基于 DES 算法与整数分解的负数据库生成算法

赵冬冬[1,2], 刘志晖[1], 廖 磊[1], 向剑文[1], 江 浩[3]

[1]武汉理工大学 计算机与人工智能学院 武汉 中国 430070
[2]武汉理工大学 重庆研究院 重庆 中国 401120
[3]安徽大学 人工智能学院 安徽 中国 230601

摘要 在计算机科学理论领域, SAT 问题(Boolean Satisfiability Problem, 布尔可满足性问题)是一项经典的数理逻辑问题, 其在计算机学科中扮演着至关重要的角色。在隐私保护领域中, 负数据库技术是一种新兴的隐私保护技术, 其主要原理是通过存储原始数据的补集来实现对数据的保护。相较于传统的加解密算法, 负数据库技术在大数据隐私保护领域展现出显著的效率优势, 因而具备巨大的潜力与前景。值得注意的是, 本文研究的负数据库方法等价于经典的 SAT 问题, 这一等价性表现在负数据库的实例表达与求解过程上, 因此本文也是对 SAT 问题的探索。现有的负数据库生成算法通常使用概率参数来进行生成, 这种方法在面对一些最新的 SAT 求解器时, 可能容易被求解。为此, 本文提出了基于 DES(Data Encryption Standard, 数据加密标准)算法的负数据库生成算法 *D*-hidden 和基于整数分解问题的负数据库生成算法 *F*-hidden。实验表明, 在 *D*-hidden 算法中, 当隐藏串的长度与基准串的长度相等且轮数不小于 5 时, 相较于目前经典的负数据库生成算法 *K*-hidden, *D*-hidden 算法生成了更加难解且更为稳定的负数据库, 并且具有更高的生成效率。在 *F*-hidden 算法中, 隐藏串长度在[50, 600]范围内相对于已有负数据库生成算法表现出显著的难解性。本工作是首个将负数据库技术与传统的加解密算法相结合的工作, 为隐私保护领域提供新的思路, 同时也为 SAT 领域提供新的研究视角。

关键词 隐私保护; 布尔可满足性问题; 负数据库
中图法分类号 TP301.6 DOI 号 10.19363/J.cnki.cn10-1380/tn.2026.01.09

Negative Database Generation Algorithm based on DES Algorithm and Integer Factorization

ZHAO Dongdong[1,2], LIU Zhihui[1], LIAO Lei[1], XIANG Jianwen[1], JIANG Hao[3]

[1] School of Computer Science and Artificial Intelligence, Wuhan University of Technology, Wuhan 430070, China
[2]Chongqing Research Institute, Wuhan University of Technology, Chongqing 401120, China
[3]School of Artificial Intelligence, Anhui University, Anhui 230601, China

Abstract In the domain of theoretical computer science, the Boolean satisfiability problem (SAT) constitutes a classical problem in mathematical logic, playing a crucial role within the field of computer science. In the domain of privacy protection, negative database technology is an emerging privacy safeguard technique, primarily based on the principle of protecting data by storing the complement of the original dataset. Compared to traditional encryption algorithms, negative database technology exhibits significant efficiency advantages in the domain of big data privacy protection, thus possessing substantial potential and prospects. It is noteworthy that the negative database method explored in this paper is equivalent to the classical SAT problem, with this equivalence manifested in the expression and solving processes of negative database instances. Consequently, this paper represents an exploration of the SAT problem. Existing negative database generation algorithms commonly employ probability parameters for generation, a method that may be susceptible to resolution when faced with some of the latest SAT solvers. To address this, the paper proposes negative database generation algorithms, namely *D*-hidden based on the data encryption standard (DES) algorithm and *F*-hidden based on the integer factorization problem. Experiments indicate that in the *D*-hidden algorithm, when the length of the hidden string is equal to the length of the baseline string and the number of rounds is not less than 5, compared to the classic negative database generation algorithm *K*-hidden, the *D*-hidden algorithm generates a more challenging and stable negative database, with higher efficiency. In the *F*-hidden algorithm, within the range of hidden string lengths [50, 600], significant difficulty is observed relative to existing negative database generation algorithms. This work represents the first integration of negative database technology with traditional encryption algorithms, providing an innovative perspective for the field of privacy protection, and concurrently offering a novel research outlook for the SAT domain.

通讯作者: 向剑文, 博士, 教授, Email: jwxiang@whut.edu.cn。
本课题得到湖北省重大科技专项(No. 2024BAA011)资助。
收稿日期: 2023-12-03; 修改日期: 2024-06-04; 定稿日期: 2025-12-09

Key words privacy protection; Boolean satisfiability problem; negative database

1 引言

随着计算机技术的不断发展, 数字化信息传播的规模不断扩大, 进而引发了大家对数据隐私安全问题的不断重视。由于网络传播的信息数据呈现多样化和复杂化的趋势, 因此如何充分保护数据隐私变得尤为重要。为了解决隐私安全问题, 目前已有多种隐私保护技术, 较为流行的有如安全多方计算、差分隐私、同态加密等。

在众多的隐私保护方法中, 负数据库(Negative Database, *NDB*)是较为新颖的一种。负数据库通过存储原数据的补集, 将原始数据进行隐藏, 从而实现对隐私数据的保护。与正数据库相似, 负数据库也支持对其存储的内容进行成员查询等类似的操作。已有工作[1-2]表明 SAT(Boolean Satisfiability Problem, 布尔可满足性问题)实例与负数据库等价, 对负数据库进行求解也是 NP(Non-deterministic Polynomial)完全问题。目前负数据库已被应用于数据隐私保护、密码领域、生物识别认证等多个领域, 具有广泛的发展前景。

负数据库的隐私保护效果源于其难解性, 而负数据库的难解性取决于被加密的字符串长度和负数据库的生成算法。已有的 *NDB* 生成算法中, 被加密字符长度较短时, 使用概率参数控制生成的负数据库无法提供令人满意的加密效果, 对于部分 SAT 求解器可能是易解的。

针对以上问题, 本文对如何生成更难解的负数据库进行探索, 结合传统加密算法 DES 算法和 RSA 算法展开了研究, 具体工作分为如下两个部分: (1) 提出了基于密钥搜索问题的负数据库生成算法 *D*-hidden; (2) 提出了基于整数分解问题的负数据库生成算法 *F*-hidden。

研究 *D*-hidden 算法的目的在于探索如何将 DES 算法与负数据库生成算法进行结合, 通过调整 DES 算法的参数, 使得负数据库的难解性与 DES 算法的难解性相关联, 从而提高负数据库的难解性。研究 *F*-hidden 算法的目的在于将整数分解与负数据库生成算法进行结合, 使得负数据库的难解性与整数分解的难解性进行关联, 从而提高负数据库的难解性。

2 国内外研究现状

2.1 RSA 与整数分解

RSA(Rivest-Shamir-Adleman 密码系统)是一种广泛应用于信息加密的算法, 该算法的核心步骤中通过选择大质数 *P* 和 *Q*, 其乘积表示为 *N*, 整数分解 *N*=*PQ* 的难解性可以有效保证 RSA 算法的难解性。目前 RSA 算法中常用的质数长度为 1024 以上, 由于计算机算力的不断发展, 1024 位长度也已逐渐满足不了要求, 部分使用者选择位数更长的质数以追求更好的安全性。RSA 算法发展至今, 对其进行攻击的研究从未停止, 但仍没有一种稳定低于指数时间复杂度的攻击算法, 因此 RSA 算法在计算机安全领域中依然有着一席之地。

王兴波等[3]在工作中对 RSA 和整数分解领域近 50 年的工作进行整理, 认为目前依然处于一个瓶颈期。他们指出目前破解整数分解的算法在大整数范围中使用最广的是数域筛法 NFS(Number Field Sieve)[4]。尽管 NFS 算法是目前应用最广的, 但其算法复杂度依然到达 $O\left(e^{(64/9*\ln(p))^{1/3}*(\ln\ln(p))^{2/3}}\right)$[5]。随着高性能计算的普及, 并行计算逐渐开始流行, 已有研究认为 NFS 在并行环境中具有优秀的效果[6-7]。此外, 整数分解领域比较经典的方法还有 Pollard 在 1975 年提出的ρ方法[8], 该方法基于随机数, 在大整数具有较小的因数时具有较好的效果。随后 Pollard 等[9]提出了基于费马小定理的 P-1 法。在 1987 年 Lenstra 提出了思想与 P-1 法相似但基于椭圆曲线的 ECM(Elliptic Curve Method)方法[10]。工作[11-12]中提出了可以高效分解十进制在 50~100 位的二次筛法 QS(Quadratic Sieve)。在量子计算机发展起来后, 付向群等[13]提出了基于量子计算机的整数分解算法。Patsakis[14]对关于 RSA 中质数暴露多少个二进制位可以在多项式时间内快速破解的工作进行了整理分析。Zhang 等[15]针对 RSA 算法的几种常见的攻击方式进行了研究。相比 NFS, 这些算法都在满足一定条件时才能够表现较好, 面对大整数时依然 NFS 表现最佳。尽管整数领域的研究从未停止, 但在算法复杂度上并没有取得质的进展。目前已有的认为 RSA 可

在多项式时间内破解的工作均基于一定的前提条件, 如 RSA 使用的模数 e=3 等。从总体来看攻击 RSA 算法若想要在多项式时间内完成, 都需要满足一些特定的条件。

由于整数分解的难解性, RSA 算法得到理论保证, 因此 RSA 算法目前依然有非常广泛的应用, 如郑钦元等将改进 RSA 算法应用于 HTTP 传输[16]、杨径山[17]将 RSA 与区块链结合用于电子病历操作记录的隐私保护。

2.2 整数分解与 SAT 实例

目前, 具有代表性的将整数分解生成 SAT 实例的开源实现有两个。其中一个由 Purdom 和 Sabry 用 Haskel 编写[18], 他们使用了两种加法电路和三种乘法电路, 分别是常见的加法电路、一种对数时间的加法电路[19]、进位保存乘法电路、Wallace-tree 乘法电路[20]以及基于 Karatsuba 快速乘法[21]的乘法电路。其中 Wallace-tree 乘法电路基于 Wallace 结构, 乘法的积是许多个部分和之和, 使用对数个加法器进行相加, 相比原始方法使用线性个加法器能够加快乘法器的速度。Karatsuba 算法主要用于加快大数相乘, 相比普通乘法的时间复杂度 $O(n^2)$, Karatsuba 算法[22]的时间复杂度为 $O(3n^{\log_2 3}) \approx O(3n^{1.585})$。

另一个开源实现则是由 Yuen 和 Bebel 使用 Python 编写[23-25]的。他们最初使用进位保存的乘法电路, 随后进行了 Wallace-tree 等优化, 对所产生的 SAT 实例的复杂度进行优化, 使得相同大小的质数产生更少的变量和子句的实例[25]。

以上两种方法生成的实例中, 变量数目的量级都在 n^2 级别。在考虑结果实例变量更少的方向中, 基于 FFT 的乘法电路[26], 使用了 FFT 优化乘法的步骤, 这种方法生成 SAT 实例变量数目可以降低至 $O(n\lg(n)\lg\lg(n))$, 这是当前已有的方法中生成变量数目复杂度最小的方法, 缺点是方法相对复杂并且存在较大的常数。

Horie 和 Watanabe[27]使用中国剩余定理, 将整数分解问题生成 SAT 实例, 生成的 SAT 实例中的变量复杂度为 $O\left(n^{(1+a)}\right)$, 其中 a 是大于零的固定常数。

2.3 负数据库

NDB 将数据以负表示的形式进行存储和操作, 以达到隐私保护和信息隐藏的目的[1-2]。对于给定的全集 $U=\{0,1\}^m$, 正数据库 $DB=\{x_1, x_2,\ldots, x_n\}$, 对应的负数据库 NDB_x 存储的信息为 *U-DB* 的数据。为更好地区分正数据库与负数据库, 负数据库中的字符串又称为“记录”。若 NDB_x 覆盖 *U-DB* 中的所有信息, 就称这个负数据库 NDB_x 是完备的。通常情况下, NDB_x 会存在大量记录, 所以常用通配符‘*’进行压缩以降低存储开销, 如“*1”是“01”和“11”两者的压缩表示。负数据库中, 一个记录中为‘1’和‘0’的位称为确定位, ‘*’称为不确定位。

目前已有不少生成 *NDB* 的算法。前缀算法[2]是第一个生成 *NDB* 的算法, 但该算法生成的 *NDB* 容易求解。Esponda 等[2]提出了生成 *NDB* 的 RNDB 算法, 其缺点是不能控制生成的 *NDB* 的不可逆性。他们在实验中表明, 将 *NDB* 重构成 *DB* 是一个 NP 完全问题, 其时间复杂度与 *NDB* 大小相关。

