上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

学士学位论文

THESIS OF BACHELOR

论文题目：祖冲之算法电路的差分功耗分析技术研究

学生姓名：于泽汉

学生学号：5142119010

专　　业：微电子科学与工程

指导教师：郭筝

学院(系)：电子信息与电气工程学院

祖冲之算法电路的差分功耗分析技术研究

摘　要

密码设备的安全性一直备受研究人员关注，其中旁路攻击技术扮演了十分重要的角色，而功耗分析攻击又是诸多旁路攻击方法中的主流手段。因此，研究功耗分析技术，不但可以加深我们对密码设备安全性的理解，更能揭示出理论安全的密码算法在实际实现时可能会出现的众多问题，从而指导我们在生产实践中采取必要的防护措施。

在传统的旁路攻击中，分组密码是主要的研究对象，这方面的研究成果也较多。由于序列密码（也称为流密码）算法中的加密变换随时间变化，因此，相比分组密码，找到序列密码中密钥和设备功耗之间的对应关系相对困难，这方面的研究成果也相对较少。

本文以祖冲之算法（也称作 ZUC 算法）作为序列密码算法的典型，力图将传统的分组密码功耗分析方法应用于序列密码算法，借此表明序列密码算法同样无法抵御功耗分析攻击，并总结出一套系统的模型建立和数据处理的流程和方法。

本研究首先在硬件和软件上实现了祖冲之算法，然后对祖冲之算法进行了分析，通过数学推导和严格论证，提出了可行的攻击方案，阐述了方案的具体实施过程。本文对一些具体细节进行了深入的剖析和说明，最后将其应用到现实设备的分析中，成功地实现了完整的功耗分析流程。

实验结果显示，通常用于攻击分组密码的差分功耗分析方法也同样适用于祖冲之算法这样的序列密码算法，并且攻击的流程和框架也都基本相同。实验实现了算法的硬件电路和软件代码，成功地攻击出了相关的密钥字节，并且详细分析了算法电路信息泄露的位置，对相关实验结果给出了可能的原因和合理的解释。

关键词：密码学　数字电路　旁路攻击　差分功耗分析　祖冲之算法

A STUDY ON DIFFERENTIAL POWER ANALYSIS OF CIRCUITS RUNNING ZUC ALGORITHM

ABSTRACT

Researchers have paid a lot of attentions to the security of cryptographic devices. Side-channel attacks have greate influences on it. Power analysis is one of the most powerful methods in side-channel attacks. Therefore, the studies on power analysis can not only deepen our understanding of the security of cryptographic devices, but can also reveal a number of potential problems when theoretically secure algorithms are used in real-world applications, which will provide much guidance on the defence of the security of cryptographic devices.

In traditional side-channel attacks, block cipher algorithms are the main objects of study and researchers have a lot of achievements on them. Compared to block cipher algorithms, it is harder for stream cipher algorithms to find the relations between the power of devices and the cipher key of the algorithm for because of the time-variant transformations during encryptions. This leads to less studies on the power analysis of stream cipher algorithms.

This paper studies the technologies used in traditional power analysis of block cipher algorithms, and applies them to stream cipher algorithms. We take the ZUC algorithm as a typical case. The results indicate that stream cipher algorithms are also vulnerable to power analysis attacks. We will build a general process of modeling and data analysis through the experiments.

This research first realizes the ZUC algorithm in both hardware and software, then analyzes the ZUC algorithm. Through mathematical deduction and strict proofs, we put forward a feasible attack scheme. We also implement the complete process of the scheme sucessfully on real-world devices. We clarify some specific details and give many explanations.

The experimental results show that the differential power analysis method, which is usually used to attack the block ciphers, also applies to the sequence cipher algorithms such as the ZUC algorithm, and the process and framework of the attack are basically the same. The experiment implements the hardware curcuits and software codes of the algorithm, successfully attacks the related key bytes, and analyzes the location of the information leakage of the algorithm circuit in detail. We also give the possible reasons and reasonable explanations for the related experimental results.

KEY WORDS: cryptography, digital circuits, side-channel attack, differential power analysis, ZUC algorithm

目 录

第一章　绪论

1.1　课题背景与研究意义

经过多年的学术研究和工业应用，密码学理论已经日趋系统和完善，各种密码算法广泛应用于各种工业设备，以保障系统和数据的安全。

目前，那些得到广泛使用的密码算法，通常都经过数学上的严格论证，并且经过了大量专家的研究和改进，因而在理论上基本是安全的。然而在现实生活中，这些算法都运行在具体的设备上，因此可能会暴露出各种各样的安全问题，研究者和攻击者可以藉由各种手段，获取密码设备中的秘密信息。

在诸多攻击密码算法和密码设备的手段中，旁路攻击是极为有效和实用的一类。“旁路”的含义是利用实际密码设备泄露的信息，而不是利用密码算法本身的漏洞。根据时间、成本以及仪器的不同，旁路攻击又可以划分为很多种类，具体的方法也多种多样。在实际应用中，旁路攻击的效果远优于传统的密码学理论分析，因此得到了攻击者和研究人员的青睐。

在诸多旁路攻击的方法中，功耗分析是研究最多的一种，其应用也最广。功耗分析攻击所需的花费极低，即使是普通的采集设备和示波器，辅以简单的软件程序，就能完成攻击。与此同时，功耗分析攻击不会对设备进行拆解动作，因此不会留下破坏的痕迹，也不会损伤设备，这既降低了研究的成本，也减少了攻击被发现的可能性。因此，作为旁路攻击的一种最为典型的方法，我们有必要详细研究这种方法。研究清楚了功耗分析攻击，再去研究其他类型的攻击方式就要容易多了。

通常的功耗分析都是以分组密码作为研究对象的，而本文则以 ZUC 密码算法为例来说明，功耗分析一样可以攻击序列密码算法。

在实施攻击之前，攻击者一定要对算法本身有充分地研究，这样才能找到可能泄露信息的地方，并加以利用。我们将介绍祖冲之算法的详细知识，试图从算法的流程中寻找可以攻击的位置。

1.2　国内外研究现状

针对 ZUC 算法的分析和攻击，国内外已经有了一些相关的研究和结果。

文献 [1] 提出了针对 ZUC 算法的差分功耗分析攻击，选取非线性函数的输出部分，攻击出部分密钥字节，再枚举其他部分，即可实现完整的攻击。

文献 [2] 研究了对嵌入设备上运行的 ZUC 算法实施频域攻击，其主要方法基于傅里叶变换。该研究表明，频域上泄露的信息比时域更加严重，因此攻击的效果也更好。

文献 [3] 和 [4] 从数学上提出了减少密钥猜测搜索空间的方法，先对密钥中的一小部分进行猜测，然后再攻击其余的部分，降低了搜索的复杂度。

文献 [5] 和 [6] 改进了传统的功耗分析方法，通过选择初始向量来提高攻击的效率。实验表明，相比使用随机的初始向量进行差分功耗攻击，选取特定的初始向量可以大大减少所需的功耗迹数目，并且攻击的效果更加显著。

文献 [7] 尝试了对 ZUC 和 SNOW 3G 等序列密码算法实施猜测决定攻击，将 ZUC 算法中和 32 比特相关的非线性函数拆分成了和 16 比特相关，减小了攻击时需要猜测的情况。

文献 [8] 和 [9] 则讨论了针对更一般的序列密码算法的攻击，分析了序列密码算法中常见的组件，比如线性反馈移位寄存器，讨论了对这类组件进行攻击的可能性。

虽然当前有不少针对 ZUC 算法的攻击方法，但是大多数研究和实验都是在仿真环境中进行的。因此，这些方法在现实场景中的可行性和具体效果如何，还缺乏可靠的证据。

因此，本研究将结合已知的攻击方法，把攻击方案应用到实际的物理设备上，从而验证攻击方法的可行性。

1.3 本文结构

第一章介绍了本课题的研究背景和重要意义，探讨了国内外分析 ZUC 算法的研究现状，这些文献中提及的思路和方法对我们的实验有一定的指导和启发作用。

第二章介绍了现代密码学和旁路攻击的基础知识，熟悉常见的密码分析方法和旁路攻击思路，并且介绍了经典的功耗分析方法，包括功耗构成、功耗仿真和功耗采集。

第三章介绍了 ZUC 算法的背景、流程和结构，在硬件和软件上实现了这一算法，并且具体讲解了几个重要的模块。

第四章介绍了对 ZUC 算法的具体攻击方案，通过数学推导给出了详细的实施流程，并且对一些细节进行了深入的分析。

第五展示了对 ZUC 算法实施差分功耗攻击的实验结果，并从不同的维度对实验结果进行了分析和解释。

第六章对全文进行了总结和回顾，并且提出了可以进一步深入研究的方向。

第二章　现代密码学与旁路攻击的基础知识

2.1　现代密码学

由于目前应用在实际生活中的绝大多数密码学理论都属于现代密码学，因此我们讨论的重点也集中在现代密码学。

现代密码学和古典密码学的一个重要区别是：现代密码学构建在严格的数学论证和完备的系统架构之上。目前普遍使用的算法也都是现代密码算法，并且经过许多分析、攻击和改进，才渐渐保证其理论上的安全性。[10]

信息安全是现代密码学的一个重要应用领域。衡量信息安全的指标包括：保密性、完整性、认证性、不可否认性以及可用性。要达成这些严格的指标，就需要同样严格的现代密码学体系支撑。这些安全指标有些是彼此掣肘的，因此在设计一个密码系统时，需要综合各方面的因素，才能达到预期的目的，保证系统的安全性。

现代密码学遵循一些基本的准则，比如柯克霍夫原则（Kerckhoffs's principle）。该原则指出，系统的安全性不应依赖于具体实现的保密性。信息论的创始人香农（Shannon）也说："敌人了解系统。"这些原则来源于历史上的经验和教训：几乎所有尝试隐藏算法和系统实现方式本身的做法，最后都失败了。

这些原则促进了通用算法的公开和评审，现实表明，公开的算法经过层层筛选和多次改进，具备更加优良的理论安全性和实际可用性。因此，当前应用广泛的许多密码算法都是公开透明的。我们在进行密码学分析和旁路攻击时，不需要在获取算法细节方面花费不必要的工夫（不过，相应地，这也意味着这些算法本身已经具有极高的安全性，想要分析和攻击它们是一件非常困难的事情）。[11]

划分现代密码学有很多依据，其中一个就是密钥的特征，分为对称密码学和非对称密码学。对称密码学应用的一个典型是 AES（Advanced Encryption Standard，高级加密标准）算法，其特点是加密和解密使用相同的密钥。非对称密码学应用的代表是 RSA（Rivest–Shamir–Adleman）算法，其特点是加密和解密使用不同的密钥。[12]

相比非对称密码学，旁路攻击在对称密码学方面的研究相对多一些。一方面可能是因为对称密码算法加解密采用同一密钥，破解密钥更有意义；另一方面是由于非对称密码算法常常要进行大数运算，速度比对称密码算法要慢很多，因而在对性能要求较高的工业密码设备中，非对称密码算法应用较少，也就造成了实际样本较少，并且研究动力不足。本文涉及的祖冲之算法，也是对称密码算法中的一种。

2.2　密码设备

数学家和密码学家们常常关注理论层面，试图从根本上提高密码算法的安全性，而工程师则更多地和现实打交道，将这些纸上的东西转化成真正能够工作的软硬件设备。也正是由于这种分工，熟悉密码学理论的数学家和密码学家们难以注意到现实设备在实现时可能存在的安全隐患，而整天从

事软硬件设备开发和维护的工程师，也鲜有人能够通晓算法背后的原理，从而留下一些潜在的可以利用的漏洞和瑕疵。

上一节我们简单提及了密码学的理论部分，因此这一节我们将讲述一些和实际密码设备相关的知识。了解密码设备的软硬件构成，有助于我们掌握实际设备的特性，以及可能出现的信息泄露点，为我们的旁路攻击打下必要的基础。

密码算法的执行通常涉及到两部分，一部分是数据的保存，另一部分是数据的操作。相应地，密码设备也可以分为两种，一种是保存数据的存储设备，另一种是操作数据的处理设备。

密码设备一般拥有这几个部分：

- **专用硬件**：比如执行特定密码算法的电路
- **通用硬件**：比如控制算法流程的控制电路
- **存储硬件**：用于存储数据和程序
- **接口**：用于规定数据的输入和输出

密码设备通常由数字电路实现，不同部分既可以在同一块芯片上完成，也可以分到多个芯片上。我们这里以单芯片的密码设备为例。通常，单芯片数字电路可以通过两种方式实现，一种是 FPGA（Field Programmable Gate Array，现场可编程门阵列），另一种是 ASIC（Application-Specific Integrated Circuit，专用集成电路）。虽然用不同方式实现的密码设备在运行性能、工作场景以及成本上有所不同，但实现流程却基本相同。

不管是 FPGA 还是 ASIC，其本质都是数字电路。数字电路的基本组成部分是各种逻辑元件。通常有两种逻辑元件，一种是组合元件，实现基本的逻辑功能，比如与、或、非，另一种是时序元件，比如触发器、锁存器以及寄存器。时序元件和组合元件最大的不同是，时序元件的输出不仅取决于当前的输入，还和元件当前的状态有关。

目前，在实现逻辑元件的工艺中，以 CMOS（Complementary Metal Oxide Semiconductor，互补金属氧化物半导体）最为常见。

2.3 旁路攻击

传统的密码学分析试图挖掘算法本身存在的问题，抑或是寻找某些数学前提和密码学假设的漏洞，从而在根本上攻破某一密码学算法或系统。一旦从理论上揭露了某种密码学算法或系统的潜在问题，那么暴露出来的问题就是致命的，使用这些算法或系统的设备要么被废弃，要么经受升级和改进，否则就不能消除安全隐患。

然而，正如上文提到的，由于现代密码学主张密码算法设计的公开化和透明化，因此，大多数现在广泛使用的密码算法和系统都经过了严格的论证、长期的研究和持续的改进。一方面，对于使用这些算法和系统的设备来说，安全性得到了极大的提升；另一方面，对于相关领域的研究人员和攻击者来说，使用传统方法破解算法或者是提取算法中的重要信息，变得日益困难。

虽然现代密码学的严格化和公开化的初衷在于更好地保障信息安全，但是这对攻击者和研究人员无疑也是巨大的挑战：攻击者需要开发更强大的武器和工具，花费更多的时间和成本，才能达成攻击目的；而研究人员也需要付出更多的精力和汗水，掌握更多的数学知识，才能驾驭日益复杂的

密码算法。

而旁路分析的出现，则提供了另一种截然不同的思路，并且打开了一扇通往新世界的大门。旁路分析的思路和方法五花八门，各有千秋，狭义上的旁路攻击通常指被动型非侵入式攻击，也就是说，这类攻击只是对密码设备的各种攻击方法的一小部分。但是广义上的旁路攻击则包含各种非传统意义上的密码分析手段。

以攻击密码设备时使用的接口作为分类依据，一般将攻击的方式划分为三种：侵入式攻击，半侵入式攻击非和侵入式攻击。

下面对这三类攻击作一些简单的介绍和说明：[13]

- **侵入式攻击（Invasive Attacks）**：能够进行侵入式攻击的人通常能够对密码设备拥有较长时间的物理接触权限，因而可以对设备进行非常仔细的拆卸和分解，能够获得设备详尽的信息和状况。不过，要完成侵入式攻击，付出的代价也相对高，比如需要探测台和激光切片器这样昂贵的仪器和技术。
- **半侵入式攻击（Semi-Invasive Attacks）**：半侵入式攻击一般也要对密码设备进行一定程度的拆卸和分解，但是通常不会破坏芯片的钝化膜，不直接接触芯片的表面。半侵入式攻击相比侵入式攻击，在仪器上花费的成本要低一些，但如何选择正确的部位以从设备中读取泄露的信息依旧要耗费大量的时间。比如常见的故障攻击，就属于半侵入式攻击的范畴。
- **非侵入式攻击（Non-Invasive Attacks）**：相比上面两种攻击，非侵入式攻击成本是最低的，并且在攻击完成后，不会有明显的痕迹。狭义上的旁路攻击，就属于非侵入式攻击，只利用设备合法的接口以及泄露的侧信道信息，就能通过数学手段提取出设备中的敏感内容。正因为非侵入式攻击所需的设备以及花费的代价很低，因此比前两种攻击应用更加广泛，威胁也更大。

从具体的实施手段上讲，常见的旁路攻击有这些方式：[14]

- **时序攻击（Timing Attack）**：设备中的程序在运行时，不同的指令和操作耗时并不是相等的。如果这些指令和操作中包含秘密参数，那么执行时间上的差异就有可能将这些信息泄露出去。如果对设备或算法的具体实现非常了解，甚至可以结合统计学的方法将秘密参数完全恢复出来。
- **功耗分析攻击（Power Analysis Attack）**：功耗分析攻击通常应用于硬件设备。在执行不同的指令或者处理不同的数据时，设备散发的功耗也是存在差异的。功耗分析通常也分为简单功耗分析和差分功耗分析。在各种各样的旁路攻击方式中，功耗分析攻击是研究最为成熟的，其方法最为系统，防护的方案也最多，是当前旁路攻击研究领域中的热门方向。
- **电磁攻击（Electro Magnetic Attack）**：除了功耗之外，设备在运行时还会发出电磁辐射。同样地，设备的指令和数据也会影响电磁辐射的特征。和功耗分析攻击类似，电磁攻击也分为简单电磁攻击和差分电磁攻击。军方很久之前就已经注意到了这种攻击，比如美国国家安全局。虽然功耗分析攻击和电磁攻击使用的方法和思路非常类似，但电磁攻击有时候比功耗分析更加强大，采用了功耗分析防护方案的设备有时候也不能抵挡电磁攻击。
- **可见光攻击（Visible Light Attack）**：攻击者们还注意到了设备的光学信息。比如有研究表明，阴极射线显像管（CRT）产生的光在投射到墙壁上，并且经过漫反射后，依旧可能泄露设备的相关信息。相同的技术也可以应用到发光二极管（LED）上。可见光攻击的一大优点就是

