中国科学 : 信息科学　2023 年　第 53 卷　第 7 期: 1348–1367
SCIENTIA SINICA Informationis
论文

# 用于大状态分组密码的深度学习辅助密钥恢复框架

陈怡[1], 包珍珍[2], 申焱天[1], 于红波[1*]

1. 清华大学计算机科学与技术系, 北京 100084
2. 清华大学网络科学与网络空间研究院, 北京 100084
* 通信作者. E-mail: yuhongbo@mail.tsinghua.edu.cn

收稿日期: 2022–07–26; 修回日期: 2022–09–11; 接受日期: 2022–09–27; 网络出版日期: 2023–07–11

国家重点研发计划 (批准号: 2017YFA0303903) 资助项目

**摘要** 深度学习辅助密钥恢复攻击是 2019 年 International Cryptology Conference (CRYPTO) 上提出的一项全新密码分析技术. 针对该技术至今无法应用于大状态分组密码的缺陷, 本文提出了一种深度学习辅助的多阶段密钥恢复框架. 该框架的核心是找到一个神经区分器组合, 分阶段进行密钥恢复攻击. 本文首先针对 Speck 的大状态成员分别训练了一组神经区分器, 通过在该框架下利用区分器组合, 设计并执行了实际密钥恢复攻击, 证实了该框架的有效性. 然后, 提出了一种在低概率差分中寻找中性比特的方法, 来把实际攻击扩展成覆盖更长轮数的理论攻击. 最终, 针对缩减轮 Speck 的最大状态成员取得了更好的密钥恢复攻击. 这项工作为使用深度学习对更多分组密码进行密码分析铺平了道路. 本文的验证代码已开源至 https://github.com/AI-Lab-Y/NAAF.

**关键词** 大状态分组密码, 深度学习, 密钥恢复攻击, 差分分析, Speck

## 1 前言

分组密码是当今应用最为广泛的一类对称密钥原语. 我们对分组密码安全性的信心源于分析分组密码对所有已知密码分析技术的抵抗力. 因此, 发展全新的密码分析技术有助于更好地理解密码算法的安全性, 这也是当前密码学的重点研究方向之一. 除了各种经典的密码分析技术 (如: 差分分析 [1] 和线性分析 [2]), 德国学者 Gohr [3] 在 2019 年 International Cryptology Conference (CRYPTO) 上提出了一种全新的技术, 即基于深度学习的密码分析技术, 并通过针对缩减轮 Speck32/64 的攻击展示了它的优势.

Gohr [3] 提出了一个基于神经网络的区分器 (也称作神经区分器 $\mathcal{ND}$), 用于区分随机密文对, 和用缩减轮 Speck32/64 生成且对应明文对满足特定明文差分的真实密文对, 神经区分器的输入是完整的

**引用格式:** 陈怡, 包珍珍, 申焱天, 等. 用于大状态分组密码的深度学习辅助密钥恢复框架. 中国科学: 信息科学, 2023, 53: 1348–1367, doi: 10.1360/SSI-2022-0298
Chen Y, Bao Z Z, Shen Y T, et al. A deep learning-aided key recovery framework for large-state block ciphers (in Chinese). Sci Sin Inform, 2023, 53: 1348–1367, doi: 10.1360/SSI-2022-0298

密文对. 基于明文差分约束 (0x40, 0x0), 文献 [3] 中给出了 5∼8 轮的神经区分器. 与经典的多差分区分器对比, Gohr 提出的神经区分器在区分准确率上均取得了不可忽略甚至显著的优势.

基于神经区分器, Gohr 首先提出了一种基础版本的密钥恢复攻击, 来恢复 $r$ 轮 Speck32/64 的轮密钥. 以恢复第 $r$ 轮轮密钥为例, 攻击的核心思想是构建一个由前置差分和神经区分器组成的 $r-1$ 轮混合区分器, 然后猜测第 $r$ 轮轮密钥, 对收集的密文对解密一轮, 并基于神经区分器对解密后的伪密文对的打分过滤错误密钥. 为了优化基础版本的密钥恢复攻击, Gohr 设计了一种基于贝叶斯优化的高级密钥猜测策略, 结合强化学习机制动态分配计算资源, 提出了一种加速版本的密钥恢复攻击. 在 11 轮 Speck32/64 上的实验表明, Gohr 提出的深度学习辅助密钥恢复攻击的计算复杂度是 $2^{38}$, 远低于此前的最优攻击 [4] 的计算复杂度 $2^{46}$.

Gohr 的工作不仅展示了深度学习应用于密码分析的巨大潜力, 也遗留了一系列的开放问题, 引起了学者们的广泛关注. 具体来说, 主要有以下几个核心问题: (1) 神经区分器学习到了哪些知识? (2) 如何增强 Gohr 提出的密钥恢复攻击? (3) Gohr 的密钥恢复攻击完全依赖于实际实验, 如何突破该瓶颈使得深度学习辅助密钥恢复攻击可以进行理论分析? (4) 如何在 Speck 家族的大状态成员、或者其他大状态分组密码上应用深度学习辅助密钥恢复攻击? 密码学者们已经在前 3 个问题上取得了一定的进展 [5∼7], 但是在第 4 个问题上却至今毫无进展, 这极大地限制了深度学习在密码分析领域的应用.

Gohr 提出的密钥恢复攻击 (包括基础版本攻击、加速版本攻击) 之所以不能直接应用于大状态分组密码算法, 有两个核心原因. 第一, Gohr 提出的神经区分器以完整的密文对作为输入, 导致猜测密钥时需要一次猜测轮密钥的所有比特. 基础版本攻击在密钥猜测阶段需要遍历整个轮密钥空间, 而加速版本攻击需要预先针对整个轮密钥空间进行预计算, 并且预计算的复杂度约为轮密钥空间的 $2^{10}$ 倍. 因此, 大状态分组密码的轮密钥长度过大, 导致基础版本攻击、加速攻击均无法在可接受的时间内运行完成. 第二, Gohr 的神经区分器虽然以完整状态对作输入, 但却只利用了部分输入比特的知识 [6]. 并且, 针对 Speck 的大状态成员构建神经区分器时, 这一现象更为明显, 甚至只有少数比特对神经区分器有影响, 进而使得与非信息比特 (即对神经区分器影响很小、甚至没影响) 相关的密钥比特很难被正确地恢复 [6]. 上述原因使得深度学习辅助密钥恢复攻击能否用于大状态分组密码的问题持续多年都悬而未决.

本文将探索该开放问题, 旨在提供一种将深度学习辅助密钥恢复攻击应用于大状态分组密码的框架, 并主要在 Speck 大状态成员上对该框架进行验证. 本文的主要贡献如下.

• 本文提出了一个深度学习辅助的多阶段密钥恢复框架. 该框架利用了多个神经区分器, 其中每个区分器以密文对的部分比特为输入, 并且不同区分器的输入依赖于不同的轮密钥比特子集. 这些轮密钥比特子集的并集包含了轮密钥的大部分甚至所有比特. 在本文提出的框架下, 分阶段应用 Gohr 提出的密钥恢复攻击恢复每个轮密钥比特子集.

• 本文提出了一种为上述多阶段密钥恢复框架寻找一组有效的神经区分器的方法. 为了利用上述框架恢复完整的轮密钥, 每一个神经区分器应该有较好的准确率且不依赖于完整的密文对. 因此, 应该有选择地基于某些密文比特训练单个神经区分器, 并仔细地组织它们. 为了验证该方法的有效性, 本文首先为 Speck 各个成员分别寻找了一组有效的神经区分器. 为了验证该方法的通用性, 本文也为其他三种分组密码 (包括 SIMON [8], PRESENT [9] 和 DES [1]) 分别寻找了一组有效的神经区分器.

• 本文提出了针对 Speck 大状态成员 (Speck64, Speck96, Speck128) 的实际密钥恢复攻击, 验证了本文提出的多阶段密钥恢复框架的有效性. 并且, 为了扩展这些攻击的轮数, 本文提出了一种适用于低概率差分的中性比特检测技术. 据我们所知, 本文是第一个提出了针对 Speck 大状态成员的基于深

**表 1　对 Speck 大状态成员的密钥恢复攻击的总结**

**Table 1**　Summary of key-recovery attacks on large-state Speck

| Target | #R | Time (#Enc) | Data (#CP) | Success rate | Configure | Ref. |
|---|---|---|---|---|---|---|
| Speck128/128 | **17/32** | $2^{113}$ | $2^{113}$ | – | $1+14r_{\mathcal{CD}}+2$ | [4] |
| | | $\mathbf{2^{78.98}}$ | $\mathbf{2^{61.28}}$ | **0.52** | $1+6r_{\mathcal{CD}}+9r_{\mathcal{ND}}+1$ | Subsection 6.2 |
| | 23/32 | $2^{125.35}$ | $2^{125.35}$ | – | $1+20r_{\mathcal{CD}}+2$ | [10] |
| Speck128/192 | **18/33** | $2^{177}$ | $2^{113}$ | – | $1+14r_{\mathcal{CD}}+3$ | [4] |
| | | $\mathbf{2^{142.98}}$ | $\mathbf{2^{61.28}}$ | **0.52** | $1+6r_{\mathcal{CD}}+9r_{\mathcal{ND}}+2$ | Subsection 6.2 |
| | 24/33 | $2^{189.35}$ | $2^{125.35}$ | – | $1+20r_{\mathcal{CD}}+3$ | [10] |
| Speck128/256 | **19/34** | $2^{241}$ | $2^{113}$ | – | $1+14r_{\mathcal{CD}}+4$ | [4] |
| | | $\mathbf{2^{206.98}}$ | $\mathbf{2^{61.28}}$ | **0.52** | $1+6r_{\mathcal{CD}}+9r_{\mathcal{ND}}+3$ | Subsection 6.2 |
| | 25/34 | $2^{253.35}$ | $2^{125.35}$ | – | $1+20r_{\mathcal{CD}}+4$ | [10] |
| Speck96/96 | 13/28 | $\mathbf{2^{52.61}}$ | $\mathbf{2^{36.60}}$ | **0.81** | $1+4r_{\mathcal{CD}}+7r_{\mathcal{ND}}+1$ | Subsection 6.2 |
| | 18/28 | $2^{82}$ | $2^{82}$ | – | $1+15r_{\mathcal{CD}}+2$ | [10] |
| Speck96/144 | 14/29 | $\mathbf{2^{100.61}}$ | $\mathbf{2^{36.60}}$ | **0.81** | $1+4r_{\mathcal{CD}}+7r_{\mathcal{ND}}+2$ | Subsection 6.2 |
| | 19/29 | $2^{130}$ | $2^{82}$ | – | $1+15r_{\mathcal{CD}}+3$ | [10] |
| Speck64/96 | 11/26 | $\mathbf{2^{41.13}}$ | $\mathbf{2^{26.47}}$ | **0.90** | $1+3r_{\mathcal{CD}}+6r_{\mathcal{ND}}+1$ | Subsection 6.2 |
| | 12/26 | $\mathbf{2^{73.13}}$ | $\mathbf{2^{26.47}}$ | **0.90** | $1+3r_{\mathcal{CD}}+6r_{\mathcal{ND}}+2$ | Subsection 6.2 |
| | 19/26 | $2^{93.56}$ | $2^{61.56}$ | – | $1+15r_{\mathcal{CD}}+3$ | [10] |
| Speck64/128 | 13/29 | $\mathbf{2^{105.13}}$ | $\mathbf{2^{26.47}}$ | **0.90** | $1+3r_{\mathcal{CD}}+6r_{\mathcal{ND}}+3$ | Subsection 6.2 |
| | 20/29 | $2^{125.56}$ | $2^{61.56}$ | – | $1+15r_{\mathcal{CD}}+4$ | [10] |

度学习的密钥恢复攻击的工作.

• 本文在缩减轮 Speck128 上取得了时间复杂度和数据复杂度均更好的密钥恢复攻击. 并提出了针对缩减轮 Speck96, Speck64 的高效密钥恢复攻击. 表 1 [4,10] 总结了本文提出的攻击和已有攻击的对比.

## 2　基础知识与符号

### 2.1　Speck 算法简介

Speck 是一组轻量级分组密码算法, 由美国国家安全局设计. 现在, Speck 系列算法已经由 ISO 标准化 (ISO/29617-22), 作为无线射频识别 (radio frequency identification, RFID) 技术的标准算法之一被广泛使用. Speck 一共包含 10 个成员, 每一个成员用 Speck2$n$/$mn$ 表示, 其中 $2n$ 表示分组长度, $mn$ 表示主密钥长度. 在本文中, Speck2$n$ 指代所有分组长度为 $2n$ 的 Speck 成员.