Jia 等[28]提出了单串负数据库生成算法 q-hidden, 该算法用于生成 3-*NDB*。算法每次迭代生成一个含 3 个确定位的字符串 w, 其中 $i(1\leq i\leq 3)$个字符与隐藏串不匹配, 算法会以概率 q_i 来接受字符串 w。Liu 等[29]提出 p-hidden 算法, 通过两个可变参数 p_1 和 p_2 来控制 *NDB* 中的不同类型的记录的比例。参数 p_1 和 p_2 的取值直接影响 *NDB* 的难解性, Barthel 等[30]也对 p_1 和 p_2 的取值对 *NDB* 难解性的影响进行了分析。Zhao 等[31]提出生成 *NDB* 的 *K*-hidden 算法, 该算法通过控制 K 的大小以及参数$\{p_1, p_2, \ldots, p_K\}$的数值来控制生成的 *NDB* 中, 记录的确定位数目和记录类型分布, 从而控制结果 *NDB* 的难解性。Zhao 等[32]中提出的 *QK*-hidden 算法, 该算法中参数$\{p_i\}$控制生成的记录类型和 *NDB* 的难解性, $\{Q_i\}$控制 *NDB* 的记录分布, 从而控制 *NDB* 应用于聚类或分类的性能。Zhao 等[32-34]表示基于 *QK*-hidden 的聚类和分类性能明显优于其他方法。随后 Zhao 和 Luo[35]又提出了实值 *NDB* 及其生成算法, 同时该工作提出了负数据库的半同态性质, 即在已知一个负数据库 NDB_x 和它的隐藏串 x 的前提下, 可以得到另一个等长 01 字符串 y 的负数据库, 步骤为 $NDB_y=NDB_x\oplus(x\oplus y)$, 其中$\oplus$表示异或操作。该性质是本文提出的 *NDB* 生成方法 *D*-hidden 和 *F*-hidden 的重要基础。

3 前置知识

3.1 布尔可满足性问题

布尔可满足性问题(Boolean Satisfiability Problem, SAT)是计算机科学的核心, 它和算法及复杂性理论中的基本问题有关。可以理解为, 给定逻辑公式, 判断该逻辑公式是否可满足。通过对解空间进行搜索, 判断是否能够找到满足逻辑公式的解, 或证明解空间内都无法满足。

形如式(1), 表示含有 m 个子句, 每个子句含有 n 个文字, 子句间进行合取, 每个子句内的文字进行析取, 该式子称为合取范式, 又称 CNF (Conjunctive Normal Formula)。

$$CNF = \wedge_{i=1}^{m}\{\vee_{j=1}^{n} l_{ij}\} \tag{1}$$

其中, l_{ij} 表示第 i 个子句中的第 j 个文字, 具体为 x 或 $\neg x$, 前者称为正文字, 后者称为负文字。在一个 SAT 公式中使用的所有变量的集合 $X=\{x_1, x_2,\ldots, x_n\}$ 称为该公式的变量集, 变量集大小 n 称为该公式的问题规模。

对于子句 $C=l_1\vee l_2\vee\ldots\vee l_n$, 若要使其可满足, 需要其中包含的 n 个文字至少一个为真。对于正文字 $l_i=x_i$, 当且仅当 x_i=true 时为真。对于负文字 $l_i=\neg x_i$, 当且仅当 x_i=false 时为真。对于合取范式 CNF=$C_1\wedge C_2\wedge\ldots\wedge C_m$, 当且仅当所有子句为真时 CNF 为真。

对于给定规模大小为 n 的变量集 X 和合取范式 CNF, 判断 CNF 是否可满足, 即 SAT 问题。对 SAT 问题进行求解是指寻找一组赋值 $\{x_1, x_2,\ldots, x_n\}$ 使得 CNF 为真。问题规模大小 $|X|=n$, 每个变量赋值可以为 true 或 false, 故解空间的大小为 2^n, 遍历整个解空间所需要的成本会非常高。因此, 定义 SAT 问题的求解算法是指在可接受的时间内能够给出 CNF 的一个具体解, 或判定 CNF 无法满足的算法, 目前较为经典的 SAT 求解算法有 DPLL[36-37]、CDCL[38]等。

3.2 SAT 与 NDB 的等价关系

Esponda 等[1-2]的研究表示, SAT 公式与 *NDB* 等价, 在表示形式与求解过程中两者均可表示为相同形式, 同时 *NDB* 的隐藏串与 SAT 公式求解的结果等价。两者在内容上, *NDB* 中存储的每一个记录, 与 SAT 实例中的每一个子句相对应。

在 *NDB* 中的每一个记录里, 由‘0’、‘1’、‘*’3 种字符组成。如表 1 所示, 在 *NDB* 记录中的第 i 位为‘0’对应在 SAT 实例的相应子句中存在 x_i 且没有$\neg x_i$; 在 *NDB* 记录中的第 i 位为‘1’, 对应在 SAT 实例的相应子句中存在$\neg x_i$ 且没有 x_i; 在 *NDB* 记录中的第 i 位为‘*’对应在 SAT 实例的相应子句中 x_i 和$\neg x_i$ 都不存在。

表 1 *DB*、*NDB* 及 SAT 之间的关系

Table 1 Relationship among *DB*, *NDB* and SAT

DB	*NDB*	SAT
100	00*	$x_1 \vee x_2$
101	*10	$\neg x_2 \vee x_3$
	*11	$\neg x_2 \vee \neg x_3$

SAT 实例转换为负数据库的方法如算法 1 所示, 算法输入为需要转换的 CNF_x, 输出为目标 *NDB*。算法在初始化时获取 CNF_x 的变量数目 n, 并且将结果 NDB_x 赋为空。然后对 CNF_x 中的所有子句进行遍历, 每次迭代首先生成一个包含 n 个通配符‘*’的字符串 x。然后遍历当前字句的文字, 若遍历到正文字, 那么字符串 x 对应位变为‘0’。相反地, 负文字则为‘1’。最后将修改完成的字符串 x 加入到 NDB_x 中。当 CNF_x 中所有子句遍历完成后, 得到 CNF_x 等价的负数据库 NDB_x。

算法 1 SATtoNDB

输入: CNF_x

输出: NDB_x

```
1  n ← the number of variable in CNF_x;
2  NDB_x ← ∅;
3  for clause in CNF_x:
4      initialize a string x with n bit '*';
5      for l in clause:
6          if (l > 0) x[|l| - 1] = 0;
7          else x[|l| - 1] = 1;
8      end for
9      NDB_x ← NDB_x ∪ x.
10 end for
11 return NDB_x.
```

3.3 DES 算法

本节介绍 DES(Data Encryption Standard)算法, 其主要包括 Feistel 结构、轮函数和子密钥生成 3 个模块, 图 1 给出了每个模块的详细步骤。

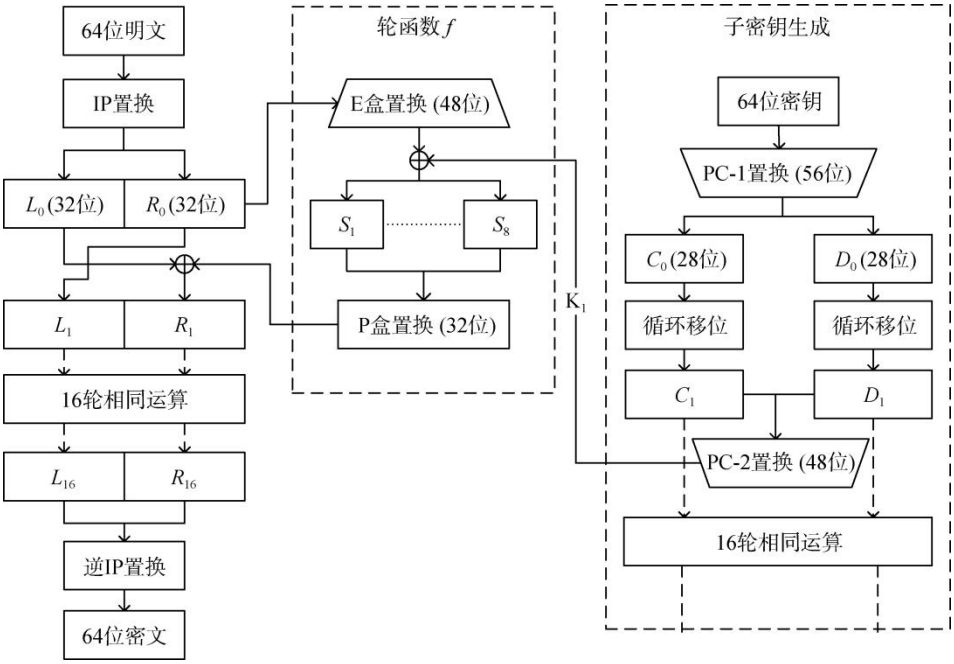

图 1　DES 算法的加密流程

Figure 1　The encryption process of DES algorithm

Feistel 结构在经过初始化后 (IP 置换) 将 64 位明文等分为两部分, 接着将其中一部分与子密钥输入轮函数并得到输出, 然后将输出与另一部分异或, 最后交换这两部分。上述过程的组合被视为一轮的操作, 其形式化描述如下:

$$\begin{aligned} L_i &= R_{i-1} \\ R_i &= L_{i-1} \oplus f\left(R_i, K_i\right) \end{aligned} \tag{2}$$

其中, 下角标 i 表示轮数, K_i 表示第 i 轮的子密钥, f 表示轮函数。完整的 DES 算法共有 16 轮, 在最后一轮中, 两部分拼接后进行逆 IP 置换得到 64 位密文。轮函数 f 首先将输入的 R_i 进行 E 盒置换扩展到 48 位, 然后与 K_i 异或, 接着将结果等分为 8 份并分别输入 8 个 S 盒 (每个 S 盒有 6 位输入, 4 位输出), 最后将 32 位输出进行 P 盒置换。子密钥生成模块首先将 64 位密钥进行 PC-1 置换压缩到 56 位, 然后将其等分为两部分, 查询移位表后分别对两部分进行移位, 最后拼接并进行 PC-2 置换压缩到 48 位作为一轮的子密钥 。

4　本文算法原理

本章介绍基于 DES 算法的负数据库生成算法和基于整数分解的负数据库生成算法。

4.1　DES 转换为 SAT 实例

如图 2 所示, *D*-hidden 生成基准组阶段用于生成包含一个基准串 x 和一个基准负数据库 NDB_x 的基准组。该阶段首先将 DES 算法编码为等价的 SAT 实例 (用 NDB_0 表示)。接着随机生成 64 位明文与 56 位密钥, 并调用加密算法得到 64 位密文。再将明文密文对和明文密钥对分别插入 NDB_0, 通过重复执行单元传播策略, 以及不同的后续处理方案, 得到等价于 DES 密钥搜索问题的负数据库 NDB_x 及其隐藏串 x。

4.1.1　DES 的 SAT 编码方案

将传统加密算法编码为 SAT 实例的想法最早由 Massacci 和 Marraro[39]提出, 本文使用的方法在编码思路上与其相似, 但在部分细节上有所不同, 具体区别将在描述时给出。

对 DES 算法的整体编码思路为: 按加密步骤依次编码, 对所有置换不进行编码。编码时将每一个比特视为一个变量, 由于置换操作的本质是对输入打乱顺序, 所以并不需要引入新的公式来表示这种约束。值得注意的是, 移位操作也可以被视为置换操作的一种, 所以也无需编码。根据图 1 可以得知, 除了置换和移位操作, 其他的操作类型仅包括异或和 S 盒, 故仅需对这两种操作进行编码即可。根据这些操作, 可以生成如下公式:

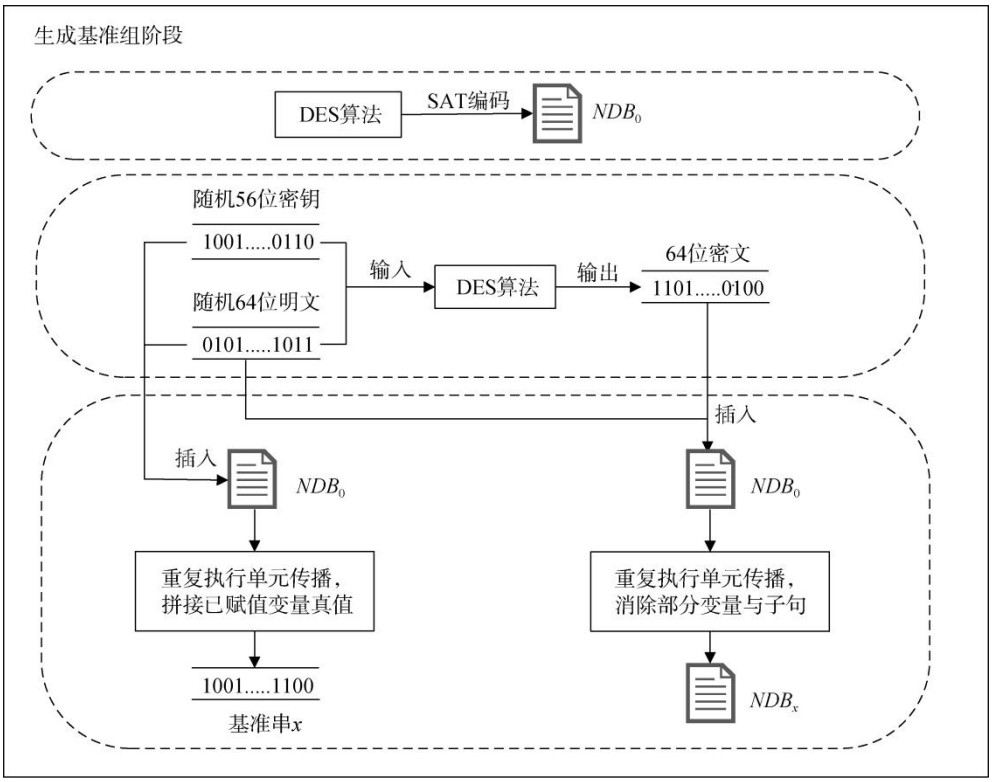

图 2　*D*-hidden 算法生成基准组阶段

Figure 2　*D*-hidden algorithm generates the benchmark group stage

$$\begin{aligned} R_i &\Leftrightarrow L_{i-1} \oplus P\left(Y_i\right) \\ X_i &\Leftrightarrow E\left(R_{i-1}\right) \oplus K_i \\ Y_i &\Leftrightarrow S\left(X_i\right) \end{aligned} \tag{3}$$