不需要物理接触设备，这是大部分其他攻击方式都不具备的。

- **声学攻击（Acoustic Attack）**：处理器在执行操作时也会泄露声学信息，通过分析声学信息的差异，也有可能获取相关的秘密信息。但这一领域尚处在早期阶段，还没有出现成熟的应用。
- **故障攻击（Fault Attack）**：大部分算法和程序都假定设备能够正常工作，很多防护方案也是基于这一前提。然而，如果设备的某些部分或者操作出现故障，一旦涉及到处理秘密信息的操作，那么就有可能泄露这方面的信息。事实上，在针对智能卡的攻击方面，故障攻击是非常实用和有效的一种。研究和实验表明，几乎所有的密码算法都无法抵御故障攻击。不过实施有效的故障攻击需要很多基本条件，因此难度相对较高。
- **错误消息攻击（Error Message Attack）**：这里的消息通常是指发给设备的指令或者参数。很多设备在处理输入参数时，需要验证格式的合法性，然后返回一定的信息。如果合理的构造某些特殊的消息输入设备，攻击者就有可能从返回结果中得到有用的信息，借此恢复秘密内容。
- **缓存攻击（Cache-based Atack）**：缓存攻击的思路某种程度上和计时攻击有些相像。数据和指令一般从内存经过缓存再到处理器，如果某些数据在处理器中没有找到（miss，或者说未命中），那么就会有一个延时，这个延时用于从内存中导入相关的数据到缓存中。通过这个延时，攻击者能够分析出缓存未命中的出现及其频率，从而进一步分析出相关信息。
- **频率攻击（Frequency-based Attack）**：在进行功耗分析攻击或电磁攻击时，如果时域上的曲线没有处理好（对齐、滤波），从频域上也能得到有用的信息。
- **扫描攻击（Scan-based Attack）**：在集成电路测试中，扫描技术很常见。到攻击者手中，则成了一个强大的武器。有研究和实验表明，在运行 DES 算法的设备上，可以通过扫描链技术恢复出对应的密钥。
- **组合旁路攻击（Combination of Side Channel Attacks）**：单一的旁路攻击技术可能不足以实施成功的攻击，因此攻击者和研究人员驶入结合多信道对设备进行分析和攻击，比如计时攻击与功耗分析攻击结合，功耗分析攻击与电磁攻击结合。
- **结合数学分析的旁路攻击（Combination of SCA and Mathematical Attacks）**：传统的密码学分析已经相当成熟，因此一旦结合旁路攻击得到部分有用的信息，整个算法就有可能被全盘攻破。数学分析与旁路攻击结合，将会是非常强大的攻击手段。

2.4 功耗分析攻击

2.4.1 功耗构成

我们在前面提到过，密码设备通常是由数字电路构成的，而数字电路的基本元件采用的是 CMOS 工艺。因此设备的功耗也就是整个 CMOS 构成的数字电路的功耗，要研究密码设备的功耗，就需要研究 CMOS 电路的功耗来源和组成。

我们这里所说的功耗，通常是指整个设备的产生的总功耗，一般采集到的也都是这类功耗。

逻辑元件的功耗一般分为静态功耗和动态功耗。

静态功耗通常来自晶体管的漏电流，其占比相对较低，不过需要注意的是，随着单个晶体管的尺寸越来越小，漏电流的比重在逐渐提高。当然，在实际的攻击中，静态功耗基本可以忽略不计。

动态功耗通常来自信号的翻转，从而造成晶体管的截断或者导通，此时等价于电流对晶体管的本征电容和寄生电容进行充放电。另一部分动态功耗来自瞬时的短路电流。还有一种需要考虑的情况是电路工作时产生的毛刺，这类毛刺往往会产生很高的瞬时功耗，而且和数据相关，因此需要特别关注。总之，动态功耗一般是元件功耗的主要组成部分，我们通常采集的功耗，也大多数是动态功耗。[13]

2.4.2 功耗仿真

在数字电路的设计阶段，设计者往往要对电路产生的功耗进行仿真。一方面是为了尽可能降低电路的功耗，以提高电路的市场竞争力；另一方面是为了避免出现较为明显的毛刺之类不利因素，影响电路的基本功能；还有一方面就是出于安全性的考虑，尽可能减少功耗泄露的秘密信息。

我们可以在不同的层级对电路的功耗进行仿真。仿真的级别越底层，粒度越细，仿真的结果就有可能越符合实际情况，当然也就需要消耗更多的计算资源；相反，仿真的级别越高层，粒度越粗，仿真的结果出现偏差的可能性就越高，不过带来的好处就是较少的计算资源和仿真时间。所以具体选择在哪个层级对电路的功耗进行仿真，需要视具体情况而定，在成本和拟真度之间做一个较好的折中。

功耗仿真的层级按照粒度从粗到细，可以分为行为级、逻辑级和模拟级。

对电路功耗最为粗糙的仿真，就是行为级的仿真。行为级的仿真通常只考虑电路的重要组成部件，比如控制单元、处理单元、存储单元以及专门的运算单元。行为级仿真通常在数据的操作或者程序的运行这一层面考虑问题，这是因为设备的功耗往往是依赖于操作的数据和运行的程序的。尽管行为级的仿真可能无法给出非常精确的结果，但是较高的仿真速度可以让开发人员在设计初期就能迅速对电路的功耗有一个大致的感觉，从而更好地指导后续的电路设计。

稍细粒度的功耗仿真是逻辑级仿真。数字电路设计软件可以在设计者画出电路的同时，生成电路的网表，这些网表表示电路中各个逻辑元件之间的连接关系，以及一些相关的参数和信息，比如信号的上升和下降延时。在这一层级，仿真通常会用到汉明距离模型或者汉明重量模型，这两类模型对功耗进行了简化，以此降低计算的复杂度，提高仿真的速度。

模拟级的功耗仿真是粒度最细的，得到的仿真结果也是三种当中最为精确的，当然也就意味着耗费的时间和成本最高。电路的各种连接关系以及详细的参数都记录在晶体管网表中，寄生电容这类实际出现的效应也会被纳入计算。然而，由于晶体管数目极为庞大，如果将所有参数都考虑进来，计算耗费的时间和成本将十分可观。因此仿真时往往会对电路的模型做一些必要的简化，这将带来一定的精度损失。所以，需要在精度和速度之间做一个折中。一般不会对整个电路做模拟级的仿真，因为这样付出的代价实在太大。只有对重要的和关键的部件才会进行模拟级的仿真，因为得到这些部分的功耗信息可能比其他部分更加有用，或者更有参考价值。

2.4.3 功耗采集

即使采用了最好的功耗模型，选取了最细的仿真粒度，仿真得到的结果有可能依旧和实际情况相去甚远。而且功耗仿真一般在设计阶段，设计者拥有较多的关于电路的参数和信息。而攻击者几乎不掌握设备的基本信息，也就无从对电路建立功耗模型，当然也就无从对电路的功耗进行仿真。所

以，对于攻击者而言，最切实可行的做法就是直接在设备运行时采集到实际的功耗。而若要采集设备的功耗，就必须配置好相关的仪器，因此我们下面介绍一下采集电路功耗要做的准备工作。

下面列举一些采集功耗所需的基本的仪器和装置：[13]

- **被攻击的密码设备**：研究人员需要将密码设备中输入输出的接口连接到对应的采集设备和分析装置上。
- **电源与适配器**：不同的密码设备所需的供电电压与工作条件不尽相同，研究人员需要根据具体的设备提供合适的电源和适配器，以保证密码设备的正常工作，从而得到正常的有效的功耗。
- **时钟信号发生装置**：要让密码设备稳定的工作，就需要收到稳定的时钟信号，因此良好的时钟信号发生装置必不可少。如果有特殊的要求，研究人员还需要自己制作特定的时钟发生电路。
- **功耗捕捉装置**：电路的功耗很难直接测得，往往体现在某些具体的信号数值上，比如供电电源线上的总电流或者测量电阻上的电压。因此，若要得到电流，就需要将测量电路接在供电装置与密码设备之间。一个比较简单的测量电路就是串接合适大小的电阻。当然，除了插入测量电路之外，也可以使用电磁探针，通过采集电磁信息，间接地得到功耗数据。总之，功耗捕捉装置可以采用不同的方案，视具体的要求和成本而定，选取合适的即可。
- **示波器**：从功耗捕捉装置中得到的信号（电压、电流、磁场）需要通过示波器来采集、呈现和记录。在功耗分析攻击中，普通的数字示波器即可满足要求。不过选取合适的示波器仍然需要考虑一些具体的参数，比如示波器的采样率、带宽以及分辨率，这个同功耗捕捉装置一样，根据具体的情况选取合适的配置。
- **计算机**：计算机的作用在于控制整套功耗采集的设备，并将采集的信息储存下来，用于后续的分析和处理。因此研究人员需要能够保证密码设备、功耗捕捉装置、示波器以及计算机之间的正常通信，这个有时候是一个并不简单的任务。

功耗采集的流程一般如下：

1. 开启密码设备、示波器以及计算机，并且保证均可正常工作和有效通信；
2. 计算机向密码设备发送特定的指令使其正常工作，运行设备内部的程序；
3. 与此同时，功耗捕捉装置产生电流、电压或者磁场信号，并通过示波器呈现和采集；
4. 计算机获取密码设备的返回数据，并储存示波器采集的功耗数据；
5. 不断重复上述过程，直到采集的功耗数据量满足研究人员的需要。

2.5 本章小结

本章首先讨论了现代密码学的背景知识，指出了密码学在当今信息安全领域的重要作用，并且讨论了现代密码学中的一些基本原则，这些原则对通用的密码算法产生了重大的影响。接着阐述了对称密码学和非对称密码学的区别，并且介绍了二者在旁路攻击领域的研究状况。

然后我们讨论了密码设备的相关内容，讲解了一些基本的硬件知识，其中涉及到一些数字集成电路方面的概念。理解密码设备的组成和结构，有利于我们更好地实施旁路攻击。

接着我们讨论了旁路攻击，指出了旁路攻击是传统密码学分析的一个重要补充，开辟了另一条分析密码算法的道路。并且对旁路攻击方法进行了分类，列举了一些常见的攻击手段。

最后我们讨论了功耗分析，讲解了数字电路中功耗的基本构成，分为静态功耗和动态功耗。静态功耗来自晶体管的漏电流，动态功耗通常来自信号的翻转。在分析时通常只关注动态功耗，忽略静态功耗。我们还涉及了功耗仿真的相关内容，讲解了功耗仿真的不同粒度和层级，并且对其优劣进行了对比，指出需要根据实际情况和具体要求选择合适的仿真方法。我们还介绍了功耗采集的准备工作和所需的设备，以及功耗采集的一般流程。

第三章　ZUC 算法的软硬件实现

3.1　算法背景

祖冲之算法，又称 ZUC 算法，是我国提出的第一个国际商用标准密码算法。

2004 年，3GPP（3rd Generation Partnership Project，第三代合作伙伴计划）提出了 LTE（Long Term Evolution，长期演进），目的是保证该计划能够继续在电信行业拥有一定的话语权。在 2010 年底，3GPP 被确立为第四代（4G）移动通信标准。[15]

安全性是通信技术中一个非常重要的指标，而密码算法又是保障安全性的一个重要工具。3GPP 之前已经拥有的两个算法是 AES 和 SNOW 3G，而 ZUC 算法则是第三个被纳入标准的算法。ZUC 算法的提出和设计历经了很多挑战，因为商用密码算法的要求非常严苛，既要保证极高的安全性，又要拥有较高地运行性能，还需要在各自环境下都方便实现。中国科学院等单位克服了重重困难，最终研制成功，经由中国通信标准化协会与工信部向 3GPP 组织提交了这一算法，并且经过了行业严格的评审，最终被批准成为 LTE 中的密码算法标准，参与到实际的商业应用。[16]

总之，ZUC 算法的提出，使我国在国际商用密码领域拥有了更多的自主权，既体现了我国在密码学领域的学术能力，又是我国参与制定国际化通信标准的重要一步。因此，研究 ZUC 算法，具有很高的现实意义。

3.2　算法流程

ZUC 算法分成三层，如下：[17]

- **线性反馈移位寄存器（Linear feedback shift register，LFSR）**：LFSR 有两种模式，初始化模式和工作模式。
- **比特重组（Bit Reorgnization, BR）**：该层将寄存器中特定位置的数据进行编排然后输出。
- **非线性函数（Nonlinear Function，F）**：这一层涉及到一个 S 盒置换，而 S 盒置换是非线性的，因此也是这个算法中非常关键的部分。我们后续的功耗分析攻击，将会重点关注这一部分。

ZUC 算法的运行过程如下：[17]

1. **初始化阶段**：这一部分包括初始密钥的装载，LFSR 和寄存器的初始化，以及重复 32 轮的打乱操作（每一轮都包含比特重组、非线性函数以及以初始模式运行一次 LFSR）。
2. **工作阶段**：这一部分包括一个一次性操作（比特重组、非线性函数以及以工作模式运行一次 LFSR）以及密钥生成阶段（每一次密钥输出都包含比特重组、非线性函数、输出密钥以及以工作模式运行一次 LFSR）。

由于密码本身相对比较复杂，这里只是粗略地介绍了一下算法的基本组成部分和大致流程。我们在 3.4 节将会讨论关于算法的更多细节，另外也可以参考文献 [17]。

作为攻击者和研究人员，了解算法的全部细节和设计理由固然重要，但是更加重要的是找到算法在实际实现中可能存在的问题。设计者和攻击者考虑问题的角度是不一样的，设计者往往拥有更

好的大局观，但在细节上往往无法挖掘太深，而攻击者只需要撬动整个系统中的某一点就足以达成目标了。

因此我们将在下一节讨论对 ZUC 算法实施差分功耗攻击的方法。

图 3–1 展示了 ZUC 算法的过程。

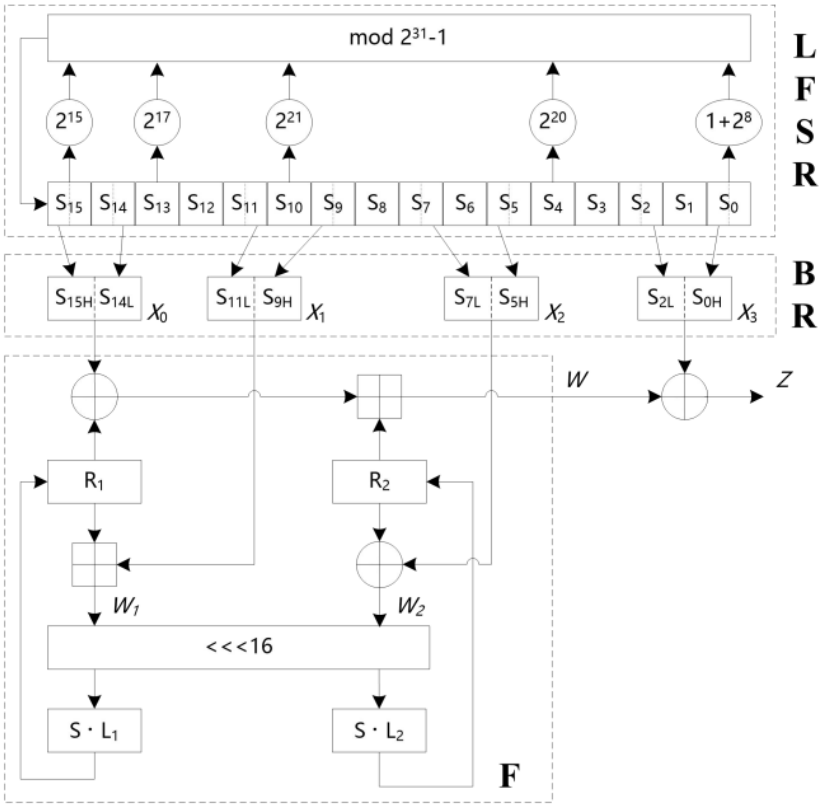


图 3–1 ZUC 算法的流程图[17]

3.3 硬件实现

硬件设计使用的软件是 ISE 14.3，使用的 FPGA 型号为：XC6SLX75-2CSG484。在完成硬件设计、仿真和综合后，在实际环境中运行 ZUC 算法，并采集功耗曲线，用于后续的分析。