通过 $r$ 轮加密, Speck2$n$ 把一个包含两个 $n$ 比特字的明文 $(x_0, y_0)$ 加密成一个密文 $(x_r, y_r)$. 每一轮加密的轮函数如下:

$$\begin{aligned} x_{i+1} &= ((x_i^{\gg\alpha}) \boxplus y_i) \oplus \mathrm{rk}_{i+1}, \\ y_{i+1} &= (y_i^{\ll\beta}) \oplus x_{i+1}, \end{aligned}$$

其中 $\mathrm{rk}_{i+1}, i \in [0, r-1]$ 是第 $i+1$ 轮的轮密钥, $\alpha$ 和 $\beta$ 是两个常量. 对于 Speck32/64, $\alpha = 7$, $\beta = 2$. 对于其他 Speck 成员, $\alpha = 8$, $\beta = 3$. 有关 Speck 的更多细节可以参考文献 [8].

### 2.2 符号

对于 Speck, 用 $n$ 表示 word size (单位是比特), $2n$ 表示状态大小 (单位是比特).

对于 Speck, 用 $(x_r, y_r)$ 表示经过 $r$ 轮加密后的左右分支的状态.

对于 Speck, 用 $\mathrm{rk}_r$ 表示第 $r$ 轮轮密钥.

$x[i]/y[i]/\mathrm{rk}[i]$: $x/y/\mathrm{rk}$ 的第 $i$ 比特 (下标从 0 开始, 最右边为 0).

$\mathrm{rk}[i_2 \sim i_1]$: $\mathrm{rk}[i_2]\|\mathrm{rk}[i_2-1]\|\cdots\|\mathrm{rk}[i_1+1]\|\mathrm{rk}[i_1]$.

对于 Speck, $[j]$ 表示状态 (即 $x\|y$) 第 $j$ 比特的下标, 其中 $y[0]$ 是第 0 比特, $x[0]$ 代表第 $n$ 比特.

$\{i_2 \sim i_1\}$: 比特下标集合 $\{i_2, i_2-1, \ldots, i_1+1, i_1\}$.

$\{[i_2] \sim [i_1]\}$: 中性比特集合 $\{[i_2], [i_2-1], \ldots, [i_1+1], [i_1]\}$.

$\oplus$: 比特异或运算; $\boxplus$: 模加运算 (模是 $2^n$).

$x^{\lll s}$ ($x^{\ggg s}$): 向左 (右) 旋转移位 $s$ 比特.

$x^{\ll s}$ ($x^{\gg s}$): 向左 (右) 移位 $s$ 比特.

对任意比特串 $x$, $y$ 和 $z$, 定义 $\mathtt{eq}(x, y, z) := (\neg x \oplus y) \wedge (\neg x \oplus z)$ (即当且仅当 $x[i] = y[i] = z[i]$ 成立时, $\mathtt{eq}(x, y, z)[i] = 1$).

对任意比特串 $x$, $y$ 和 $z$, 定义 $\mathtt{xor}(x, y, z) := x \oplus y \oplus z$.

对任意整数 $n$, 令 $\mathtt{mask}(n) := 2^n - 1$.

符号 $i \in [1, n]$ 表示 $i$ 是一个处于 $[1, n]$ 之间的整数.

$\Delta_{[i]}$ 表示一个单比特差分, 即只有第 $i$ 比特是活跃比特.

$\mathrm{hw}(x, y)$: $x$ 和 $y$ 的汉明距离.

### 2.3 中性比特

考虑一个加密函数 $f : \mathbb{F}_2^n \to \mathbb{F}_2^n$ 和一个差分 $\Gamma \to \Delta$. Biham 等 [11] 发现明文一些比特的值不影响差分传播. 具体来说, 对于任何符合上述差分的明文对 $(P, P \oplus \Gamma)$, 如果同时翻转 $P$ 和 $P \oplus \Gamma$ 的第 $i$ 比特, 并且得到的密文对差分仍然是 $\Delta$, 那么比特 $i$ 就是一个中性比特. 如果翻转后得到的密文对以概率 $p$ 满足差分 $\Delta$, 就把 $i$ 称作概率性中性比特,$p$ 是该比特的中性度.

Bao 等 [5] 提出了一个新的概念: simultaneous-neutral bit-sets (SNBS). 给定一个包含 $d$ 比特的比特集合 $[i_1, i_2, \ldots, i_d]$, 对于任意符合上述差分的明文对 $(P, P \oplus \Delta)$, 如果同时翻转 $P$ 和 $P \oplus \Gamma$ 的 $d$ 个比特, 得到的密文差分仍然是 $\Delta$, 那么 $[i_1, i_2, \ldots, i_d]$ 就是一个 SNBS.

本文接下来把中性比特、SNBS 统称为中性比特. 如果一个明文对符合给定的一个差分, 就可以用该明文对和 $k$ 个中性比特生成 $2^k$ 个符合给定差分的明文对.

### 2.4 神经区分器

神经区分器本质上是一个执行二分类任务的神经网络, 可以表示为 $Z = \mathcal{ND}(X)$, $Z \in [0, 1]$, 其中 $Z$ 代表神经区分器的输出, $X$ 代表神经区分器的输入 (即密文对). 当 $Z > 0.5$ 时, 代表神经区分器的分类结果为 1; 否则, 代表分类结果为 0.

Gohr [3] 提出了一个深度残差网络, 并基于该深度残差网络成功地构建了针对 Speck32/64 的神经区分器. 在文献 [12, 13] 中, 研究人员也证实了 Gohr 构建神经区分器的方法也适用于其他密码算法. 本文也采用 Gohr 提出的方法构建神经区分器. 所以本小节将回顾 Gohr 提出的构建神经区分器的方法.

**训练数据和输入.** 给定一个密码算法和一个明文差分 $\Delta$, 深度残差网络被训练用于区分两类密文对. 一类密文对对应的明文对的差分为 $\Delta$, 该类密文对也称作真实密文对. 另一类密文对对应的明文对差分为一个随机值, 该类密文对也称作随机密文对. 所以, 训练数据的每一个样本包含一个密文对和标签. 标签为 1, 代表密文对是真实密文对. 标签为 0, 代表密文对是随机密文对. 为了训练神经区分器, 需要生成一个训练集和一个测试集, 两个数据集均包含一半标签为 1 的正类样本, 和一半标签为 0 的负类样本.

假设密码算法的分组长度是 $mn$ 比特, 一个密文对用一个 $m \times n$ 的矩阵表示, 矩阵的每一个元素代表一个比特, 取值为 0 或者 1. 残差神经网络的输入层包含 $mn$ 个神经元, 并且这 $mn$ 个神经元也被安排成 $m \times n$ 的矩阵形式. 当为不同分组长度的密码算法训练神经区分器时, 本文只改变输入层的神经元数量, 其他参数保持不变. 关于深度残差神经网络的更多细节参考文献 [3].

**训练框架.** 在文献 [3] 中一共有 3 种训练框架, 本文介绍并使用其中最基本的训练框架. 深度残差神经网络在包含 $10^7$ 个样本的训练集上一共训练 50 个周期. 在每一个周期里, 数据集被随机划分成多组, 每一组包含 5000 个样本. 深度残差神经网络的误差函数是均方误差损失和 L2 权重正则项的和, 其中正则化因子是 $10^{-5}$. 优化方法是 Keras[14] 提供的 Adam[15] 算法, 参数是默认参数. 循环学习率框架也被采用, 第 $i$ 周期的学习率 $l_i$ 是 $l_i = a + \frac{(n-i) \bmod (n+1)}{n} \times (b-a)$, 其中 $a = 10^{-4}$, $b = 2 \times 10^{-3}$, $n = 9$.

### 2.5 Gohr 的密钥恢复攻击

本小节简要介绍 Gohr 的基础版本密钥恢复攻击的核心原理. 给定一个 $r_1$ 轮前置差分 $\Gamma \xrightarrow{p} \Delta$ 和一个 $r_2$ 轮神经区分器 $\mathcal{ND}$. 目标是恢复第 $(r_1 + r_2 + 1)$ 轮轮密钥 rk.

Gohr 提出的基础版本密钥恢复攻击[3] 如下:

(1) 随机生成一个明文对 $(P, P \oplus \Gamma)$, 并基于前置差分中的 $k$ 个中性比特把该明文对扩展为包含 $2^k$ 个明文对的明文结构, 最后收集对应的密文结构.

(2) 猜测 rk, 对每一个可能的值 kg:

(a) 用 kg 对密文结构解密一轮, 并把解密后的 $2^k$ 个伪密文对送入 $\mathcal{ND}$, 收集对应的分数 $Z_i, i \in [1, 2^k]$.

(b) 使用如下公式联合分数:

$$v_{\text{kg}} := \sum_{j=1}^{2^k} \log_2 \left( \frac{Z_j}{1 - Z_j} \right),$$

并作为 kg 的排名分数.

(c) 如果 $v_{\text{kg}}$ 大于等于一个阈值 $c$, 就保存 kg 作为一个可能的候选密钥.

(3) 返回步骤 (1), 并重复上述步骤, 直至发现候选密钥.

## 3 用于大状态分组密码的深度学习辅助密钥恢复框架

本节将首先介绍深度学习辅助密钥恢复框架的核心思想, 然后给出应用该框架的攻击过程、复杂度分析. 在后文, 我们将直接把本节给出的攻击过程应用于对 Speck 的攻击中. 最后, 本节将会对一些应用该框架时需要注意的因素进行讨论.

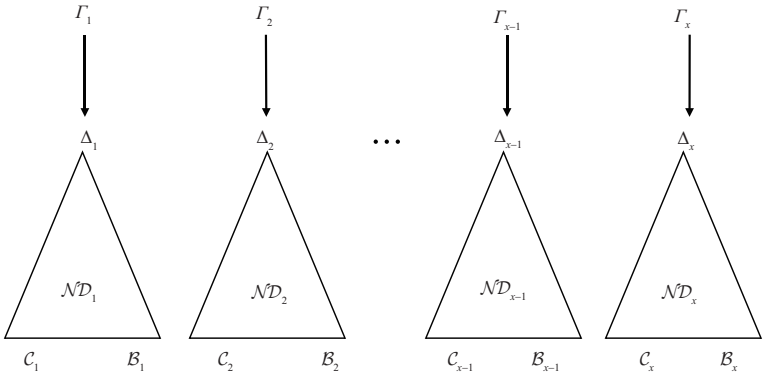


**图 1 用于大状态分组密码的多阶段密钥恢复框架的示意图. 一共使用 $x$ 个神经区分器 $\mathcal{ND}_i$, 其中区分器 $\mathcal{ND}_i$ 采用的明文差分是 $\Delta_i$. 每一个区分器的前面添加了一个前置差分 $\mathcal{CD}_i$, 分别是 $\Gamma_i \to \Delta_i, i \in [1, x]$. 每一个神经区分器在部分状态比特 $\mathcal{C}_i$ 上训练, 并用于恢复部分密钥比特 $\mathcal{B}_i, i \in [1, x]$. 在理想情况下, 并集 $\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_x$ 覆盖了完整的状态, 而并集 $\mathcal{B}_1 \cup \cdots \cup \mathcal{B}_x$ 包含了所有待恢复的密钥比特.**

**Figure 1** The schematic diagram of the multi-stage key-recovery framework for large-state block ciphers. A total of $x$ neural distinguishers $\mathcal{ND}_i$ are used, whose input differences are $\Delta_i$; each $\mathcal{ND}$ is prepended with a $\mathcal{CD}$, and $\mathcal{CD}_i$ is defined as $\Gamma_i \to \Delta_i$ for $i \in [1, x]$. Each $\mathcal{ND}_i$ is trained on partial state bits $\mathcal{C}_i$, and is used to recover partial key bits $\mathcal{B}_i, i \in [1, x]$. In the ideal scenario, the union $\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_x$ covers the full state, the union $\mathcal{B}_1 \cup \cdots \cup \mathcal{B}_x$ contains all the key bits to be recovered.

### 3.1 核心思想

该框架的本质思想是通过多阶段过程恢复完整的子密钥. 每个阶段采用单独的神经区分器 (前面可以加一个前置差分) 来恢复部分密钥比特. 为了达到这个目的, 需要选择一组神经区分器. 其中的每一个神经区分器以密文的部分比特为输入, 进而每一个神经区分器只受一个密钥比特子集影响. 只要保证所有比特子集的并集包含完整的密钥, 我们就可以借助该框架恢复完整密钥.

该框架的多个阶段可以并行或者串行地执行, 这取决于目标密码算法. 在每一个阶段中, 可以应用 Gohr 提出的基础版本, 或者加速版本的密钥恢复攻击. 图 1 是该多阶段密钥恢复框架的示意图.

