系统硬件安全

李琦
清华大学网研院

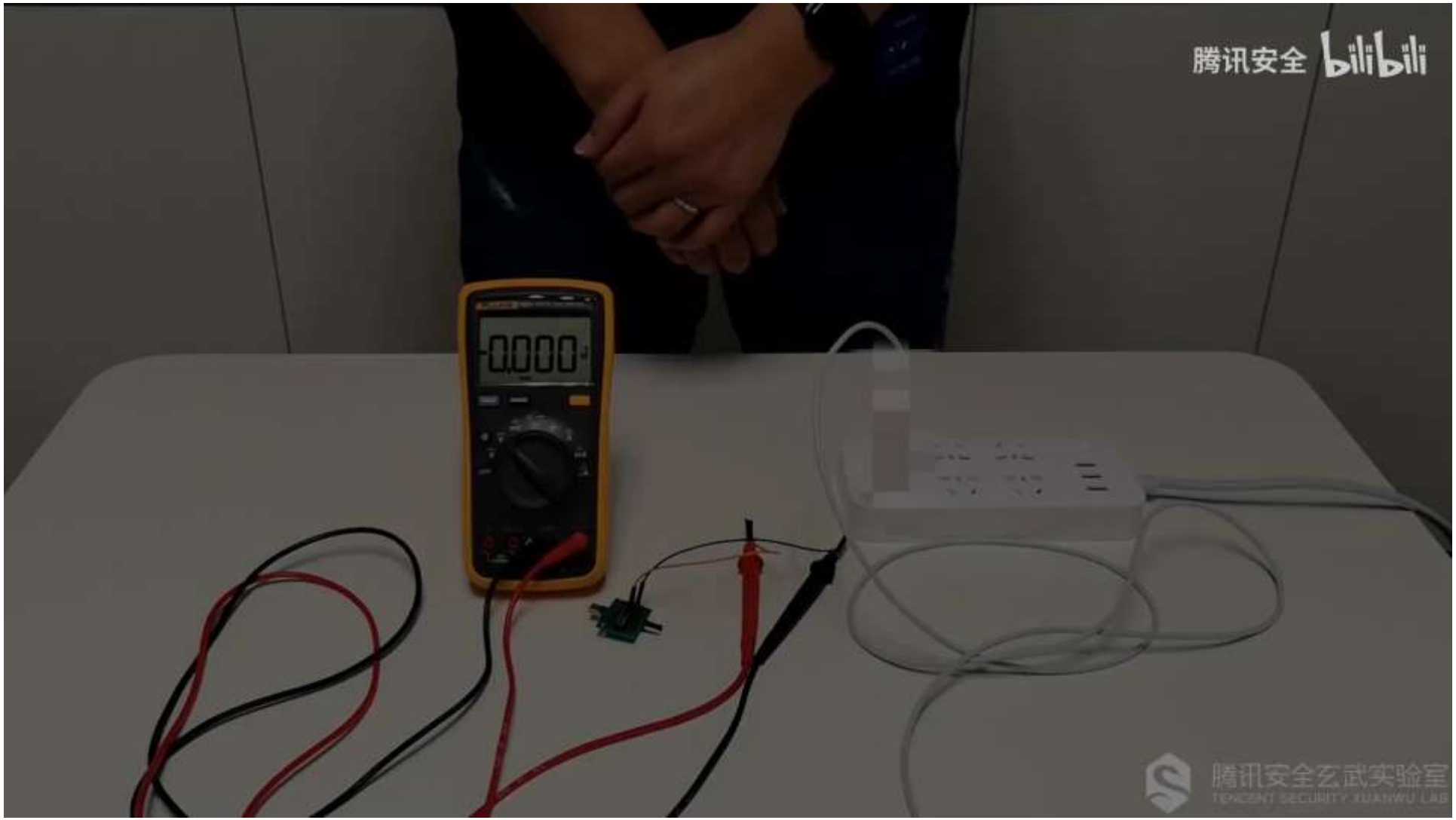
讨论

畅所欲言

CPU等硬件是难以修改的，所以相对来说更安全，

你同意这个观点吗?