这部分的代码实现和 3.4 节中所述的基本相同，但因为软件层面的代码可读性更强，因此我们将具体的实现细节留在 3.4 中详细阐述。

硬件电路的输入和输出端口如图 3–2 所示。

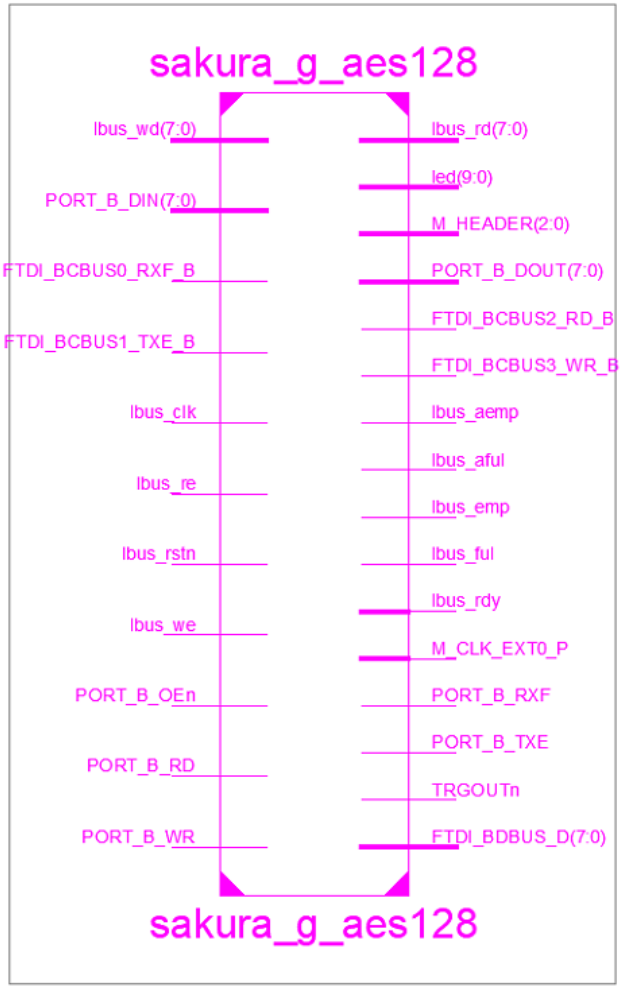


图 3–2 硬件电路的输入和输出端口

硬件电路的内部结构如图 3–3 所示。

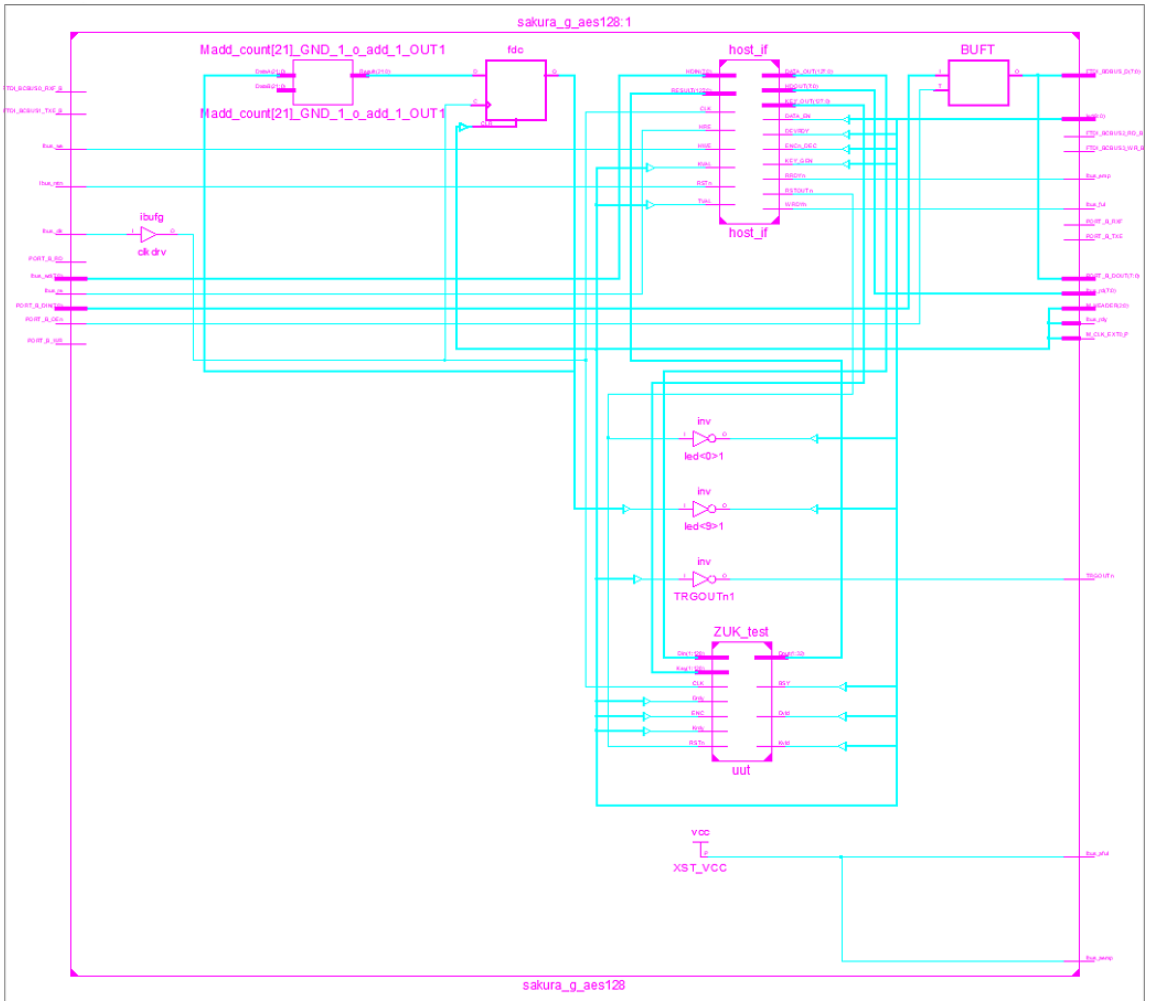


图 3–3 硬件电路的内部结构

3.4 软件实现

软件实现使用的语言为 Python 3.6，运行平台为 Windows 10。软件部分的作用是在功耗分析过程中计算中间值以及对应的假设功耗值，并进行相关系数攻击和更多的处理。软件代码开源在本人的 GitHub 仓库中：https://github.com/Hansimov/zuc-attack。

下面对软件实现的一些重要模块进行说明，作为对 3.2 节的补充。

3.4.1 符号和函数说明

为了方便讨论，这里的符号基本保持和文献 [17] 中的一致。

表 3–1 ZUC 算法中符号的说明

`s[0] ,s[1], ..., s[15]`	线性反馈移位寄存器模块的 16 个 31 比特寄存器单元变量
`x[0], x[1], x[2], x[3]`	比特重组模块输出的 4 个 32 比特字
`r1, r2`	非线性函数模块中的 2 个 32 比特记忆单元变量
`w`	非线性函数模块输出的 32 比特字
`w1`	r1 与 x[1] 进行模 2^{32} 加法运算输出的 32 比特字
`w2`	r2 与 x[2] 按比特位逐位异或运算输出的 32 比特字
`z`	算法每拍输出的 32 比特密钥字
`k`	初始种子密钥
`v`	初始向量，可以视为输入的明文
`d[0], d[1], ..., d[15]`	给定的 16 个 15 比特的字符串常量

表 3–2 ZUC 算法中函数的说明

`lsfrInit`	线性反馈移位寄存器模块的初始化模式
`lfsrWork`	线性反馈移位寄存器模块的工作模式
`nonLinearFunction`	非线性函数模块
`linearTransform`	非线性函数模块中的线性变换函数
`sboxOfZuc`	非线性函数模块中的 S 盒置换函数
`bitReorganization`	比特重组模块
`varsInit`	所有变量初始化
`zucInit`	算法运行的初始化阶段
`zucWork`	算法运行的工作阶段
`outputkey`	密钥输出

3.4.2 三层模块

3.4.2.1 线性反馈移位寄存器模块

线性反馈移位寄存器模块拥有两个模式，分别是初始化模式和工作模式，二者唯一的区别在于 `s[16]` 的赋值方式。其中用到了素域上的模 $2^{31}-1$ 加法，需要具备一些抽象代数的基础知识。这部分的主要操作就是不断打乱和混淆寄存器中存储的变量。

该模块中的素域 $GF(2^{31}-1)$ 上的本原序列可以视为二元域 $GF(2)$ 上的非线性序列，具备一系列优点，比如线性复杂度高、随机性强、权位序列平移等价以及可以一定程度上抵抗现有的基于二元域的密码分析（区分攻击、相关攻击和代数攻击）。[18]

代码 3–1 线性反馈移位寄存器的初始化模式和工作模式

```
def lfsrInit():
    global k_hex, k, v_hex, v, d, s, w
    shift_bits_list  = [15, 17, 21, 20, 8, 0]
    shift_index_list = [15, 13, 10, 4, 0, 0]
    xv = [0] * 31
    for i in range(0, len(shift_bits_list)):
        s_i_shifted = circShiftLeft(s[shift_index_list[i]], shift_bits_list[i])
        xv = modAdd_2e31m1(xv, s_i_shifted)
    s[16] = modAdd_2e31m1(shiftLeft(w, -1), xv) # The only diffrence of lfsrWork() and
       lfsrInit()
    if s[16] == [0]*31:
        s[16] = [1]*31
    for i in range(0,16):
        s[i] = s[i+1]

def lfsrWork():
    global k_hex, k, v_hex, v, d, s
    shift_bits_list =  [15, 17, 21, 20, 8, 0]
    shift_index_list = [15, 13, 10, 4, 0, 0]
    xv = [0] * 31
    for i in range(0, len(shift_bits_list)):
        s_i_shifted = circShiftLeft(s[shift_index_list[i]], shift_bits_list[i])
        xv = modAdd_2e31m1(xv, s_i_shifted)
    s[16] = xv # The only diffrence of lfsrWork() and lfsrInit()
    if s[16] == [0]*31:
        s[16] = [1]*31
    for i in range(0,16):
        s[i] = s[i+1]
```

3.4.2.2 比特重组模块

比特重组模块的实现最为简单，其操作基本就是字符串的剪切和粘贴，建立起了上一层线性反馈移位寄存器和下一层非线性函数之间的桥梁。

简单的设计使这一模块便于在软硬件上实现，其作用是改变线性反馈移位寄存器部分的线性结构，并且加强对素域 $GF(2^{31}-1)$ 上密码分析的防御能力。[18]

代码 3–2 比特重组

```
def bitReorganization():
    global x, s
    x[0] = s[15][0:16] + s[14][-16:]
    x[1] = s[11][-16:] + s[9][0:16]
    x[2] = s[7][-16:] + s[5][0:16]
    x[3] = s[2][-16:] + s[0][0:16]
```

3.4.2.3 非线性函数模块

非线性函数模块的实现相对复杂一些，其中的 S 盒置换函数和分组密码的设计思想很相似。其中的操作较多，包括比特加、比特异或、模 2^{32} 加、字符串移位以及 S 盒置换。

这一模块中的复杂运算和操作，从根本上改变了源序列在素域 $GF(2^{31}-1)$ 上的线性代数结构，而精心设计的 S 盒则具备很强的非线性性和扩散性。因此这一模块使整个算法的安全性得到了极大地提升。[18]

代码 3–3 非线性函数

```
def nonLinearFunction():
    global w, x, r1, r2
    w = binaryAdd(binaryXor(x[0], r1), r2)
    w1 = binaryAdd(r1, x[1])
    w2 = binaryXor(r2, x[2])
    r1 = sboxOfZuc(linearTransform(w1[-16:]+w2[0:16], 1))
    r2 = sboxOfZuc(linearTransform(w2[-16:]+w1[0:16], 2))
```

非线性函数模块中有两个重要的函数：`linearTransform` 和 `sboxOfZuc`。

线性变换函数 `linearTransform` 主要是将字符串的不同部分进行移位和比特异或。

代码 3–4 线性变换函数

```
def linearTransform(list_in, typex):
    if typex == 1:
        shift_bits_list = [0, 2, 10, 18, 24]
    else:
        shift_bits_list = [0, 8, 14, 22, 30]

    list_out = [0] * len(list_in)
    for i in range(0, len(shift_bits_list)):
        list_out = binaryXor(list_out, circShiftLeft(list_in, shift_bits_list[i]))

    return list_out
```

S 盒置换函数 `sboxOfZuc` 主要是提供非线性性，将每个输入的字节映射到另一个截然不同的字节，其核心是两张 S 盒置换表。限于篇幅，这里没有将 S 盒中的所有元素都写出来。

代码 3–5 S 盒置换函数

```
def sboxOfZuc(list_in):
    sbox = [[]] * 4
    sbox[0] = [
        ['3E', '72', '5B', '47', 'CA', 'E0', '00', '33', '04', 'D1', '54', '98', '09', 'B9', '6D', 'CB'],
        ...
        ]; # 16x16 list
    sbox[1] = [
        ['55', 'C2', '63', '71', '3B', 'C8', '47', '86', '9F', '3C', 'DA', '5B', '29', 'AA', 'FD', '77'],
        ...
        ];  # 16x16 list
    sbox[2] = sbox[0]
    sbox[3] = sbox[1]
    list_out = [0] * 32

    for i in range(0, 4):
        list_tmp = list_in[8*i:8*(i+1)]
        hex_tmp = binvec2hex(list_tmp)
        row_tmp, col_tmp = hex2dec(hex_tmp[0]), hex2dec(hex_tmp[1])
        hex_tmp_s = sbox[i][row_tmp][col_tmp]
        list_out[8*i:8*(i+1)] = hex2binvec(hex_tmp_s)
    return list_out
```

3.4.3 运行过程

3.4.3.1 变量初始化

变量初始化部分就是给算法中要用到的变量赋初值，其中就包括将初始密钥装入寄存器的操作。

代码 3–6 变量初始化

```
k_hex = [''] * 16
v_hex = [''] * 16
k = [''] * 16 # initial key - 8 bit x 16
v = [''] * 16 # initial vector - 8 bit x 16
d = [''] * 16 # constant data - 15 bit x 16
s = [[0]*31 for _ in range(17)] # combined bit list - 31 bit x 16+1 (The last is used to update)
r1 = [0] * 32 # register 1 - 32 bit
r2 = [0] * 32 # register 2 - 32 bit
x = [[0]*32 for _ in range(4)] # list - 32 bit x 4
w = [0] * 32 # var in nonLinearFucntion(), used by zucInit()
d = [ '100010011010111', '010011010111100', '110001001101011', '001001101011110',
      '101011110001001', '011010111100010', '111000100110101', '000100110101111',
      '100110101111000', '010111100010011', '110101111000100', '001101011110001',
      '101111000100110', '011110001001101', '111100010011010', '100011110101100']

def varsInit():
    global k_hex, k, v_hex, v, d, s, r1, r2
    k_hex = ['00','00','00','00','00','00','00','00','00','00','00','00','00','00','00','00']
```

```
    for i in range(0, 16):
        k[i] = hex2binstr(k_hex[i])

    v_hex = ['00','00','00','00','00','00','00','00','00','00','00','00','00','00','00','00']
    for i in range(0, 16):
        v[i] = hex2binstr(v_hex[i])

    for i in range(0, 16):
        s[i] = k[i] + d[i] + v[i]
        s[i] = list(map(int,s[i]))
    r1 = [0] * 32
    r2 = [0] * 32
```

3.4.3.2 算法运行

算法运行分为两个阶段，分为初始化阶段和工作阶段。唯一的不同就是，初始化阶段执行的次数是固定的 32 轮，而工作阶段的执行次数取决于要输出多少次密钥。

算法运行过程中，需要不断地执行上一小节中提及的三个模块。

代码 3–7 算法初始化阶段和工作阶段

```
def zucInit():
    global k_hex, k, v_hex, v, d, s, r1, r2, x, w
    for i in range(0,32):
        bitReorganization()
        nonLinearFunction()
        lfsrInit()

def zucWork(i=0):
    global k_hex, k, v_hex, v, d, s, r1, r2, x, w
    bitReorganization()
    nonLinearFunction()
    key_bin = binaryXor(w, x[3])
    key_hex = binvec2hex(key_bin)
    if i != 0:
        print('Key {:>02} {}'.format(i, key_hex.lower()))
    lfsrWork()
```

3.4.3.3 密钥输出

这一部分非常简单，其核心就是进入整个算法的工作阶段，然后不断循环，输出密钥。

代码 3–8 算法初始化阶段和工作阶段

```
def outputKey(num=1):
    global k_hex, k, v_hex, v, d, s, r1, r2, x, w
    for i in range(0, num+1):
        zucWork(i)
```

3.4.3.4 运行结果

运行如下代码，则会输出 3 次密钥。虽然工作阶段实际运行了 4 次，但是算法规定第 0 次输出的密钥要舍弃。

代码 3–9 完整的算法执行过程

```
if __name__ == '__main__':
    varsInit()
    zucInit()
    outputKey(4)
```

上述代码中，种子密钥和初始向量都是全 0 序列。

算法运行产生的输出如下：

代码 3–10 算法运行的输出结果

```
Key 01 27bede74
Key 02 018082da
Key 03 87d4e5b6
Key 04 9f18bf66
```

3.5 本章小结

本章首先讨论了 ZUC 算法的提出背景，介绍了其和通信行业发展的密切联系，并且指出了该算法对我国的重要意义。

然后我们讨论了 ZUC 算法的流程和结构，分别介绍了 ZUC 算法的三个部分，包括线性移位反馈寄存器、比特重组和非线性函数，并且讲解了初始化和工作的运行过程。

接着我们展示了 ZUC 算法的硬件实现情况和所用设备，给出了硬件电路的原理图。

最后我们详细讲解了算法的软件实现，分析了各个模块的操作和作用，这是我们后续对算法进行攻击的基础工作。

第四章　对 ZUC 算法的功耗分析方案

4.1　差分功耗分析的一般步骤

功耗分析通常包含简单功耗分析（Simple Power Analysis）和差分功耗分析（Differential Power Analysis）。

简单功耗分析通常只需要少量的功耗曲线，就能揭示密码设备中的有用信息。简单功耗分析通常适用于功耗曲线特征较为明显的密码设备，比如出现明显的波峰和波谷，以及呈现出多个周期性的重复段落。如果攻击者对密码设备中运行的程序有一定的预备知识，那么就能推测出功耗曲线的不同段落在执行何种操作，就有可能进一步掌握设备的更多信息。