**选择标准.** 对于一个包含 $x$ 阶段的密钥恢复攻击, $x$ 个神经区分器是基于以下标准选择:

(1) 每一个神经区分器 $\mathcal{ND}_i$ 以部分状态作输入, 并承担恢复部分密钥的任务. 部分状态比特的集合、部分密钥比特的集合分别记作 $\mathcal{C}_i, \mathcal{B}_i, i \in [1, x]$.

(2) 并集 $\mathcal{B}_1 \cup \cdots \cup \mathcal{B}_x$ 应该包含尽可能多的密钥比特. 理想情况[1)] 下应该包含所有待恢复的密钥比特.

(3) 每一个神经区分器的输入长度, 即 $\mathcal{C}_i$ 包含的状态比特数目应该适中, 以避免神经区分器的准确率太小的同时, 确保每一个神经区分器恢复部分密钥的工作量是可行且平衡的. 换言之, 集合的大小 $|\mathcal{C}_i|$ 和 $|\mathcal{B}_i|$, $i \in [1, x]$ 需要合理分配.

基于上述 $x$ 个神经区分器, 整个密钥恢复攻击被划分成 $x$ 阶段. 在阶段 $i \in [1, x]$, $|\mathcal{B}_i|$ 个密钥比特通过 $\mathcal{ND}_i$ 恢复. 每个阶段可以等价地视作攻击一个小状态密码.

### 3.2 攻击过程

假设攻击者已经选定 $x$ 个神经区分器 $\mathcal{ND}_i$ 和 $x$ 个前置差分 $\mathcal{CD}_i := \Gamma_i \xrightarrow{p_i} \Delta_i$, 其中 $p_i, i \in [1, x]$

1) 在非理想情况下 (即输入并集没有包含全部密文比特), 只要输入并集中的比特受所有密钥比特影响, 就也有一定概率成功恢复密钥, 如: 后文针对缩减轮 Speck128 的攻击.

是前置差分的概率. 目标是恢复最后一轮的轮密钥 rk.

整个攻击的 $x$ 个阶段串行执行, 在阶段 $i$, rk 的 $|\mathcal{B}_i|$ 个密钥比特是在获取前 $i-1$ 个阶段的密钥恢复结果的基础上进行恢复的. 也就是说, 在第 $i$ 阶段, 攻击者已经获取 $\bigcup_{j\in[1,i-1]}\mathcal{B}_j$ 的有关知识. 恢复 $\mathcal{B}_i$ 可以用 Gohr 提出的基础攻击, 或者加速攻击. 不失一般性, 本文在每个阶段采用最基础的攻击. 假设 $\epsilon$ 表示一个小的常数, 整个攻击过程就如算法 1 所示.

**算法 1** 串行执行的深度学习辅助多阶段密钥恢复攻击

**输入:** $\mathcal{B}_i, c_i, \beta_i, i\in[1,x]$, $\epsilon$;

**输出:** The final key guess rk;

1: **for** $i\in[1,x]$ **do**
2: 通过选择 $\frac{\epsilon}{p_i}$ 个满足差分 $\Gamma_i$ 的明文对发起阶段 $i$. 利用存在于 $\mathcal{CD}_i$ 中的 $\log_2(N_i)$ 个中性比特把每个明文对扩展成明文结构;
3: **for** $\frac{\epsilon}{p_i}$ 个明文结构中的每一个明文结构 **do**
4: 询问加密算法获取对应的密文结构 (每个密文结构中包含 $N_i$ 个密文对);
5: 初始化一个空列表 $L_i\leftarrow\emptyset$;
6: 从前 $i-1$ 个阶段恢复的密文比特集合 $\bigcup_{j\in[1,i-1]}\mathcal{B}_j$ 对应的部分密钥中选择 $\beta_i$ 个排名分数最高的部分密钥猜测;
7: **for** 上述 $\beta_i$ 个密钥猜测的每一个, 记作 $\mathbf{kg}_{i-1}:=\mathrm{kg}_{i-1}\|\cdots\|\mathrm{kg}_1$ (对于阶段 1, 令 $\beta_1=1$ and $\mathbf{kg}_0=\emptyset$) **do**
8: **for** $\mathcal{B}_i$ 对应的 $2^{|\mathcal{B}_i|}$ 个可能值的每一个 **do**
9: $\mathbf{kg}_i=\mathrm{kg}_i\|\mathrm{kg}_{i-1}\|\cdots\|\mathrm{kg}_1$;
10: 用 $\mathbf{kg}_i$ 对 $N_i$ 个密文对进行部分解密, 提取 $\mathcal{C}_i$ 对应的部分伪密文对;
11: 把 $N_i$ 个部分伪密文对送入神经区分器 $\mathcal{ND}_i$, 获取 $N_i$ 个预测分数 $Z_j, j\in[1,N_i]$;
12: 使用如下公式结合 $N_i$ 个分数计算密钥 $\mathbf{kg}_i$ 的排名分数:

$$v_{\mathbf{kg}_i}:=\sum_{j=1}^{N_i}\log_2\left(\frac{Z_j}{1-Z_j}\right); \tag{1}$$

13: **end for**
14: 如果 $v_{\mathbf{kg}_i}>c_i$, 把 $(\mathbf{kg}_i, v_{\mathbf{kg}_i})$ 添加到列表 $L_i$ 中;
15: **end for**
16: 如果 $L_i\neq\emptyset$, 根据排名分数对 $L_i$ 中的元素进行排序, 并且把排名分数最高的 $\beta_{i+1}$ 个值作为 $\bigcup_{j\in[1,i]}\mathcal{B}_j$ 对应的密钥比特的猜测值. 返回步骤 2, 进入下一阶段;
17: **end for**
18: 如果 $\frac{\epsilon}{p_i}$ 个密文结构全部用完, 并且没有任何值 $\mathbf{kg}_i$ 的排名分数超过 $c_i$, 则终止攻击输出 $\perp$;
19: **end for**
20: 返回阶段 $x$ 中排名分数最高的密钥猜测 $\mathbf{kg}_x$ 作为 rk 的值.

### 3.3 复杂度分析

本小节讨论算法 1 的复杂度上界, 即最坏情况下的复杂度.

在最坏的情况下, 每一个阶段 $i\in[1,x]$ 的 $\frac{\epsilon}{p_i}$ 个密文结构均被使用. 所以算法 1 的数据复杂度上界是

$$\sum_{i=1}^{x}\frac{\epsilon}{p_i}\times N_i. \tag{2}$$

在算法 1 中, 一共有 3 个最基本的操作 (在步骤 8 的最内层循环内):

(1) 使用一个密钥猜测在一个密文对上执行一轮解密;

(2) 把解密后的伪密文对送入神经区分器, 并获得预测分数 $Z$;

(3) 计算 $\log_2(\frac{Z}{1-Z})$ 的值.

假设把上述 3 个操作的整体看作一个原子操作, 则算法 1 的计算复杂度 (即原子操作的数量) 上界是

$$\sum_{i=1}^{x} \beta_i \times 2^{|\mathcal{B}_i|} \times \frac{\epsilon}{p_i} \times N_i, \tag{3}$$

其中 $N_i$ 是阶段 $i$ 中一个密文结构所包含的密文对数量, $\beta_1 = 1$.

假设第 $i$ 阶段的原子操作等价于目标密码算法的 $\eta_i$, $i \in [1, x]$ 次加密. 算法 1 的计算复杂度 (即等价的加密次数) 上界是

$$\sum_{i=1}^{x} \beta_i \times 2^{|\mathcal{B}_i|} \times \frac{\epsilon}{p_i} \times N_i \times \eta_i. \tag{4}$$

### 3.4 适用场景

除了需要精细地选择 $x$ 个神经区分器外, 攻击者在应用本文的多阶段密钥恢复框架时可能还需要考虑以下因素:

• 第 1 个因素是 $x$ 个神经区分器 $\mathcal{ND}_i$ 使用的明文差分 $\Delta_i, i \in [1, x]$. 这些明文差分中的多个甚至所有差分可以是相同的, 这样可以在不同阶段复用数据, 但是针对大状态密码不太容易找到这样的明文差分. 尽管如此, 即使 $x$ 个明文差分各不相同, 对数据复杂度的影响也并不显著, 甚至只是一个常数 $x$ 的区别.

• 第 2 个因素是关于前置差分 $\mathcal{CD}$s (如图 1 所示, 在 $\mathcal{ND}_i$ 前加一个差分 $\mathcal{CD}_i := \Gamma_i \to \Delta_i, i \in [1, x]$). 具体来说有两点: 所有前置差分 $\mathcal{CD}_i$ 和对应的神经区分器 $\mathcal{ND}_i$ 一起覆盖的加密轮数相同; 存在于每一个前置差分 $\mathcal{CD}_i$ 中的中性比特数量对于对应的神经区分器 $\mathcal{ND}_i$ 是足够的.

• 第 3 个因素是 $x$ 阶段的执行顺序. 当 $x$ 个阶段的攻击结果互不影响时, 可以并行地执行 $x$ 个阶段的攻击. 否则, 就必须以串行的方式执行. 以 Speck 为例, 本文对 Speck 的攻击采用算法 1, 攻击是从最低位 (相对于 $\boxplus$) 到最高位逐阶段进行. 一旦阶段 $i$ 猜错了 $\mathcal{B}_i$ 中包含的部分密钥比特, 就可能对后续阶段中猜测 $\bigcup_{j\in[i+1,x]} \mathcal{B}_i$ 包含的密钥比特产生负面影响, 影响的大小取决于目标密码算法的特点. 对于 Speck (或者说 ARX 密码) 来说, 影响因子包含猜错的密钥比特的数量、位置. 在本文对 Speck 进行的实验中, 即使之前的阶段猜错了部分密钥比特, 仍然有较高概率猜对剩余的密钥比特.

除了上述因素外, 在使用本文提出的多阶段密钥恢复框架时没有其他严格的限制.

## 4 针对大状态 Speck 的神经区分器

本节将首先介绍一种寻找一组适用于多阶段密钥恢复框架的神经区分器的方法, 然后再介绍利用该方法为每一个 Speck 大状态成员寻找的一组区分器. 由于 Speck 的密钥长度对神经区分器没有影响, Speck 成员中密钥长度不同但是分组长度 ($2n$ 比特) 相同的成员将统称为 Speck$2n$.

### 4.1 一种构建一组神经区分器的方法

为了找到一组合适的神经区分器来发起一个多阶段密钥恢复攻击, 本文设计了如下方法:

(1) 使用明文差分 $\Delta_{[i]}$, $i \in [0, b-1]$ 训练 $b$ 个神经区分器, 其中 $\Delta_{[i]}$ 代表单比特差分 (即只有第 $i$ 比特是活跃比特[2)]), $b$ 代表明文长度. 这些神经区分器的输入是完整的密文对, 即 $2b$ 个比特. 对于 Speck$2n$, $b = 2n$.

2) 更一般地, 此处可以不局限于单比特明文差分, 只要保证得到的区分器的信息比特集合不同即可.

**表 2　针对缩减轮 Speck 大状态成员的神经区分器** [a)]

**Table 2**　Neural distinguishers against round-reduced large-state Speck[a)]

| | 9-round Speck128 | | | 7-round Speck96 | | | 6-round Speck64 | | |
|---|---|---|---|---|---|---|---|---|---|
| $\mathcal{ND}_i$ | $\Delta_i$ | $\mathcal{C}_i$ | Acc. | $\Delta_i$ | $\mathcal{C}_i$ | Acc. | $\Delta_i$ | $\mathcal{C}_i$ | Acc. |
| $\mathcal{ND}_1$ | $\Delta_{[64]}$ | $\{22\sim18, 14\sim9\}$ | 0.559 | $\Delta_{[53]}$ | $\{19\sim8\}$ | 0.633 | $\Delta_{[42]}$ | $\{17\sim8\}$ | 0.613 |
| $\mathcal{ND}_2$ | $\Delta_{[76]}$ | $\{34\sim30, 26\sim21\}$ | 0.586 | $\Delta_{[65]}$ | $\{31\sim20\}$ | 0.621 | $\Delta_{[47]}$ | $\{29\sim18\}$ | 0.677 |
| $\mathcal{ND}_3$ | $\Delta_{[90]}$ | $\{48\sim44, 40\sim34\}$ | 0.609 | $\Delta_{[77]}$ | $\{43\sim32\}$ | 0.628 | $\Delta_{[33]}$ | $\{31, 30, 7\sim0\}$ | 0.653 |
| $\mathcal{ND}_4$ | $\Delta_{[105]}$ | $\{63\sim59, 55\sim49\}$ | 0.616 | $\Delta_{[89]}$ | $\{47\sim44, 7\sim0\}$ | 0.634 | | | |
| $\mathcal{ND}_5$ | $\Delta_{[117]}$ | $\{11, 7, 4, 3, 0\}$ | 0.559 | | | | | | |

a) $\Delta_i$: the input difference; Acc.: the accuracy of the $\mathcal{ND}_i$; $\mathcal{C}_i$: the index of bits of $x_r$ (at the same time, take the same index of bits of $y_r$) that are fed into $\mathcal{ND}_i$, where $x_r$ (resp. $y_r$) is the left (resp. right) $n$-bit word of a full $r$-round output state.