其中, P、E 和 S 分别表示 P 盒置换, E 盒置换和 S 盒操作, X_i 和 Y_i 分别表示第 i 轮的 S 盒输入和输出。此外, 还需要将式(3)中的内容转化为 CNF 范式, Massacci 和 Marraro[39]选择了一个多值逻辑函数的最小化工具对其进行编码。本文选择根据真值表直接进行编码, 这是一种最朴素也是最直接的方式。

表 2 给出了根据异或操作的真值表直接得出对应 CNF 范式的示例, 其中 x_1 和 x_2 表示输入, y 表示输出, 对应地可以将公式 $y \Leftrightarrow x_1 \oplus x_2$ 编码为 4 个长度为 3 的子句的合取范式。进一步地, 对于一个具有 m 位输入, n 位输出的真值表, 可以将其编码为 $2^m \times n$ 个长度为 m+1 的子句的合取范式。基于此, 对每个 S 盒编码后可以得到 $2^6 \times 4 = 256$ 个长度为 7 的子句的 CNF 范式。

通过上述编码方式, 最终可以得到一个对应于 DES 算法本身的 SAT 实例, 为了便于描述, 后文将其称为原始 SAT 实例 (或原始负数据库) 。原始 SAT 实例的变量由明文变量、密钥变量、密文变量和其他中间变量组成。将完整的 16 轮 DES 算法编码后的 SAT 实例共包含 1912 个变量, 其中不同变量的分布如表 3 所示。

表 2　编码异或示例

Table 2　Example of coding anomalies

x_1	x_2	y	子句
0	0	0	$x_1 \vee x_2 \vee \bar{y}$
0	1	1	$x_1 \vee \bar{x_2} \vee y$
1	0	1	$\bar{x_1} \vee x_2 \vee y$
1	1	0	$\bar{x_1} \vee \bar{x_2} \vee \bar{y}$

表 3　完整轮数的 DES 算法编码后的变量分布

Table 3　Distribution of variables after encoding by DES algorithm for the complete number of rounds

密钥变量	明文变量	中间变量	密文变量
1~56	57~120	121~1768 1801~1880	1769~1800 1881~1912

表 3 中的数字范围表示对应变量的索引区间。其中, 密钥变量和明文变量属于初始索引, 二者的

相对顺序由具体的代码实现决定, 并无影响。在表 3 中, 密文变量的索引区间并不连续, 这是因为在最后一轮拼接的两部分中, 一部分来自上一轮的 R_i, 另一部分由上一轮的 L_i 和轮函数的输出异或而来, 而异或会引入新的变量和公式, 从而导致密文变量的索引被划分为两部分。

4.1.2 生成基准组阶段: *D*-hidden 算法

为了将负数据库生成与 DES 算法相结合, 首先需要将 DES 算法对应的 SAT 实例表示为基准组, 以便使用负数据库的半同态性质对指定字符串生成对应的负数据库。该过程如算法 2 所示, 算法的输入为 64 位明文 P、56 位密钥 K、轮数 *round* 和等价于 DES 算法本身的负数据库 NDB_0, 输出为串 x 及其对应负数据库 NDB_x (即一个基准组)。

首先, 算法的第 1 步根据明文 P、密钥 K 和轮数 *round* 调用 DES 算法加密得到密文 C, 接着第 2~3 步将 P 和 C 插入 NDB_0 得到 NDB_x, 并在 NDB_x 上重复执行单元传播, 消除部分变量和子句。随后第 4~5 步将 P 和 K 插入 NDB_0 得到 NDB_t, 并在 NDB_t 上重复执行单元传播, 然后拼接 NDB_t 中已赋值的变量的真值得到串 t。最后, 算法的第 6 步根据 t, 找到 NDB_x 中剩余未赋值的变量的真值并拼接得到串 x, 然后返回 x 和 NDB_x。

算法 2	***D*-hidden 算法生成基准组**
输入:	64 位明文 P, 56 位密钥 K, 轮数 *round*, 原始负数据库 NDB_0
输出:	x 和 NDB_x 组成的基准组
1.	C←DES(P, K, *round*) //调用不同轮数的 DES 算法加密
2.	NDB_x←向 NDB_0 中插入 P 和 C
3.	在 NDB_x 上重复执行单元传播, 消除已赋值的变量和子句
4.	NDB_t←向 NDB_0 中插入 P 和 K
5.	t←在 NDB_t 上重复执行单元传播, 并拼接已赋值变量的真值
6.	x←在 t 中找到 NDB_x 未赋值变量的真值并拼接
7.	**return** x, NDB_x

需要说明的是, 算法的第 2 步和第 4 步中的插入操作需先找到指定变量的索引 (根据表 3 中的变量分布), 然后根据其真值向 NDB_0 中插入对应的文字。例如, 某个密钥变量在 NDB_0 中的索引为 1, 56 位密钥中对应真值为 0, 则此时向 NDB_0 中插入负文字 $\overline{x_1}$(即单元子句), 真值若为 1 则插入正文字 x_1。此外, 算法的第 2 步获得的 NDB_x 等价于 DES 算法的密钥搜索问题 (即已知 P 和 C, 求 K) , 对 NDB_x 进行单元传播后仅能消除部分变量, 最后会剩下密钥变量和部分中间变量。而第 4 步获得的 NDB_t 等价于 DES 算法的加密过程 (即已知 P 和 K, 求 C) , 对 NDB_t 进行单元传播后会扩散到整个实例, 进而 NDB_t 中的所有变量均会被赋值, 这意味着拼接后的串 t 会包括 NDB_x 所有变量的真值。基于此, 算法的第 6 步才能根据 t 找到 NDB_x 中所有变量的真值并拼接得到串 x, 而 x 正是 NDB_x 的唯一隐藏串 (得益于 DES 密钥的唯一性) 。

当参数 *round* 不同时, 算法 2 生成的基准组有较大差异。表 4 给出了根据不同轮数的 DES 算法编码获得的原始负数据库(NDB_0)和基准负数据库(NDB_x)的详细信息。可以发现, 当轮数大于 3 时, NDB_x 的串长(即基准串长)与记录数相比于 NDB_0 都分别减少了 128 和 448, 而当轮数为 2 或 3 时则减少得更多。这是因为当轮数大于 3 时, 向 NDB_0 中插入明文变量和密文变量的真值后, 执行单元传播仅能影响到前两

表 4 不同轮数的基准组信息

Table 4 Baseline group information for different number of rounds

round	NDB_0 串长	NDB_0 记录数	NDB_x 串长	NDB_x 记录数
2	344	4376	152	2270
3	456	7104	296	5632
4	568	9472	440	9024
5	680	11840	552	11392
6	792	14208	664	13760
7	904	16576	776	16128
8	1016	18944	888	18496
9	1128	21312	1000	20864
10	1240	23680	1112	23232
11	1352	26048	1224	25600
12	1464	28416	1336	27968
13	1576	30784	1448	30336
14	1688	33152	1560	32704
15	1800	35520	1672	35072
16	1912	37888	1784	37440

轮与最后两轮的部分异或操作生成的公式, 这导致会减少 2×64 = 128 个变量和 2×(32×2×2+48×2) = 448 条记录。而当轮数为 2 或 3 时, 由于轮数过少, 部分含有明文变量和密文变量的子句会包含相同的中间变量, 这导致单元传播会进一步扩散, 从而减少更多的变量和子句数量。

4.1.3 生成目标负数据库阶段

如图 3 所示, 生成目标负数据库阶段用于生成隐藏了用户输入串 s 的负数据库。该阶段首先将隐藏串 s 的长度与基准串 x 对齐, 接着将 s 与 x 异或得到中心串 k, 然后将 k 与基准负数据库 NDB_x 异或得到目标负数据库 NDB_s。根据负数据库的半同态异或性质可知, NDB_s 正是隐藏了串 s 的负数据库。

算法 3 给出了 D-hidden 算法生成目标负数据库阶段的伪代码。算法的输入为一个基准组 、隐藏串 s 和模板串 s_0, 其中 s_0 是一个长度为 1912 的固定且公开的随机串, 用于补齐隐藏串。算法的输出为 s 的负数据库 NDB_s。

首先, 算法的第 1 步计算 s 和 x 的长度差, 并赋值给 u, 接着第 2 步在 s 的末尾拼接 s_0 的前 u 位。算法的第 3 步将 s 和 x 异或得到串 k, 接着在第 4 步将 NDB_s 初始化为空集。然后, 第 5~8 步遍历 NDB_x 的每条记录, 将其与 k 异或后添加到 NDB_s 中作为一条新记录。需要说明的是, 串 k 与一条记录异或时, 仅计算确定位的结果, 非确定位'*'不做操作直接保留。最后, 算法的第 9 步返回 NDB_s, 即隐藏了串 s 的负数据库。

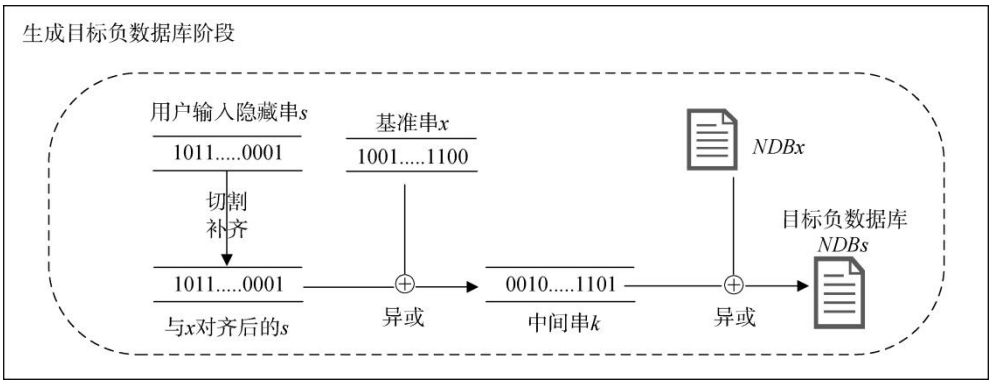

图 3 生成目标负数据库

Figure 3 Generate target negative database

通过算法 3 生成的负数据库要求隐藏串的长度不大于基准串的长度, 因此当隐藏串的长度过长时, 需要对隐藏串做一定长度的截取, 然后再调用该算法。根据负数据库的半同态异或性质, NDB_s 会保留 NDB_x 的 NP 难性质, 这意味着逆转 NDB_s 的难度将等价于求解由 DES 密钥搜索问题编码而来的 SAT 实例。

4.1.4 安全性与效率分析

D-hidden 负数据库的安全性与 DES 密钥搜索问题的难解性息息相关。但由于算法将密钥搜索问题编码为了 SAT 实例, 且通过单元传播策略消除了明文与密文, 在这种场景下, 常见的一些针对密钥搜索问题的攻击方法(例如差分密码分析和线性密码分析等)无法适用, 故这里仅讨论 D-hidden 负数据库对于 SAT 求解器的安全性。

实际上, D-hidden 负数据库的安全性仅依赖于 n 位隐藏串中的密钥变量对应的位, 这意味着实际搜索空间为 2^{56}。由于 SLS 求解器的搜索机制并不包含变量间的蕴含推导, 即使已知密钥变量所在位, 其仍无法依靠密钥变量推导出其他变量的真值, 故搜索空间仍为 2^n, 预计仅能在负数据库规模较小(即 *round* 较小)时求解成功。而 CDCL 求解器通过一些先进的启发式搜索算法可以将搜索空间逐步定位到密钥变量, 并进一步通过单元传播与冲突学习的机制求解, 但预计这个过程的耗时会随着 *round* 的增加而增加。对于搜索空间较小的问题, 可以考虑编码更复杂的加密算法来替换 DES(例如 3DES、AES 等), 但通常更复杂的约束会引入更多的公式, 这导致负数据库的规模进一步增加, 带来更多的运算与存储开销。此外, 当隐藏串 s 与基准串 x 的长度不一致时, 基于模板串 s_0 的补齐操作可能会降低 D-hidden 负数据

库的安全性, 因为 s_0 的信息是公开的, 而暴露的补齐位的信息会破坏 DES 算法的结构, 并被攻击者利用。而当 s 与 x 的长度一致且 $round$ 足够时, 预计 D-hidden 负数据库将会保持在较高的安全性。

算法 3	***D*-hidden 算法生成目标负数据库**
输入:	x, NDB_x 长度为 n 的隐藏串 s 模板串 s_0
输出:	NDB_s
1.	$u \leftarrow \|x\|-n$
2.	$s \leftarrow s$ 拼接 s_0 的前 u 位
3.	$k \leftarrow s \oplus x$
4.	$NDB_s \leftarrow \emptyset$
5.	**for** $v \in NDB_x$ **do**
6.	$t \leftarrow k \oplus v$
7.	$NDB_s \leftarrow NDB_s \cup t$
8.	**end for**
9.	**return** NDB_s

在效率方面, 编码 DES 的时间复杂度为常数级。算法 3 的第 3 步和第 5 步的时间复杂度为 $O(m)$, 第 6 步的时间复杂度为 $O(n)$, 其中 n 表示基准串 x 的长度, m 表示基准负数据库 NDB_x 的记录数, 具体大小见表 4。而通常情况下, n 小于 m, 故算法 3 的整体时间复杂度为 $O(m)$。但由于该算法涉及多次读写文件, 故实际耗时可能较多。算法 3 的第 1、3、4 步的时间复杂度为常数级, 第 2 步的时间复杂度为 $O(n)$, 第 5~8 步的时间复杂度为 $O(6.59m)$, 其中 6.59 为 NDB_x 的平均记录长度。故算法 3 的整体时间复杂度为 $O(m)$。需要说明的是, 算法 3 的操作可以离线进行从而提前存储基准组信息, 实际生成隐藏串的负数据库时, 仅需考虑算法 3 的耗时即可。