差分功耗分析则需要大量的功耗迹。大量功耗数据带来的好处就是更强大的分析和攻击能力，也不需要对设备的构造和执行的程序有详细的了解，一般情况下，只要掌握设备运行的算法流程就足够实施分析和攻击了。

因此，我们的关注重点就放在差分功耗分析上。

下面我们来介绍一下差分功耗分析的一般流程：[13]

1. **选取合适的算法中间值位置：**一个好的中间值，应该尽可能地区错误的猜测和正确的猜测。因此，在密码算法中，通常选择非线性函数的输出作为差分功耗分析的中间值。由于在运行算法和采集功耗时，攻击者往往只能获得明文或者密文，因此通常只能对算法的第一轮加密或者最后一轮加密进行攻击。因此，选取的中间值最好能够出现在第一轮或者的最后一轮。选取不同的中间值，会对攻击效果产生很大的影响，因此需要根据不同的算法和具体的实验条件，选取最合适的算法中间值。
2. **采集设备运行时的实际功耗曲线：**这一部分没有什么技术难度，不过值得一提的是，如果合理地选择功耗曲线采集和结束的位置，就能得到对齐较好的曲线，方便后续的分析和处理。采集环境也要尽可能地排除外界因素的干扰，以提高功耗曲线同数据和操作的相关性，增大信号的信噪比。更多具体的细节已经在上一小节阐述了。
3. **根据算法计算理论中间值：**对某个具体的密码算法而言，密钥通常是最重要也是最机密的信息，攻击者唯一无法知晓的也是这一部分。对全部位数的密钥进行穷举猜测是不可能做到的，因此攻击者常常需要在选择合适的中间值的前提下，尽可能地降低中间值和全部密钥之间的相关性。或者说，攻击者应该尽可能选取只依赖少部分密钥的中间值，这样就能大大减少猜测的可能情况，提高攻击的效率。由于密码算法通常是公开透明的，因此已知明文和猜测密钥的情况下，是可以计算出适合的理论中间值的。
4. **使用合适的功耗模型将理论中间值转换为假设功耗值：**算法的中间值通常是某个字节或者比特，和算法有关。由于中间值的值域很大，因此对所有可能的中间值建立一个具体的模型是不现实的。所以有必要采用合适的功耗模型，缩小猜测空间，将中间值转换成假设功耗值。常用的功耗模型包括汉明重量模型、汉明距离模型以及零值模型。功耗模型之间各有利弊，需要根据实际的实验情况和攻击效果选取最合适的模型。

5. **分析假设功耗值和实际功耗曲线，挖掘所需的信息：**这部分通常涉及到一定的统计学知识，需要攻击者具备较好的数学基础。在分析曲线的特征之前，通常还要对功耗曲线进行预处理，比如对齐和滤波，减少噪声，提高信噪比，从而能够更好地利用功耗曲线中的有效信息。此外，高效地处理大量的数据也是一个需要仔细考量的问题，差分功耗分析往往会采集成千上万甚至是百万条曲线，如何编写性能优异的算法，或者是并行化处理，都会很大程度上影响分析的速度和效果。除了常用的相关系数攻击之外，模板攻击也很有效。攻击者应该尽可能地设计好的算法，从而减少所需的功耗曲线条数，这样就能大大减少攻击的时间和成本。

图 4–1 展示了差分功耗分析攻击的第 3 – 5 步。

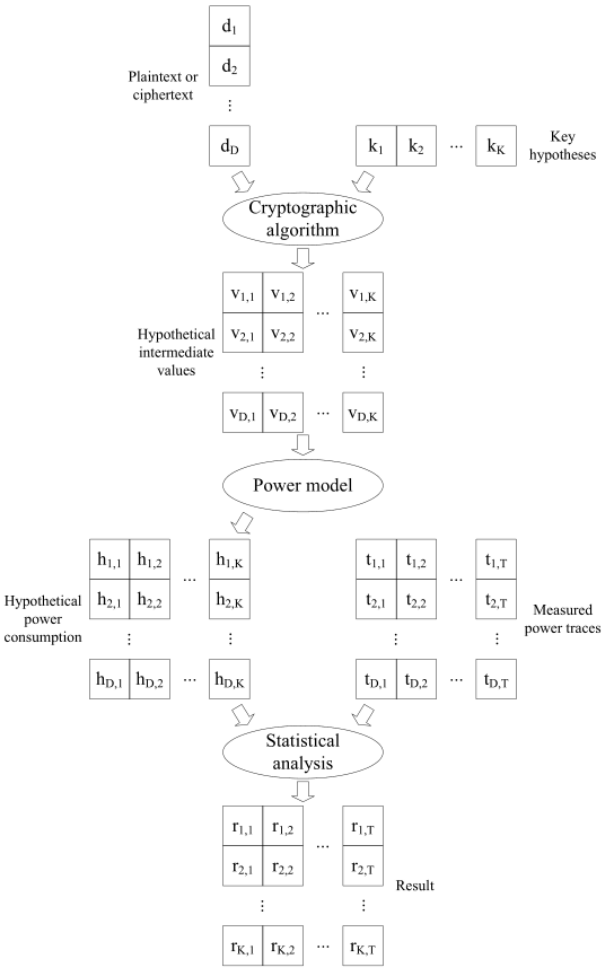


图 4–1 差分功耗分析的典型流程[19]

4.2 寻找中间值

在进行软件分析之前，我们首先要在硬件上实现 ZUC 算法电路，并采集其运行时的功耗。这部分相对简单，我们已经在 3.3 节中实现了硬件电路，而采集功耗的过程也很容易，因此这里不再赘述。

在 ZUC 算法中，唯一未知的信息就是初始的种子密钥，其他的常量和明文都是已知的。因此我们的攻击目的就是得到密钥的信息，也就是种子密钥的各个字节。

差分功耗攻击最核心的思想是，假设功耗值和实际功耗值之间是有关联的。而要想假设功耗值尽可能贴合实际功耗值，就需要选择合适的中间值。

一般而言，中间值通常选择算法中非线性变换的部分，因为如果输入稍有不同，非线性变换的输出就会出现较大的差异，从而正确的输入和错误的输入产生的差异将比线性变换更加明显，就可以有效地区分出正确的输入和错误的输入。

因此，我们第一步想到的是，把目光放在 ZUC 算法的非线性函数模块，看看能否从这个模块的附近找到合适的中间值。

由于 ZUC 算法是分成三层的，第三层的非线性函数模块的输入，取决于第一层的线性反馈移位寄存器模块和第二层的比特重组模块的输出，而上面两层都对算法中的变量作了很多操作和运算，比如移位、剪切、粘贴、比特加、比特异或、素域模加，这就导致不同密钥字节彼此间的关联度很高。

密钥字节关联度很高的后果就是，我们很难对单个密钥字节进行猜测（只需要穷举 2^8 种情况），这就导致我们必须穷举更多的情况（N 个密钥字节就对应 $2^{8\times N}$ 种情况），一旦关联的密钥字节数较多，就几乎不可能对其穷举。

所以我们的首要任务，就是找到算法中尽可能独立的密钥字节。换句话说，如果在某个时刻，非线性函数模块的输出只和单个密钥字节相关，那么这个密钥字节就具有极强的独立性，我们也只需要穷举 2^8 种情况就能攻出这个密钥字节。

我们需要对算法进行自顶向下地剖析，观察不同阶段哪些密钥字节参与了什么运算。

我们从图 3–1 中截取一部分，得到图 4–2，该图展示了线性反馈移位寄存器模块的输出、比特重组模块的全部以及非线性函数模块的输入。

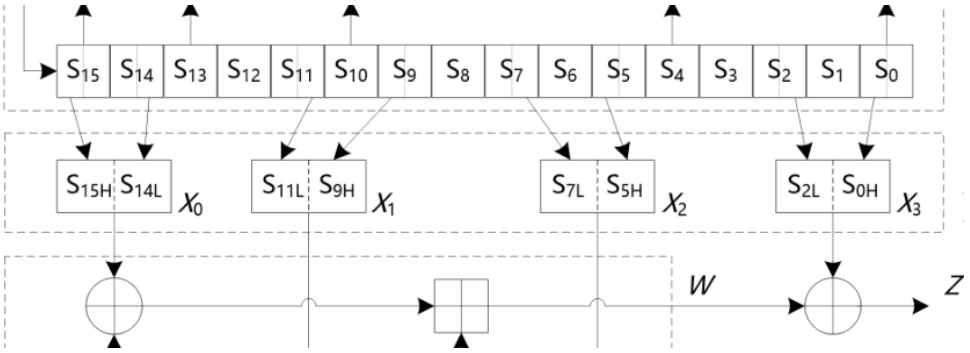


图 4–2 线性反馈移位寄存器模块的输出、比特重组模块的全部以及非线性函数模块的输入

从图中可以看出，并不是任何时刻所有种子密钥字节都参与运算，以初始化阶段为例，zucInit 的每一轮中，线性反馈移位寄存器的输出只和 8 个寄存器单元相关。线性反馈移位寄存器输出的便是比特重组的输入，见代码 3–2，我们这里将其再次写出来，方便参阅：

```
def bitReorganization():
    global x, s
    x[0] = s[15][0:16] + s[14][-16:]
    x[1] = s[11][-16:] + s[9][0:16]
    x[2] = s[7][-16:] + s[5][0:16]
    x[3] = s[2][-16:] + s[0][0:16]
```

那么这里的寄存器单元 s[i] 又和什么有关呢？在变量初始化的代码 3–6 中，我们可以找到如下语句：

```
...

def varsInit():
    ...
    for i in range(0, 16):
        s[i] = k[i] + d[i] + v[i]
    ...
```

也就是说，在 ZUC 算法中，线性反馈移位寄存器的每个 31 比特单元变量 s[i] 是由种子密钥字节 k[i]、给定的字符串常量单元 d[i] 以及初始向量（输入的明文）字节 v[i] 连接而成的。（可以参考表 3–1 中各符号的说明。）

那么我们就可以对输入比特重组模块的 8 个字节进行拆解，找到和其相关的密钥字节：

```
    x[0] = s[15][0:16] + s[14][-16:]
         = (k[15][:] + d[15][0:8]) + (d[14][-8:] + v[14][:])

    x[1] = s[11][-16:] + s[9][0:16]
         = (d[11][-8:] + v[11][:]) + (k[9][:] + d[9][0:8])

    x[2] = s[7][-16:] + s[5][0:16]
         = (d[7][-8:] + v[7][:]) + (k[5][:] + d[5][0:8])

    x[3] = s[2][-16:] + s[0][0:16]
         = (d[2][-8:] + v[2][:]) + (k[0][:] + d[0][0:8])
```

这里提一句，在 Python 中，s[15][0:16] 表示 s[15] 的前 16 比特，也就是第 0 到 15 比特，而不是第 0 到 16 比特，相当于索引的区间是左闭右开。s[14][-16:] 则表示 s[14] 的末 16 比特，k[15][:] 表示 k[15] 的全部比特（8 位）。

从拆解结果可以看到，相关的密钥字节有且仅有：k[15], k[9], k[5], k[0]。

那么，在非线性函数模块中，是不是这 4 个又都参与运算了呢？并不是。

我们再截取图 3–1 的另外一部分，得到图 4–3，稍作观察。

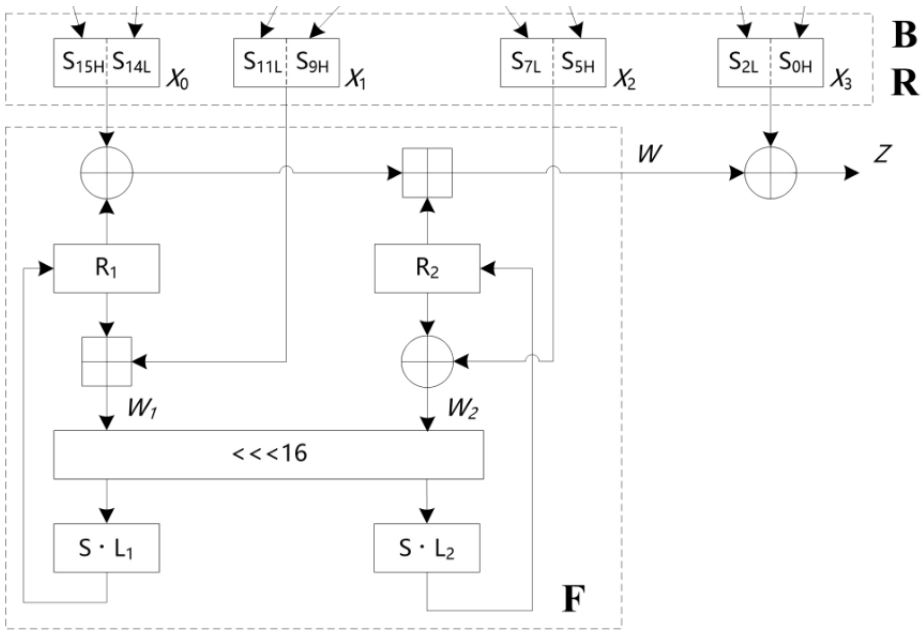


图 4–3 比特重组模块和非线性函数模块

从图中的数据流向可以看出，`x[0]` 和 `x[3]` 并没有参与非线性函数模块的内部操作，而只是和记忆单元 `r1` 和 `r2` 作了简单的运算，用于输出密钥流。

因此，真正参与非线性函数模块的内部操作的，只有 `x[1]` 和 `x[2]`，与之相关的密钥字节便是 `k[9]` 和 `k[5]`。

下面我们再来看一下 `k[9]` 和 `k[5]` 又是如何参与非线性函数中的运算的。

这里我们将非线性函数模块的代码 3–3 重写一遍，方便查看：

```
1  def nonLinearFunction():
2      global w, x, r1, r2
3      w = binaryAdd(binaryXor(x[0], r1), r2)
4      w1 = binaryAdd(r1, x[1])
5      w2 = binaryXor(r2, x[2])
6      r1 = sboxOfZuc(linearTransform(w1[-16:]+w2[0:16], 1))
7      r2 = sboxOfZuc(linearTransform(w2[-16:]+w1[0:16], 2))
```

从第 6 – 7 行可以看出，`r1` 和 `r2` 的值与 `w1` 和 `w2` 均相关，而 `w1` 与 `x[1]` 相关，`w2` 与 `x[2]` 相关。

让我们将几个变量做一下拆解，以便更好地理清它们之间的关系：
（我们用 $a \Leftarrow b$ 表示 a 取决于 b）

```
r1 ⇐ w1[−16:] + w2[0:16]
   ⇐ x[1][−16:] + x[2][0:16]
   ⇐ (k[9][:] + d[9][0:8]) + (d[7][−8:] + v[7][:])

r2 ⇐ w2[−16:] + w1[0:16]
   ⇐ x[2][−16:] + x[1][0:16]
   ⇐ (k[5][:] + d[5][0:8]) + (d[11][−8:] + v[11][:])
```

我们惊喜地发现：**r1 仅和密钥字节 k[9] 相关，而 r2 仅和密钥字节 k[5] 相关！**
图 4–4 展示了这一重要的关系。

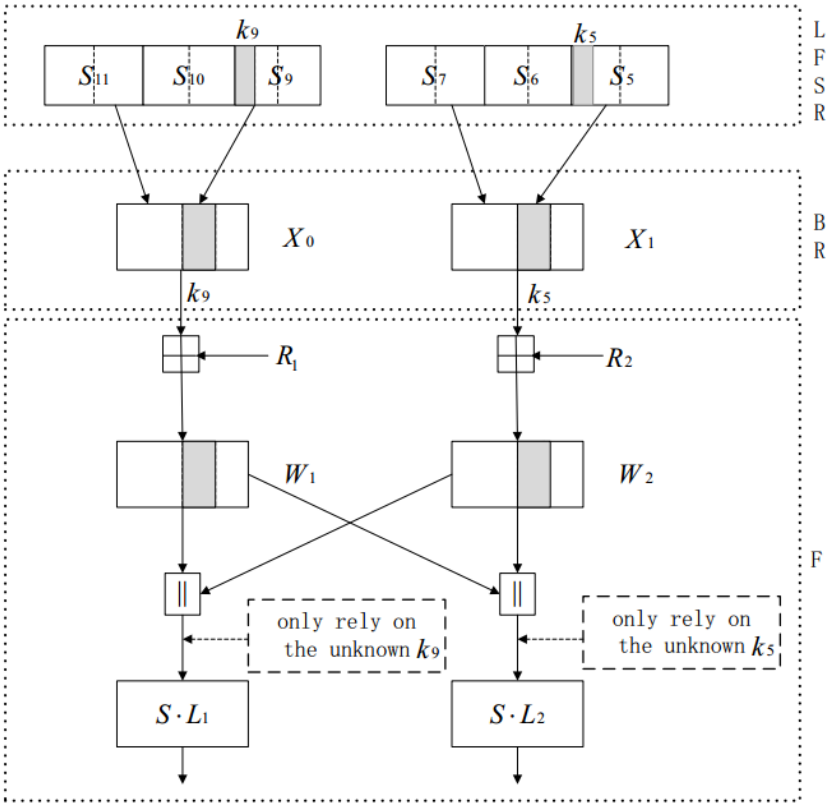


图 4–4 在 zucInit 第 1 轮中，r1 仅和密钥字节 k[9] 相关，而 r2 仅和密钥字节 k[5] 相关[1]