BadPower

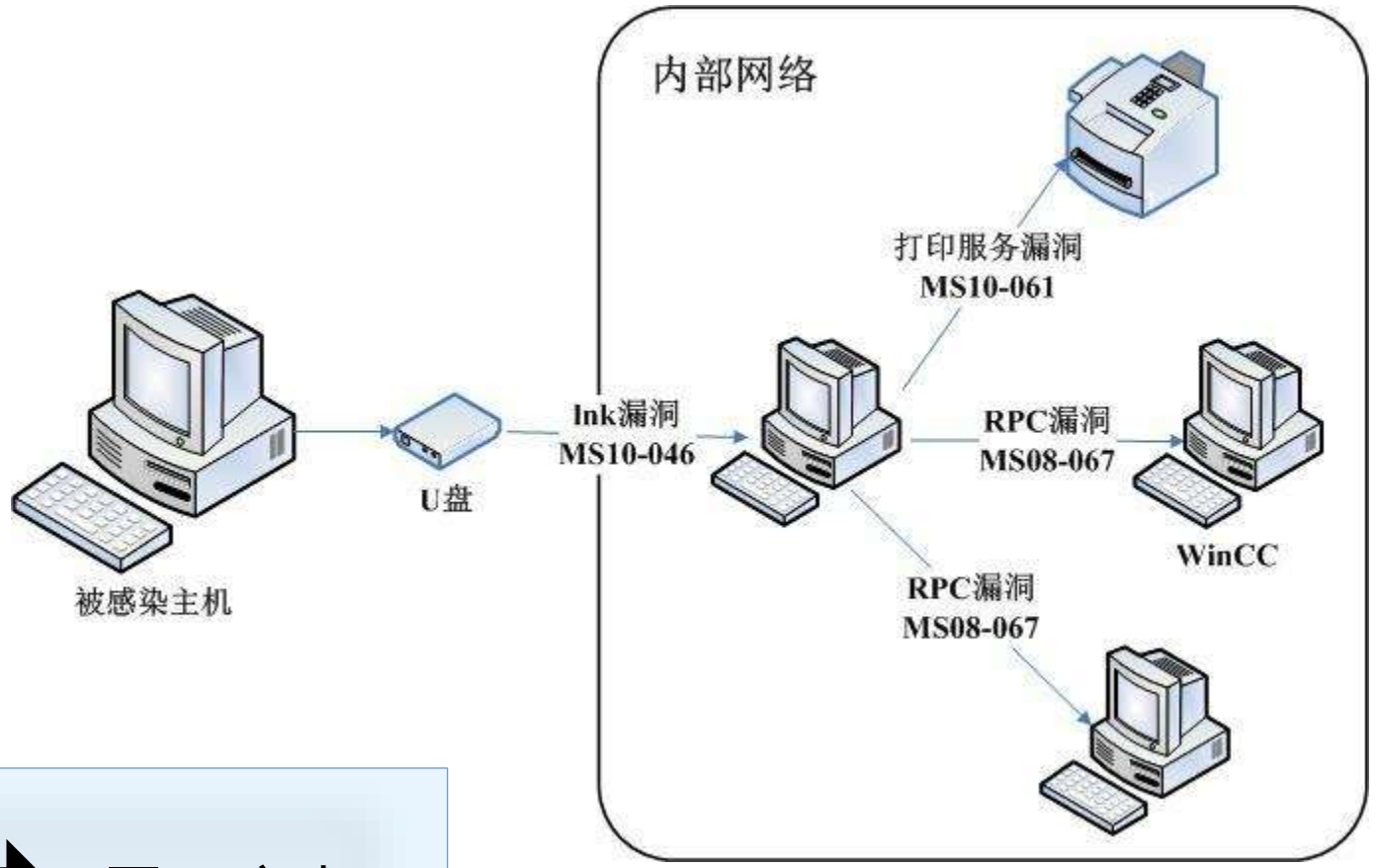


伊朗核设施

伊朗的核设施位于纳坦兹，不与外网连接，并有严密的物理隔离，却遭受了「震网」病毒攻击

震网病毒

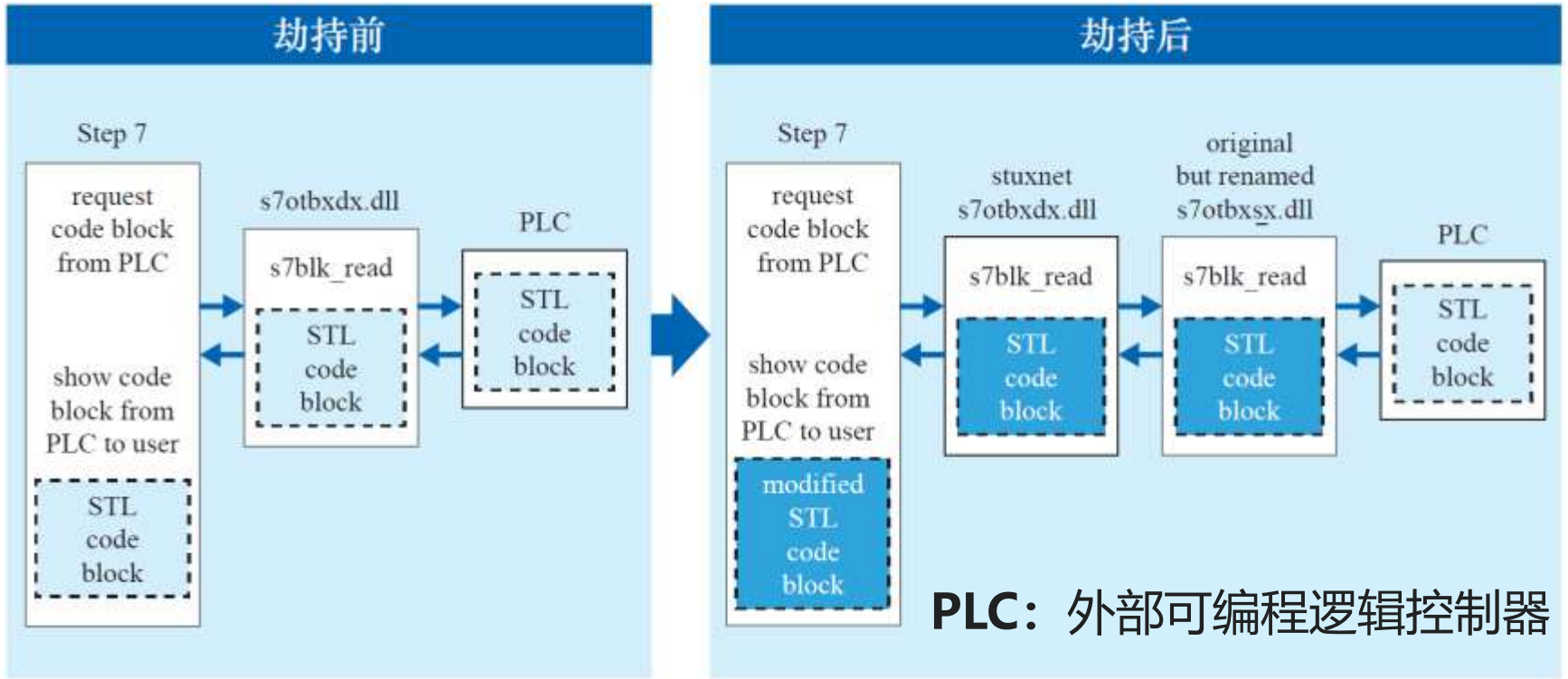
西门子公司产的工业控制系统被广泛运用于伊朗核设施的工业控制系统之中



震网病毒传播方式

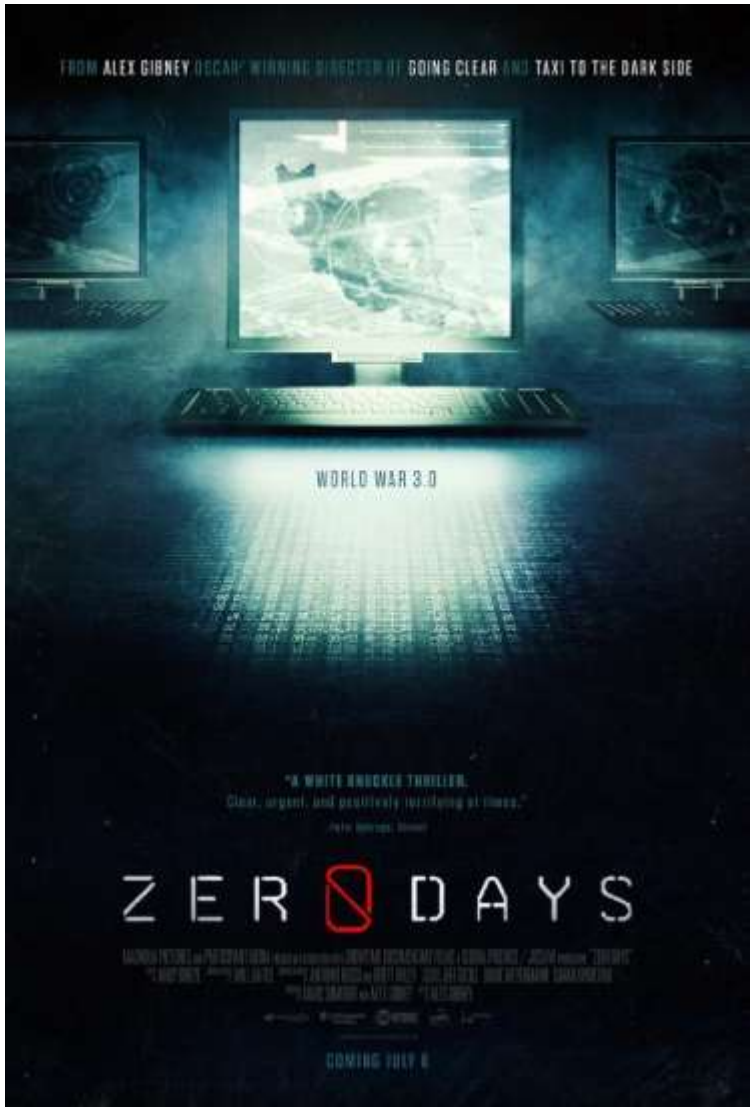
西门子 (7漏洞) + Windows (2漏洞) → 震网病毒

震网病毒



- 工程师主要的作用是用来设计自动化程序，并将程序上传到PLC上运行。震网病毒攻击的特定软件是西门子PLC的STEP7
- 震网将原始的STEP7 DLL文件（名为“s7otbxdx.dll”）替换成自己的恶意版本，劫持所有与PLC通信的函数，然后将恶意代码注入PLC

Zero Days



纪录片《零日》中讲述了美国与以色列的情报机构如何借助「震网」病毒之手，瘫痪了伊朗的核设施，这也凸显了潜在危害全球工业系统的风险

网络空间的里程碑

震网被认为是第一个以现实世界中的关键工业基础设施为目标的恶意代码，通过西门子的硬件设备漏洞，达到了预设的攻击目标，其证实了通过网络空间手段进行攻击，可以达成与传统物理空间攻击（甚至是火力打击）的等效性

	传统物理攻击伊拉克	网络空间攻击伊朗