4.2 整数分解转换为 SAT 实例

本节给出整数分解问题转换为等价的 SAT 问题的方法, 流程如图 4 所示。若该 SAT 问题求解成功, 表示整数分解问题也求解成功, 结果可以从 SAT 问题的解中进行等价转换。

4.2.1 整数分解的 SAT 编码方案

一个拥有 L+1 位二进制的数 P, 它的命题变量可以表示成 $P_0, P_1, \ldots, P_L$。例如 P=8, 它的二进制表示是$(0101)_2$, 对应的变量为$\neg P_3 \wedge P_2 \wedge \neg P_1 \wedge P_0$。

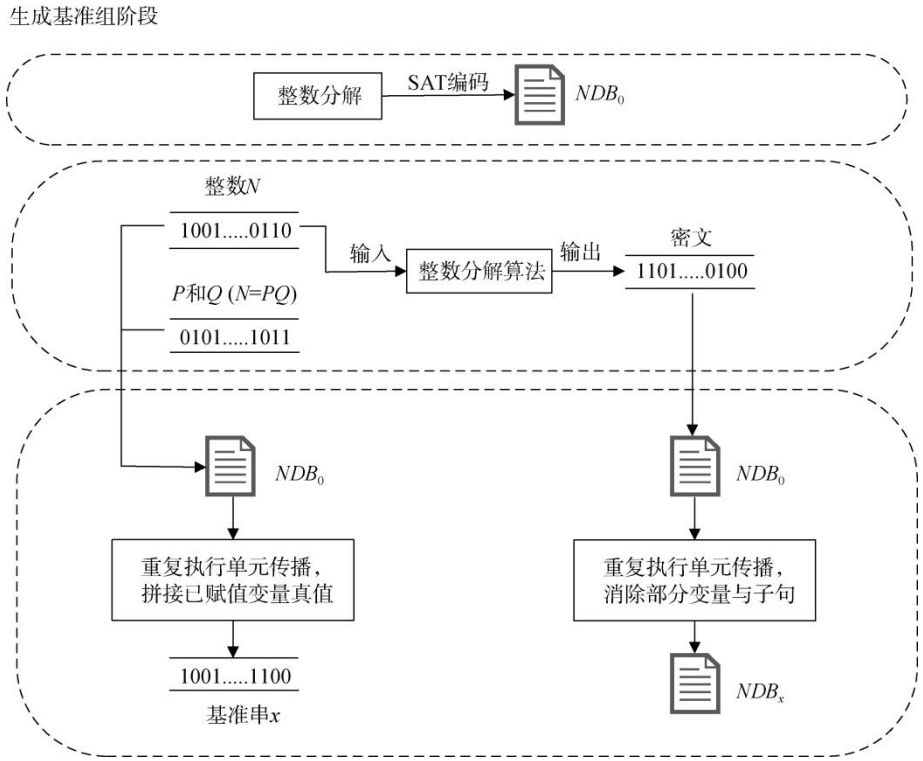

图 4 整数分解生成基准组阶段

Figure 4 Integer decomposition generates the base group stage

对于两数相等 R=P, 考虑数的二进制位, 两数相等表示它们每个二进制位都为 0 或都为 1。对于长度为 L+1 的两个数 R 和 P, 它们的二进制位的关系如式(4)所示, 转换为 CNF 如式(5)所示。

$$\wedge_{i=0}^{L}\{(R_i \wedge P_i) \vee (\neg R_i \wedge \neg P_i)\} \tag{4}$$

$$\wedge_{i=0}^{L}\{(P_i \vee \neg R_i) \wedge (\neg P_i \vee R_i)\} \tag{5}$$

在乘法运算中, 使用$R_{i+j} = P_i Q_j$表示 P 的第 i 个二进制位乘上 Q 的第 j 个二进制位。该过程的二进制的关系如式(6)所示, 转换为 CNF 如式(7)所示。

根据加法运算过程可以知道, 进位器 C 是有 L+1 位的二进制数, 在加法运算中, C_i 表示第 i 位运算是否产生进位。对于最低位的运算, C_0 表示第 0 位是否产生进位, 只需要判断 P 和 Q 的最低位是否同时为 1 即可, 否则不产生进位。该过程如式(8)所示, 转换为 CNF 如式(9)所示。对于进位器的第 i (i > 0)位, 此时只要 P 的第 i 位、Q 的第 i 位以及进位器的第 i–1 位这三者中至少两者为 1, 那么就有 C_i=1, 否则 C_i=0, 其关系如式(10)所示, 转换为 CNF 如式(11)所示。对于加法 R=P+Q, 三者都是 L+1 位的二进制数, 防止加法计算过程中溢出, 在 P 和 Q 的最高二进制位添加 0。这时可知进位器最高位 C_L 为 0, 因为对它有贡献的只有 P_L、Q_L、C_L–1, 这三者中前两者都是 0, 故不会产生进位。因此可以直接添加 SAT 公式$\neg C_L$到结果。

结合进位器 C, 考虑最低位的加法结果, R_0 为 1 的情况只有 P_0 和 Q_0 两者其中一个为 1 时成立, P_0 和 Q_0 两者都为 0 或者两者都为 1 则有 R_0=0, 该过程可以表示为式(12), 将其转换为 CNF 如式(13)所示。

$$\wedge_{i=0}^{L}\{\wedge_{j=0}^{L}\{\left(R_{i+j} \wedge \left(P_i \wedge Q_j\right)\right) \vee \left(\neg R_{i+j} \wedge \neg\left(P_i \wedge Q_j\right)\right)\}\} \tag{6}$$

$$\wedge_{i=0}^{L}\{\wedge_{j=0}^{L}\{\left(Q_j \vee \neg R_{i+j}\right) \wedge \left(P_i \vee \neg R_{i+j}\right) \wedge \left(R_{i+j} \vee \neg Q_j \vee \neg P_i\right)\}\} \tag{7}$$

$$(C_0 \wedge P_0 \wedge Q_0) \vee \left(\neg C_0 \wedge \neg(P_0 \wedge Q_0)\right) \tag{8}$$

$$\left(P_0 \vee \neg C_0\right) \wedge \left(Q_0 \vee \neg C_0\right) \wedge \left(\neg P_0 \vee \neg Q_0 \vee C_0\right) \tag{9}$$

$$\begin{aligned}&\wedge_{i=1}^{L}\{(C_i \wedge ((P_i \wedge Q_i) \vee (P_i \wedge C_{i-1}) \vee (Q_i \wedge C_{i-1})))\\ &\vee \left(\neg C_i \wedge \left(\left(\neg P_i \wedge \neg Q_i\right) \vee \left(\neg P_i \wedge \neg C_{i-1}\right) \vee \left(\neg Q_i \wedge \neg C_{i-1}\right)\right)\right)\}\end{aligned} \tag{10}$$

$$\begin{aligned}&\wedge_{i=1}^{L}\{(P_i \vee Q_i \vee \neg C_i) \wedge (P_i \vee \neg C_i \vee C_{i-1}) \wedge\\ &(Q_i \vee \neg C_i \vee C_{i-1}) \wedge (\neg P_i \vee \neg Q_i \vee C_i) \wedge\\ &(\neg P_i \vee C_i \vee \neg C_{i-1}) \wedge (\neg Q_i \vee C_i \vee \neg C_{i-1})\}\end{aligned} \tag{11}$$

$$\left(R_0 \wedge \left(\left(P_0 \wedge \neg Q_0\right) \vee \left(\neg P_0 \wedge Q_0\right)\right)\right) \vee \left(\neg R_0 \wedge \left((P_0 \wedge Q_0) \vee \left(\neg P_0 \wedge \neg Q_0\right)\right)\right) \tag{12}$$

$$\left(P_0 \vee Q_0 \vee \neg R_0\right) \wedge \left(P_0 \vee \neg Q_0 \vee R_0\right) \wedge (\neg P_0 \vee Q_0 \vee R_0) \wedge (\neg P_0 \vee \neg Q_0 \vee \neg R_0) \tag{13}$$

$$\begin{aligned}&\wedge_{i=1}^{L}\{(R_i \wedge ((P_i \wedge Q_i \wedge C_{i-1}) \vee (P_i \wedge \neg Q_i \wedge \neg C_{i-1}) \vee\\ &(\neg P_i \wedge Q_i \wedge \neg C_{i-1}) \vee (\neg P_i \wedge \neg Q_i \wedge C_{i-1})))\\ &\vee\ (\neg R_i \wedge ((\neg P_i \wedge \neg Q_i \wedge \neg C_{i-1}) \vee (P_i \wedge Q_i \wedge \neg C_{i-1}) \vee\\ &(P_i \wedge \neg Q_i \wedge C_{i-1}) \vee (\neg P_i \wedge Q_i \wedge C_{i-1})))\}\end{aligned} \tag{14}$$

$$\begin{aligned}&\wedge_{i=1}^{L}\{(P_i \vee Q_i \vee R_i \vee \neg C_{i-1}) \wedge (P_i \vee Q_i \vee \neg R_i \vee C_{i-1}) \wedge\\ &(P_i \vee \neg Q_i \vee R_i \vee C_{i-1}) \wedge (P_i \vee \neg Q_i \vee \neg R_i \vee \neg C_{i-1}) \wedge\\ &(\neg P_i \vee Q_i \vee R_i \vee C_{i-1}) \wedge (\neg P_i \vee Q_i \vee \neg R_i \vee \neg C_{i-1}) \wedge\\ &(\neg P_i \vee \neg Q_i \vee R_i \vee \neg C_{i-1}) \wedge (\neg P_i \vee \neg Q_i \vee \neg R_i \vee C_{i-1})\}\end{aligned} \tag{15}$$

对于 i > 0 的位, 需要考虑进位的影响, 当有 P_i、Q_i 和 C_{i-1} 三者都为 1, 或者三者中只有一个 1 的情况下有 R_i=1; 当 P_i、Q_i 和 C_{i-1} 三者都为 0 或只有一个 0 的情况有 R_i=0, 该过程如式(14)所示, 转换为 CNF 如式(15)所示。

Patsakis[14]指出, 在 N=PQ 的整数分解中, 假设 P 的有效位长度不大于 Q 的有效位长度, 则有 P 的二进制表示中接近一半高位为 0。在 N=PQ 中, 假设 N 的有效位为[0, L_0], 在限制 P 的有效位长度不大于 Q 的有效位长度的前提下, 可以得到 P 的最高非零位

范围是[0, ⌊L_0/2⌋], 此处⌊x⌋表示对 x 下取整, Q 的最高非零位范围为[0, L_0−1]。

在计算中需要设置数字的最大位大于 L_0, 因为 P 和 Q 的最高非零位相乘, 中间结果的最高非零位会到达⌊L_0/2⌋+ L_0−1, 需要设置数字的有效位范围为[0, ⌊L_0/2⌋+ L_0−1]。因此计算过程中间变量的有效位设置为[0, L], 其中 L 如式(16)所示。同时对 P 和 Q 确定为零的位进行设置, 如式(17)和式(18)所示。此外, 还需要对 N 确定为零的位进行设置, 如式(19)所示。

对于乘法 N=PQ, 将其等式拆解得到如所式(20)所示, 其中$P_i = 2^i$。若 P 的第 i 位为 1, 则表示 Q 乘上 2^i, 否则表示 Q 乘上 0, 最后将各项相加即可得到 N。

$$L = \left\lfloor \frac{L_0}{2} \right\rfloor + L_0 - 1 \tag{16}$$

$$P[i] = 0, i \in \left[\left\lfloor \frac{L_0}{2} \right\rfloor + 1, L \right] \tag{17}$$

$$Q[j] = 0, j \in [L_0, L] \tag{18}$$

$$N[k] = 0, k \in [L_0 + 1, L] \tag{19}$$

$$N = P_0 Q + P_1 Q + P_2 Q + \ldots + P_L Q \tag{20}$$

在 N 的计算过程中, 只需要考虑 P 的最高有效位即可, 该操作目的是尽可能减少最后生成的子句数目。使用 M_i 表示乘法中得到的每一项 P_iQ, 用 R_i 作为累加器, 累加得到的结果即为 N。在该过程中使用到的 3 种关系表示: 两数相等、乘法中一个乘数的某一个二进制位乘上另一个乘数、两数相加, 三者都已在第 2 节给出详细的 SAT 公式生成方法, 因此可以将整数分解 N=PQ 转换为 SAT 实例。

根据前文讨论的 P 的零确定位可以知道中间变量 M 共有⌊L_0/2⌋+1 个, 设置累加器的范围为[0, m], 其中 m∈[0,⌊L_0/2⌋]。因此可以得到如式(21)所示的整数分解对应 SAT 公式。

$$\left(\wedge_{i=0}^{m}\{M^i = P_i Q\}\right) \wedge (R_0 = M_0) \wedge \left(\wedge_{i=1}^{m}\{R_i = R_{i-1} + M_i\}\right) \wedge (R_m = N) \tag{21}$$

4.2.2 生成基准组阶段

为了将负数据库生成与整数分解相结合, 首先需要将整数分解算法对应的 SAT 实例表示为基准组, 以便使用负数据库的半同态性质对指定字符串生成对应的负数据库。该过程如算法 4 所示, FactToSAT 算法通过给定两个质数 P、Q, 将整数分解问题 N=PQ 生成 SAT 实例, 具体过程如算法 4 所示。根据选定的质数 P、Q, 得到乘积 N=PQ, 通过模拟进位保存的乘法过程将 N 的整数分解过程转换为 SAT 实例。