也就是说，如果将 r1 或者 r2 的输出作为中间值的话，我们就只需要猜测单个字节，也就是 2^8 种情况，所花的代价非常小，可行性也非常高。

4.3 构建密钥推算链条

然而，我们在上一节中的方案遗漏了两个细节。

第一个遗漏的细节是，在非线性函数模块的代码的第 4 – 5 行中，`w1` 不仅取决于 `x[1]`，还取决于 `r1`，同样地，`w2` 不仅取决于 `x[2]`，还取决于 `r2`。而 `r1` 和 `r2` 又是和上一轮非线性函数的结果相关的。

这里就出现了一个问题：如果 `r1` 和 `r2` 和上一轮非线性函数的结果相关，而要想知道上一轮非线性函数的结果，又需要知道上一轮中相关密钥字节，可是密钥字节正是我们未知的信息。似乎陷入了一个死循环。

经过思索，我们终于找到了问题的突破口：在第 1 轮算法初始化阶段 `zucInit` 的非线性函数模块 `nonLinearFunction` 中，`r1` 和 `r2` 的初始值是全 0。

可以参考变量初始化的代码 3–6 的这一段：

```
...

def varsInit():
    ...
    r1 = [0] * 32
    r2 = [0] * 32
```

这样，我们就可以将全 0 作为 `r1` 和 `r2` 的初始值，代入到第 1 轮 `zucInit` 的 `nonLinear-Function` 中计算中间值。

换句话说，在 `zucInit` 的第 1 轮中，除了密钥字节，其他的变量值都是已知的。这样，我们就可以计算出中间值，进行功耗分析，攻击出对应密钥字节，再将攻出来的字节代入到算法中，得到下一轮的相关值，这样就可以攻击出后续的密钥字节。

然而事实并没有我们想象的这么简单，事实上，以 `r1` 和 `r2` 作为中间值，只能攻击 `zucInit` 的前 5 轮。也就是说，通过这种方法可以攻击出来的密钥字节为：`k[9], k[5]; k[10], k[6]; k[11], k[7]; k[12], k[8]; k[13](, k[9])`。（第 5 轮中攻出的 `k[9]` 已经在第 1 轮攻出。）

为什么只能攻击前 5 轮呢？又为什么是这几个密钥字节呢？

这也就引出了第二个遗漏的细节：在 `zucInit` 的每一轮中，线性反馈移位寄存器的每个寄存器单元都会向右平移一个单元。参见代码 3–1 中的这一段：

```
def lfsrInit():
    ...
    for i in range(0,16):
    s[i] = s[i+1]

def lfsrWork():
    ...
    for i in range(0,16):
    s[i] = s[i+1]
```

那么，我们之前所谈论的 `k[9]` 和 `k[5]` 只是线性反馈移位寄存器中对应位置（第 9 个记忆单元和第 5 个记忆单元）的密钥字节，而不是原始的 `k[9]` 和 `k[5]`，只有在 `zucInit` 的第 1 轮 `nonLinearFunction` 中，才是真正的 `k[9]` 和 `k[5]`。

事实上，在 `zucInit` 的第 N 轮 `nonLinearFunction` 中，和 `r1` 相关的应当是原始密钥字节中的 `k[8+N]`，和 `r2` 相关的应当是原始密钥字节中的 `k[4+N]`。这也就是我们之前所列的，前 5 轮中可以攻出来的密钥字节的递推公式。

这里我们就碰到了关键的问题，到第 6 轮时，第 1 轮中的 $\texttt{s[16]}_1$ 已经移动到了这里的 $\texttt{s[11]}_6$ 处。（为了便于之后区分初始化阶段不同轮中线性反馈移位寄存器的单元变量，我们用$\texttt{s[i]}_n$ 表示第 n 轮中线性反馈移位寄存器第 i 个记忆单元变量的值，事实上，$\texttt{s[16]}_j = \texttt{s[11]}_{j+5}$。）

此时，如果想得到 `r1` 的值，就必须知道 `x[1]` 的值，而 `x[1]` 的值是取决于 $\texttt{s[11]}_1$ 的，见代码 3–2 的这一段：

```
def bitReorganization():
    ...
    x[1] = s[11][-16:] + s[9][0:16]
    ...
```

问题的根源就在于此：第 6 轮中的 $\texttt{s[11]}_6$ 对应的是第 1 轮的 $\texttt{s[16]}_1$，而 $\texttt{s[16]}_1$ 的计算过程牵扯到几个其他密钥字节，参见代码 3–1 的这一段：

```
def lfsrInit():
    global k_hex, k, v_hex, v, d, s, w
    shift_bits_list  = [15, 17, 21, 20, 8, 0]
    shift_index_list = [15, 13, 10, 4, 0, 0]
    xv = [0] * 31
    for i in range(0, len(shift_bits_list)):
        s_i_shifted = circShiftLeft(s[shift_index_list[i]], shift_bits_list[i])
        xv = modAdd_2e31m1(xv, s_i_shifted)
    s[16] = modAdd_2e31m1(shiftLeft(w, -1), xv)
    if s[16] == [0]*31:
        s[16] = [1]*31
    ...
```

因此，从 `zucInit` 的第 6 轮开始，就不能再简单地用 `r1` 和 `r2` 作为中间值了。因为相关的密钥字节不再只有单个，而是多个，所需要猜测的可能大大增加。

事实上，在 `zucInit` 的第 6 轮中，需要同时穷举 4 个密钥字节：`k[15]`, `k[14]`, `k[4]`, `k[0]`。也就是说，需要猜测 2^{32} 种情况。

我们解释一下为什么第 6 轮需要同时穷举这 4 个密钥字节。我们在上文指出，`zucInit` 第 6 轮中要用到的 $s[11]_6$ 对应的第 1 轮中的 $s[16]_1$，而第 1 轮中的 $s[16]_2$ 和 `k[15], k[13], k[10], k[4], k[0]` 相关。可以参考上面线性反馈移位寄存器初始化模式代码的第 4 行：

```
1  def lfsrInit():
2      ...
3      shift_bits_list  = [15, 17, 21, 20, 8, 0]
4      shift_index_list = [15, 13, 10, 4, 0, 0]
5      ...
```

`shift_index_list` 列出了需要移位的字节，`shift_bits_list` 列出了对应位置的字节需要移多少位，并且相关的字节在移位后还要参与后续的素域模 $2^{31} - 1$ 加运算，因此无法单独攻出各个密钥字节，必须同时穷举所有的未知字节。

在第 1 轮中，这里的移位的 $s[i]_1$ 和 `k[i]` 相关。事实上，在第 N 轮中，发生移位的 $s[i]_N$ 和 `k[i+N-1]` 相关。

因此，从 `shift_index_list` 就能够得知和第 1 轮中 $s[16]_1$ 相关的密钥字节是：`k[15], k[13], k[10], k[4], k[0]`。不过由于 `k[10]` 可以在第 2 轮中攻出，`k[13]` 可以在第 5 轮中攻出，因此实际上未知的只有 `k[15], k[4], k[0]`。

又由于第 6 轮中，和 `r1` 相关的密钥字节是 `k[14]`（未知），和 `r2` 相关的密钥字节是 `k[10]`（已攻出），故需要猜测的密钥字节还要再加上 `k[14]`。

因此，第 6 轮中需要同时对 `k[15], k[14], k[4], k[0]` 这 4 个密钥字节进行穷举，共计 2^{32} 种情况。

此时，截至第 6 轮结束，我们可以攻出的字节有：`k[9], k[5]; k[10], k[6]; k[11], k[7]; k[12], k[8]; k[13]; k[15], k[14], k[4], k[0]`。剩下的未知的密钥字节只有：`k[1], k[2], k[3]`。

类似第 6 轮中的思路，在第 7 轮中的 $s[11]_7$ 对应第 2 轮输出的 $s[16]_2$。我们在上文提过，在第 N 轮中，发生移位的 $s[i]_N$ 和 `k[i+N-1]` 相关，因此 $s[16]_2$ 又依据 `shift_index_list`，可以得知和 $s[16]_2$ 相关的密钥字节为 `k[14], k[11], k[5], k[1]`，这里仅有 `k[1]` 未知。又第 7 轮中的 `r1` 和密钥字节 `k[11]` 相关，`r2` 和密钥字节 `k[15]` 相关，而 `k[11]` 和 `k[11]` 都是已知的。因此只需要猜测 `k[1]`，也就是穷举 2^8 种情况。

仿照上面的流程，我们可以在第 8 轮中攻出 `k[2]`，在第 9 轮中攻出 `k[3]`。

至此，我们就将所有的密钥字节攻击出来了，所需要猜测的情况总数为：$2^8 \times (9 + 3) + 2^{32}$。

4.4 计算假设功耗值

选好了中间值的位置，我们就可以计算出所有密钥猜测（从 0 到 255，也即 16 进制的 00 到 FF）对应的中间值。

然后我们要选择合适的功耗模型，通过中间值计算出理论功耗值（或者称作假设功耗值）。这一步骤的目的是将中间值映射到一个更小的空间，便于区分不同特征的曲线。

我们这里选择汉明重量作为功耗模型。一个仅由 0 和 1 构成的向量的汉明重量，等于该向量中 1 的个数。

因此，给定一个中间值向量，就能得到一个对应的汉明重量，也就是其假设功耗值。

汉明重量模型有效的前提假设是：设备产生的功耗，与其操作的数据中 1 的个数是相关的。

选择汉明重量模型的原因是，计算相对简单，而且其有效性假设在现实中一般都能成立。

4.5 计算相对相关系数

在采集到实际功耗曲线，并通过中间值算出假设功耗值后，我们就可以计算假设功耗值矩阵和实际功耗值矩阵之间的相关系数了。

相关系数表征了假设功耗值和实际功耗值之间的关联程度，相关系数越高，二者之间的关联度就越高。

计算相关系数的公式如下：

$$r = \frac{\sum_{i=1}^{n}(X_i - \overline{X})(Y_i - \overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \overline{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \overline{Y})^2}}$$

其中，r 表示样本相关系数，X_i 表示假设功耗值的样本点，Y_i 表示实际功耗值的样本点，n 表示样本点数。

我们对相关系数矩阵稍作处理，得到“相对相关系数”，目的是为了更好地挖掘有效信息。处理过程如下：

先对相关系数矩阵取绝对值，这是因为不论相关系数是正是负，只要绝对值够大，就表明有显著的相关性。再取每一行中最大的前 5 个值，计算平均值，即可得到“相对相关系数”。这是因为实际功耗曲线上只有极少数的时间点有功耗信息的泄露，我们只需要泄露最明显的那一些点，如果让很多不相关的时刻上的相关系数也参与运算，就会降低信噪比，掩盖掉有用的信息。

最后我们比较不同密钥猜测的相对相关系数，值最大的即对应最优的密钥猜测。

4.6 实现完整的差分功耗分析

我们在前面详细讲解了攻击的方案，这里我们为了突出功耗分析的重要思想，选取了一个易于理解的典型，仅以 `zucInit` 第 1 轮中 `r2` 的输出作为中间值，来攻击 `k[5]`。

将上面的方案串联起来，就能得到完整的功耗分析流程，代码 4–1 是这一流程的软件实现。

代码 4–1 差分功耗分析攻击的完整流程

```
# Read traces
    sample_mat = [[]]*trace_num
    for i in range(0, trace_num):
        sample_mat[i] = getTraceSample(trs_file, header_end, i, sample_num, offset)
    hw_mat = [[0]*trace_num for _ in range(256)]

# Calculate inter values and hammingweight
    for i in range(0, trace_num):
        for k in range(0, 256):
            inter_val = calcInterValue(i, k)
            hw_mat[k][i] = calcHammingWeight(inter_val)

# Calculate relative correlation matrix of guessing key
    sample_mat = np.array(sample_mat)
    hw_mat = np.array(hw_mat)
    for n in [100, 200, 500, 1000, 2000, 5000, 10000, 20000]:
        corr_maxn = [[0]*5 for _ in range(256)]
        for k in range(0,256):
            for j in range(0, sample_num):
                corr_mat[k][j] = abs(np.corrcoef(hw_mat[k][0:n], sample_mat[0:n, j])[0][1])
            corr_maxn[k] = heapq.nlargest(len(corr_maxn[0]), corr_mat[k])
              corr_avg[k] = np.mean(corr_maxn[k])

          corr_max = max(corr_avg)
          key_max = corr_avg.index(corr_max)
```

由于篇幅限制，这里略去了很多细节，只展示了算法中最核心的部分。

我们将在下一章给出实验的具体结果以及详细分析。

4.7 本章小结

本章首先讨论了功耗分析中简单功耗分析和差分功耗分析的异同，并且重点介绍了差分功耗攻击的基本流程和一般方法。

其次我们试图寻找合适的中间值，对算法进行了深入的挖掘，并且发现了合适的中间值。

然后我们对算法进行了进一步的剖析，探讨了遗漏的细节，并且补全了对剩余密钥字节的攻击方案。

接着我们选取了合适的功耗模型，并进行相关系数攻击。

最后我们以一个典型的密钥字节作为攻击对象，展示了将功耗分析技术完整地用于现实场景的过程。

第五章　实验结果与分析

5.1　功耗曲线的特征

按照在 2.4.3 节所述的流程，在密码算法电路运行时，采集其功耗，并记录存储到计算机上。本次实验一共采集了 100 万条功耗曲线，实际用到的为前 2 万条。密码算法跑在普通的 FPGA 上，这里的功耗值对应的具体物理量，实际上是电路电源线上的总电流。[20][21]

由于我们的攻击针对的是 ZUC 算法的初始化阶段，因此只储存了初始化阶段的功耗点。初始化阶段一共有 32 轮，这里只记录了前 27 轮，不过这也已经足够我们使用了。

图 5–1 展示了第 1、20、200、2000 和 20000 条功耗曲线的波形。

从图中可以得到这样一些信息：

1. **时钟频率以及电路运行状况稳定。**
   图中的曲线在各个时间点处的峰值基本对齐，说明时钟频率非常稳定，并且每次运行算法所花费的时间几乎是相同的。从抽取的曲线来看，波形也几乎一致。这意味着我们可以免去对齐曲线这部分预处理的工作，降低了工作量，清晰的轮廓也从侧面反映了采集得到的功耗曲线质量较高。因此得到这样的曲线是非常令人满意的。
2. **采集到的曲线包含了密钥装载阶段和前 27 轮初始化。**
   前面一段持续的高功耗，应该是算法电路的启动，以及相关数据载入内存并交由控制器的部分，其中的一个尖峰，可能是密钥装载的阶段。之后可以看到 27 个特征相似的波形，毫无疑问，这是算法中初始化阶段的 27 轮循环操作。图中的尖峰特征还是非常明显的，然而这并不代表一定会有密钥信息的泄露。诚然，在很多情况下，功耗的尖峰和明显的起伏，很可能是泄露点，但是具体的泄露位置以及泄露程度，还是需要后续的分析才能确定。

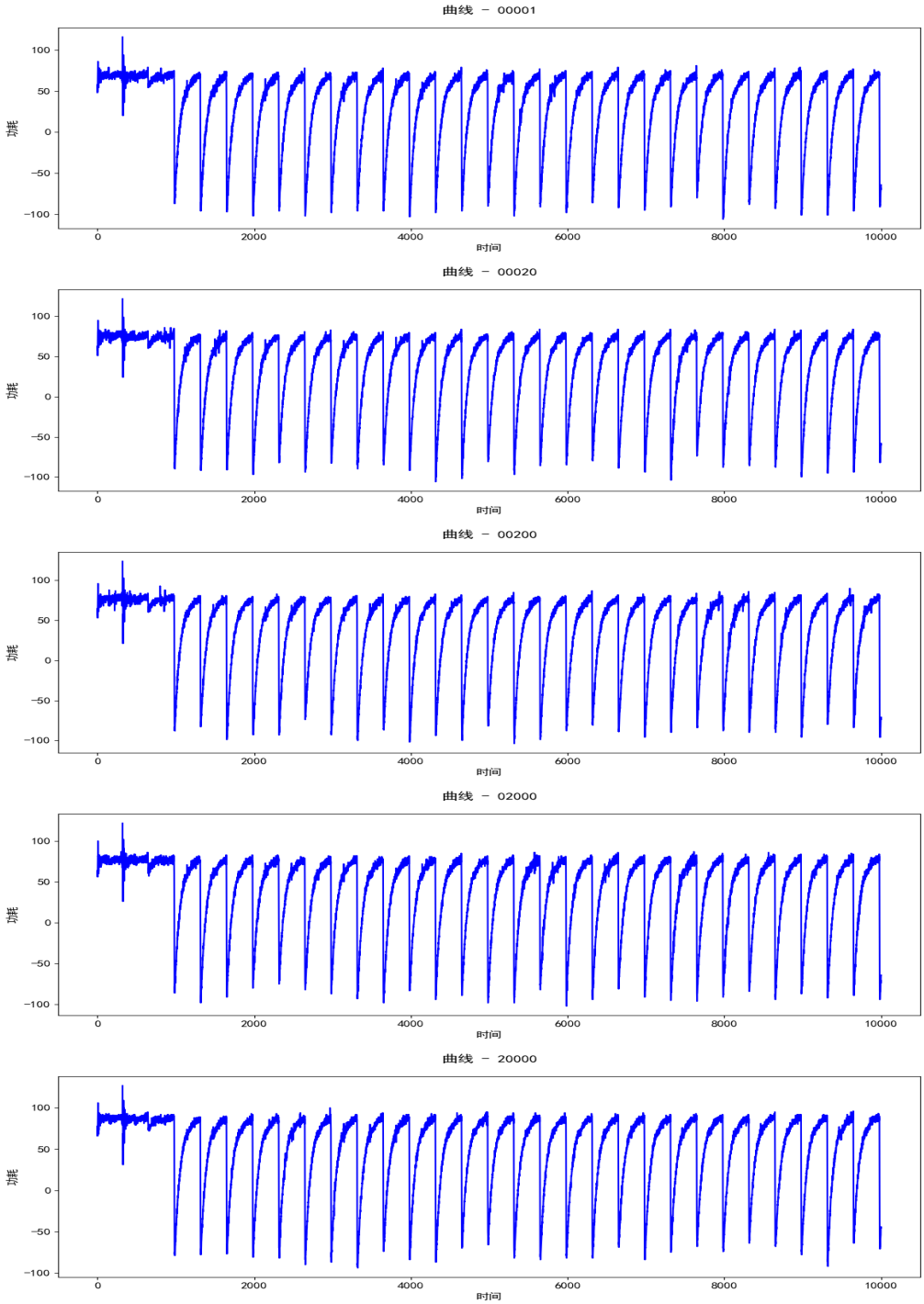


图 5–1 ZUC 算法初始化阶段硬件电路的功耗曲线

5.2 不同密钥猜测的相对相关系数

按照在上一章中分析的流程，猜测不同的 k[5] 对应的中间值，总共有 256 种可能的情况。然后利用汉明重量模型，将其转换成假设功耗值，求解与实际功耗值之间的相关系数。由于选取的中间值在初始化阶段第一轮末尾，因此我们将攻击的部位选取为覆盖第一轮和第二轮的功耗点，因此实际参与计算的功耗点数在 500 左右。