被攻击目标	伊拉克核反应堆	伊朗纳坦兹铀离心设施
时间周期	1977-1981年	2006-2010年
人员投入	以色列空军、特工人员、伊朗空军、美国空军和情报机构	美、以情报和军情领域的软件和网络专家，工业控制和核武器的专家
作战准备	多轮前期侦查和空袭，核反应堆情报	战场预制、病毒的传播和相关核设施情报
前期成本	18个月模拟空袭训练，训练2架F-4鬼怪攻击坠毁，3名飞行员阵亡	跨越两位总统任期，经历了5年的持续开发和改进
毁伤效果	反应堆被炸毁，吓阻了法国供应商继续提供服务，伊拉克核武器计划永久迟滞	导致大量离心机瘫痪，铀无法满足武器要求，几乎永久性迟滞了伊朗核武器计划
效费比	打击快速，准备期长，耗资巨大，消耗大，行动复杂，风险高	周期长，耗资相对军事打击低，但更加精准、隐蔽，不确定性后果更低

影响与启示

如果说，伊拉克战争向世人展现了美国只用空中打击就能灭亡一国的军事实力，那么「震网」病毒所呈现的就是美国在互联网时代强大的攻击能力，以及他国与之的能力「代差」

- 网络空间安全很重要
- 系统硬件安全很关键

身边的硬件安全问题

Meltdown

Spectre

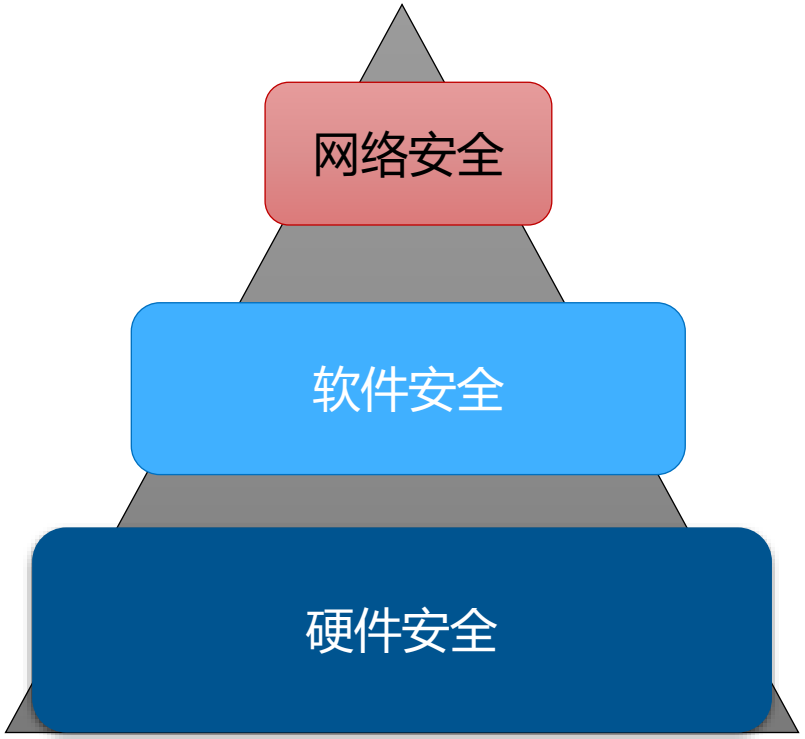
Foreshadow

ZombieLoad

硬件漏洞

VoltJockey

硬件安全



硬件安全是网络安全的基石

硬件安全包括哪些内容?