质数的二进制长度表示为 n, N 的二进制最高非零位为 L_0=2n-1, 因此 N 的有效范围为[0, L_0]。step 4 中设置的 m 为累加器的范围; step 5 中根据式(15)设置计算过程中所有数字的最高有效位, 即每个运算变量二进制可用位区间为[0, L]。step 6 中将 CNF_x 赋为空集。step 7~10 初始化确定每一个变量在需要生成的 SAT 实例中的变量的位置。例如对于变量 N、P、Q 若设置 L=7, 便对应 Var1~Var8 表示 N 的位, Var9~Var16 和 Var17~Var24 分别表示 P 和 Q 的位。最后求解的结果中, 若 Var_i=i, 则表示该位对应的变量值为 1, Var_i=-i 表示该位为 0。在 step 11~13 把表示 M_i=P[i]*Q 的 SAT 公式加入到 CNF_x 中。*Multi*(a, b, c)函数表示, a 和 c 都含多个 01 位, b 表示 0 或 2i, 返回表示 a=0*c 或 a=2i *c 的 SAT 公式。step 14 把表示 R_0=M_0 的 SAT 公式加入到 CNF_x 中。step 15~17 把表示 R_i=R_{i-1}+M_i 关系的 SAT 公式加入到 CNF_x 中。函数 *Add*(a, b, c)的作用是获取表示 a=b+c 关系的 SAT 公式并将其返回。step 18 把 R_m=N 关系的 SAT 公式加入到 CNF_x 中。step 19 把 N 的确定位加入到 CNF_x 中。step 20 把数值为 0 的特殊位的 SAT 公式加入到 CNF_x 中。根据式(16)~(18), 将 N、P、Q 三者高位确定的 0 加入结果。

此时可以得到 N=PQ 的 SAT 实例。在 step 21~22 中, 可以加入 P 和 Q 的值计算 SAT 实例中的中间变量 m, 将 P、Q 和中间变量 m 拼接得到最后生成的 SAT 实例的解串 x。最后可以得到整数分解 N=PQ 的 SAT 实例和该实例对应的解串 x。

算法 4	FactToSAT
输入:	n-bit prime numbers P,Q
输出:	CNF_x
1.	$N \leftarrow P * Q$
2.	$n \leftarrow$ the size of P or Q // both bit size are n
3.	$L_0 \leftarrow 2n - 1$
4.	$m \leftarrow \lfloor L_0/2 \rfloor$
5.	$L = \lfloor L_0/2 \rfloor + L_0 - 1$ // the bit size of

number in calculating
6. $CNF_x \leftarrow \emptyset$
7. Initialize N,P,Q
8. Initialize $\{M_0,M_1,M_2,...,M_m\}$
9. Initialize $\{R_0,R_1,R_2,...,R_m\}$
10. Initialize $\{C_0,C_1,C_2,...,C_{m-1}\}$
11. **for** i from 0 to m:
12. $CNF_x \leftarrow CNF_x \wedge Multi(M_i,P[i],Q)$ //$means\ M_i = P[i] * Q$
13. **end for**
14. $CNF_x \leftarrow CNF_x \wedge Equal(R_0,M_0)$
15. **for** i from 1 to m:
16. $CNF_x \leftarrow CNF_x \wedge Add(R_i,R_{i-1},M_i)$ // means $R_i = R_{i-1} + M_i$
17. **end for**
18. $CNF_x \leftarrow CNF_x \wedge Equal(R_m,N)$
19. $CNF_x \leftarrow CNF_x \wedge Num(N)$
20. $CNF_x \leftarrow$ special position with bit $= 0$
21. $m \leftarrow$ use the value of P and Q to calculate the middle variables
22. $x \leftarrow P + Q + m$
23. **return** CNF_x, x

4.2.3 生成目标负数据库阶段: *F*-hidden 算法

F-hidden 算法根据选取的长度为 *n* 的两个质数 *P* 和 *Q*, 对长度为 2*n* 的 01 字符串进行加密, 生成对应的负数据库, 用 SAT 形式进行表示为 CNF_s'。

F-hidden 步骤如算法 5 所示, 首先选定质数 *P*、*Q*, 使用 FactToSAT 算法生成对应整数分解的 SAT 实例 CNF_x 和该实例对应的解串 *x*。*F*-hidden 算法要求选定质数 *P* 和 *Q* 的二进制长度都为 *n*, 被加密的字符串 *s* 长度为 2*n*。

算法 5	*F*-hidden
输入:	an *m*-bit string *x*; CNF_x; a 2*n*-bit string *s*
输出:	CNF_s
1	$CNF_{s'} \leftarrow FactToSAT(s)$
2	$s' \leftarrow$ the hidden string of $CNF_{s'}$
3	$k \leftarrow x \oplus s'$
4	$CNF_s \leftarrow \emptyset$
5	**for** *clause* in CNF_x:
6	$z \leftarrow clause \oplus k$
7	$CNF_s \leftarrow CNF_{s'} \wedge z$
8	**end for**
9	**return** CNF_s

在 step 1~2 中, 将长度为 2*n* 的 01 字符串 *s* 当作伪质数乘积, 作为 FactToSAT 算法中 *P*、*Q* 的乘积, 可以得到一个将 *s* 对应的整数分解的 CNF_s', 并使用求解器求得 *s*'的隐藏串。在 step 3 中将 *s'*与 *x* 进行异或, 得到串 *k*。step 4 中先将结果实例 CNF_s 初始化为空集。step 5~8 中, 将 CNF_x 中的所有子句与串 *k* 进行异或, 异或的结果放入 CNF_s 中, 得到的 CNF_s 即为最终结果。

根据负数据库的半同态异或性质, 最终 CNF_s 对应的 NDB_s 会保留 NDB_x 的 NP 难性质, 这意味着逆转 NDB_s 的难度将等价于求解 *N*=*PQ* 的整数分解问题编码而来的 SAT 实例。

4.2.4 效率分析

在 FactToSAT 算法设计中, 对数字有效位数进行严格设置, 目的是尽可能减少最终 SAT 实例中变量的数目。

Patsakis[14]指出, 在整数分解的两个质数中, 必定有一个数接近一半的位是 0。这是用于攻击整数分解的方法, 本文中进行使用是为了使得最后生成的 SAT 实例尽可能具有唯一解。并且还需设置两个质数的二进制有效位长度不一致, 且 *P*<*Q*。本文虽然限制了 *P* 的二进制有效位长度不小于 *Q* 的二进制有效位长度, 但依然存在两者二进制有效位长度相同, 但有 *P*>*Q* 的情况。如 $P=(1101)_2=13$, $Q=(1011)_2=11$, 两者是二进制有效位长度相同的质数, 但 *P*>*Q*, 这时 FactToSAT 算法生成的 SAT 实例存在两个不同的合法解。

在算法中表示每个变量需要用到 *L*+1 个位, *N*、*P*、*Q* 三个变量需要用到 3*(*L*+1)位; *M* 和 *R* 分别有 $\lfloor L_0/2 \rfloor$+1 个变量, 共有 2*($\lfloor L_0/2 \rfloor$+1)*(*L*+1)位; 另外还有用于表示加法进位的变量 *C*, 在过程中用到了 $\lfloor L_0/2 \rfloor$次加法, 所以有$\lfloor L_0/2 \rfloor$*(*L*+1)位。

结合式(21)进行分析, 可以知道最终负数据库的变量中共有(3+2($\lfloor L_0/2 \rfloor$+1)+ $\lfloor L_0/2 \rfloor$)(*L*+1)位。根据选定的质数 *P* 和 *Q*, 在选定的质数长度同等, 均为 *n* 的情况下, 根据前面讨论, *N* 的有效位范围为 L_0=2*n*-1, 再结合式(16)可以得到 *L*=3*n*-3。因此可以得到所有变量位数之和为 $9n^2$-4。即在选定质数长度均为 *n* 的前提

下, 最终生成的 $N=PQ$ 对应整数分解的 SAT 实例产生的变量数目为 $O(9n^2-4)$。

5 实验设置

5.1 DES 生成 SAT

本文使用 SLS 类型求解器 WalkSAT[40]和 CDCL 类型求解器 Kissat[41]对负数据库进行求解。其中 WalkSAT 的版本为 2018 年 5 月发布的 v56 版本, Kissat 的版本为 2020 年 SAT 国际竞赛提交版, 同时 Kissat 也是本次竞赛中主赛道的第一名。在求解器的参数设置上, WalkSAT 被设置为最多重启 104 次, 每次最多执行 104 次翻转, 行走概率 (噪声) 为 0.5。Kissat 被设置为以“sat”模式运行, 最长执行时间为 3600 s。不同算法生成的负数据库将同时作为两个求解器的输入, 其中 WalkSAT 的总翻转次数和 Kissat 的运行时间被用作评估指标。更多的翻转次数和更久的运行时间意味着生成的负数据库的更难被求解。

5.2 整数分解生成 SAT

5.2.1 SAT 实例变量与子句数量对比实验

整数分解生成 SAT 实例的效果主要对比内容为生成的 SAT 实例中的变量数目和子句数目。通过给定质数 P 和 Q, 计算 $N=PQ$, 将 3 个数作为输入得到 SAT 实例, 比较我们提出的方法与基准方法的效果。

对于每次给定的质数 P 和 Q, 最终得到的 SAT 实例都是唯一的, 因此实验设置每种长度的质数选取 10 对进行实验, 使用这 10 对质数得到的 SAT 实例中的变量数目和子句数目的均值作为结果进行比较。

目前在整数分解生成 SAT 实例的工作中, 具有代表性的是 Purdom 和 Sabry[18]的方法, 实验中我们将本文方法与之进行对比。Purdom 和 Sabry 的实例中实现了两种加法电路和三种乘法电路, 进行组合共有 6 种电路可以使用。加法电路为普通常用的加法电路, 以及一种对数时间的加法器[19]; 乘法电路则有普通的进位保存乘法电路、Wallace-tree 乘法电路[20]以及 Recursive 电路[21]。

实验将本文的方法与 Purdom 和 Sabry 的实例中的 6 种组合方法进行对照, 比较使用相同的质数 P 和 Q 作为输入时, FactToSAT 方法是否能生成更少的变量数目以及子句。若 FactToSAT 方法能生成更少的变量数目以及子句, 那么在同等或近似大小的 SAT 实例中, FactToSAT 实例时具有更难求解的性质, 因为此时对应的整数分解的质数长度更大。

5.2.2 难解性对比实验

实验目的是验证 *F*-hidden 算法生成的 *NDB* 相比目前已有的负数据库生成算法, 对相同长度的字符串进行加密得到的 *NDB* 是否具有更好的难解性。

实验设置中使用控制变量法则, 通过设置字符串相同, 实验对相同长度的 01 字符串使用不同的负数据库生成算法 *F*-hidden、*p*-hidden、*q*-hidden、*K*-hidden、*QK*-hidden 进行处理。求解器使用 CDCL 类型求解器 Kissat[41], Kissat 的版本为 2020 年 SAT 国际竞赛提交版, 为以“sat”模式运行, 最长执行时间为 1000 s。将 SAT 求解器 Kissat 对不同方法的求解时间进行对比。

为了进一步探究算法的加密效果, 我们对所需要加密的 01 字符串的长度设置在[40, 8000]内进行探究, 以寻找不同方法在加密字符串的不同长度的优势。*F*-hidden 设置需要保护的数据长度与两个质数的长度之和相同。

在设置负数据库中隐藏串大小相同的前提下, 根据求解器 Kissat 对各个方法生成的负数据库的求解时间进行对比, 求解使用时间更少者效果更优。

6 实验结果

6.1 DES 生成 SAT

本小节分析了 *D*-hidden 算法变化轮数 *round* 和隐藏串长 *n* 对生成的负数据库的难解性的影响。

6.1.1 参数变换实验

(1) 同时变化参数 *round* 和 *n*

算法 2.1 的参数 *round* 设置为 2~16, 步长为 1。算法 3 中隐藏串的长度 *n* 设置为与每一轮基准串的长度一致(即无需补齐), 具体可见表 2~表 4 中的“NDB_x 串长”列。每组实验重复执行 30 次, 实验结果见图 5 和图 6。需要说明的是, 图中矩形内部的横线表示中位数, 符号“×”表示平均数, 圆点表示异常值。

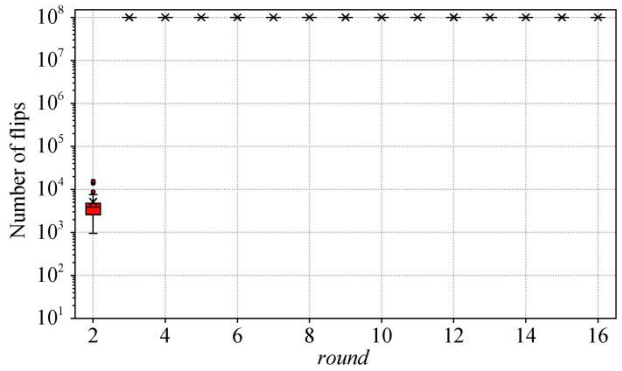

图 5　同时变化 *round* 与 *n* 时, WalkSAT 求解 *D*-hidden 负数据库的结果

Figure 5　Results of WalkSAT solving the *D*-hidden negative database when varying *round* and *n* simultaneously