我们将这些点上的相关系数求绝对值后，取最大的前 5 个值求平均，得到可以参考的平均相关系数。这样处理的目的是为了减小误差，防止某些无效的相关系数值遮盖了有效的值。为了和真正的“相关系数”作区分，我这里将其称为“相对相关系数”。

图 5–2 展示了使用不同功耗曲线条数进行攻击，得到的 256 个密钥猜测的相对相关系数。横轴是密钥字节猜测的 10 进制值，纵轴是相对相关系数，绿色的虚线代表相对相关系数最高的猜测字节。

从图中可以得到这样一些信息：

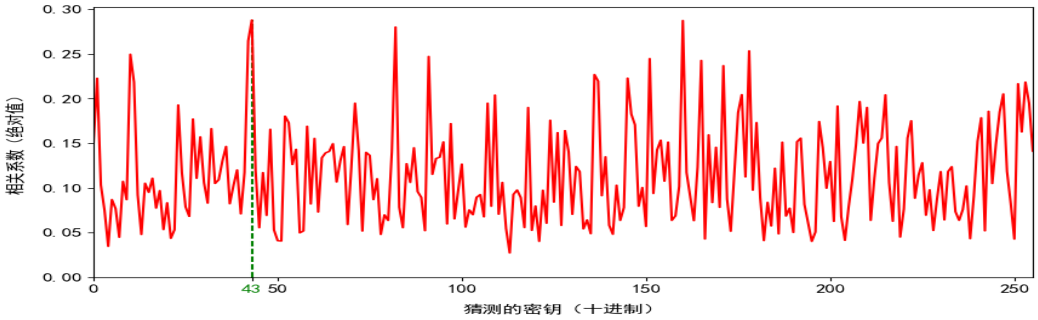
1. **当攻击的功耗曲线数目达到一定值之后，字节 171（AB）的相对相关系数保持最高。**
   因此 k[5] 最有可能是 171（AB）。而实验预设的密钥是“01 23 45 67 89 AB CD EF 01 23 45 67 89 AB CD EF”（k[0] 到 k[15]），k[5] 恰好是“AB”，这也就表明我们的最佳猜测是正确的，攻击成功。
2. **在使用的功耗曲线条数为 100 时，根据相对相关系数猜测的最佳密钥字节是 43（2B），而这个猜测结果是错误的。**
   这表明当攻击使用的功耗曲线数目较少时，得到的结果可信度较低，这时最佳密钥字节猜测还不够稳定。因此在攻击时一定要测试猜测结果的稳定性，选用的方法就是使用更多的功耗曲线。
   一个值得注意的细节是，尽管猜测错误，但是“2B”对应的二进制值是“00101011”，和正确的“AB”对应的二进制值“10101011”仅仅相差一个比特。虽然不同字节在非线性变换后，输出的差异一般非常大，因此错误猜测和正确结果之间的关联一般没有参考价值，但是这里的细节可能表明，密码算法的非线性变换部分或许有微小的瑕疵，从而造成了某些不同的字节在输出后差异并没有想象中的那么大，攻击者有可能通过选择合适的明文，来扩大非线性变换中的漏洞，更高效地破解相关密钥字节。不过尚缺乏充分的实验来证明这一猜想。
3. **当使用的功耗曲线条数越来越多时，正确的密钥字节对应的相对相关系数明显高于错误的密钥字节。**
   这个现象的原因是显然的，随着功耗曲线条数的增多，错误密钥字节产生的中间值与正确密钥字节产生的中间值差异愈发明显，偶然性渐渐被必然性覆盖。因此，选用尽可能多的功耗曲线条数，一般可以提高攻击的成功率。
4. **随着功耗曲线条数的增多，相对相关系数的最大值呈减小趋势。**
   比如使用的功耗曲线条数为 1000 时，最高相对相关系数约为 0.25，而当使用的功耗曲线条数到 20000 时，最高相对相关系数已经降低到大概 0.15 了。事实上，不仅最高相对相关系数在减小，所有密钥字节猜测对应的相对相关系数，整体上都在减小。这是因为，尽管假设功耗值（依据理论中间值和功耗模型计算得到）和实际功耗值之间存在一定的联系，但这种联系不一定是强正相关的。所以当攻击采用的功耗曲线条数很多时，假设功耗值与实际功耗值

之间不相关的部分占比就会变大。

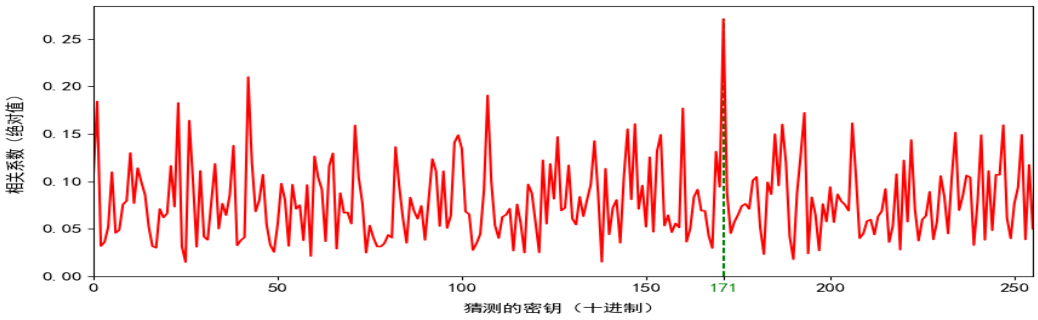
为什么假设功耗值与理论功耗值有不相关的地方呢？直观的原因是，电路执行某个操作的时间并不是每一次都完全对齐的，或者某些频率的噪声对曲线的波形产生了影响。还有一些更底层更根本的原因，比如选取的功耗模型并不能真正反映实际的功耗值大小，我们此次采用的汉明重量模型，是基于“设备的功耗和操作的数据中的 1 的个数有关”这一假设，而事实可能并非如此：设备的功耗可能不仅仅和数据中 1 的个数有关，既可能和数据中 0 的个数有关，也可能和从 0 到 1 或者从 1 到 0 的比特数目有关，而且设备的功耗也不完全取决于操作的数据，还可能和控制命令也有关系，或者操作相关数据的同时，还可能在运行其他的指令、操作其他不相关的数据。

既然这样，那么为什么本次实验中，功耗曲线条数越多，正确密钥猜测的相对相关系数越突出呢？这是因为，正确密钥字节对应的假设功耗值和实际功耗值中相关的部分比例高于不相关部分的比例。功耗曲线数目的增加并不会改变这种比例，因此尽管整体上相关系数的绝对数值降低了，但是正确的密钥字节对应的假设功耗值仍旧比错误的密钥字节更接近于实际功耗值。

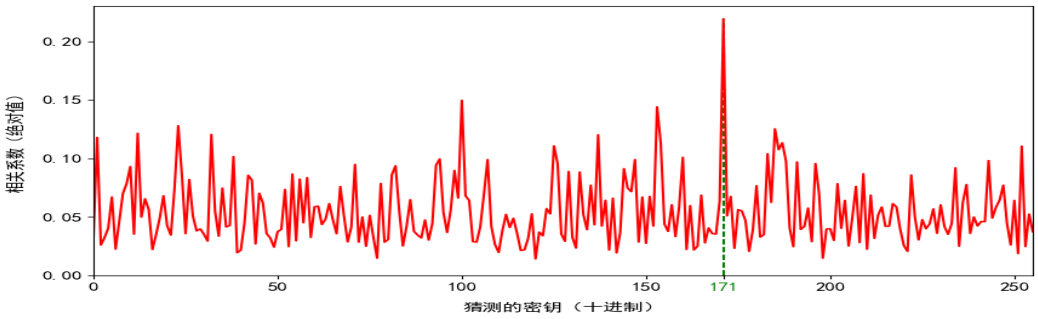
因此，尽管随着功耗曲线条数的增加，整体的相对相关系数的数值存在减小趋势，但是这也降低了偶然因素的影响，使得正确密钥字节对应的假设功耗值与实际功耗值之间的关联度，更加稳定地优于错误的密钥猜测。



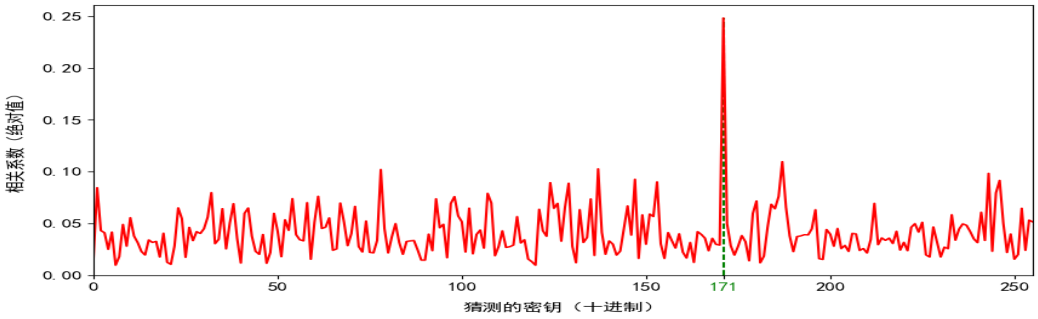
(a) 攻击使用的功耗曲线条数：100



(b) 攻击使用的功耗曲线条数：200

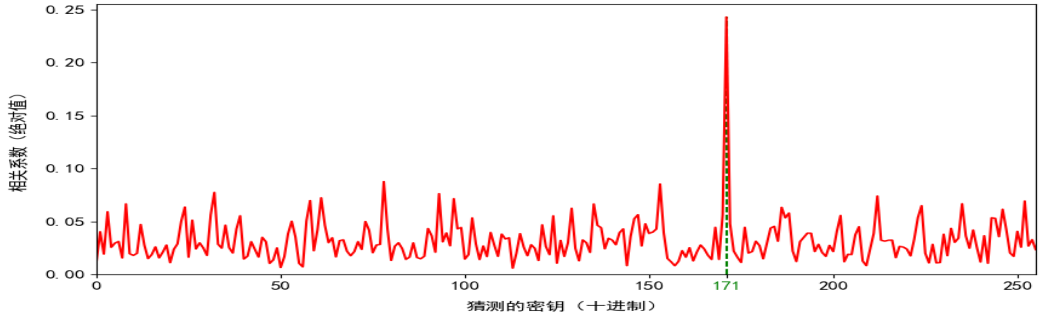


(c) 攻击使用的功耗曲线条数：500

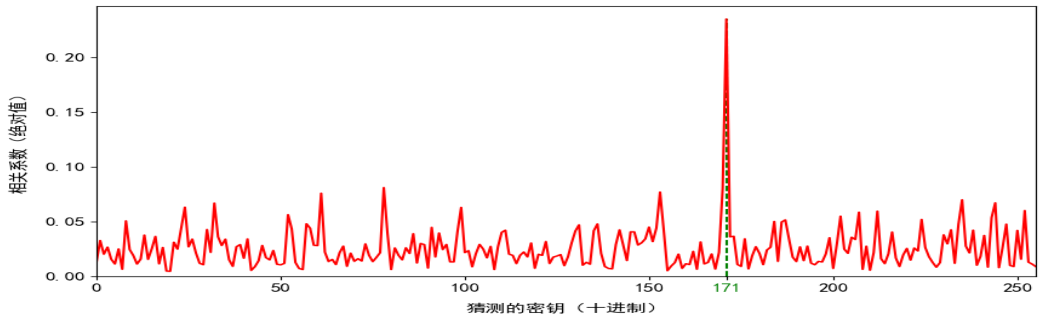


(d) 攻击使用的功耗曲线条数：1000

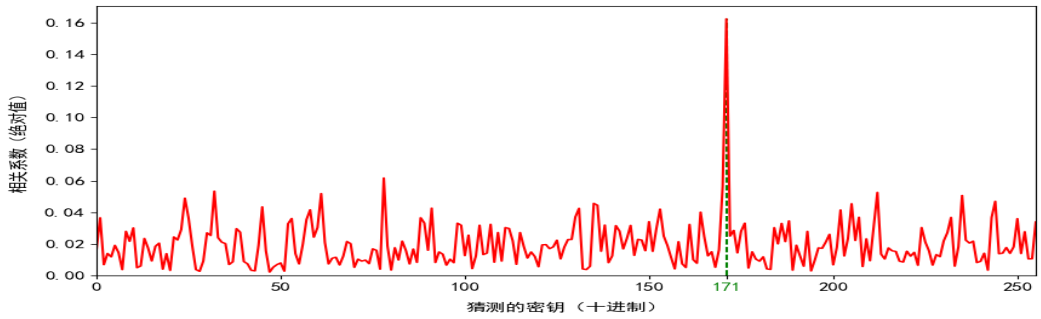
图 5–2 使用不同功耗曲线条数进行攻击，得到 256 个密钥猜测的相对相关系数（1）



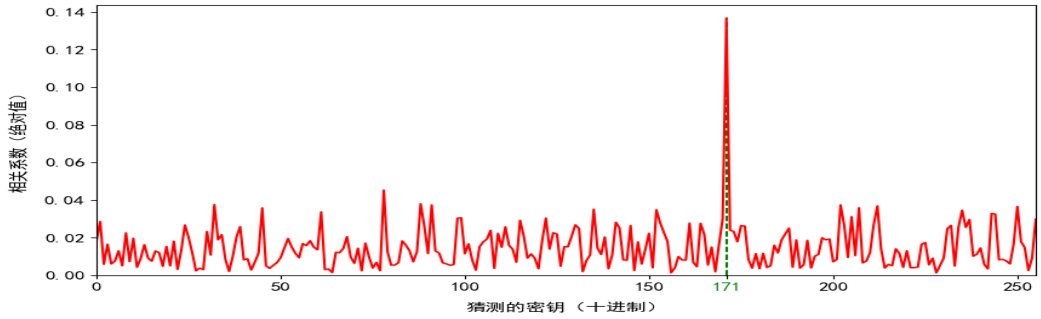
(e) 攻击使用的功耗曲线条数：2000



(f) 攻击使用的功耗曲线条数：5000



(g) 攻击使用的功耗曲线条数：10000



(h) 攻击使用的功耗曲线条数：20000

图 5–2 使用不同功耗曲线条数进行攻击，得到 256 个密钥猜测的相对相关系数（2）

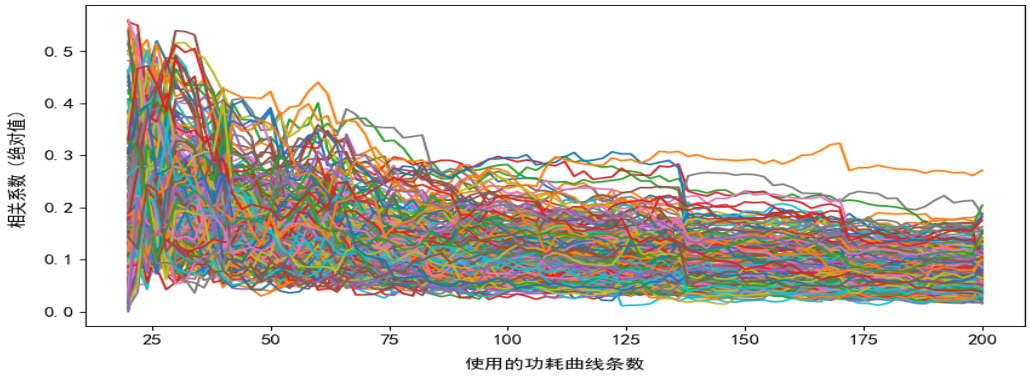
5.3 功耗曲线条数对相关系数的影响

上一节中从侧面反映了不同猜测密钥字节的相对相关系数和功耗曲线条数的关系，这一小节将更加完整地展现二者之间的联系。

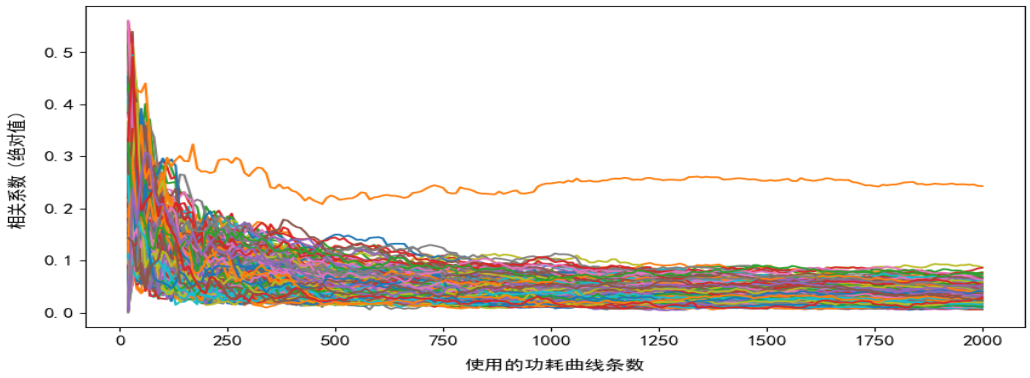
图 5–3 展示了不同密钥猜测的相对相关系数随功耗曲线条数变化情况。图中不同颜色的线代表不同的密钥猜测。图 5–3b 和图 5–3c 中相对相关系数最高的曲线对应的密钥字节为 171（AB），也即正确的密钥字节。