硬件安全概述

- 硬件指计算机系统中由电子、机械和光电元件等组成的各种物理装置的总称。例如，CPU、存储器和传感部件等

- 硬件安全作为网络空间安全的重要研究方向，涉及硬件设计，访问控制，安全计算，安全密钥存储、硬件供应链安全等诸多方面

系统硬件安全

- 系统硬件
  - 硬件攻击技术
    - 漏洞攻击
    - 硬件木马攻击
    - 故障注入攻击
      - 基于软件
      - 基于硬件
    - 侧信道攻击
      - 物理侧信道
      - 微架构侧信道
  - 硬件防护技术
    - 密码技术
      - 对称密码
      - 非对称密码
      - PUF技术
    - 隔离技术
      - 存储器隔离
      - TrustZone
      - SGX
    - 侧信道防护
      - 旁路信道隐藏
      - 掩码技术
      - 设计分区
      - 物理安全
    - 木马检测
      - 破坏性检测
      - 非破坏性检测

本章的内容组织

第一节
硬件攻击技术

- 漏洞攻击
- 硬件木马
- 硬件故障
- 侧信道攻击

学习硬件攻击的基本方法，掌握系统硬件攻击原理

第二节
硬件防护技术

- 木马检测技术
- 隔离技术
- 密码技术
- 侧信道防护

熟悉硬件防护技术，理解系统硬件防御的基本策略

第三节
典型漏洞分析

- MOLES
- Meltdown
- Spectre
- VoltJockey

结合典型漏洞，深入分析硬件攻击技术的具体实现

知己知彼，百战不殆，了解攻击手段才能制定安全策略

实践出真知，在实践中剖析攻击技术，进而构建切实可靠的防御体系

第1节 硬件攻击技术

- 漏洞攻击
- 硬件木马
- 硬件故障
- 侧信道攻击

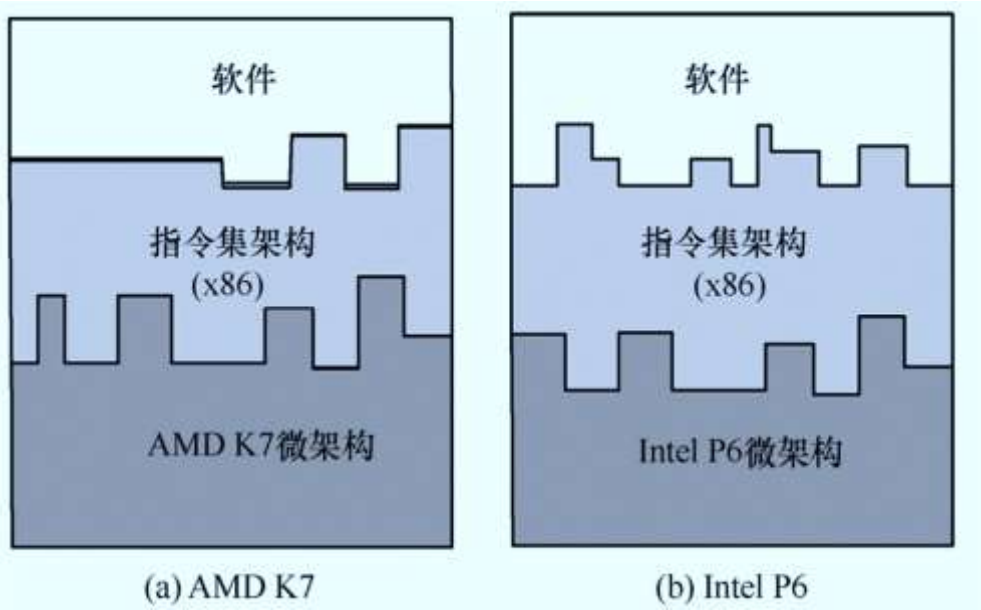
威胁一：设计缺陷

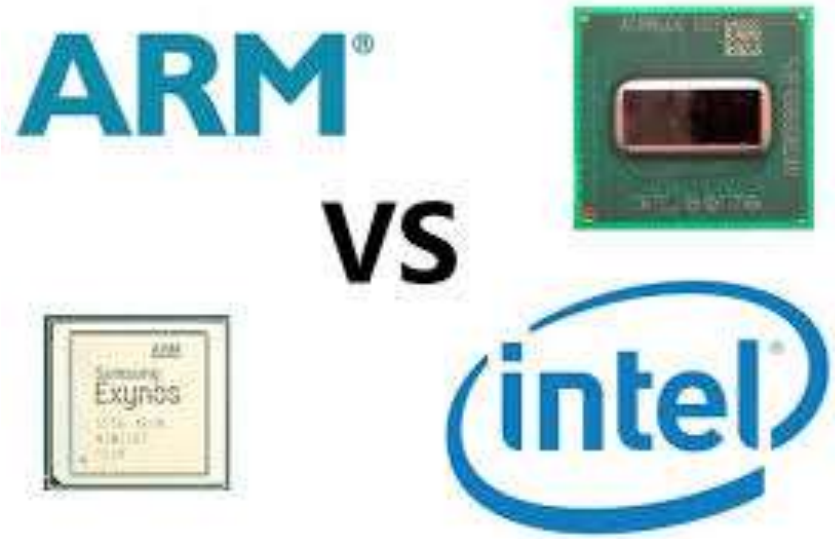
漏洞攻击：利用硬件设计中存在的缺陷

硬件漏洞

硬件漏洞主要指硬件设计和生产中存在的缺陷，这些缺陷大多由设计说明不完善、设计人员编码不规范、硬件性能优化、测试与验证覆盖率不够高等因素引起

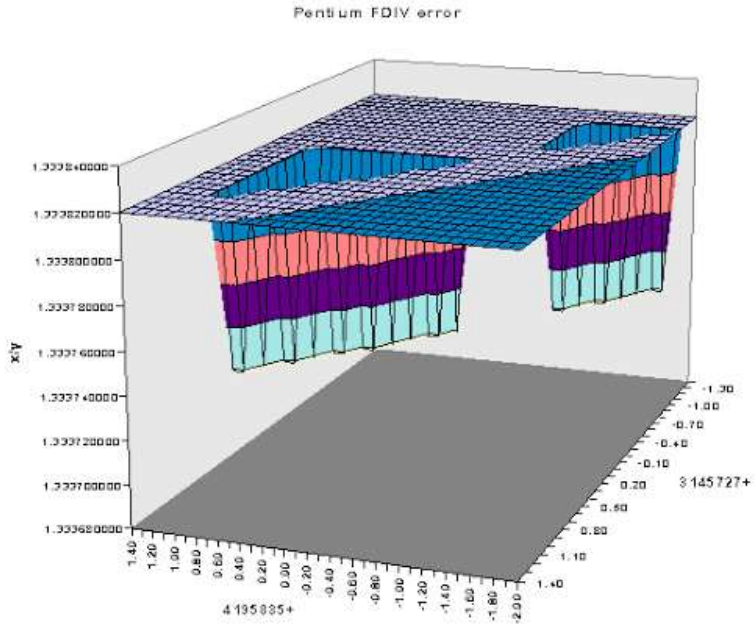
指令集漏洞





- 不同系统架构对相同指令集的实现方式有所不同
- 指令集漏洞是**设计人员编码不规范**的典型代表,通常是由于处理器开发人员在指令集的硬件实现过程中，对一些特殊情况考虑不周导致的。各大处理器厂商会定期发布产品的勘误表。Intel每个版本的勘误信息平均有150条左右，ARM平均每个版本有20条左右