(2) 识别信息比特, 即对神经区分器的准确率有显著影响的密文比特, 把信息比特集合记作 $\mathcal{C}_i$, $i \in [0, b-1]$. 信息比特的识别通过文献 [6] 中提出的比特敏感性测试完成.

(3) 根据 $\mathcal{C}_i$, $i \in [0, b-1]$、相关的密钥比特集合和第 3.1 小节中提出的选择标准, 从 $b$ 个神经区分器中选择一个包含 $x$ 个神经区分器的合适组合.

(4) 重新训练选出来的 $x$ 个神经区分器. 值得注意的是, 现在每个区分器的输入不是完整的密文对, 而是信息比特集合中包含的密文比特. 对于 Speck2$n$ 来说, 新训练的区分器的输入是一对长度为 $|\mathcal{C}_i|$ 的部分密文.

### 4.2　神经区分器

使用第 4.1 小节中设计的方法, 本文为 Speck128, Speck96, Speck64 分别找到了一个合适的神经区分器组合. 其中, 针对 Speck128 的区分器组合包含 5 个 9 轮神经区分器, 基于以下 5 个明文差分训练得到:

$$\Delta_1 = \Delta_{[64]},\ \Delta_2 = \Delta_{[76]},\ \Delta_3 = \Delta_{[90]},\ \Delta_4 = \Delta_{[105]},\ \Delta_5 = \Delta_{[117]}. \tag{5}$$

针对 Speck96 的神经区分器组合包含 4 个 7 轮神经区分器, 基于以下 4 个明文差分训练得到:

$$\Delta_1 = \Delta_{[53]},\ \Delta_2 = \Delta_{[65]},\ \Delta_3 = \Delta_{[77]},\ \Delta_4 = \Delta_{[89]}. \tag{6}$$

针对 Speck64 的神经区分器组合包含 3 个 6 轮神经区分器, 基于以下 3 个明文差分训练得到:

$$\Delta_1 = \Delta_{[42]},\ \Delta_2 = \Delta_{[47]},\ \Delta_3 = \Delta_{[33]}. \tag{7}$$

表 2 总结了针对 Speck 三类大状态成员的神经区分器组合的更多细节. 这些神经区分器组合将会在第 5 节被应用于密钥恢复攻击.

## 5　对大状态 Speck 的实际密钥恢复攻击

本节将利用第 4 节提出的区分器组合发展针对 Speck 大状态成员的实际密钥恢复攻击, 验证本文提出的多阶段密钥恢复框架.

**表 3 在表 2 中的神经区分器前添加的一轮前置差分**

**Table 3** One-round classical differentials to be prepended to the $\mathcal{ND}$s in Table 2

| | 1-round Speck128 | | | 1-round Speck96 | | | 1-round Speck64 | | |
|---|---|---|---|---|---|---|---|---|---|
| $\mathcal{CD}_i$ | $\Gamma_i \to \Delta_i$ | NB's | $p_i$ | $\Gamma_i \to \Delta_i$ | NB's | $p_i$ | $\Gamma_i \to \Delta_i$ | NB's | $p_i$ |
| $\mathcal{CD}_1$ | $\Delta_{[72,69,61]} \to \Delta_{[64]}$ | $\{[20]\sim[11]\}$ | $2^{-2}$ | $\Delta_{[61,58,2]} \to \Delta_{[53]}$ | $\{[25]\sim[16]\}$ | $2^{-2}$ | $\Delta_{[50,47,7]} \to \Delta_{[42]}$ | $\{[39]\sim[30]\}$ | $2^{-2}$ |
| $\mathcal{CD}_2$ | $\Delta_{[84,81,9]} \to \Delta_{[76]}$ | $\{[32]\sim[23]\}$ | $2^{-2}$ | $\Delta_{[73,70,14]} \to \Delta_{[65]}$ | $\{[37]\sim[28]\}$ | $2^{-2}$ | $\Delta_{[55,52,12]} \to \Delta_{[47]}$ | $\{[39]\sim[30]\}$ | $2^{-2}$ |
| $\mathcal{CD}_3$ | $\Delta_{[98,95,23]} \to \Delta_{[90]}$ | $\{[46]\sim[37]\}$ | $2^{-2}$ | $\Delta_{[85,82,26]} \to \Delta_{[77]}$ | $\{[49]\sim[40]\}$ | $2^{-2}$ | $\Delta_{[41,38,30]} \to \Delta_{[33]}$ | $\{[29]\sim[20]\}$ | $2^{-2}$ |
| $\mathcal{CD}_4$ | $\Delta_{[113,110,38]} \to \Delta_{[105]}$ | $\{[61]\sim[52]\}$ | $2^{-2}$ | $\Delta_{[94,49,38]} \to \Delta_{[89]}$ | $\{[61]\sim[52]\}$ | $2^{-2}$ | | | |
| $\mathcal{CD}_5$ | $\Delta_{[125,122,50]} \to \Delta_{[117]}$ | $\{[73]\sim[64]\}$ | $2^{-2}$ | | | | | | |

### 5.1 对 12 轮 Speck128 的实际攻击

应用第 3 节提出的多密钥恢复攻击框架, 结合第 4 节发现的神经区分器组合, 本文发展了一个针对 12 轮 Speck128 的实际攻击, 可以恢复完整的最后一轮轮密钥.

具体来说, 5 个 9 轮神经区分器 (表 2) 的前面都添加一个 1 轮前置差分 $\Gamma_i \to \Delta_i$ (表 3), 分别组成一个 10 轮的混合区分器 $\mathcal{HD}_i$. 因为 Speck 的第一个非线性操作之前没有白化密钥, 所以每个 10 轮混合区分器可以再免费在顶部上扩 1 轮. 因此, 在最后一轮上面一共有 11 轮, 而攻击的目标是恢复第 12 轮的轮密钥 $\mathrm{rk}_{12}$. 为了完成该攻击, 第 3.2 小节的算法 1 被采用, 具体的攻击参数设定如下所示.

#### 5.1.1 攻击设定

轮密钥 $\mathrm{rk}_{12}$ 的 64 比特通过 5 个阶段恢复:

(1) 阶段 1 猜测 $|\mathcal{B}_1| = 15$ 比特, 即 $\mathrm{rk}_{12}[14 \sim 0]$.

(2) 阶段 2 猜测 $|\mathcal{B}_2| = 12$ 比特, 即 $\mathrm{rk}_{12}[26 \sim 15]$.

(3) 阶段 3 猜测 $|\mathcal{B}_3| = 14$ 比特, 即 $\mathrm{rk}_{12}[40 \sim 27]$.

(4) 阶段 4 猜测 $|\mathcal{B}_4| = 15$ 比特, 即 $\mathrm{rk}_{12}[55 \sim 41]$.

(5) 阶段 5 猜测 $|\mathcal{B}_5| = 8$ 比特, 即 $\mathrm{rk}_{12}[63 \sim 56]$.

表 3 总结了将要使用的 5 个 1 轮差分、对应的差分概率 $p_i$ 和使用的 10 个中性比特 (所以 $N_i = 2^{10}$, $i \in [1,5]$). 常量的值设定为 $\epsilon = 4$, 意味着阶段 $i$ 最多生成 $\frac{4}{p_i}$, $i \in [1,5]$ 个密文结构. 5 个阶段中用于过滤错误密钥的阈值是 $c_1 = c_2 = c_3 = c_4 = c_5 = 10$. 在阶段 $i \in [1,4]$, 保留至下一阶段的幸存密钥的数量上界为 $\beta_{i+1} = 3$ (注意 $\beta_1$ 始终为 1).

#### 5.1.2 实验结果

在上述攻击设定下, 执行了 100 次实验, 实验环境是一台装有一张现代显卡 (GeForce GTX 1080 Ti GPU) 的电脑. 在使用一张显卡、一个 CPU 核心的前提下, 上述攻击的平均时间消耗是 5060 s.

在 100 次实验中, 有 80 次实验返回了一个最终密钥猜测 kg. 把返回的密钥猜测 kg 和正确轮密钥 rk 的汉明距离记作 $\mathrm{hw}(\mathrm{kg},\mathrm{rk})$. 则在上述 80 次实验中, 汉明距离 $\mathrm{hw}(\mathrm{kg},\mathrm{rk})$ 的平均值是 3.2. 关于 $\mathrm{hw}(\mathrm{kg},\mathrm{rk})$ 的具体统计结果总结在表 4 中. 当 $\mathrm{hw}(\mathrm{kg},\mathrm{rk}) \leqslant 3$ 成立时, 本文即把攻击视作成功. 从表 4 中可以看出, 12 轮密钥恢复攻击的成功率是 0.52.

我们把 $\mathrm{hw}(\mathrm{kg},\mathrm{rk}) \leqslant 3$ 作为判断攻击成功的标准有两个理由. 第一, 正确地恢复至少 $(64-3)$ 个密钥比特, 等价于把轮密钥空间从 $2^{64}$ 降低到了 $C(64,3) \times 2^3 = 2^{18.3}$. 实际上, 剩余的密钥空间可以更

**表 4　对 12 轮 Speck128 的攻击的 100 次实验中 hw(kg, rk) 的统计结果**

**Table 4**　Statistic on hw(kg, rk) over 100 trials of the 12-round attacks on Speck128

| hw(kg, rk) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| # trials | 4 | 12 | 17 | 19 | 8 | 9 | 7 | 3 | 0 | 1 |

**表 5　对 10 轮 Speck96 的攻击的 100 次实验中 hw(kg, rk) 的统计结果**

**Table 5**　Statistic on hw(kg, rk) over 100 trials of the 10-round attacks on Speck96

| hw(kg, rk) | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| # trials | 23 | 30 | 18 | 10 | 3 | 3 |

小, 因为我们观察到有一些密钥比特比其他密钥比特被猜错的概率大得多[3)].

第二, 矫正返回的密钥猜测中的少数错误比特并恢复其他的轮密钥, 所需要的复杂度预计很小. 为了实现上述两个任务 (即: 矫正少数错误比特、恢复其他轮密钥), 攻击者可以利用更短轮数的神经区分器 (在本小节的 12 轮攻击中, 即使用 $i$ 轮神经区分器, 其中 $i \leqslant 8$), 并复用已经生成的密文结构. 在通过 $x$ 阶段攻击恢复最后一轮子密钥时, 只要最终返回了一个密钥猜测, 就可以认为找到了 $x$ 个正确的密文结构 (即对应的明文结构通过了前置差分). 原因是 $x$ 个阈值 $c_i$ 会被设置的足够大, 确保当使用错误密文结构时, 不会有密钥猜测的排名分数超过阈值 (攻击中使用的中性比特数目足够多, 导致神经区分器的信号足够强. 所以, 可以很容易地设定合适的阈值来过滤掉所有错误密文结构). 这样一来, 与通过 $x$ 阶段恢复最后一轮轮密钥的复杂度相比, 完成上述两个任务的代价就很小. 而且, 前置差分的概率越小, 后者的复杂度相对而言就会更小.

### 5.2　对 Speck96, Speck64 的实际攻击

与针对缩减轮 Speck128 的实际攻击一样, 也可以利用第 3 节提出的多密钥恢复攻击框架, 结合第 4 节发现的神经区分器组合, 发展针对缩减轮 Speck96, Speck64 的实际密钥恢复攻击, 来恢复完整的最后一轮的轮密钥.

#### 5.2.1　对 10 轮 Speck96 的实际攻击

结合 4 个 7 轮神经区分器 (表 2) 和 4 个 1 轮前置差分 (表 3), 可以得到 4 个 8 轮混合区分器. 再免费上扩一轮, 即可发展针对 10 轮 Speck96 的密钥恢复攻击, 目标是恢复第 10 轮的轮密钥 $\mathrm{rk}_{10}$.

在 4 个阶段中, 每个阶段都猜测轮密钥 $\mathrm{rk}_{10}$ 的 12 比特, 分别是 $\mathrm{rk}_{10}[11 \sim 0]$, $\mathrm{rk}_{10}[23 \sim 12]$, $\mathrm{rk}_{10}[35 \sim 24]$, $rk_{10}[47 \sim 36]$. 表 3 总结了将要使用的 4 个 1 轮差分、对应的差分概率 $p_i$ 和使用的 10 个中性比特 (所以, $N_i = 2^{10}$, $i \in [1, 4]$). 其他攻击参数和第 5.1 小节针对 Speck128 的 12 轮攻击中使用的参数一样, 即 $\epsilon = 4$, $c_i = 10$, $i \in [1, 4]$, 和 $\beta_i = 3$, $i \in [2, 4]$.