从图中可以得到这样一些信息：

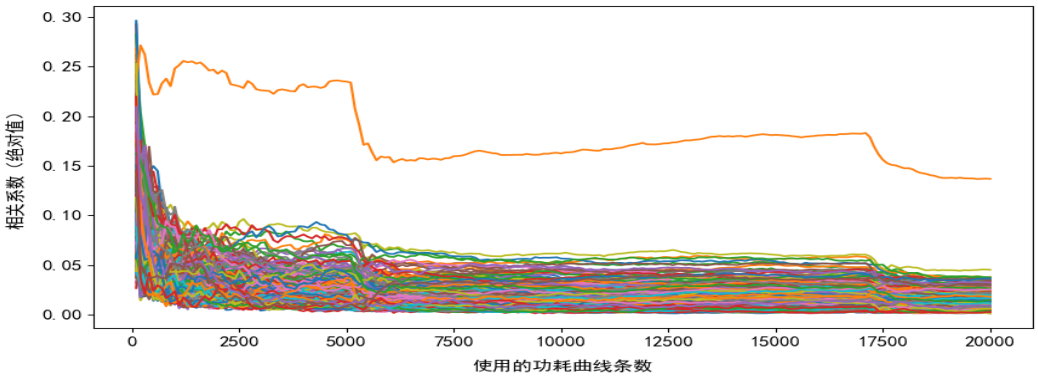
1. **攻击使用的功耗曲线条数较少时，无法分析出正确的密钥字节。**这一点已经在上一节详细阐述过，不再赘述。
2. **当攻击使用的功耗曲线条数超过一定值时，能够稳定地分析出正确的密钥字节。**这一点也已经在上一节详细阐述过，不再赘述。
3. **在攻击使用的曲线条数到达 5000 和 17500 左右时，相对相关系数的值均出现了明显的下降。**相对相关系数整体随着功耗曲线条数的增加而减小的原因，已经在上一节讨论过了，但是这里关注的问题是，为什么在这两个节点出现了相对相关系数的剧烈下降？而且不单是正确的密钥字节，所有密钥猜测的相对相关系数都同时出现陡降。一个合理的解释是，在这两个节点，采集的功耗曲线出现了在时间上微弱的偏移，导致后面的曲线组与前面曲线组不对齐。由于相对系数的计算和所有的功耗曲线都相关，因此如果后面采集的功耗曲线与前面的不完全对齐，就有可能造成攻击效果下降。
4. **在攻击使用的曲线条数从 5000 到 17500 的区间内，相对相关系数的值出现了缓慢的爬升。**这一现象与上一点并不矛盾，反而是支持功耗曲线出现时间偏移的有力证据。相对相关系数在第 5000 条到 17500 条这一段中出现了爬升，一个合理的解释是，尽管这一段曲线与第 1 到 5000 条是不对齐的，但是段内的曲线是对齐的，因此分析这一段时间内采集到的曲线，会渐渐不足之前不对齐造成的影响，因而出现相对相关系数的缓慢爬升。



(a) 攻击使用的功耗曲线条数变化范围：20 – 200



(b) 攻击使用的功耗曲线条数变化范围：20 – 2000



(c) 攻击使用的功耗曲线条数变化范围：20 – 20000

图 5–3 不同密钥猜测的相对相关系数随功耗曲线条数变化情况

5.4 密钥信息的泄露位置

尽管我们之前已经从理论上推测密钥信息泄露的位置应该在初始化阶段第一轮末尾，但是我们在攻击时是取覆盖第一轮和第二轮的一段区间的。如果我们想得到泄露密钥信息的更精确的位置，就需要计算正确密钥字节产生的中间值，然后计算出假设功耗值，求出其与实际功耗值的相关系数。从相关系数的变化中就能看出密钥信息的泄露位置。

图 5–4 展示了密钥字节 k5 的功耗信息泄露情况。图 5–5 则将泄露位置放大，以进行更加细致的观察。

从图中可以得到如下信息：

1. **攻击使用的功耗曲线条数较少时（100 条），密钥字节几乎无泄露。**
2. **攻击使用的功耗曲线超过 200 条时，就已经出现了较为明显的泄露。**
3. **随着攻击使用的功耗曲线条数的增加，对应位置功耗泄露的情况越发明显。**
4. **泄露的位置在初始化阶段第一轮的末尾和第二轮的开始之间。**

这些结论都是显而易见的，原因也都在上面的结果中分析过了，因此这里不再重复。

既然找到了泄露的位置，下一步的工作就是要对该部分进行防护了。由于算法本身不能修改，因此可以做的就是在硬件实现上进行改进。由于泄露的中间值是由于 S 盒的非线性变换引起的，因此可以对 S 盒做一些防护措施，比如最常用的就是加入掩码。不过这超出了本次实验的内容，可以留到今后进行更加深入的研究和分析。

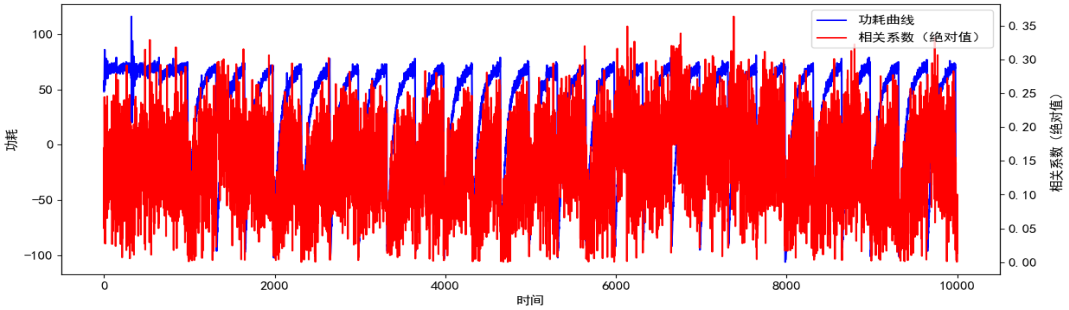
5.5 本章小结

本章首先展示了 ZUC 算法电路实际运行时采集的功耗曲线，简单分析了功耗曲线的特征，推测出了一些基本的信息。

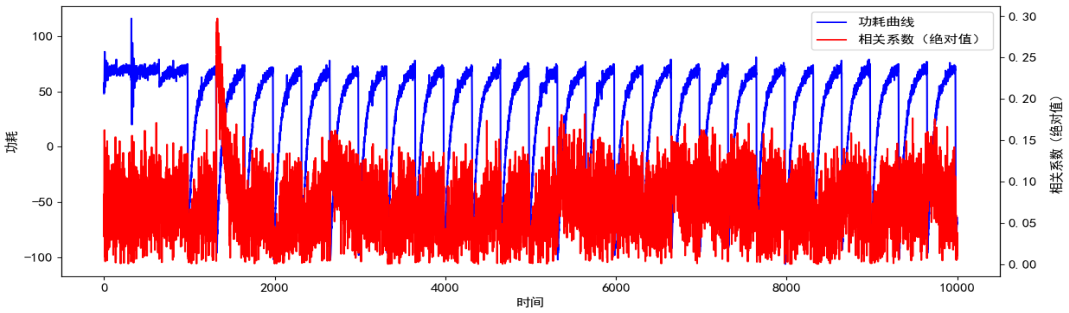
然后我们对采集到的功耗曲线实施了差分功耗分析攻击，得到了不同密钥猜测对应的相对相关系数。实验结果表明我们的攻击是成功的，得到了正确的密钥字节。与此同时，我们还发现了相对相关系数和功耗曲线条数有关的现象。

接着我们讨论了攻击使用的功耗曲线条数对相关系数的影响，指出了相对相关系数随功耗曲线条数增加时出现的变化，并且根据事实和观察给出了可能的原因和合理的解释。

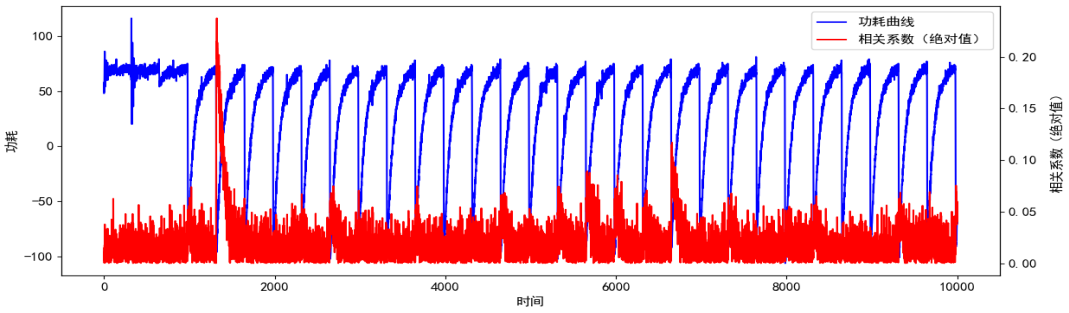
最后我们分析了密钥信息的泄露位置，发现功耗泄露的位置确实是理论分析得到的地方，说明我们的思路和方法是正确的。