浮点除(FDIV)的实现漏洞



- 为提高运算速度，因特尔在奔腾第五代x86处理器中嵌入了一个**查询表**，用于计算迭代浮点除法所必需的中间商。但是2048个乘法数字中，有5个输入错误

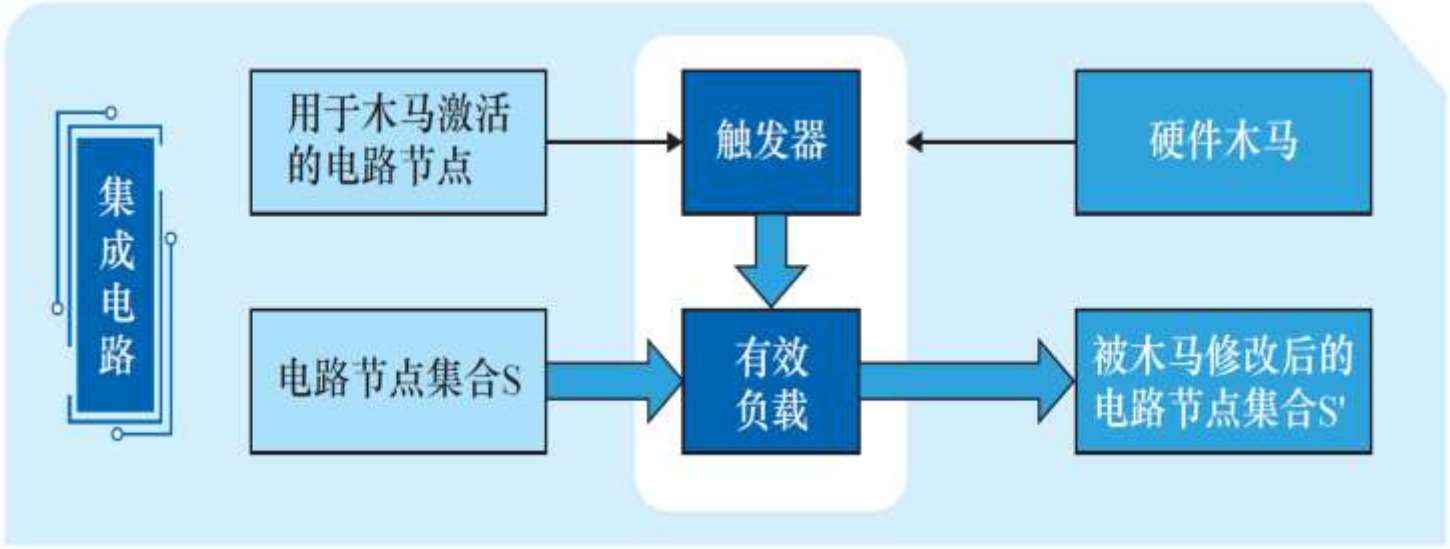
- 此图是在具有FDIV（Floationg Point Divide，浮点除）误差的奔腾上计算出的比率4195835/3145727的3D图。凹陷的三角形区域指示已计算出错误值的位置。正确的值全部取整为1.3338，但是返回的值是1.3337

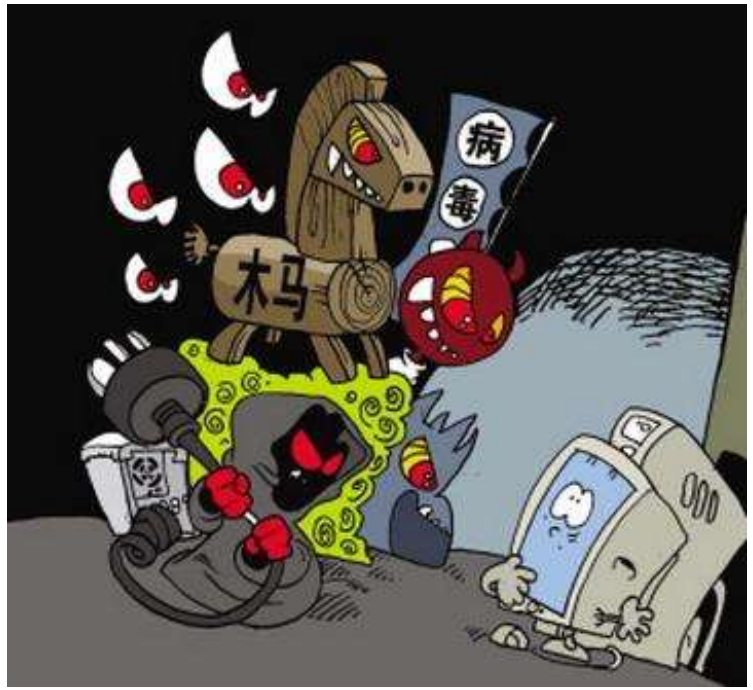
性能优化缺陷

Meltdown和Spectre源于**乱序执行和分支预测**（提高CPU性能）

威胁二：硬件木马

硬件木马是指被第三方故意植入或设计者有意留下的特殊模块和电路，硬件木马一般由触发器和有效负载两部分组成。这些模块或电路潜伏在正常硬件中，仅在特定条件下被触发，实施恶意功能（如窃取敏感信息、拒绝服务等）