在上述攻击设置下, 一共执行了 100 次实验. 在相同的实验环境 (即配备一个 GeForce GTX 1080 Ti GPU 的电脑) 下, 该 10 轮密钥恢复攻击的平均时间消耗是 825 s. 在这 100 次实验中, 一共有 87 次实验返回了一个密钥猜测 kg. 关于返回密钥猜测和正确的轮密钥的汉明距离的具体统计结果 hw(kg, rk) 总结在表 5 中. 同样的, 当 hw(kg, rk) $\leqslant 3$ 时, 就认为攻击成功. 从表 5 中可以看出, 10 轮攻击的成功率是 0.81.

3) 这个现象在对其他 Speck 成员的攻击中也出现了. 但是, 容易猜错的密钥比特的位置在不同 Speck 成员间没有共性.

表 6 对 9 轮 Speck64 的 100 次实验中 $\mathrm{hw}(\mathrm{kg},\mathrm{rk})$ 的统计结果

**Table 6** Statistic on $\mathrm{hw}(\mathrm{kg},\mathrm{rk})$ over 100 trials of the 9-round attacks on Speck64

| $\mathrm{hw}(\mathrm{kg},\mathrm{rk})$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| # trials | 22 | 30 | 29 | 9 | 4 | 1 | 2 | 1 |

#### 5.2.2 对 9 轮 Speck64 的实际攻击

结合 3 个 6 轮神经区分器 (表 2) 和 3 个 1 轮前置差分 (表 3), 可以得到 3 个 7 轮混合区分器. 再免费上扩一轮, 即可发展针对 9 轮 Speck96 的密钥恢复攻击, 目标是恢复第 9 轮的轮密钥 $\mathrm{rk}_9$.

在密钥恢复攻击中, 3 个阶段分别猜测轮密钥 $\mathrm{rk}_9$ 的 10, 12, 10 比特, 分别是 $\mathrm{rk}_9[9\sim 0]$, $\mathrm{rk}_9[21\sim 10]$, $\mathrm{rk}_9[31\sim 22]$. 表 3 总结了将要使用的 3 个 1 轮差分、对应的差分概率 $p_i$ 和使用的 10 个中性比特 (所以 $N_i=2^{10}$, $i\in[1,3]$). 其他攻击参数和第 5.1 小节针对 Speck128 的 12 轮攻击中使用的参数一样, 即 $\epsilon=4$, $c_i=10$, $i\in[1,3]$ 和 $\beta_i=3$, $i\in[2,3]$.

在上述攻击设置下, 一共执行了 100 次实验. 在相同的实验环境 (即配备一个 GeForce GTX 1080 Ti GPU 的电脑) 下, 该 10 轮密钥恢复攻击的平均时间消耗是 90 s. 在这 100 次实验中, 一共有 98 次实验返回了一个密钥猜测 kg. 关于返回密钥猜测和正确的轮密钥的汉明距离的具体统计结果 $\mathrm{hw}(\mathrm{kg},\mathrm{rk})$ 总结在表 6 中. 同样的, 当 $\mathrm{hw}(\mathrm{kg},\mathrm{rk})\leqslant 3$ 时, 就认为攻击成功. 从表 6 中可以看出, 10 轮攻击的成功率是 0.90.

## 6 对大状态 Speck 的理论分析

### 6.1 在神经区分器前置多轮差分

通过使用覆盖更长轮数的前置差分, 可以把第 5 节提出的实际攻击转换成理论分析. 可以使用的前置差分的轮数受中性比特的数量限制. 具体来说, 前置差分中的中性比特数量需要足够多, 以便于使得神经区分器的信号足够强.

#### 6.1.1 在低概率差分路径中寻找中性比特

给定一个差分, 为了找到其中的中性比特, 一般分成两步: 一、收集足够多的正确对 (即满足差分的明文对); 二、把正确对的目标比特翻转或者把目标比特集合包含的比特全部翻转, 并检查新的明文对还是正确对的概率. 当差分概率较低时, 第一步的计算复杂度很高. 因此, 本小节提出了一个高效的方法来收集正确对, 并检测中性比特.

给定一个密码算法和一个差分. 假设在各个轮密钥全部独立随机生成 (记作随机轮密钥设定) 时一个比特是中性比特, 那么当使用密钥生成框架生成轮密钥 (记作真实轮密钥设定) 时, 该比特也是中性比特. 所以接下来的方法将把真实轮密钥设定简化为随机轮密钥设定, 然后通过调整随机采样的轮密钥来为给定差分生成正确对. 通过该方法收集的正确对将被用来识别中性比特.

首先, 我们推断了当明文对是正确对时, 异或差分通过模加传播时的条件. 本文中用 $\delta=(\alpha,\beta\mapsto\gamma)$ 表示通过模加运算 (模是 $2^n$) 的异或差分, 用 $\mathrm{DP}^{+}(\alpha,\beta\mapsto\gamma):=\Pr_{(x,y)\in\mathbb{F}_2^n\times\mathbb{F}_2^n}[(x\boxplus y)\oplus((x\oplus\alpha)\boxplus(y\oplus\beta))=\gamma]$ 表示该差分的概率.

**命题1** 用 $\delta=(\alpha,\beta\mapsto\gamma)$ 表示通过模加运算 (模是 $2^n$) 的异或差分 ($\boxplus$). 如果 $(x,y)$ 和 $(x\oplus\alpha,y\oplus\beta)$

是满足差分 $\delta$ 的正确对, 则 $x$ 和 $y$ 应该满足如下条件. 对于 $i\in[0,n-2]$, 如果 $\mathtt{eq}(\alpha,\beta,\gamma)[i]=0$:

$$\left.\begin{array}{ll} x[i]\oplus y[i]=\mathtt{xor}(\alpha,\beta,\gamma)[i+1]\oplus\alpha[i], & \\ \left.\begin{array}{ll} x[i]\oplus c[i]=\mathtt{xor}(\alpha,\beta,\gamma)[i+1]\oplus\alpha[i], & \text{if } \alpha[i]\oplus\mathtt{xor}(\alpha,\beta,\gamma)[i]=0,\\ y[i]\oplus c[i]=\mathtt{xor}(\alpha,\beta,\gamma)[i+1]\oplus\beta[i], & \text{if } \alpha[i]\oplus\mathtt{xor}(\alpha,\beta,\gamma)[i]=1, \end{array}\right\} & \text{if } \alpha[i]\oplus\beta[i]=1, \end{array}\right\} \text{if } \alpha[i]\oplus\beta[i]=0,$$

其中 $c[i]$ 表示第 $i$ 个进位比特, $\mathtt{eq}(\cdot,\cdot,\cdot)$ 和 $\mathtt{xor}(\cdot,\cdot,\cdot)$ 的定义可以在第 2 节找到.

**证明** 证明参见附录 A.

基于命题 1, 给定一个 Speck 的差分路径, 使用算法 2 生成正确对, 并识别中性比特. 通过调整轮密钥使得大部分正确对需要满足的条件成立, 算法 2 中步骤 1(a)∼(e) 的重复次数远小于 $N/\mathrm{DP}^{+}(\delta)$. 所以, 算法 2 比纯粹的随机采样高效.

**算法 2** 为 Speck 的差分路径生成正确对并过滤候选中性比特

输入: (1) $r$ 轮差分路径 $\delta=(\delta x_0,\delta y_0)\mapsto(\delta x_1,\delta y_1)\mapsto\cdots\mapsto(\delta x_r,\delta y_r)$; (2) 候选中性比特 $I_s=[i_1,i_2,\cdots,i_j]$.
输出: $I_s$ 的中性度 $p$.
(1) 使用如下步骤为 $\delta$ 生成 $N$ 个正确对.
(a) 随机生成一个输入 $(x_0,y_0)$.
(b) 针对输入, 选择第 $i-1$ 个轮密钥来进行第 $i$ 轮轮函数加密. 具体来说, 在进行第 $i$ 轮加密时, 先确定第 $(i-1)$ 个轮密钥的部分比特以确保通过第 $i$ 个 $\boxplus$ 的输入满足命题 1 中列出的条件, 然后随机生成其他比特. 一些取决于进位比特的条件无法通过简单地调整一个密钥比特且不影响其他比特相关条件而被满足. 对于这些条件, 就随机选择密钥比特来绕过它们. 用这种方式计算进行 $r$ 轮加密并得到最终输出 $(x_r,y_r)$.
(c) 使用上一步中生成的轮密钥来计算输入 $(x_0\oplus\delta x_0,y_0\oplus\delta y_0)$ (记作 $(x'_0,y'_0)$) 经过 $r$ 轮加密后的输出 (记作 $(x'_r,y'_r)$).
(d) 如果 $(x'_r,y'_r)$ 满足 $x_r\oplus x'_r=\delta x_r$ 和 $y_r\oplus y'_r=\delta y_r$, 存储 $((x_0,y_0),(x'_0,y'_0))$ 和所有的轮密钥作为一个正确对.
(e) 重复步骤 2 直到收集到 $N$ 个正确对.
(2) 对每一个正确对, 翻转 $I_s$ 中的比特, 然后测试得到的消息对是否还是正确对 (注意需要使用与正确对关联的轮密钥加密). 统计反转比特后, 得到的仍然是正确对的次数, 记作 cnt.
(3) 返回 $p\leftarrow \mathrm{cnt}/N$ 作为 $I_s$ 的中性度.

至于 $N$ 的选择, 使用 $N\in\{2^{10}\sim 2^{13}\}$ 进行了一些初步实验. 结果显示, 令 $N=2^{10}$ 时测得的比特中性度和采用更大的 $N$ 测出来的中性度没有显著差异. 所以, 本文在应用算法 2 时均采用 $N=2^{10}$ 的设定.

### 6.1.2 大状态 Speck 的差分和差分中的中性比特

**大状态 Speck 中的差分.** 对于每一个 Speck 成员, 前置差分的概率可以看作是由一条差分路径所主导. 所以, 本文会用概率最大的差分路径代替前置差分. 添加在不同神经区分器前方的前置差分路径可以从一条基本差分路径中推导出来, 具体方法是把中间状态差分向左旋转 (即循环左移). 旋转后, 只要所有通过模加运算的差分的最低位满足 $\mathtt{xor}(\alpha,\beta,\gamma)[0]=0$ 的条件, 得到的新差分路径就是有效的[4)]. 新的差分路径和基本差分路径的差分概率权重 (即 $-\log_2(p)$, 其中 $p$ 是差分路径概率) 最多相差 $t$, 其中 $t=\max_{i,0\leqslant i<n}(\sum_{j=1}^{r}\neg\mathtt{eq}(\alpha_j,\beta_j,\gamma_j)[i])$, 并且 $(\alpha_j,\beta_j,\gamma_j)$ 是该 $r$ 轮差分路径中通过第 $j$ 个 $\boxplus$ 的输入/输出差分. 在本文接下来的案例中, $t$ 的最大值是 3. 接下来的理论攻击中利用的基本差分路径展示在了表 7 中.

4) 实际上存在比本文使用的差分路径概率更大 (最多相差 $2^2$ 倍) 的差分路径, 但是这些差分路径均是弱密钥路径. 所以本文使用的是非弱密钥路径中概率最大的差分路径. 如果利用弱密钥路径, 攻击的复杂度可以有一点改进, 但是只对部分密钥有效.

表 7 用于 Speck 的差分路径 [a)]

**Table 7** Differential trails used for Speck[a)]

| R | Speck128 ($w$ 43 ∼ 46) | | | Speck96 ($w$ 20 ∼ 22) | | | Speck64 ($w$ 10 ∼ 12) | | |
|---|---|---|---|---|---|---|---|---|---|
| | $\delta x_i$ | $\delta y_i$ | $w_i$ | $\delta x_i$ | $\delta y_i$ | $w_i$ | $\delta x_i$ | $\delta y_i$ | $w_i$ |
| −6 | 4041041440401000 | 0240402406 40d010 | | | | | | | |
| −5 | 0200012012009000 | 1002000020061080 | 14 | | | | | | |
| −4 | 1000000100141010 | 9010000000249410 | 10 | 100100141010 | 901000249410 | | | | |
| −3 | 8000000001248000 | 0080000000002084 | 9 | 800001248000 | 008000002084 | 9 | 81248000 | 00802084 | |
| −2 | 0000000000010404 | 0400000000000024 | 6 | 000000010404 | 040000000024 | 6 | 00010404 | 04000024 | 6 |
| −1 | 0000000000000120 | 2000000000000000 | 4 | 000000000120 | 200000000000 | 4 | 00000120 | 20000000 | 4 |
| 0 | 0000000000000001 | 0000000000000000 | 2 | 000000000001 | 000000000000 | 2 | 00000001 | 00000000 | 2 |

a) The listed are base trails. Other trails can be obtained by rotating (by word) the base trails to the left. The notation ($w$ $w_1 \sim w_2$) in the first row represents the range of the differential weights of the trails that can be obtained by rotating the base trails.