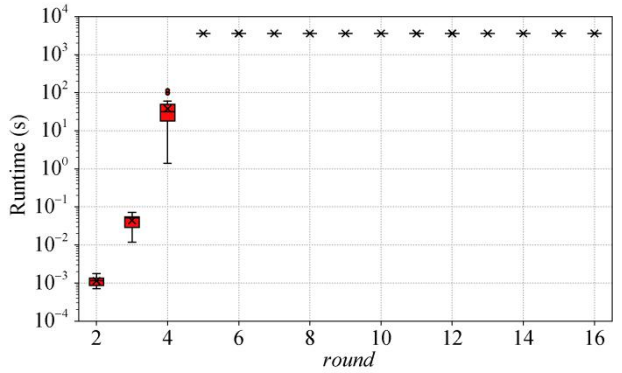

图 6　同时变化 *round* 与 *n* 时, Kissat 求解 *D*-hidden 负数据库的结果

Figure 6　Results of Kissat solving the *D*-hidden negative database when varying *round* and *n* simultaneously

如图 5 所示, WalkSAT 在 10^8 次翻转次数限制下, 仅能成功求解 *round* 为 2 的 *D*-hidden 负数据库, 对应的隐藏串长为 152 , 翻转次数大致在 4000 次左右。当轮数大于 2 时, 在 30 次重复实验中, WalkSAT 均无法在规定翻转次数内逆转 *D*-hidden 负数据库, 而 *round* 为 3 的 *D*-hidden 负数据库对应的隐藏串长仅有 296 位, 此时传统的基于记录分布模型的负数据库生成算法 (例如 *K*-hidden 算法) 将很难生成稳定难解的负数据库。因为串长越小, 意味着求解器的搜索空间也越小, 局部搜索策略可以很容易找到负数据库的隐藏串。但 *D*-hidden 负数据库给隐藏串的各个位置之间建立了强约束关系 (异或和 S 盒) , 这种约束关系随着 *round* 的增加层层传递, 导致局部搜索策略在翻转某个变量时, 很容易陷入局部最优的情况, 即反复在某一轮引入的变量集上翻转, 从而无法找到隐藏串。

Kissat 的求解情况如图 6 所示, 相比基于局部搜索策略的 WalkSAT, Kissat 的表现要更好, *round* 小于 5 的 *D*-hidden 负数据库均可以在 100 s 内求解。而当 *round* 不小于 5 时, Kissat 无法求解出任何 *D*-hidden 负数据库, 且 5 轮的 *D*-hidden 负数据库对应的隐藏串长仅为 552。虽然 *D*-hidden 负数据库的实际搜索空间仅为 2^{56}, 且不会随着 *round* 的增加而增加, 但 SAT 求解器不知道该信息, 尽管有一些先进的启发式搜索策略, 它的初始搜索空间依然是 2^n。所以当 *round* 较少时 (例如小于 5) , Kissat 利用学习子句与冲突分析的机制可以逐步将搜索空间定位到密钥变量上, 从而加快求解速度。但当 *round* 增加时, 随着引入的中间变量越来越多, Kissat 在发生冲突时更难定位到密钥变量, 更多的时间花费在求解中间变量上, 导致求解失败。

(2) 变化参数 *n*

参数 *round* 设置为 5, 对应的基准串长为 552。参数 *n* 设置为 100~500, 步长为 20。需要说明的是, 为了让 SAT 求解器能充分利用模板串 s_0 的信息, 实验时将隐藏串末尾补齐的相应长度的串额外插入生成的负数据库中, 使其以单元子句的形式暴露给求解器。上述每组参数设置下的实验重复执行 30 次 , 实验结果见图 7 和图 8。

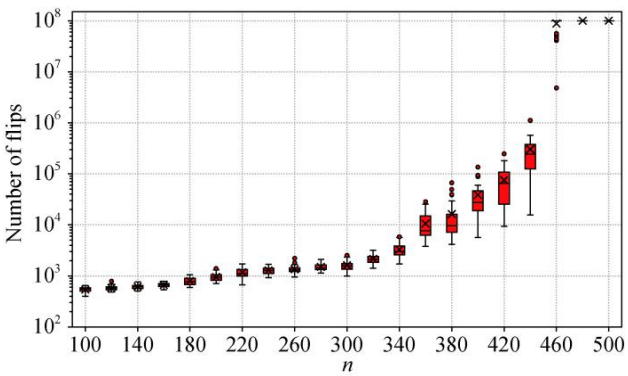

图 7　变化 *n* 时 WalkSAT 求解 *D*-hidden 负数据库结果

Figure 7　Results of WalkSAT solving the *D*-hidden negative database when varying *n*

如图 7 所示, 随着 n 的增加, 需要补齐的位数(即暴露的信息)随之减少, WalkSAT 求解 *D*-hidden 负数据库的难度逐渐增加, 二者间呈现出曲线相关的趋势。当 n 不小于 460 时(即补齐的位数不大于 92), WalkSAT 无法在规定翻转次数内逆转 *D*-hidden 负数据库。而在图 5 的结果中, *D*-hidden 负数据库在串长为 296 时就已无法被 WalkSAT 成功求解。这表明补齐的位进一步影响了其他位的信息, 使得整个负数据库的难解性下降。根据上述结果可以推测出, 当隐藏串中补齐的位数小于 100 时, *D*-hidden 负数据库仍然能对 WalkSAT 保持较高的难解性。

图 8 展示了 Kissat 的求解情况, 可以发现, 暴露部分信息后的 *D*-hidden 负数据库对于 Kissat 是十分易解的。相比于串长刚好等于 552 时 Kissat 无法求解任意 *D*-hidden 负数据库的情况, 二者的效果相差较大。可能的原因是当轮数较低时, 补齐的位中包含部分密钥信息, 导致 Kissat 可以通过单元传播机制将该部分信息迅速传播, 从而实现快速求解。

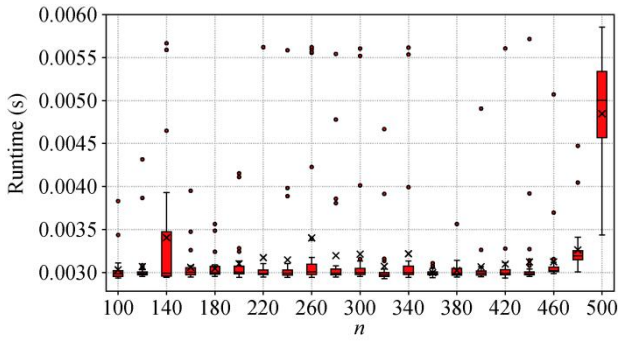

图 8 变化 n 时, Kissat 求解 *D*-hidden 负数据库的结果

Figure 8 The results of Kissat solving the *D*-hidden negative database when varying n

(3) 变化参数 *round*

参数 *round* 设置为 4~16, 步长为 1。参数 n 设置为 100。同样地, 实验时将补齐的信息暴露给求解器。上述每组参数设置下的实验重复执行 30 次, 实验结果见图 9 和图 10。

如图 9 所示, 当 *round* 增加时, 需要补齐的位数随之增加, WalkSAT 求解 *D*-hidden 负数据库的难度也逐渐增加。当 *round* 大于 6 时, WalkSAT 无法在规定翻转次数内逆转 *D*-hidden 负数据库。这表明虽然轮数增加会使暴露的信息更多, 但负数据库的总记录数的增加似乎阻止了暴露信息的扩散, 即使 WalkSAT 已知补齐位的信息, 其仍无法通过局部搜索策略成功求解出 s 的前 100 位。根据实验结果可以得出, 当隐藏串长与基准串长不一致时, 将隐藏串补齐到 *round* 大于 6 时的基准串长会更优。

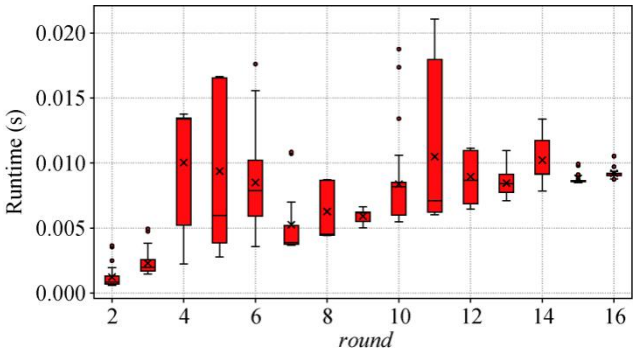

图 9 变化 *round* 时, WalkSAT 求解 *D*-hidden 负数据库的结果

Figure 9 Results of WalkSAT solving the *D*-hidden negative database when varying *round*

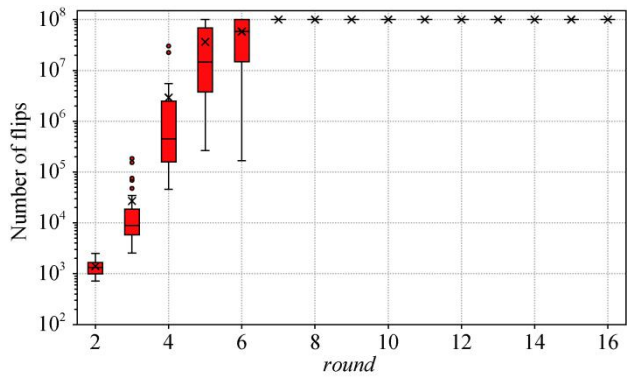

图 10 变化 *round* 时, Kissat 求解 *D*-hidden 负数据库的结果

Figure 10 Results of Kissat solving the *D*-hidden negative database when varying *round*

图 10 中 Kissat 的求解情况与仅变化 n 时的实验结果一致。无论 *round* 如何变化, 暴露的位数增加或总记录数增加, Kissat 均能在秒级内成功求解 *D*-hidden 负数据库, 耗时极少。结合前文的实验结果表明, 对于 Kissat 而言, *D*-hidden 算法的补齐操作会严重降低其生成的负数据库的难解性, 只有当隐藏串长与基准串长一致且 *round* 足够时(例如不小于 5), *D*-hidden 算法才能生成对 Kissat 难解的负数据库。

6.1.2 对比试验

本小节比较了 *K*-hidden 负数据库与 *D*-hidden 负数据库的难解性, 以及两种生成算法的效率。为了公平起见, 这里选择先进行寻找 *K*-hidden 算法的最优参数组合的实验, 然后再将其与 *D*-hidden 算法对比。

(1) 寻找 *K*-hidden 算法的最优参数组合

Zhao 等[31]指出, 当 *K*-hidden 算法的 *K* 个概率参数满足式(22)时, 可以生成很难被局部搜索策略求解的负数据库。

$$f=\sum_{i=1}^{K}(K-2i)p_i>0 \tag{22}$$

其中, p_i 表示类型为 i 的记录在负数据库中的占比。对于 CDCL 求解器而言, 负数据库的 r 值(记录数与串长之比)与其难解性的关联十分密切。但之前的工作[31]仅分析了固定概率参数时, 变化 r 值的情况, 并未给出 *K*-hidden 负数据库最难解的参数区间, 故本文设计了这个实验。

具体地, 实验中 *K*-hidden 算法的参数 r 设置为 4~6, 步长为 0.05, 参数 f 设置为 0~1, 步长为 0.05, 每个 f 值对应一组 *K* 个概率参数, 对应关系见表 5。其他参数设置为 $n=500$, $K=3$。每组实验重复执行 10 次, 实验结果见图 11。

如图 11 所示, 对于 WalkSAT 而言, 随着 f 值的增加, *K*-hidden 负数据库的难解区域逐渐变小。整体来看, *K*-hidden 算法的 f 值设置为比 0 略大时最优, 而对于不同的 f 值, 都存在一个不同的阈值, 当 r 大于该阈值时, WalkSAT 将很难在可接受的翻转次数内逆转 *K*-hidden 负数据库, 且该阈值随着 f 的增加而增加。对于 Kissat 而言, 仅当 f 小于 0.2 且 r 位于 4.5~5 之间时, *K*-hidden 负数据库对于 Kissat 是难解的, 其他情况下都属于易解。当 f 小于 0.2 时, 随着 r 值的增加, *K*-hidden 负数据库呈现出“易-难-易”的模式, 难解的区间在 r 为 4.6 附近。而最难解的负数据库由 f 为 0.1, r 为 4.5 的 *K*-hidden 算法生成。

(2) 难解性对比

D-hidden 算法的参数 *round* 设置为 2~16, 步长为 1, 隐藏串的长度 n 设置为与每一轮基准串的长度一致。相应地, *K*-hidden 算法的参数 n 也随之变化。对于 *K*-hidden 算法的其他参数, 参考前文的实验结果, 选择了一组最优的参数组合: $p_1=0.663$, $p_2=0.225$, $p_3=0.112$, $r=4.5$ 和 $K=3$。每组实验重复执行 30 次, 图 12 和图 13 给出了实验结果。

图 12 表明, 对于 WalkSAT 而言, *K*-hidden 负数据库仅在 *round* 为 2 时比 *D*-hidden 负数据库的难解性更高。当 *round* 不小于 6 时, 两种算法生成的负数据库的难解性相当, 并持续保持较高的难解性, 且 WalkSAT 在重复实验中无法在指定翻转次数内成功求解任意负数据库。当 *round* 位于 3~5 之间时, *D*-hidden 算法相对表现出更高的安全性和稳定性, 在 30 次重复实验中, 虽然 WalkSAT 求解 *K*-hidden 负数据库的翻转次数的中位数和平均数已经接近 10^8 次上限, 但仍有部分负数据库(约 3~10 个)能被 WalkSAT 成功求解。值得注意的是, 当 *round* 不小于 3 时, *D*-hidden 负数据库的难解性已经达到实验中对 WalkSAT 的设置上限, 且重复实验中没有出现部分负数据库被成功求解的情况。