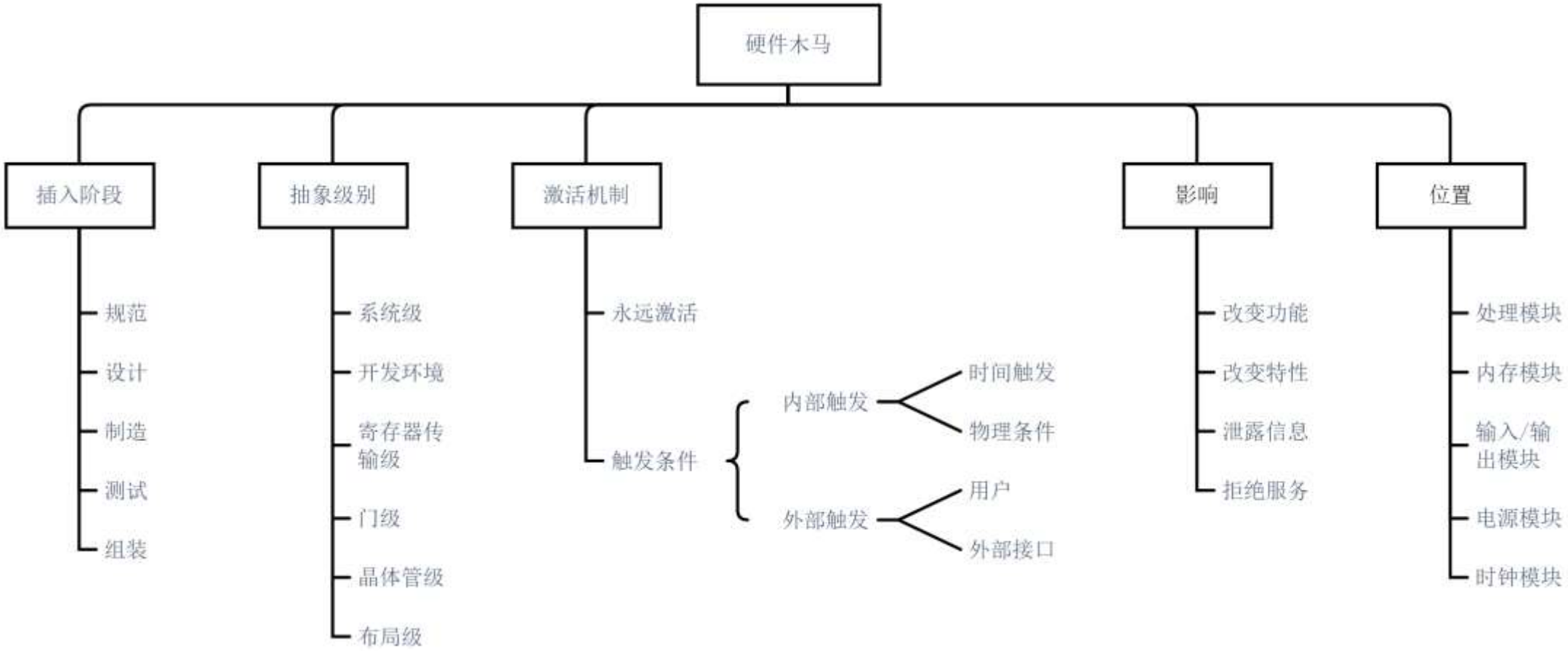


不可信的产业链

硬件木马通常由不可信合作厂商引入，目前我国的高端服务器、操作系统等核心软硬件大部分进口，木马和后门问题不容乐观

木马分类

- 硬件木马可以基于5个特征来分类：**插入阶段、抽象级别、激活机制、影响和位置**



木马≠故障

- **电路故障**是指在电路的生产加工过程中由于工艺或者流程不完美造成的电路缺陷
- **硬件木马**是指由人为蓄意加入的指定功能之外的恶意电路模块。电路故障通常能够被检测到而硬件木马的检测比较困难

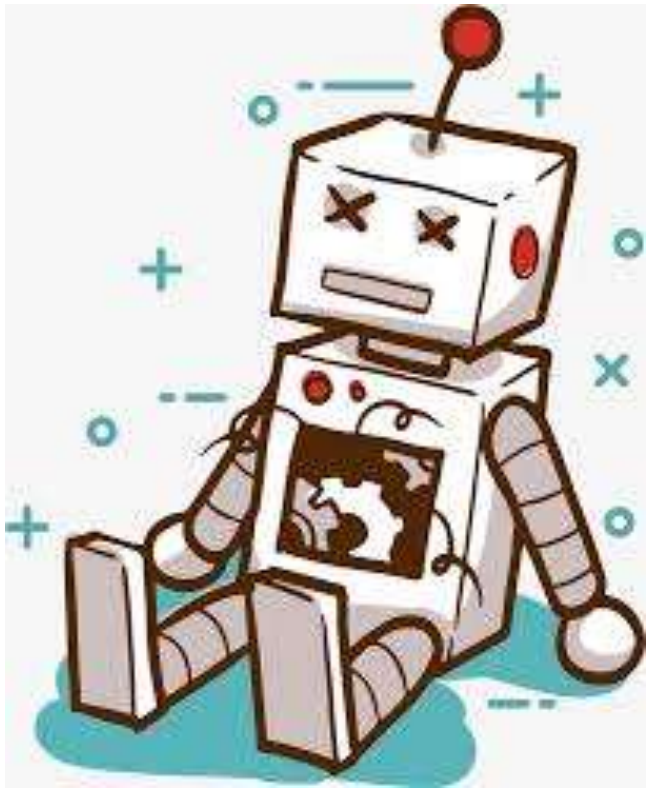
	电路故障	硬件木马
激活	通常在已知的功能状态下	任意电路中间节点特定状态的组合或者序列（可以为数字或者模拟信号）
植入	偶然（制造过程的某些缺陷）	蓄意（由产业链的攻击者植入）
作用	电路功能或参数错误	电路功能参数错误、泄漏信息等

软件木马与硬件木马

- **软件木马**是一种带有恶意代码的软件，与硬件木马攻击目标相似，软件木马通常可以在应用领域内解决，而硬件木马一旦植入就很难移除

	软件木马	硬件木马
激活	存在恶意代码的软件，在代码运行过程中激活	存在集成硬件中，在硬件运行过程中激活
感染	用户交互过程中传播	由集成硬件设计流程中不可信的参与方植入
补救措施	通过软件更新移除	流片一旦加工成型就不能移除