**差分中的中性比特.** 表 8 列出了对基本差分旋转得到的差分路径的信息, 包括差分概率、中性比特数量 (应用算法 2 搜索中性比特, 最多同时翻转 3 个比特). 其中, 对于 Speck128/Speck96/Speck64 中的差分, 只统计中性度大于等于 0.7/0.8/0.8 的中性比特. 表 9 列出了本文使用的差分路径和对应的中性比特.

有趣的是, 和差分路径一样, 我们发现旋转得到的差分路径中的中性比特也可以通过把基本差分中的中性比特向左旋转得到. 例如, 对于表 9 中 'Speck128' 列用符号 $\mathcal{CD}_{[64]}$ 表示的差分路径来说, 比特集合 $[26, 37, 98]$ 是一个中性比特. 通过把 $\mathcal{CD}_{[64]}$ 向左旋转 53 比特可以得到差分路径 $\mathcal{CD}_{[117]}$, 而比特集合 $[26, 37, 98]$ 向左旋转 53 比特后的比特集合

$$[(26+53)\text{mod}64, (37+53)\text{mod}64, 64+((98+53)\text{mod}64)]$$

(也就是 $[15, 26, 87]$) 也是差分路径 $\mathcal{CD}_{[117]}$ 的中性比特.

总的来说, 对于 Speck128/Speck96/Speck64 的 6/4/3 轮差分, 都可以找到超过 10 个可用的中性比特. 对于更长的差分, 具有高中性度的中性比特均不超过 3 个.

### 6.2 对扩展轮数的 Speck128, Speck96, Speck64 的理论分析

对于 Speck128/128, 通过在 5 个 9 轮神经区分器 (表 2) 前面分别添加一个 6 轮差分 (表 9), 可以得到 5 个 15 轮的混合区分器 $\mathcal{HD}s$. 这 5 个混合区分器可以分别表示为 $6r_{\mathcal{CD}} - \Delta_{[64]} - 9r_{\mathcal{ND}}$, $6r_{\mathcal{CD}} - \Delta_{[76]} - 9r_{\mathcal{ND}}$, $6r_{\mathcal{CD}} - \Delta_{[90]} - 9r_{\mathcal{ND}}$, $6r_{\mathcal{CD}} - \Delta_{[105]} - 9r_{\mathcal{ND}}$, $6r_{\mathcal{CD}} - \Delta_{[117]} - 9r_{\mathcal{ND}}$. 利用这 5 个 15 轮混合区分器, 可以发起一个多阶段密钥恢复攻击, 来恢复 17 轮 Speck128/128 的最后一轮的轮密钥. 这个 17 轮攻击的每一阶段均由顶部的一轮自由可逆轮、中间的 15 轮混合区分器、和末尾的 1 轮部分轮密钥猜测组成 (记作 $1r + 6r_{\mathcal{CD}} - \Delta_{[\cdot]} - 9r_{\mathcal{ND}} + 1r$).

该 17 轮攻击的实现采用第 3.2 小节提出的算法 1, 并且攻击参数和第 5.1 小节提出的针对 12 轮 Speck128 的实际攻击的参数相同. 和针对 12 轮 Speck128 的实际攻击相比, 本小节的 17 轮攻击只有两点不同: (1) 每一阶段需要生成更多的明文结构, 因为 6 轮差分路径的概率远低于 1 轮差分的概率; (2) 每一阶段使用的中性比特的中性度不完全是 1, 所以也导致每一阶段需要更多的明文结构. 除此之外, 两个攻击没有本质区别. 在恢复最后一轮的轮密钥过程中, 可以找到 5 个满足前置差分的正确密

**表 8　由表 7 中的基本差分路径旋转 (向左旋转 rol 比特) 得到的差分路径的差分概率权重 (用符号 $w$ 表示) 和其中独立的中性比特的数量 (用 #NB 表示)**

**Table 8** The differential weights (denoted by $w$) and the numbers of independent NBs (denoted by #NB) of the differential trails obtained by rotating (to the left by rol bits) the base trail in Table 7

| Speck128 (#NB with $\mathrm{Pr_{nb}} > 0.7$) | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB |
| **0** | **45** | **12** | 8 | 45 | 12 | 16 | 46 | 13 | 24 | 46 | 13 | 32 | 46 | 13 | 40 | 46 | 12 | 48 | 44 | 12 | 56 | 44 | 12 |
| 1 | 46 | 12 | 9 | 45 | 12 | 17 | 45 | 13 | 25 | 46 | 13 | 33 | 46 | 13 | **41** | **45** | **12** | 49 | 45 | 12 | 57 | 46 | 12 |
| 2 | 45 | 12 | 10 | 46 | 12 | 18 | 46 | 12 | **26** | **45** | **14** | 34 | 45 | 12 | 42 | 45 | 12 | 50 | 45 | 12 | 58 | 45 | 12 |
| 3 | 44 | 12 | 11 | 45 | 12 | 19 | 46 | 12 | 27 | 46 | 13 | 35 | 45 | 12 | 43 | 45 | 12 | 51 | 43 | 12 | 59 | 43 | 12 |
| 4 | 46 | 12 | **12** | **46** | **12** | 20 | 46 | 12 | 28 | 46 | 13 | 36 | 46 | 12 | 44 | 46 | 12 | 52 | 46 | 12 | 60 | 46 | 12 |
| 5 | 45 | 12 | 13 | 46 | 12 | 21 | 46 | 12 | 29 | 45 | 12 | 37 | 45 | 12 | 45 | 44 | 12 | **53** | **44** | **12** | 61 | 44 | 12 |
| 6 | 45 | 12 | 14 | 45 | 13 | 22 | 46 | 12 | 30 | 46 | 13 | 38 | 45 | 12 | 46 | 45 | 12 | 54 | 46 | 12 | 62 | 46 | 12 |
| 7 | 46 | 12 | 15 | 46 | 15 | 23 | 45 | 14 | 31 | 45 | 13 | 39 | 45 | 12 | 47 | 45 | 12 | 55 | 45 | 12 | 63 | 45 | 12 |
| Speck96 (#NB with $\mathrm{Pr_{nb}} > 0.8$) | | | | | | | | | | | | | | | | | | | | | | | |
| rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB |
| 0 | 21 | 25 | 6 | 22 | 23 | 12 | 22 | 23 | 18 | 22 | 21 | 24 | 22 | 22 | 30 | 22 | 23 | 36 | 22 | 21 | 42 | 21 | 22 |
| 1 | 22 | 23 | 7 | 22 | 23 | 13 | 22 | 23 | 19 | 22 | 22 | 25 | 22 | 23 | 31 | 21 | 24 | 37 | 20 | 22 | 43 | 21 | 21 |
| 2 | 21 | 24 | 8 | 21 | 24 | 14 | 22 | 22 | 20 | 22 | 22 | 26 | 21 | 23 | 32 | 21 | 22 | 38 | 22 | 21 | 44 | 22 | 21 |
| 3 | 21 | 23 | 9 | 22 | 26 | 15 | 22 | 22 | 21 | 22 | 21 | 27 | 22 | 22 | 33 | 22 | 24 | 39 | 21 | 22 | 45 | 20 | 24 |
| 4 | 22 | 23 | 10 | 22 | 24 | 16 | 22 | 21 | 22 | 22 | 21 | 28 | 22 | 23 | 34 | 21 | 21 | 40 | 21 | 21 | 46 | 22 | 22 |
| **5** | **21** | **23** | 11 | 21 | 24 | **17** | **22** | **22** | 23 | 21 | 23 | **29** | **21** | **22** | 35 | 21 | 21 | **41** | **22** | **21** | 47 | 21 | 25 |
| Speck64 (#NB with $\mathrm{Pr_{nb}} > 0.8$) | | | | | | | | | | | | | | | | | | | | | | | |
| rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB | rol | $w$ | #NB |
| 0 | 12 | 18 | 4 | 12 | 16 | 8 | 11 | 16 | 12 | 12 | 14 | 16 | 12 | 15 | 20 | 12 | 15 | 24 | 11 | 14 | 28 | 12 | 14 |
| **1** | **12** | **16** | 5 | 11 | 17 | 9 | 12 | 14 | 13 | 12 | 14 | 17 | 12 | 15 | 21 | 11 | 15 | 25 | 12 | 15 | 29 | 10 | 16 |
| 2 | 11 | 18 | 6 | 12 | 17 | **10** | **12** | **14** | 14 | 12 | 14 | 18 | 11 | 14 | 22 | 12 | 17 | 26 | 11 | 15 | 30 | 12 | 15 |
| 3 | 12 | 15 | 7 | 12 | 15 | 11 | 12 | 14 | **15** | **11** | **16** | 19 | 12 | 15 | 23 | 11 | 15 | 27 | 12 | 14 | 31 | 11 | 26 |

文结构, 然后可以使用短轮神经区分器 (如: 8 轮神经区分器) 在正确密文结构上恢复其他轮 (如: 倒数第 2 轮) 的轮密钥.

同样的, 基于第 5.2 小节提出的针对 10 轮 Speck96、9 轮 Speck64 的实际攻击, 通过把表 3 中展示的 1 轮差分和中性比特转换成表 9 中展示的长轮差分和中性比特, 可以设计对 13 轮 Speck96、11 轮 Speck64 的理论攻击.

根据第 3 节中的复杂度分析, 理论攻击中最差情况下的数据复杂度、时间复杂度可以直接用式 (2) 和 (4) 计算. 表 10 展示了理论攻击的复杂度分析, 其中 $\epsilon$, $N_i$, $\beta_i$, $|\mathcal{B}_i|$ 和第 5 节中的实际攻击相同. $\mathrm{Pr_{nb}}$'s 代表由中性比特的中性度带来的因子, 具体值可以在表 9 的最后一列找到. 因子 $\eta_i$ 是攻击中一个原子操作 (第 3 节) 和加密一个明文的时间比. 这些时间比是通过 $2^{12}$ 次实验得到, 每一次实验处理 $2^{10}$ 明文对. 理论攻击的成功率预期和对应实际攻击的成功率一致, 即对 Speck128/128 的 17 轮理论攻击对应 12 轮实际攻击, 对 Speck96/96 的 13 轮理论攻击对应 10 轮实际攻击, 对 Speck64/96 的 11 轮理论攻击对应 9 轮实际攻击.

表 9 神经区分器的前置差分中的中性比特和对应的概率 [a)]

**Table 9** The concrete NBs and their empirical probabilities for the $\mathcal{CD}$s to be prepended to the $\mathcal{ND}$s[a)]