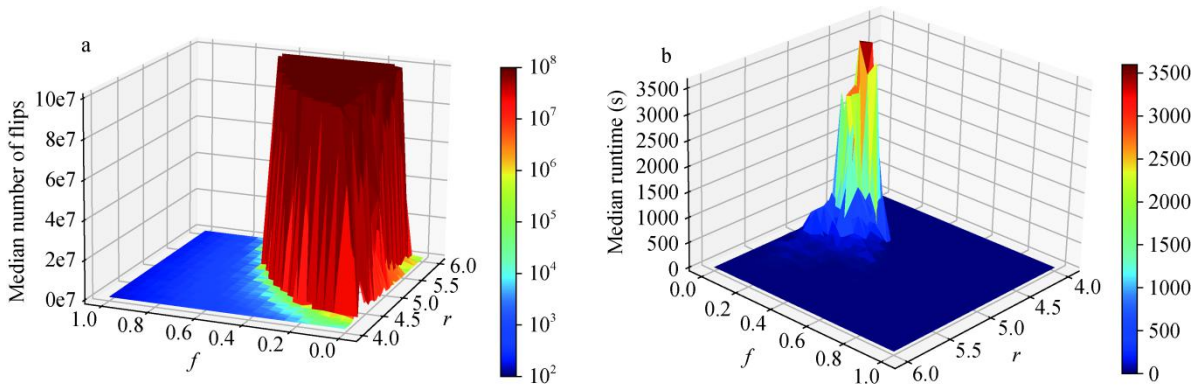

图 11 WalkSAT(a)和 Kissat(b)求解不同参数组合下的 *K*-hidden 负数据库的情况

Figure 11 WalkSAT (a) and Kissat (b) solve for the *K*-hidden negative database with different parameter combinations

表 5 *K*-hidden 算法的参数设置

Table 5 Parameters setting of *K*-hidden algorithm

p_1	p_2	p_3	f	p_1	p_2	p_3	f
0.625	0.250	0.125	0.00	0.831	0.113	0.056	0.55
0.644	0.238	0.118	0.05	0.850	0.100	0.050	0.60
0.663	0.225	0.112	0.10	0.869	0.088	0.043	0.65
0.681	0.213	0.106	0.15	0.888	0.075	0.037	0.70
0.700	0.200	0.100	0.20	0.906	0.063	0.031	0.75
0.719	0.188	0.093	0.25	0.925	0.050	0.025	0.80
0.737	0.175	0.088	0.30	0.944	0.037	0.019	0.85
0.756	0.162	0.082	0.35	0.963	0.025	0.012	0.90
0.775	0.150	0.075	0.40	0.981	0.012	0.007	0.95
0.794	0.137	0.069	0.45	1.000	0.000	0.000	1.00
0.813	0.125	0.062	0.50				

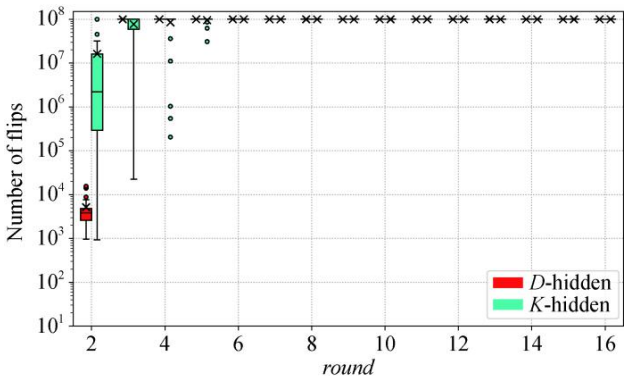

图 12 WalkSAT 求解 *D*-hidden 负数据库和 *K*-hidden 负数据库的结果对比

Figure 12 Comparison of the results of WalkSAT solving *D*-hidden negative database and *K*-hidden negative database

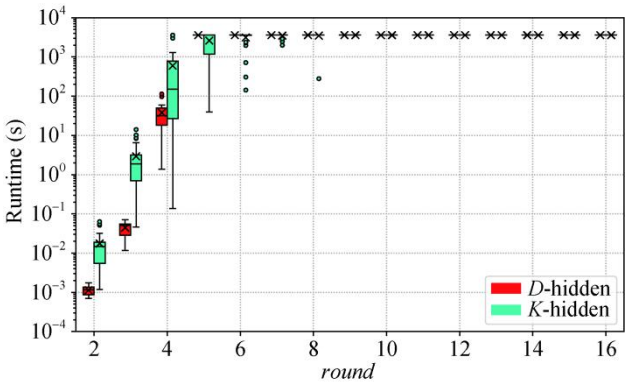

图 13 Kissat 求解 *D*-hidden 负数据库和 *K*-hidden 负数据库的结果对比

Figure 13 Comparison of Kissat's results for solving *D*-hidden negative database and *K*-hidden negative database

如图 13 所示, *D*-hidden 负数据库对于 Kissat 的难解性同样随着 *round* 的增加而增加, 但相比对于 Walksat 的难解性, 增加的过程持续时间更长, 直到 *round* 不小于 5 时, *D*-hidden 负数据库对于 Kissat 才是难解的, 而当 *round* 小于 5 时, *K*-hidden 算法的表现都要比 *D*-hidden 算法更优。当 *round* 位于 6~8 之间时, 重复实验中 Kissat 可以成功求解 *K*-hidden 负数据库约 1~8 次, 但其依然无法求解任意的 *D*-hidden 负数据库。随着轮数的进一步增加, 两种算法生成的负数据库都保持在最高的难解性, 此时 Kissat 无法在 1 h 内成功求解。整体来看, 当 *round* 偏小时(例如小于 5), *K*-hidden 算法的效果更好, 当 *round* 属于中等大小时(例如 5~8), *D*-hidden 算法的效果更优, 其他情况下二者生成的负数据库的难解性相当。

(3) 效率对比

两种算法的参数设置与前文的难解性对比实验中的参数一致。实验重复执行 30 次, 分别记录两种算法耗时的平均值, 单位为秒, 实验结果见表 6。

如表 6 所示, *D*-hidden 算法生成基准组的耗时要远大于生成目标负数据库耗时和 *K*-hidden 算法耗时。这是因为 *D*-hidden 算法在生成基准组时, 涉及对 DES 算法的 SAT 编码, 重复执行单元传播, 以及多次读写文件操作, 而 IO 操作的耗时较长, 故生成基准组的耗时要高出一个量级。且随着 *round* 的增加, 该部分的耗时呈正相关。但需要说明的是, 在实际中生成基准组的操作可以离线进行, 即提前生成多组不同 *round* 下的基准组。因为基准组的生成过程往往

是重复的, 且相同 *round* 下不同基准组间的区别较小。因此, *D*-hidden 算法的实际运行时间可以仅参考生成目标负数据库的耗时。

表 6　*D*-hidden 算法与 *K*-hidden 算法的效率对比

Table 6　Efficiency comparison of *D*-hidden algorithm and *K*-hidden algorithm

round	生成基准组耗时	生成目标负数据库耗时	*K*-hidden 算法耗时
2	0.357	**0.010**	0.036
3	0.471	**0.008**	0.057
4	0.598	**0.014**	0.031
5	0.515	**0.011**	0.047
6	0.682	**0.013**	0.055
7	0.773	**0.029**	0.050
8	0.873	**0.010**	0.070
9	1.204	**0.027**	0.098
10	1.067	**0.034**	0.088
11	1.964	**0.029**	0.115
12	1.936	**0.039**	0.133
13	2.223	**0.062**	0.120
14	2.702	**0.050**	0.377
15	3.044	**0.082**	0.346
16	4.291	**0.131**	0.417

同时, 无论 *round* 如何变化(对应的隐藏串长也在变化), *D*-hidden 算法生成目标负数据库的耗时都要低于 *K*-hidden 算法耗时。且随着 *round* 的增加, 二者的耗时呈大致增加的趋势。*D*-hidden 算法在生成目标负数据库时, 仅涉及两次异或操作, 虽然相比于 *K*-hidden 算法需要的迭代次数更多, 但其操作相对简单。而 *K*-hidden 算法在每次迭代时的执行的操作更多, 包括随机挑选变量、随机生成指定类型且不匹配隐藏串的记录等, 故耗时相对较多。

6.2　整数分解生成 SAT

6.2.1　SAT 实例变量与子句数量对比实验

实验使用长度 *n* 为 32、64、128、256 的质数进行实例生成分析, 结果中使用的符号含义如表 7 所示。使用不同长度的质数, 生成的实例在不同方法中变量数目和子句数目如图 14 和图 15 所示。

以变量数目作为比较标准时, 从图 14 中可以看到, FactToSAT 方法与其他 6 种方法相比, 只有常规加法器与 Recursive 乘法电路的组合产生的 SAT 实例变量数是明显更少的, 且需要在质数长度 *n* 在 3 位数以上才具有明显优势。在其他 5 种方法生成的 SAT 实例中, FactToSAT 生成的 SAT 实例变量明显是最少的。因此可以推断, 在 SAT 实例大小相似的情况下, 除了常规加法器与 Recursive 乘法电路的组合, 与其他 5 种方法相比, FactToSAT 对应的整数分解的难解性最高。

表 7　符号含义

Table 7　Symbol meaning

符号	含义	符号	含义
n	质数长度	②	对数时间加法器
Var	变量数目	*A*	进位保存乘法电路
Cla	子句数目	*B*	Wallace-tree 电路
X	FactToSAT	*C*	Recursive 电路
①	普通加法器		

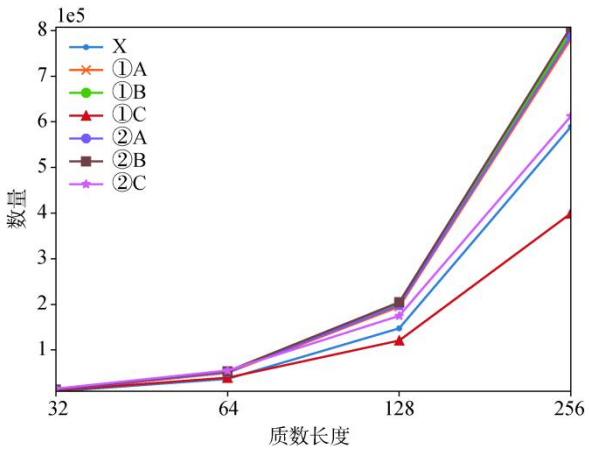

图 14　SAT 变量数目对比

Figure 14　Comparison of the number of SAT variables

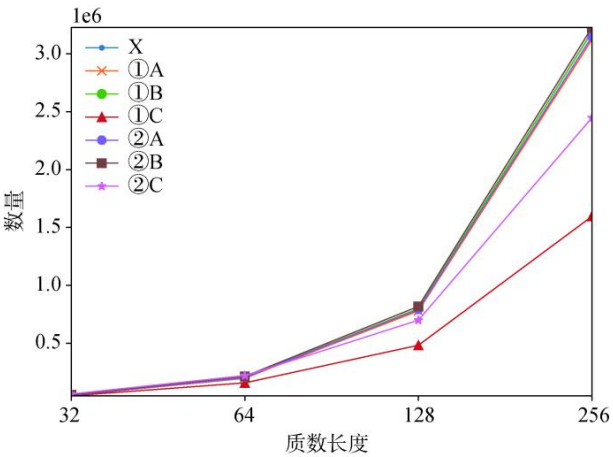

图 15　SAT 子句数目对比

Figure 15　Number of clauses comparison

从图 15 可以知道, 以子句数目为标准进行比较时, FactToSAT 方法生成的 SAT 实例的子句数目, 与其他方法相比都是基本一样的。并且可以看出, 乘法电路使用 Karatsuba 算法时, 子句会明显减少。本文考虑是因为 Karatsuba 算法通过分治优化减少了乘法运算的步骤, 从而使得可以在使用更少的逻辑约束的情况下表示乘法过程。但 FactToSAT 相比其他 4 种方法的子句数目不相上下。

将各自生成的 SAT 实例中的变量数目与子句数目进行比较, FactToSAT 相比除使用了 Recursive 乘法电路外的其他方法, 子句数目数量基本一致, 但变量数目更少更优, 故可以认为本文提出的FactToSAT 方法具有一定优势。并且可以推断, 在 SAT 实例大小相当时, 相比大部分方法, FactToSAT 实例对应的整数分解问题 $N=PQ$ 更难求解。

综上, 可以认为 FactToSAT 方法生成的实例的效果是令人满意的, 且在一定情况下不弱于当前已有的整数分解转换为 SAT 实例的大部分主流方法。而对比使用了 Recursive 乘法电路的方法, FactToSAT 则具有易于理解、更容易设计实现的优点。

6.2.2 难解性对比实验

将被加密字符串的长度作为横轴坐标, 各个负数据库生成算法生成的 *NDB* 的求解时间作为纵轴。

根据图 16 可以知道使用 *F*-hidden 算法在串长到达 50 以后就可以到达难解, 但 *q*-hidden 和 *p*-hidden 算法需要在被保护的串长到达 600 以后, 才能到达难解; *K*-hidden 算法需要在串长到达 800 以后, 才到达难解; 而 *QK*-hidden 算法要求更高, 需要在串长到达接近 2000 才能够到达难解。

因此可以认为在串长较小时, 加密算法 *q*-hidden 算法、*p*-hidden 算法、*K*-hidden 算法和 *QK*-hidden 算法, 无法提供较好的隐私保护, 其难解性严重依赖于串长的大小, 需要串长达到比较高时算法的安全性才体现出来。