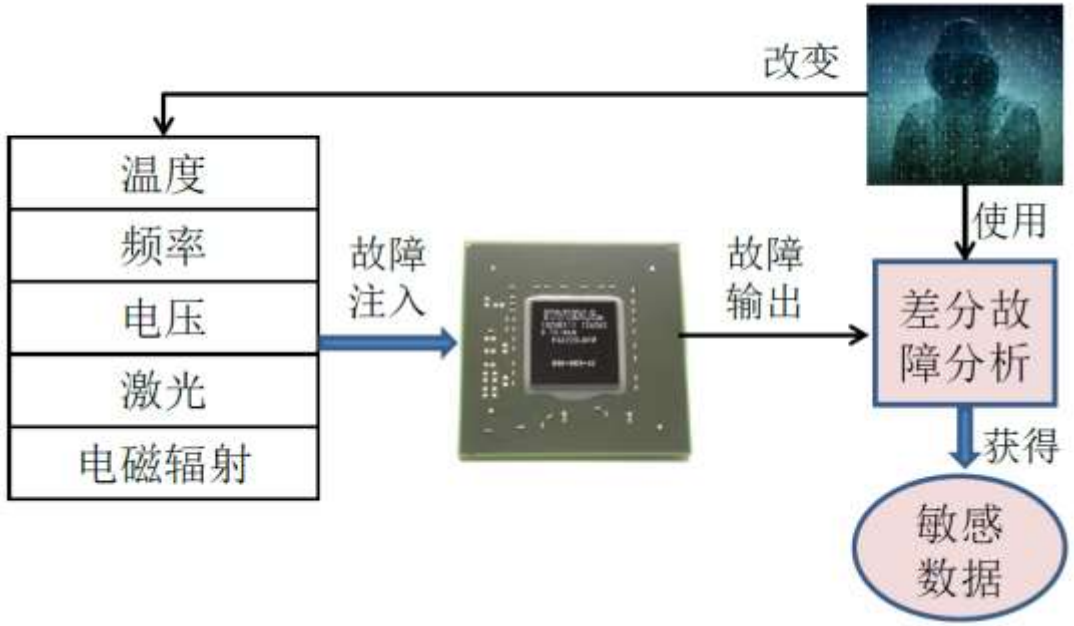
硬件故障



硬件故障是常见硬件问题，攻击者利用合适的硬件故障就能窃取敏感信息、实施硬件攻击

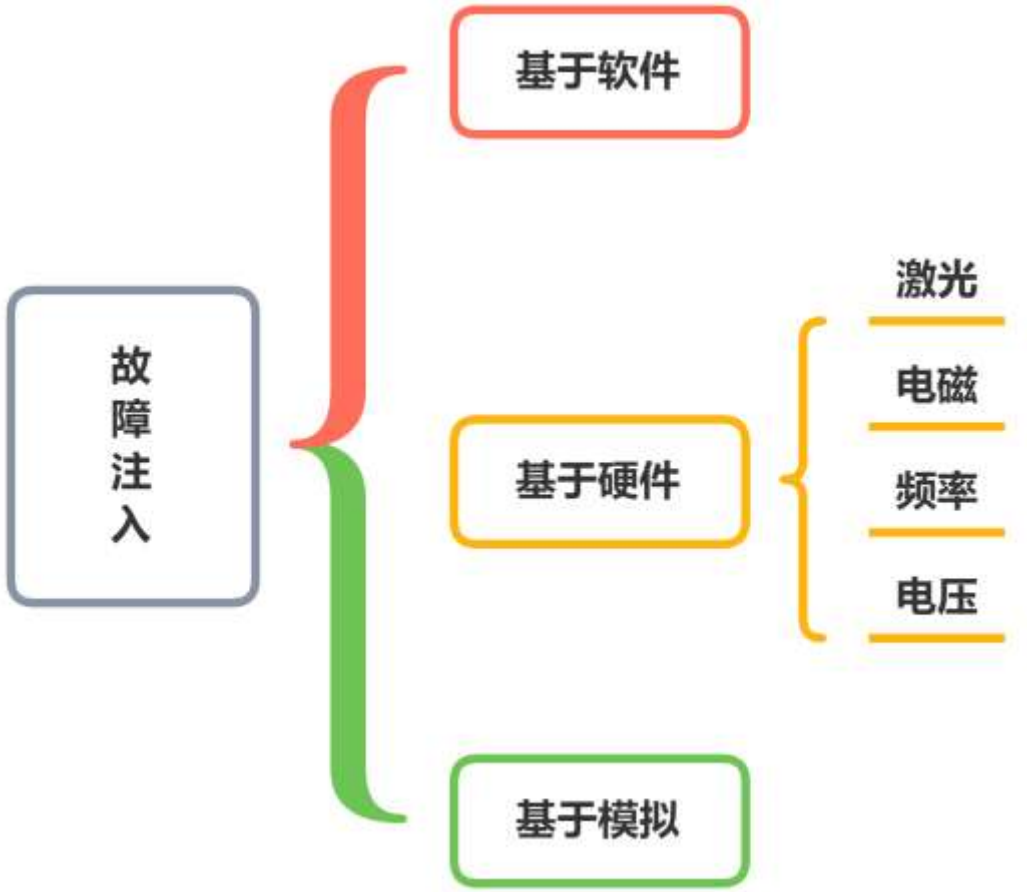
威胁三：故障注入攻击

故障注入技术最初被用来验证系统的可靠性和可用性，并在电子器件的整个制造过程中得到广泛使用，主要做法是试图通过受控实验改变正在运行的系统的工作环境，从而引入故障，并根据系统的工作状态评价系统的可靠性和可用性



故障注入检测→故障注入攻击

故障注入分类



基于硬件的故障注入攻击**实现简单**、**成功率比较高**、故障注入过程也**比较可控**，是一类被广泛研究的故障注入攻击方法

光故障注入

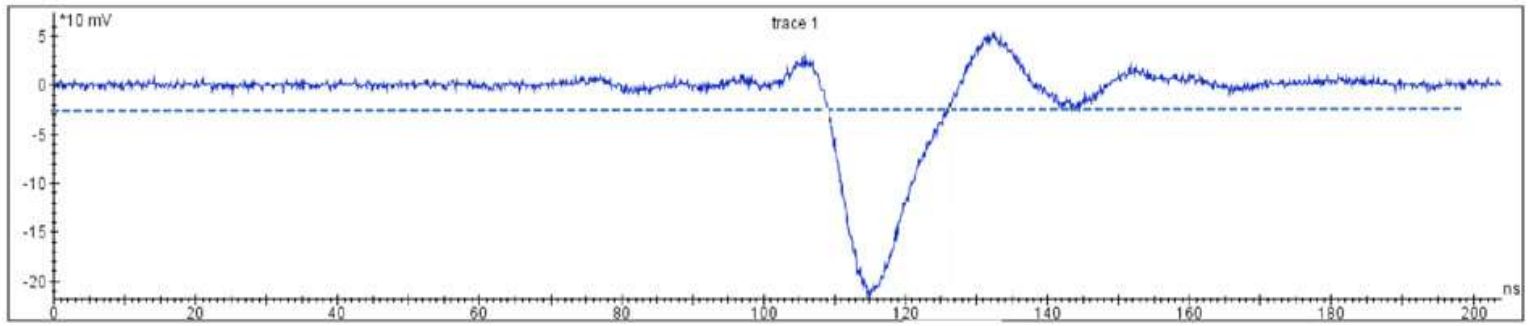
- **集成电路芯片**的主要材料是基于硅材料掺杂的N型半导体和P型半导体。光故障注入利用光电效应来激发半导体硅片产生大量的自由电子和空穴，造成晶体管的导通，从而对被攻击的电子设备引入故障