| Speck128 (10 NBs for each of the five $\mathcal{CD}$s) | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\mathcal{CD}$ | rol | $[26,37,98]^{\lll \mathsf{rol}}$ | $[34,106]^{\lll \mathsf{rol}}$ | $[41]^{\lll \mathsf{rol}}$ | $[42]^{\lll \mathsf{rol}}$ | $[43]^{\lll \mathsf{rol}}$ | $[112]^{\lll \mathsf{rol}}$ | $[113]^{\lll \mathsf{rol}}$ | $[114]^{\lll \mathsf{rol}}$ | $[115]^{\lll \mathsf{rol}}$ | $[116]^{\lll \mathsf{rol}}$ | $\mathrm{Pr}_{\mathsf{nb}}$ |
| $\mathcal{CD}_{[64]}$ | 0 | 1.000 | 0.998 | 0.947 | 0.901 | 0.818 | 1.000 | 0.981 | 0.954 | 0.924 | 0.852 | 0.567 |
| $\mathcal{CD}_{[76]}$ | 12 | 0.999 | 1.000 | 0.970 | 0.947 | 0.840 | 0.999 | 0.979 | 0.964 | 0.929 | 0.873 | 0.649 |
| $\mathcal{CD}_{[90]}$ | 26 | 0.994 | 1.000 | 0.935 | 0.873 | 0.778 | 0.995 | 0.977 | 0.932 | 0.883 | 0.833 | 0.502 |
| $\mathcal{CD}_{[105]}$ | 41 | 0.995 | 0.997 | 0.951 | 0.872 | 0.785 | 0.994 | 0.979 | 0.951 | 0.917 | 0.836 | 0.516 |
| $\mathcal{CD}_{[117]}$ | 53 | 0.996 | 0.998 | 0.936 | 0.896 | 0.768 | 0.996 | 0.975 | 0.944 | 0.896 | 0.820 | 0.512 |
| Speck96 (10 NBs for each of the four $\mathcal{CD}$s) | | | | | | | | | | | | |
| $\mathcal{CD}$ | rol | $[10,21,66]^{\lll \mathsf{rol}}$ | $[25]^{\lll \mathsf{rol}}$ | $[26]^{\lll \mathsf{rol}}$ | $[27]^{\lll \mathsf{rol}}$ | $[28]^{\lll \mathsf{rol}}$ | $[80]^{\lll \mathsf{rol}}$ | $[81]^{\lll \mathsf{rol}}$ | $[82]^{\lll \mathsf{rol}}$ | $[83]^{\lll \mathsf{rol}}$ | $[84]^{\lll \mathsf{rol}}$ | $\mathrm{Pr}_{\mathsf{nb}}$ |
| $\mathcal{CD}_{[53]}$ | 5 | 1.000 | 1.000 | 0.999 | 0.996 | 0.998 | 1.000 | 1.000 | 0.999 | 0.999 | 0.999 | 0.993 |
| $\mathcal{CD}_{[65]}$ | 17 | 1.000 | 1.000 | 0.999 | 0.999 | 0.993 | 1.000 | 0.999 | 1.000 | 1.000 | 1.000 | 0.990 |
| $\mathcal{CD}_{[77]}$ | 29 | 1.000 | 1.000 | 0.999 | 0.997 | 0.994 | 1.000 | 1.000 | 0.999 | 0.997 | 0.999 | 0.987 |
| $\mathcal{CD}_{[89]}$ | 41 | 1.000 | 0.998 | 0.997 | 0.998 | 0.994 | 1.000 | 1.000 | 1.000 | 0.997 | 0.995 | 0.987 |
| Speck64 (10 NBs for each of the three $\mathcal{CD}$s) | | | | | | | | | | | | |
| $\mathcal{CD}$ | rol | $[7,47]^{\lll \mathsf{rol}}$ | $[13,53]^{\lll \mathsf{rol}}$ | $[17]^{\lll \mathsf{rol}}$ | $[18]^{\lll \mathsf{rol}}$ | $[19]^{\lll \mathsf{rol}}$ | $[56]^{\lll \mathsf{rol}}$ | $[57]^{\lll \mathsf{rol}}$ | $[58]^{\lll \mathsf{rol}}$ | $[59]^{\lll \mathsf{rol}}$ | $[60]^{\lll \mathsf{rol}}$ | $\mathrm{Pr}_{\mathsf{nb}}$ |
| $\mathcal{CD}_{[42]}$ | 10 | 1.000 | 1.000 | 0.998 | 0.992 | 0.967 | 1.000 | 0.999 | 0.998 | 0.996 | 0.998 | 0.951 |
| $\mathcal{CD}_{[47]}$ | 15 | 1.000 | 1.000 | 0.983 | 0.971 | 0.938 | 1.000 | 0.990 | 0.982 | 0.971 | 0.948 | 0.854 |
| $\mathcal{CD}_{[33]}$ | 1 | 1.000 | 1.000 | 0.983 | 0.981 | 0.969 | 1.000 | 0.991 | 0.982 | 0.975 | 0.955 | 0.887 |

a) The $\mathcal{CD}$s can be obtained by rotating by word (to the left by rol bits) the base trail in Table 7 (corresponding to the ones in bold in Table 8). For each $\mathcal{CD}$, its NBs can be obtained by rotating by word (to the left by rol bits) the NBs listed in the first row for each version of Speck. Concretely, for Speck2$n$, suppose $i < n$ and $j \geqslant n$, then $[i,j]^{\lll k}$ means $[(i+k)\%n,\ n+(j+k)\%n]$. The column titled $\mathrm{Pr}_{\mathsf{nb}}$ shows the empirical probability that all $2^{10}$ pairs in a structure created from a conforming pair using 10 NBs are conforming pairs. Each empirical probability in this table is obtained by performing 1024 independent trials.

对 17 轮 Speck128/128 的攻击可以直接转换成对 18 轮 Speck128/192 (或者是 19 轮 Speck128/256) 的攻击. 为了实现这一点, 只需要先猜测第 18 轮 (或者是第 19 和 18 轮) 的轮密钥, 并最多重复 17 轮攻击 $2^{64}$ (或者是 $2^{128}$) 次. 预计只有正确猜测最后一轮 (或者最后两轮) 的轮密钥时, 内存循环的 17 轮攻击才会返回一个密钥猜测. 相似的分析也适用于 13 轮 Speck96/96 (或者是 14 轮 Speck96/144) 和 13 轮 Speck64/128 (参考表 10 和 1).

### 6.3 对 Speck64 的 11 轮攻击的验证

本小节验证了针对 11 轮 Speck64 的密钥恢复攻击. 我们在一个配备专业显卡 (Tesla V100-SXM2-32GB GPU) 的服务器上执行了 55 次实验. 在一个 CPU 核心和一张显卡的环境上, 该实验的平均运行时间时 19395 s. 每次实验平均使用了 7891 个密文结构, 等价于 $2^{23.95}$ 个选择明文.

在 55 次实验中, 有 53 次实验返回了密钥猜测并且汉明距离 $\mathrm{hw}(\mathrm{kg},\mathrm{rk})$ 的平均值是 2.0, 有关 $\mathrm{hw}(\mathrm{kg},\mathrm{rk})$ 的具体统计结果总结在了表 11 中. 当 $\mathrm{hw}(\mathrm{kg},\mathrm{rk}) \leqslant 3$ 成立时, 则认定攻击成功, 所以成功率是 0.87. 从以上结果可知, 11 轮攻击的实际验证结果和表 10 中的理论分析结果基本一致.

**表 10　对 Speck 大状态成员的理论攻击的复杂度** [a)]

**Table 10**　Complexity of the theoretical attacks using multiple-round $\mathcal{CD}s$ with the $\mathcal{ND}s$[a)]

| 17-round Speck128/128 | | | | | |
|---|---|---|---|---|---|
| Attack stage$_i$ | DC | | TC | | Ps. |
| | $\epsilon/(p_i \times \Pr_{\mathsf{nb}}) \times N_i$ pairs | #CP | $\beta_i \times 2^{\lvert\mathcal{B}_i\rvert} \times$ DC pairs $\times\eta_i$ | #Enc | |
| $1r + 6r_{\mathcal{CD}} - \Delta_{[64]} - 9r_{\mathcal{ND}} + 1r$ | $4/(2^{-45} \times 0.567) \times 2^{10}$ | $2^{58.82}$ | $1 \times 2^{15} \times 2^{57.82} \times 10.69$ | $2^{76.24}$ | |
| $1r + 6r_{\mathcal{CD}} - \Delta_{[76]} - 9r_{\mathcal{ND}} + 1r$ | $4/(2^{-46} \times 0.649) \times 2^{10}$ | $2^{59.62}$ | $3 \times 2^{12} \times 2^{58.62} \times 10.41$ | $2^{75.58}$ | |
| $1r + 6r_{\mathcal{CD}} - \Delta_{[90]} - 9r_{\mathcal{ND}} + 1r$ | $4/(2^{-45} \times 0.502) \times 2^{10}$ | $2^{58.99}$ | $3 \times 2^{14} \times 2^{57.99} \times 10.77$ | $2^{77.00}$ | |
| $1r + 6r_{\mathcal{CD}} - \Delta_{[105]} - 9r_{\mathcal{ND}} + 1r$ | $4/(2^{-45} \times 0.516) \times 2^{10}$ | $2^{58.95}$ | $3 \times 2^{15} \times 2^{57.95} \times 10.90$ | $2^{77.98}$ | |
| $1r + 6r_{\mathcal{CD}} - \Delta_{[117]} - 9r_{\mathcal{ND}} + 1r$ | $4/(2^{-44} \times 0.512) \times 2^{10}$ | $2^{57.97}$ | $3 \times 2^{8} \times 2^{56.97} \times 9.74$ | $2^{69.84}$ | |
| $\sum_i$ #CP | – | $\mathbf{2^{61.28}}$ | $\sum_i$ #Enc | $\mathbf{2^{78.98}}$ | **0.52** |
| 13-round Speck96/96 | | | | | |
| Attack stage$_i$ | DC | | TC | | Ps. |
| | $\epsilon/(p_i \times \Pr_{\mathsf{nb}}) \times N_i$ pairs | #CP | $\beta_i \times 2^{\lvert\mathcal{B}_i\rvert} \times$ DC pairs $\times\eta_i$ | #Enc | |
| $1r + 4r_{\mathcal{CD}} - \Delta_{[53]} - 7r_{\mathcal{ND}} + 1r$ | $4/(2^{-21} \times 0.993) \times 2^{10}$ | $2^{34.01}$ | $1 \times 2^{12} \times 2^{33.01} \times 12.00$ | $2^{48.59}$ | |
| $1r + 4r_{\mathcal{CD}} - \Delta_{[65]} - 7r_{\mathcal{ND}} + 1r$ | $4/(2^{-22} \times 0.990) \times 2^{10}$ | $2^{35.01}$ | $3 \times 2^{12} \times 2^{34.01} \times 11.89$ | $2^{51.17}$ | |
| $1r + 4r_{\mathcal{CD}} - \Delta_{[77]} - 7r_{\mathcal{ND}} + 1r$ | $4/(2^{-21} \times 0.987) \times 2^{10}$ | $2^{34.02}$ | $3 \times 2^{12} \times 2^{33.02} \times 12.12$ | $2^{50.20}$ | |
| $1r + 4r_{\mathcal{CD}} - \Delta_{[89]} - 7r_{\mathcal{ND}} + 1r$ | $4/(2^{-22} \times 0.987) \times 2^{10}$ | $2^{35.02}$ | $3 \times 2^{12} \times 2^{34.02} \times 12.30$ | $2^{51.23}$ | |
| $\sum_i$ #CP | – | $\mathbf{2^{36.60}}$ | $\sum_i$ #Enc | $\mathbf{2^{52.61}}$ | **0.81** |
| 11-round Speck64/96 | | | | | |
| Attack stage$_i$ | DC | | TC | | Ps. |
| | $\epsilon/(p_i \times \Pr_{\mathsf{nb}}) \times N_i$ pairs | #CP | $\beta_i \times 2^{\lvert\mathcal{B}_i\rvert} \times$ DC pairs $\times\eta_i$ | #Enc | |
| $1r + 3r_{\mathcal{CD}} - \Delta_{[42]} - 6r_{\mathcal{ND}} + 1r$ | $4/(2^{-12} \times 0.951) \times 2^{10}$ | $2^{25.07}$ | $1 \times 2^{10} \times 2^{24.07} \times 11.93$ | $2^{37.65}$ | |
| $1r + 3r_{\mathcal{CD}} - \Delta_{[47]} - 6r_{\mathcal{ND}} + 1r$ | $4/(2^{-11} \times 0.854) \times 2^{10}$ | $2^{24.23}$ | $3 \times 2^{12} \times 2^{23.23} \times 12.40$ | $2^{40.45}$ | |
| $1r + 3r_{\mathcal{CD}} - \Delta_{[33]} - 6r_{\mathcal{ND}} + 1r$ | $4/(2^{-12} \times 0.887) \times 2^{10}$ | $2^{25.17}$ | $3 \times 2^{10} \times 2^{24.17} \times 12.00$ | $2^{39.34}$ | |
| $\sum_i$ #CP | – | $\mathbf{2^{26.47}}$ | $\sum_i$ #Enc | $\mathbf{2^{41.13}}$ | **0.90** |

a) DC: data complexity; TC: time complexity; Ps.: success rate; #CP: number of chosen plaintexts; #Enc: number of equivalent encryptions under the targeted cipher.

**表 11　对 11 轮 Speck64 的 55 次攻击中 $\mathrm{hw}(\mathrm{kg}, \mathrm{rk})$ 的统计结果**

**Table 11**　Statistic on $\mathrm{hw}(\mathrm{kg}, \mathrm{rk})$ over 55 trials of the 11-round attacks on Speck64

| $\mathrm{hw}(\mathrm{kg}, \mathrm{rk})$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| # trials | 4 | 16 | 12 | 16 | 5 |

## 7　在更多密码上的应用

为了证明多阶段密钥恢复框架的通用性, 在其他分组密码算法上 (包括 SIMON [8], PRESENT [9] 和 DES [1]) 也进行了实验, 并分别找到了适合进行密钥恢复攻击的神经区分器组合. 表 12 展示了一组针对 16 轮 SIMON128 的神经区分器、一组针对 5 轮 PRESENT 的神经区分器和一组针对 5 轮 DES 的神经区分器. 如果将多轮差分放在这些神经区分器前面, 只要有足够的中性比特就可以发展覆盖更多轮数的密钥恢复攻击.