而 *F*-hidden 算法的难解性则主要依赖于对应整数分解的难解性, 在串长为 58 时, 即使用两个长度分别为 29 的质数, 此时整数分解已具有一定难解性, 再将其用于负数据库的加密, 所以可以在字符串较短时得到很好的难解性。

从图 16 中可以看出, *F*-hidden 算法使用部分质数进行负数据库生成时产生易解的 *NDB*, 考虑是选取的质数的二进制结构与使用的 SAT 求解器结构存在巧合, 因为 SAT 求解器在解空间的搜索中使用了各种加速和优化, 所以考虑部分质数的结构会被使用的 Kissat 求解器的某种搜索方式命中, 从而变得易解。因此使用 *F*-hidden 算法用于加密时, 需要首先使用多个求解器筛选出较好的质数, 从而达到加密效果。

综上, *F*-hidden 算法对串长较小的加密效果非常显著, 但对使用质数的长度有一定的要求。

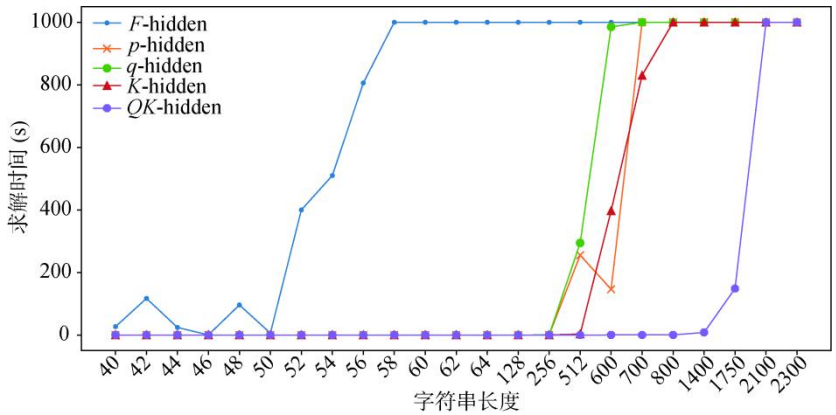

图 16 算法效果比较

Fig. 16 Comparison of algorithms effects

7 总结

本文将传统加解密方法与负数据库进行结合, 提出了新的负数据库生成算法, 也是一种 SAT 实例的生成方法。两种负数据库生成算法如下:

(1) 基于 DES 算法的 *D*-hidden 算法: 首先将

DES 算法编码为 SAT 实例(称为原始 *NDB*), 接着生成一组随机明文和密钥并加密得到密文, 然后分批将其插入原始 *NDB*, 并进一步利用单元传播策略得到包括一个基准串和一个等价于密钥搜索问题的负数据库的基准组, 最后利用负数据库的半同态异或性质生成隐藏串的负数据库。我们通过实验结果展示了 *K*-hidden 负数据库在不同参数组合下的难解区域分布。在此基础上, 本章将 *D*-hidden 算法与最优参数设置下的 *K*-hidden 算法进行了对比, 实验结果表明当隐藏串的长度与基准串的长度一致且轮数足够时(例如不小于 5), *D*-hidden 算法可以比 *K*-hidden 算法生成更难解且更稳定的负数据库, 生成效率也更高。

(2) 基于整数分解的 *F*-hidden 算法: 选取合适的两个质数, 再将整数分解的逻辑过程使用 SAT 实例进行表示, 根据选取的质数的值对 SAT 实例使用单元传播策略可以求得负数据库的基准组, 最后利用负数据库的半同态异或性质可以对隐藏串进行保护。本文在实验中验证文本长度在[50, 600]时, *F*-hidden 算法相比目前已有的负数据库生成算法具有明显优势。在文本长度达到 50 时, *F*-hidden 通过挑选合适的质数, 可以使得生成的负数据库达到难解, 而已有的其他算法需在长度到达 600 后才具有难解性。

本工作是首个将负数据库与传统加解密算法进行结合的工作, 在隐私保护方向提供了一个可供参考的新思路, 同时也为 SAT 领域提供新的研究视角。此外, 由于 *D*-hidden 方法的安全性受到 DES 算法的制约, 在未来工作中我们会尝试使用更加安全的加密算法(包括最新的分组密码、非对称密码、同态加密算法等)生成负数据库, 探究是否能生成更加难解且特性丰富的负数据库。

参考文献

[1] Esponda F, Ackley E S, Helman P, et al. Protecting data privacy through hard-to-reverse negative databases[J]. *International Journal of Information Security*, 2007, 6(6): 403-415.

[2] Esponda F, Forrest S, Helman P. Enhancing privacy through negative representations of data[R]. University of New Mexico, 2004.

[3] Wang X B, Tang C M, Li J H. Study on algorithms of big integer factorization in public-key cryptosystem[J]. *Modern Information Technology*, 2020, 4(16): 125-133.
(王兴波, 唐春明, 李建辉. 公钥密码体制中大整数分解算法研究[J]. *现代信息科技*, 2020, 4(16): 125-133.)

[4] Lenstra A K, Lenstra H W Jr, Manasse M S, et al. The Number Field Sieve[C]. *The Twenty-Second Annual ACM Symposium on Theory of Computing - STOC '90*, 1990: 564-572.

[5] Dong Q, Wu N. Recent progress of integer factorization algorithms and challenges faced by the traditional cryptology[J]. *Computer Science*, 2008, 35(8): 17-20.
(董青, 吴楠. 整数质因子分解算法新进展与传统密码学面临的挑战[J]. *计算机科学*, 2008, 35(8): 17-20.)

[6] Yang L T, Xu L, Lin M. Integer Factorization by a Parallel GNFS Algorithm for Public Key Cryptosystems[C]. *Embedded Software and Systems*, 2005: 683-695.

[7] Daoud S, Gad I. A parallel line sieve for the GNFS algorithm[J]. *International Journal of Advanced Computer Science and Applications*, 2014, 5(7): 178-185.

[8] Pollard J M. A monte carlo method for factorization[J]. *BIT Numerical Mathematics*, 1975, 15(3): 331-334.

[9] Pollard J M. Theorems on factorization and primality testing[J]. *Mathematical Proceedings of the Cambridge Philosophical Society*, 1974, 76(3): 521-528.

[10] Lenstra H W. Factoring integers with elliptic curves[J]. *The Annals of Mathematics*, 1987, 126(3): 649.

[11] Carl P. The Quadratic Sieve Factoring Algorithm[C]. *Advances in Cryptology*, 1985: 169-182.

[12] Montgomery P L. A Block Lanczos Algorithm for Finding Dependencies over GF(2)[C]. *Advances in Cryptology — EUROCRYPT '95*, 1995: 106-120.

[13] Fu X Q, Bao W S, Zhou C, et al. Quantum algorithm for prime factorization with high probability[J]. *Acta Electronica Sinica*, 2011, 39(1): 35-39.
(付向群, 鲍皖苏, 周淳, 等. 具有高概率的整数分解量子算法[J]. *电子学报*, 2011, 39(1): 35-39.)

[14] Patsakis C. RSA private key reconstruction from random bits using SAT solvers[C]. *IACR Cryptol. ePrint Arch.*, 2013, 2013: 26.

[15] Zhang W B, Feng M, Li Q, et al. Research on RSA encryption algorithm based on python and its several cracking methods[J]. *China CIO News*, 2020(12): 132-134.
(张文博, 冯梅, 李青, 等. 基于 Python 的 RSA 加密算法及其几种破解方法的研究[J]. *信息系统工程*, 2020(12): 132-134.)

[16] Zheng Q Y, Zhao N D. Research and implementation of RSA asymmetric encryption algorithm for quadruple Prime numbers[J]. *Network Security Technology & Application*, 2022(5): 38-40.
(郑钦元, 赵乃东. 四重素数 RSA 非对称加密算法的研究与实现[J]. *网络安全技术与应用*, 2022(5): 38-40.)

[17] Yang J S. Privacy protection scheme of medical record operation records based on blockchain[J]. *Network Security Technology & Application*, 2022(4): 29-31.
(杨径山. 基于区块链的病历操作记录的隐私保护方案[J]. *网络安全技术与应用*, 2022(4): 29-31.)

[18] Purdom P, Sabry A. CNF generator for factoring problems[EB/OL]. http://www.cs.indiana.edu/cgi-pub/sabry/cnf.html. Accessed 20 May 2022.

[19] Alfred V. Aho, Jeffrey D. Ullman Foundations of Computer Science with C New York[M]. W. H. Freeman/Computer Science Press, 1995: 716-722.

[20] Wallace C S. A suggestion for a fast multiplier[J]. *IEEE Transactions on Electronic Computers*, 1964, EC-13(1): 14-17.

[21] Karatsuba A, Ofman Y. Multiplication of many-digital numbers by

automatic computers[J]. *Doklady Akademii Nauk SSSR*, 1962, 145(2): 293-294.

[22] Karatsuba A A. The complexity of computations[J]. *Proceedings of the Steklov Institute of Mathematics*, 2007, 211(2): 169-183.

[23] Yuen H, Bebel J. Toughsat[EB/OL]. https://github.com/joebebel/toughsat. Accessed 20 May 2022.

[24] Bebel J, Yuen H. Hard SAT instances based on factoring[C]. *Proceedings of SAT Competition 2013: Solver and Benchmark Descriptions*, 2013: 102.

[25] Bebel J. Harder SAT Instances from Factoring with Karatsuba and Espresso[C]. *Proceedings of SAT Race 2019: Solver and Benchmark Descriptions*, 2019: 47-48.

[26] Schönhage A, Strassen V. Schnelle multiplikation großer zahlen[J]. *computing*, 1971, 7(3): 281-292.

[27] Horie S, Watanabe O. Hard Instance Generation for SAT: Extended Abstract[M]. Algorithms and Computation, 1997: 22-31.

[28] Jia H X, Moore C, Strain D. Generating hard satisfiable formulas by hiding solutions deceptively[J]. *Journal of Artificial Intelligence Research*, 2007, 28(1): 107-118.

[29] Liu R, Luo W, Yue L. The p-hidden algorithm: Hiding single databases more deeply[J]. *Immune Computation*, 2014, 2(1): 43-55.

[30] Barthel W, Hartmann A K, Leone M, et al. Hiding solutions in random satisfiability problems: A statistical mechanics approach[J]. *Physical Review Letters*, 2002, 88(18): 188701.

[31] Zhao D D, Luo W J, Liu R, et al. A Fine-Grained Algorithm for Generating Hard-Toreverse Negative Databases[C]. *2015 International Workshop on Artificial Immune Systems*, 2016: 1-8.

[32] Zhao D D, Hu X Y, Xiong S W, et al. A Fine-Grained Privacy-Preserving K-Means Clustering Algorithm Upon Negative Databases[C]. *2019 IEEE Symposium Series on Computational Intelligence*, 2020: 1945-1951.

[33] Zhao D D, Hu X Y, Xiong S W, et al. *k*-means clustering and kNN classification based on negative databases[J]. *Applied Soft Computing*, 2021, 110: 107732.

[34] Zhao D D, Zhang P C, Xiang J W, et al. NegDL: Privacy-Preserving Deep Learning Based on Negative Database[C]. *2022 4th International Conference on Data Intelligence and Security*, 2022: 118-126.

[35] Zhao D D, Luo W J. Real-Valued Negative Databases[C]. *European Conference on Artificial Life*, 2013

[36] Ouyang M. How good are branching rules in DPLL?[J]. *Discrete Applied Mathematics*, 1998, 89(1/2/3): 281-286.

[37] Harrison J. Handbook of Practical Logic and Automated Reasoning[M]. Cambridge: Cambridge University Press, 2009: 79-90.

[38] Biere A, Heule M, van Maaren H, et al. Conflict-driven clause learning sat solvers[J]. *Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications*, 2009, 131-153.

[39] Massacci F, Marraro L. Logical cryptanalysis as a SAT problem[J]. *Journal of Automated Reasoning*, 2000, 24(1): 165-203.

[40] Selman B, Kautz H, Cohen B. Local Search Strategies for Satisfiabilitytesting[M]. Cliques, Coloring, and Satisfiability. Providence, RhodeIsland: American MathematicalSociety, 1996: 521-531.

[41] Biere A, Fazekas K, Fleury M, et al. Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020[C]. *Proceedings of SAT Competition 2020: Solver and Benchmark Descriptions*, 2020: 50-53.

赵冬冬 于 2016 年在中国科学技术大学计算机应用技术专业获得博士学位。现任武汉理工大学计算机与人工智能学院副教授、CCF 会员。研究领域为隐私保护、虹膜识别、数据挖掘。Email: zdd@whut.edu.cn

刘志晖 于 2025 年在武汉理工大学软件工程专业获得工学硕士学位。现在武汉大学软件工程专业攻读博士学位。研究领域为软件工程。研究兴趣包括布尔可满足性问题。Email: zhihuiliu@whu.edu.cn

廖磊 于 2023 年在武汉理工大学软件工程专业获得硕士学位。研究领域为隐私保护。研究兴趣包括布尔可满足性问题。Email: liaolei666@ whut.edu.cn

向剑文 于 2005 年在日本北陆先端科学技术大学院大学情报(计算机)科学专业获得博士学位。现任武汉理工大学计算机与人工智能学院教授、湖北省创新群体负责人、CCF 会员。研究领域为可信计算、软件工程、网络安全、机器学习。Email: jwxiang@ whut.edu.cn

江浩 于 2019 年中国科学技术大学计算机系统结构专业获博士学位。现任安徽大学人工智能学院副教授。研究领域为隐私保护、智能优化。研究兴趣包括敏感信息收集、敏感信息隐藏、进化计算理论及应用。Email: haojiang@ahu.edu.cn