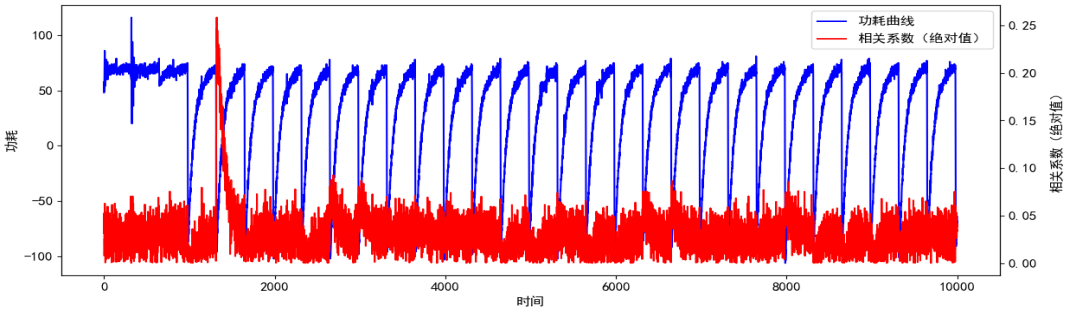
(a) 攻击使用的功耗曲线条数：100



(b) 攻击使用的功耗曲线条数：200

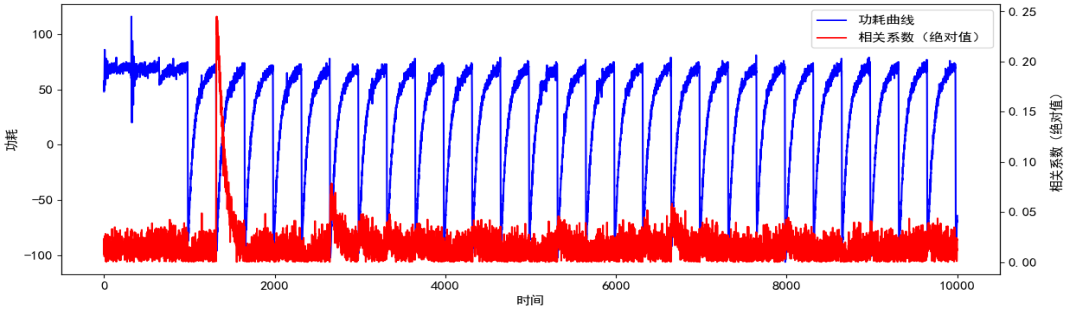


(c) 攻击使用的功耗曲线条数：500

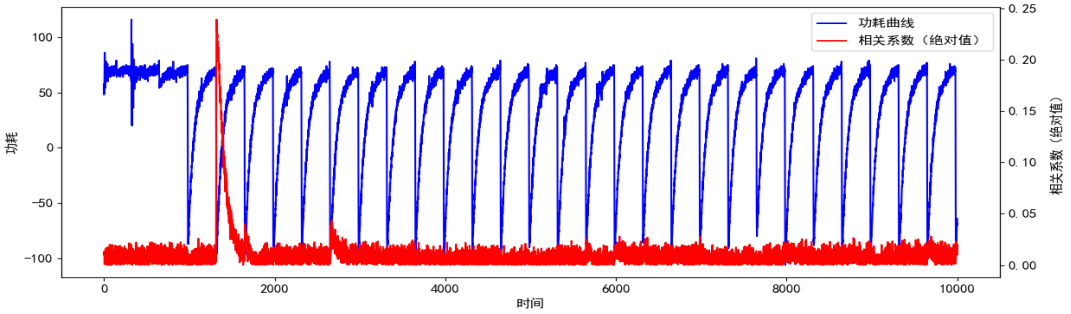


(d) 攻击使用的功耗曲线条数：1000

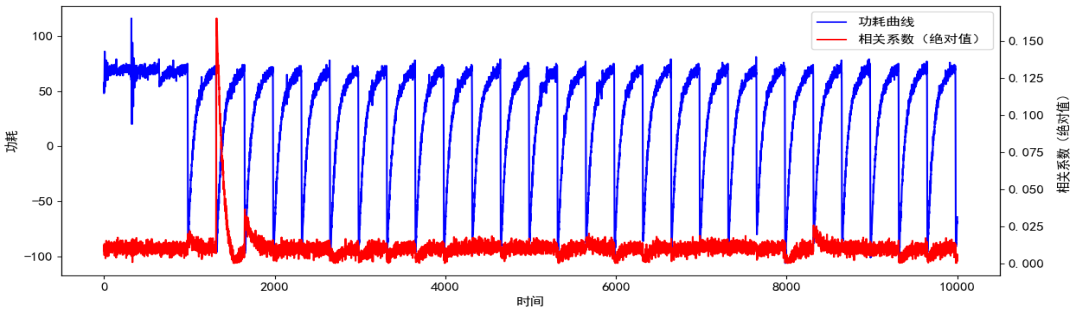
图 5–4 正确密钥字节对应的假设功耗值与实际功耗值的相关系数（1）



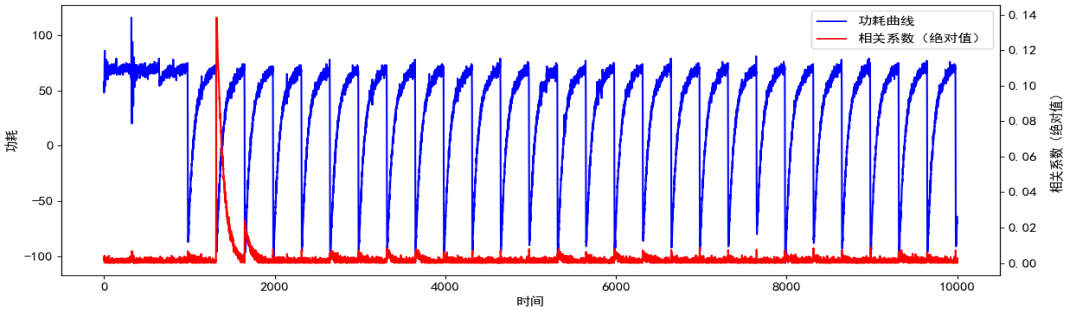
(e) 攻击使用的功耗曲线条数：2000



(f) 攻击使用的功耗曲线条数：5000

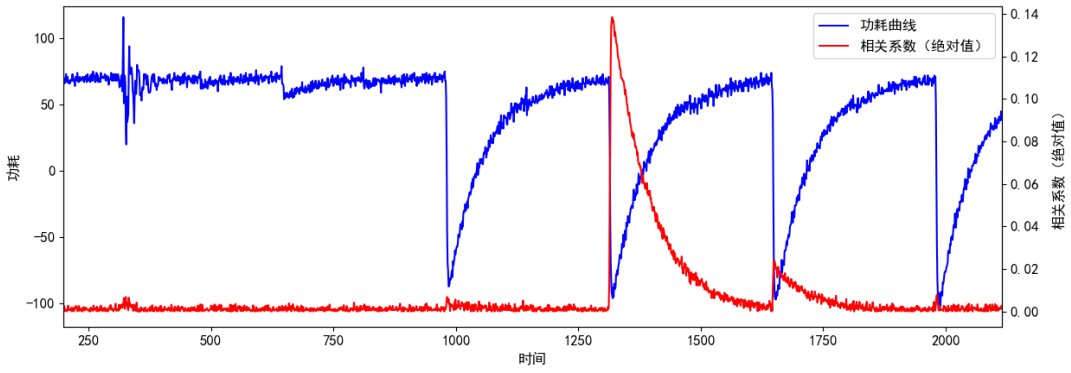


(g) 攻击使用的功耗曲线条数：10000

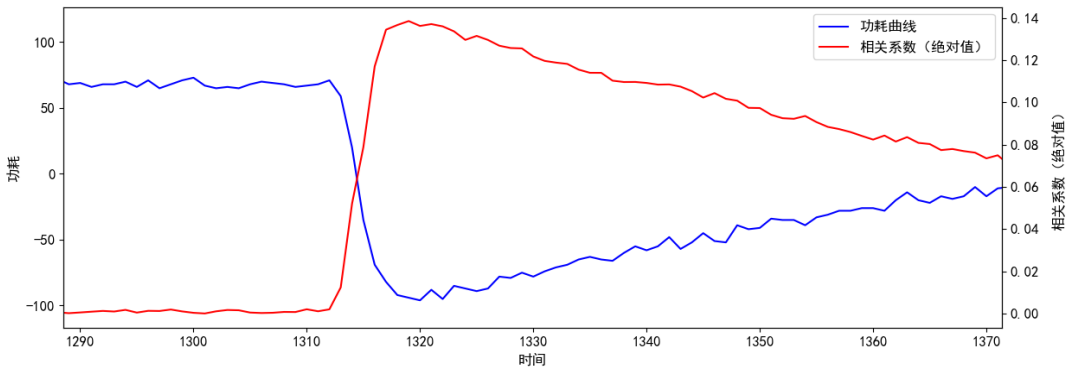


(h) 攻击使用的功耗曲线条数：20000

图 5–4 正确密钥字节对应的假设功耗值与实际功耗值的相关系数（2）



(a) 使用 20000 条功耗曲线时的泄露情况（放大 5 倍）



(b) 使用 20000 条功耗曲线时的泄露情况（放大 140 倍）

图 5–5 密钥字节功耗泄露情况的细节（使用 20000 条功耗曲线）

第六章　总结和展望

6.1　研究总结

本次毕业设计研究了旁路攻击方法在祖冲之算法上的应用，实现了硬件电路，完成了软件分析，得到了预期结果，并且解释了诸多现象的原因。

首先我们了解了现代密码学的基本概念，对现代密码学在信息安全领域的应用有了较为感性和初步的认识；了解了旁路攻击的基本分类和常见手段，以及不同手段的具体适用场景；了解了密码设备的组成部件和基本结构，对数字电路的专业知识做了简单的整理。

然后我们掌握了旁路攻击中最经典的方法——功耗分析攻击，并且介绍了功耗分析攻击的一般流程；了解了数字电路功耗的构成、功耗仿真的方式，以及功耗采集所需的设备与步骤；掌握了功耗分析中最常用的方法——差分功耗分析，详细介绍了差分功耗分析的流程和步骤。

接着我们熟悉了 ZUC 算法的背景和原理，并且讲解了 ZUC 算法的工作方式；完成了 ZUC 算法的硬件电路，采集了设备运行时的功耗，提出了对 ZUC 算法的差分功耗分析方案，并且在软件层面对功耗曲线进行了详细地分析，验证了方案的正确性和可行性。

最后我们展示了实验结果，简单分析了功耗曲线的特征，成功地依据制定的方案完成了攻击，得到了正确的密钥字节。我们还发现了攻击使用的功耗曲线条数对相关系数的有一定的影响，并且根据事实和观察给出了可能的原因和合理的解释。并且根据得到的结果反推出密钥信息的泄露位置，发现实际情况和我们的假设结论是相符的。

6.2　未来展望

尽管最终的实验结果达到了预期的目标，但是还有很多可以改进和深入的地方。

首先是可以对 ZUC 算法的分析方法进行改进。选取文中所述的中间值和功耗模型，只是差分功耗分析中的一个可行方案，而差分功耗分析只是诸多功耗分析中的一种，功耗分析又是很多旁路攻击方法中的一种。比如之后可以尝试其他可能的中间值，或者选用不同的功耗模型。还可以尝试功耗分析之外的其他方法，比如电磁攻击，很可能电磁信息会泄露算法中更多的信息。，然后是可以对分析方法的适用范围进行改进。本次实验仅仅是完成了对 ZUC 算法的攻击，如果能够将这些方法中的核心思路和关键技术抽象出来，把他们应用到更多的序列密码算法电路中，可能会有更高的实用价值。

最后是可以针对 ZUC 算法在实验中暴露的问题给出具体的防护方案。比如可以针对 S 盒进行掩码，掩盖设备运行过程中泄露的信息，或者是在算法的某些地方随机插入空操作，以打乱功耗曲线的时序，让攻击者难以分析出功耗泄露的具体位置。

总而言之，本次毕业设计还有很大的提升空间，如果想要提高和改进实验的结果，就需要我在今后的时间里，学习更多的知识，进行更加深入的研究。所谓学无止境，即使到达百尺竿头，亦需要更进一步。

参考文献

[1] Ming T, PingPan C, ZhenLong Q. Differential power analysis on ZUC algorithm. 2012. https://eprint.iacr.org/2012/299.pdf.

[2] 唐明, 高剑, 孙乐昊, 等. 嵌入式平台下 ZUC 算法的侧信道频域攻击[J]. 山东大学学报 (理学版), 2014.

[3] 杜红红, 张文英. 祖冲之算法的安全分析[J]. 计算机技术与发展, 2012.

[4] Orhanou G, Hajji S E, Lakbabi A, et al. Analytical evaluation of the stream cipher ZUC[C]//2012 International Conference on Multimedia Computing and Systems. [S.l.]: [s.n.], 2012: 927-930. DOI: 10.1109/ICMCS.2012.6320128.

[5] 严迎建, 杨昌盛, 李伟, 等. ZUC 序列密码算法的选择 IV 相关性能量分析攻击[J]. 电子与信息学报, 2015.

[6] Wu H, Huang T, Nguyen P H, et al. Differential Attacks against Stream Cipher ZUC[C]//Wang X, Sako K. Advances in Cryptology – ASIACRYPT 2012. Ed. by Wang X, Sako K. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012: 262-277. ISBN: 978-3-642-34961-4.

[7] 关杰, 丁林, 刘树凯. SNOW3G 与 ZUC 流密码的猜测决定攻击[J]. 软件学报, 2013.

[8] Fischer W, Gammel B M, Kniffler O, et al. Differential Power Analysis of Stream Ciphers[C]//Abe M. Topics in Cryptology – CT-RSA 2007. Ed. by Abe M. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006: 257-270. ISBN: 978-3-540-69328-4.

[9] Qu B, Gu D, Guo Z, et al. Differential power analysis of stream ciphers with LFSRs[J/OL]. Computers & Mathematics with Applications, 2013, 65(9): 1291-1299. http://www.sciencedirect.com/science/article/pii/S0898122112001381. DOI: https://doi.org/10.1016/j.camwa.2012.02.024. ISSN: 0898-1221.

[10] Goldreich O. Foundations of Cryptography:A Primer[M/OL]. [S.l.]: Now Foundations, Trends, 2005: 132-. https://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=8187550. DOI: 10.1561/0400000001. ISBN: 9781933019024.

[11] Konheim A G. Basic Concepts of Cryptography[M/OL]//Hashing in Computer Science:Fifty Years of Slicing and Dicing. [S.l.]: Wiley Telecom, 2010: 300-. https://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=8041771. DOI: 10.1002/9780470630617.ch6. ISBN: 9780470630617.

[12] Katz J, Lindell Y. Introduction to Modern Cryptography[M]. [S.l.]: Chapman & Hall/CRC, 2007. ISBN: 1584885513.

[13] Mangard S, Oswald E, Popp T. 能量分析攻击[M]. 冯登国, 周永彬, 刘继业, 译. 北京: 科学出版社, 2010.

[14] Zhou Y, Feng D. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. 2005. http://eprint.iacr.org/2005/388.

[15] 冯秀涛. 3GPP LTE 国际加密标准 ZUC 算法[J]. 信息安全与通信保密, 2011.

[16] 陈珺, 田云飞, 崔斌. 题名: 祖冲之算法的国际标准化进展和终端一致性测试[J]. 电信网技术, 2012.

[17] GM/T 0001-2012. 祖冲之序列密码算法. [R]. 中国, 2012.

[18] 冯秀涛. 祖冲之序列密码算法[J]. 信息安全研究, 2016.

[19] Mangard S, Oswald E, Popp T. Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)[M]. Berlin, Heidelberg: Springer-Verlag, 2007. ISBN: 0387308571.

[20] 李歌, 陶琳, 高献伟. 基于 FPGA 的祖冲之算法研究与实现[J]. 北京电子科技学院学报, 2012.

[21] Kitsos P, Sklavos N, Skodras A N. An FPGA Implementation of the ZUC Stream Cipher[C]//2011 14th Euromicro Conference on Digital System Design. [S.l.]: [s.n.], 2011: 814-817. DOI: 10.1109/DSD.2011.109.

致　谢

感谢郭筝和刘军荣两位老师，是你们让我明确了研究的方向，并且给我提供了硬件和平台上的支持。两位老师在我入门这个方向时给了我很多指导，让我得以迅速掌握基础知识，从而踏入这个领域的大门。

感谢张驰和朱文锋两位学长，你们协助我解决了一路上碰到的不少棘手问题，在做毕业设计的过程中用到的许多技术，都是建立在你们之前工作的基础之上的。如果没有学长提供的思路方法和工程经验，我肯定会花费很多不必要的精力，走更多弯路。

感谢我的几位挚友，王文铮、文奕扉、孙一璇、杜威，祝愿大家都有美好的未来。

感谢爸妈，无论什么时候，你们都一如既往地支持我鼓励我，让我能够坚定自己的信念和理想。

感谢本科四年中教过我的很多老师和帮助过我的很多学长，虽然毕业设计和所学的知识并非和各科课程直接相关，但是你们教给我的学术能力和工程思维，让我能够在面临新的领域和问题时能够充满信心、毫不气馁。

感谢微电子系的同学，我们一起度过了本科四年愉快的时光。

感谢交大，我在这里真正感受到了精神上的自由，交大教给我的一切将是我一生的财富。

A STUDY ON DIFFERENTIAL POWER ANALYSIS OF CIRCUITS RUNNING ZUC ALGORITHM

This paper studies the technologies used in traditional power analysis of block cipher algorithms and applies them to a famous stream cipher algorithm — ZUC algorithm. The results of our experiments indicate that ZUC algorithm is also vulnerable to power analysis attacks.

We introduce the basic knowledge of modern cryptography, cryptographic devices and side-channel attacks.

Modern cryptography is based on rigorous mathematical conclusions and provements, so its security is improved a lot compared to traditional cryptography. One famous principle in modern cryptography is Kerckhoff's principle. The principle suggests that the cipher algorithms should be open and evaluated by the public. This principle breaks people's prejudices on the confidentiality of cipher algorithms and it is proved to be true as time goes on. So the famous cipher algorithms are harder to attack and break with traditional methods of cryptanalysis.

However, the side-channel attacks appeared in the 1990s. This kind of attacks does not aim to discover the bugs or flaws in the cipher algorithms with theoretical provements, but open the gate to attack the cryptographic devices in reality. People using this kind of attacks believe that, although the cipher algorithm is nearly perfect and it is impossible to find their theoretical flaws, the devices running the cipher algorithms are not perfect. Vulnerabilities are common in the real-world implementations of the cipher algorithms.

Since side-channel attacks have large threats to the security of the real-world cryptographic devices, a lot of researchers are attracted and this kind of attacks becomes more and more popular. And the techniques and ideas of side-channel attacks are flourishing and amazing today. From the early timing attack to the latest acoustic attack, from the famous power analysis attack to the less known light attack, side-channel attacks have become the most effective and powerful methods to break all kinds of cryptographic devices.

And one important part of side-channel attacks is the real-world circuits and devices. It is necessary to learn about the design and manufacture of physical devices. The basic elements of cryptographic devices are digital integrated circuits. So the researchers should not only be good at mathematics, but also have a good understanding on microelectronics. This is why side-channel attacks can do some things that traditional cryptanalysis cannot do — knowledge of different areas is applied.

We discuss the power analysis attacks, and one technique of this kind is differential power analysis, the abbreviation for which is DPA.

Before attacking the devices with power analysis, we must learn that in the circuits, what is consuming power. The power consumption of logic components is usually divided into static power and dynamic power consumption.

Static power consumption usually comes from the leakage current of transistors, which is relatively low, but it needs to be noted that as the size of a single transistor becomes smaller and smaller, the proportion of leakage current is gradually increased. Of course, static power consumption is negligible in actual attacks.

The dynamic power consumption usually comes from the reversal of the signal, which causes the truncation or conduction of the transistor, which is equivalent to the charge and discharge of the intrinsic and parasitic capacitance of the transistor. Another part of the dynamic power consumption comes from transient short-circuit current. One other thing to consider is the burr produced in the circuit, which often produces high instantaneous power consumption and is related to the data, so special attention needs to be paid. In short, dynamic power consumption is generally a major component of component power consumption, and most of the power consumption we collect is mostly dynamic power consumption.

In the design stage of digital circuits, designers often need to simulate the power consumption of circuits. On the one hand, it is to reduce the power of the circuit as much as possible to improve the market competitiveness of the circuit, on the other hand, to avoid more obvious disadvantageous factors such as burr, affect the basic function of the circuit, and to reduce the information of power leakage as far as possible.

Power analysis usually consists of simple power analysis (SPA) and differential power analysis (DPA).

Simple power analysis usually requires a small number of power curves to reveal useful information in cryptographic devices. Simple power analysis is usually applicable to cryptographic devices with more obvious characteristics of the power curve, such as obvious peaks and valleys, and multiple periodic repetitions. If an attacker has certain preparatory knowledge of the program running in the password device, then it is possible to speculate on what operation of the different paragraphs of the power curve, and more information on the device may be further mastered.

Differential power analysis requires a large number of power traces. The benefits of a large amount of power data are more powerful analysis and attack capabilities, and there is no need to have a detailed understanding of the construction and execution of the device. In general, as long as the algorithm flow of the device runs is sufficient to carry out analysis and attack.

The general process of differential power analysis includes the following steps. Firstly, select the appropriate location of the intermediate value of the algorithm; secondly, collect the actual power consumption curve of the device; thirdly, calculate the theoretical intermediate value according to the algorithm and use the appropriate power model to convert the theoretical middle value to the hypothesis of power consumption; finally, analyze the assumption of the power and the actual power curve, and dig the information needed.

We talk about the ZUC algorithm and propose a scheme to attack it with differential power analysis.

Zu Chongzhi algorithm, also known as ZUC algorithm, is the first international commercial standard cipher algorithm proposed by our country. The proposed ZUC algorithm has made our country have more autonomy in the field of international commercial cryptography, which not only embodies the academic ability of our country in the field of cryptography, but also is an important step for our country to participate in the formulation of international communication standards. Therefore, the study of ZUC algorithm has high practical significance.

Because the password itself is relatively complex, although I have implemented the ZUC algorithm

on both hardware and software, and verified the correctness of the implementation, it is unnecessary to explain the details of some implementations, and it is just a rough introduction to the basic composition of the algorithm. Part and general flow. If you want to know more about the algorithm, you can refer to the literature.

As an attacker and researcher, it is important to know all the details of the algorithm and the reason for the design, but it is more important to find the possible problems in the actual implementation of the algorithm. A designer and an attacker consider a different point of view. The designer often has a better view of the overall situation, but it is often unexcavated in detail, and an attacker only needs to pry a point in the whole system to reach the goal.

In the ZUC algorithm, the only unknown information is the initial key, and the other constants and plaintexts are known. Therefore, the purpose of our attack is to get the key information. We choose the output of the right half of the nonlinear function in the first round of the initialization phase as the intermediate value, and try to attack the value of fifth byte of the original keys.

We apply the differential power analysis attack to a real-world device and the results indicate that ZUC algorithm is vulnerable to differential power analysis.

When the number of power consumption curves used by the attack is small, the result is less reliable, and the best key byte guess is not stable enough. Therefore, it is necessary to test the stability of guessing results when attacking. The method of selection is to use more power curves.

When the number of power consumption curves is increasing, the relative correlation coefficient of the correct key byte is obviously higher than that of the wrong key byte.

The reason for this phenomenon is obvious. With the increase of the number of power consumption curves, the difference between the intermediate value produced by the error key byte and the correct key bytes is more obvious, and the contingency is gradually covered by the inevitability. Therefore, the selection of as many power curve bars as possible can generally increase the attack success rate.

With the increase of the number of power curve, the maximum of relative correlation coefficient decreases. In fact, not only the highest relative correlation coefficient is reduced, but all key bytes guess the corresponding relative coefficient, which is decreasing as a whole.

This is because, although there is a certain relationship between the assumption of the power value (based on the theoretical median and the power model calculated) and the actual power consumption, this connection is not necessarily strongly positive correlation. Therefore, when the number of power consumption curves used by the attack is large, it is assumed that the proportion of the uncorrelated parts between the power consumption value and the actual power consumption will increase.

In conclusion, we prove that the power analysis is effective and powerful to attack the ZUC algorithm. We have developped a general process of problem modeling and data analysis. The experience can also be applied to other stream cipher algorithms and that is the most important result.