表 12 针对缩减轮 SIMON, PRESENT, DES 的神经区分器组合 [a)]

**Table 12** Neural distinguishers against round-reduced SIMON, PRESENT, and DES[a)]

| | 16-round SIMON128 | | | 5-round PRESENT | | | 5-round DES | | |
|---|---|---|---|---|---|---|---|---|---|
| $\mathcal{ND}_i$ | $\Delta_i$ | $\mathcal{C}_i$ | Acc. | $\Delta_i$ | $\mathcal{S}_i$ | Acc. | $\Delta_i$ | $\mathcal{S}_i$ | Acc. |
| $\mathcal{ND}_1$ | $\Delta_{[8]}$ | $\{9\sim0\}$ | 0.669 | $\Delta_{[9]}$ | $\{2,4,7,10,12,15\}$ | 0.832 | $\Delta_{[41]}$ | $\{1\}$ | 0.669 |
| $\mathcal{ND}_2$ | $\Delta_{[21]}$ | $\{22\sim10\}$ | 0.717 | $\Delta_{[22]}$ | $\{11\}$ | 0.838 | $\Delta_{[45]}$ | $\{2,4,7\}$ | 0.717 |
| $\mathcal{ND}_3$ | $\Delta_{[34]}$ | $\{35\sim23\}$ | 0.717 | $\Delta_{[25]}$ | $\{0,1,3,8,9\}$ | 0.839 | $\Delta_{[62]}$ | $\{3,5,6,8\}$ | 0.717 |
| $\mathcal{ND}_4$ | $\Delta_{[47]}$ | $\{48\sim36\}$ | 0.717 | $\Delta_{[34]}$ | $\{5,6,13,14\}$ | 0.800 | | | |
| $\mathcal{ND}_5$ | $\Delta_{[54]}$ | $\{55\sim49\}$ | 0.718 | | | | | | |
| $\mathcal{ND}_5$ | $\Delta_{[62]}$ | $\{63\sim56\}$ | 0.711 | | | | | | |

a) $\Delta_i$: the input difference; Acc.: the accuracy of the $\mathcal{ND}_i$; $\mathcal{C}_i$: the index of bits of $x_r$ (at the same time, take the same index of bits of $y_r$) that are fed into $\mathcal{ND}_i$, where $x_r$ (resp. $y_r$) is the left (resp. right) $n$-bit word of a full $r$-round output state; $\mathcal{S}_i$: the index of S-boxes fed into $\mathcal{ND}_i$.

## 8 总结

本文提出了一个基于神经区分器的多阶段密钥恢复框架, 使得深度学习辅助密钥恢复攻击可以应用于大状态分组密码. 为了证明该框架的有效性, 本文在 Speck 大状态成员上应用该框架进行密码分析. 首先, 本文训练了多个神经区分器, 并基于这些区分器设计了针对 Speck128, Speck96, Speck64 的基础攻击. 通过设计一种为低概率差分寻找中性比特的方法, 并基于这些实际攻击, 设计了覆盖更多轮数的理论攻击. 最终, 针对缩减轮的 Speck 的最大状态成员 (即 Speck128), 取得了计算复杂度、数据复杂度均更优的密钥恢复攻击. 同时, 在缩减轮 Speck96, Speck64 上, 也取得了高效的密钥恢复攻击.

## 参考文献


1 Biham E, Shamir A. Differential cryptanalysis of the full 16-round DES. In: Proceedings of Annual International Cryptology Conference, 1992. 487–496

2 Matsui M. Linear cryptanalysis method for DES cipher. In: Proceedings of Workshop on the Theory and Application of of Cryptographic Techniques, 1993. 386–397

3 Gohr A. Improving attacks on round-reduced Speck32/64 using deep learning. In: Proceedings of Annual International Cryptology Conference, 2019. 150–179

4 Dinur I. Improved differential cryptanalysis of round-reduced speck. In: Proceedings of International Conference on Selected Areas in Cryptography, 2014. 147–164

5 Bao Z Z, Guo J, Liu M C, et al. Conditional differential-neural cryptanalysis. IACR Cryptol ePrint Arch, 2021, 2021: 719

6 Chen Y, Shen Y, Yu H. Neural-aided statistical attack for cryptanalysis. Comput J, 2022, doi: 10.1093/comjnl/bxac099

7 Benamira A, Gerault D, Peyrin T, et al. A deeper look at machine learning-based cryptanalysis. In: Proceedings of Annual International Conference on the Theory and Applications of Cryptographic Techniques, 2021. 805–835

8 Beaulieu R, Shors D, Smith J, et al. The SIMON and SPECK lightweight block ciphers. In: Proceedings of the 52nd Annual Design Automation Conference, 2015

9 Bogdanov A, Knudsen L R, Leander G. PRESENT: an ultra-lightweight block cipher. In: Proceedings of International Workshop on Cryptographic Hardware and Embedded Systems, 2007. 450–466

10 Song L, Huang Z J, Yang Q Q. Automatic differential analysis of ARX block ciphers with application to SPECK and LEA. In: Proceedings of the 21st Australasian Conference on Information Security and Privacy, 2016

11 Biham E, Chen R. Near-collisions of SHA-0. In: Proceedings of Annual International Cryptology Conference, 2004. 290–305

12 Baksi A, Breier J, Chen Y, et al. Machine learning assisted differential distinguishers for lightweight ciphers. In: Proceedings of Design, Automation & Test in Europe Conference & Exhibition (DATE), 2021. 176–181
13 Chen Y, Shen Y, Yu H, et al. A new neural distinguisher considering features derived from multiple ciphertext pairs. Comput J, 2022, doi: 10.1093/comjnl/bxac019
14 Chollet F, et al. Keras. 2015. https://github.com/fchollet/keras
15 Kingma D P, Ba J. Adam: a method for stochastic optimization. In: Proceedings of the 3rd International Conference for Learning Representations, 2015

## 附录 A 命题 1 的证明

**性质 A1** (模加的基本性质) 如果 $(x,y) \in \mathbb{F}_2^n \times \mathbb{F}_2^n$, 则 $x \boxplus y = x \oplus y \oplus \mathtt{carry}(x,y)$, 其中加法 $x \boxplus y$ 的进位 $\mathtt{carry}(x,y) := c \in \mathbb{F}_2^n$ 定义如下. 第一, $c[0] := 0$. 第二, 对于 $i \geqslant 0$, $c[i+1] := (x[i] \wedge y[i]) \oplus (x[i] \wedge c[i]) \oplus (y[i] \wedge c[i])$[5].

**引理 A1** $\mathrm{DP}^{+}(\alpha, \beta \mapsto \gamma) = \Pr_{(x,y)\in\mathbb{F}_2^n\times\mathbb{F}_2^n}[\mathtt{carry}(x,y) \oplus \mathtt{carry}(x\oplus\alpha, y\oplus\beta) = \mathtt{xor}(\alpha,\beta,\gamma)]$, 其中 $\mathtt{xor}(x,y,z) := x \oplus y \oplus z$[5].

**定理 A1** 令 $\delta = (\alpha, \beta \mapsto \gamma)$ 表示一个通过对 $2^n$ 的模加的异或差分. 算法 A1 在 $\Theta(\log n)$ 的时间内返回 $\mathrm{DP}^{+}(\delta)$. 更准确地说, 算法 A1 的执行时间为 $\Theta(1) + t$, 其中 $t$ 是计算 $\mathtt{w_h}$ 的时间 [5].

**算法 A1** 计算 $\mathrm{DP}^{+}(\delta)$

输入: $\delta = (\alpha, \beta \mapsto \gamma)$.
输出: $\mathrm{DP}^{+}(\delta)$.
(1) 如果 $\mathtt{eq}(\alpha^{\ll 1}, \beta^{\ll 1}, \gamma^{\ll 1}) \wedge (\mathtt{xor}(\alpha,\beta,\gamma) \oplus (\beta^{\ll 1})) \neq 0$ 就返回 0;
(2) 返回 $2^{-\mathtt{w_h}(\neg\mathtt{eq}(\alpha,\beta,\gamma)\wedge\mathtt{mask}(n-1))}$.

**证明** (Proof of 命题 1) 用 $\Delta c$ 表示 $\mathtt{carry}(x,y) \oplus \mathtt{carry}(x\oplus\alpha, y\oplus\beta)$ by. 根据引理 A1, 对于一个可能的差分 $\delta := (\alpha, \beta \mapsto \lambda)$ 的任何一个正确对, $(x,y)$ and $(x\oplus\alpha, y\oplus\beta)$, 可知

$$\begin{aligned}
&\mathtt{xor}(\alpha,\beta,\gamma)[i+1] \\
&\quad = \Delta c[i+1] \\
&\quad = (x[i]y[i] \oplus x[i]c[i] \oplus y[i]c[i]) \\
&\qquad \oplus ((x[i]\oplus\alpha[i])(y[i]\oplus\beta[i]) \oplus (x[i]\oplus\alpha[i])(c[i]\oplus\Delta c[i]) \oplus (y[i]\oplus\beta[i])(c[i]\oplus\Delta c[i])) \\
&\quad = x[i]\beta[i] \oplus y[i]\alpha[i] \oplus \alpha[i]\beta[i] \oplus (x[i]\oplus y[i]\oplus\alpha[i]\oplus\beta[i])\Delta c[i] \oplus (\alpha[i]\oplus\beta[i])c[i].
\end{aligned}$$

因为 $\delta$ 是一个可能的差分, 根据定理 A1, 只需要考虑比特 $i$, $0 \leqslant i < n-1$, 且 $\mathtt{eq}(\alpha,\beta,\gamma)[i] = 0$. 对于这些比特 $i$, 可知如下结论. 如果 $\alpha[i] \oplus \beta[i] = 0$, 则 $\alpha[i] = \beta[i] \neq \lambda[i] = \Delta c[i]$ (注意 $\mathtt{xor}(\alpha[i],\beta[i],\lambda[i]) = \Delta c[i]$). 所以, $\alpha[i]\beta[i] = 0$ 和 $\alpha[i] \oplus \Delta c[i] = 1$ 成立. 进一步, 可知

$$\begin{aligned}
&\mathtt{xor}(\alpha,\beta,\gamma)[i+1] = (x[i]\oplus y[i])(\alpha[i]\oplus\Delta c[i]) \oplus \alpha[i], \text{ thus} \\
&x[i] \oplus y[i] = \mathtt{xor}(\alpha,\beta,\gamma)[i+1] \oplus \alpha[i];
\end{aligned}$$

如果 $\alpha[i] \oplus \beta[i] = 1$, 则有

$$\begin{aligned}
&\mathtt{xor}(\alpha,\beta,\gamma)[i+1] = (x[i]\oplus y[i])(\alpha[i]\oplus\Delta c[i]) \oplus x[i] \oplus \Delta c[i] \oplus c[i], \text{ thus} \\
&x[i] \oplus c[i] = \mathtt{xor}(\alpha,\beta,\gamma)[i+1] \oplus \alpha[i], \quad \text{if } \alpha[i] \oplus \mathtt{xor}(\alpha,\beta,\gamma)[i] = 0, \\
&y[i] \oplus c[i] = \mathtt{xor}(\alpha,\beta,\gamma)[i+1] \oplus \beta[i], \quad \text{if } \alpha[i] \oplus \mathtt{xor}(\alpha,\beta,\gamma)[i] = 1.
\end{aligned}$$

5) Lipmaa H, Moriai S. Efficient algorithms for computing differential properties of addition. In: Proceedings of International Workshop on Fast Software Encryption, 2001. 336–350

# A deep learning-aided key recovery framework for large-state block ciphers

Yi CHEN[1], Zhenzhen BAO[2], Yantian SHEN[1] & Hongbo YU[1*]

1. *Department of Computer Science and Technology, Tsinghua University, Beijing* 100084*, China*;
2. *Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing* 100084*, China*
* Corresponding author. E-mail: yuhongbo@mail.tsinghua.edu.cn

**Abstract** The deep learning-aided key recovery attack is a new cryptanalysis technique published in CRYPTO'2019. The drawback of this technique is that it does not apply to large-state block ciphers. To overcome the drawback, this paper proposes a deep learning-based multistage key recovery framework. The core of this technique is to find a combination of neural distinguishers (NDs) for performing key recovery attacks at each stage. To apply this multistage key recovery framework to large-state members of Speck, multiple NDs are trained and combined into groups. Employing the groups of NDs under the multistage key recovery framework, practical attacks are designed and trialed to show framework effectiveness. The practical attacks are then extended to theoretical attacks, covering more rounds by prepending longer differentials before NDs. Moreover, to boost signals from NDs, an efficient algorithm is proposed to find neutral bits for differentials with low probability. Therefore, considerable improvement is observed in terms of both time and data complexities of differential key recovery attacks on round-reduced Speck with the largest state. This work paves the way for performing cryptanalysis using deep learning on more block ciphers. The related code is available at https://github.com/AI-Lab-Y/NAAF.

**Keywords** large-state block ciphers, deep learning, key recovery attack, differential cryptanalysis, Speck