)$ is dominated by the case where there are exactly t distinctive elements occurring twice in the complete binary tree. We estimate the expected number of solutions to $\mathsf{LGBP}(n, 2^k - 2t)$ as:

$$\mathbf{E}\left[\left|L^{(k)}(t)\right|\right] \approx \binom{2^k-t}{t}\binom{N}{2^k-t}\frac{(2^k)!}{(2!)^t\cdot 2^{2^k-1}}\left(\frac{2}{N}\right)^{2^k}. \tag{16}$$

Remark 5. The estimation given in Eq. (16) is based on the assumption that all the permutations of 2^k elements can construct a valid complete collision binary tree as depicted in Wagner's algorithm with equal probability. However, this assumption is not true especially when t is large. For example, let $t = 1$, $s = (a, a, a_1, \cdots, a_{2^k-2})$ is definitely impossible in the single-list algorithm since we need to discard the trivial solutions. Meanwhile, candidates with overlapping indices like $s' = (a, b_1, a, b_2, a, b_1, \cdots)$ are less likely to be a valid solution compared to 2^k distinctive elements. Therefore, the single-list algorithm can yield the expected number of solutions only if the N^K search space is primarily concentrated on cases with no or minimal index overlap.

Specifically, we conduct experiments to verify the expected number of solutions to $\mathsf{LGBP}(n, 2^k - 2t)$ for $t = 0, 1, \ldots, 2^{k-1}$ where $k = \sqrt{n/2+1}$ and $\ell = n/(k+1)$ are both integers. The theoretical estimation and experimental results are shown in Table 4.

Table 4: Expected number of primary and secondary solutions to $\mathsf{LGBP}(n, K-2t)$ produced by the single-list algorithm in Algorithm 4.

$\mathsf{LGBP}(n, K)$	Source	#Sol.	#Sec-Sol.		Total
		$t=0$	$t=1$	$t\geq 2$	
$(96, 2^7 = 128)$	Eq. (16)	0.7377	0.7435	0.5088	1.9900
	Exp.	0.7254	0.7222	1.1738	2.6214
$(160, 2^9 = 512)$	Eq. (16)	0.7362	0.7377	0.5235	1.9974
	Exp.	0.7414	0.7404	1.2935	2.7753

We solved 10,000 random instances of $\mathsf{LGBP}(96, 2^7)$ and $\mathsf{LGBP}(160, 2^9)$. The experimental results align with theoretical estimates only for $t = 0, 1$, as shown in Table 4. A notable and unexpected finding is that the total number of solutions can exceed the theoretical upper bound of 2. We hypothesize that certain intermediate list elements in the middle layer of the single-list algorithm may interact in such a way that additional solutions emerge. This behavior appears analogous to the exponential growth in list size observed in the presence of trivial solutions. Understanding this interaction remains an open problem. In our experiments, Algorithm 4 produced as many as 37 solutions in the most favorable random instance, with a runtime of 2.75 seconds, compared to an average runtime of 1.5 seconds in regular cases. In this instance, the maximum layer size reached 474,984, which is approximately $3.62 \times N$. Further investigation into the single-list algorithm is necessary to uncover the mechanisms behind these secondary solutions.