- 2019年11月6日，据国外媒体报道，最新研究发现，通过**将激光调至精确频率**并对准智能语音设备，其可以像用户声音一样激活语音助理并进行交互，从而解锁汽车、打开车库门等等。这种方法的作用距离甚至可以达到**一百多米**

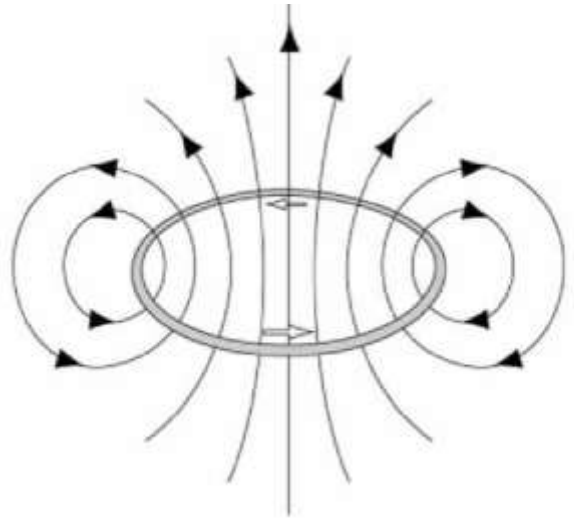
[1]https://www.163.com/tech/article/ET9HOLFQ00097U80.html

电磁故障注入

- CMOS工艺特征尺寸的不断缩小，一方面降低了芯片的电源电压和功耗，提高了芯片的工作频率，另一方面导致芯片的电磁抗扰度较低，对电磁攻击更为敏感



待攻击芯片上所产生的瞬态感应电压波形



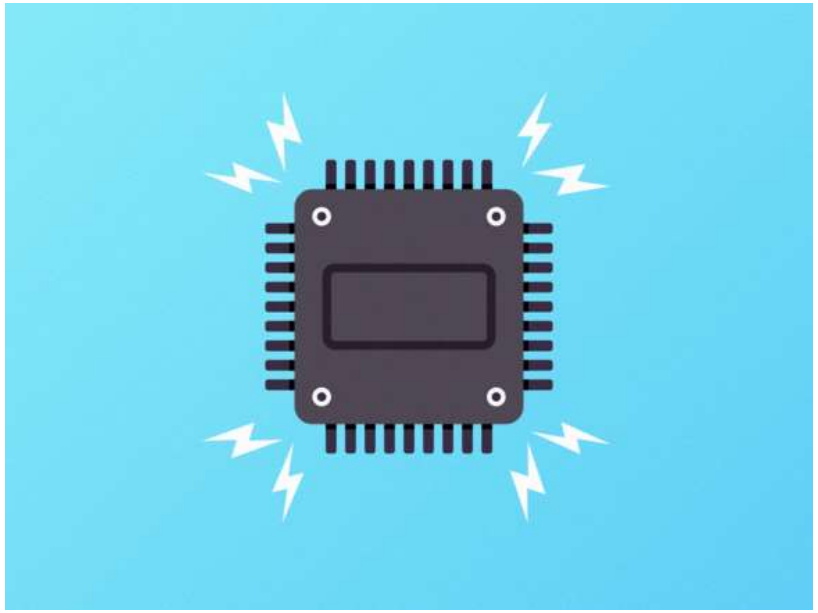
电磁故障注入产生的电磁场会对待攻击芯片的内部引入毛刺信号

芯片的电磁故障注入

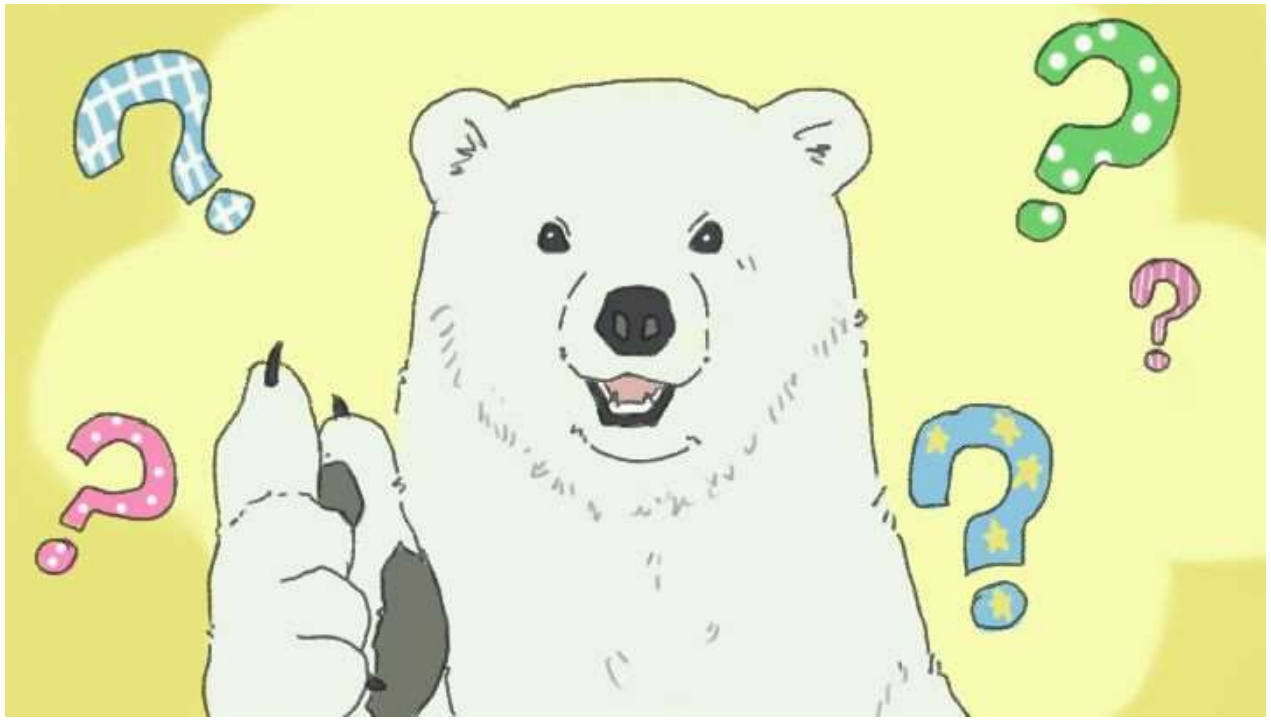
单粒子翻转现象

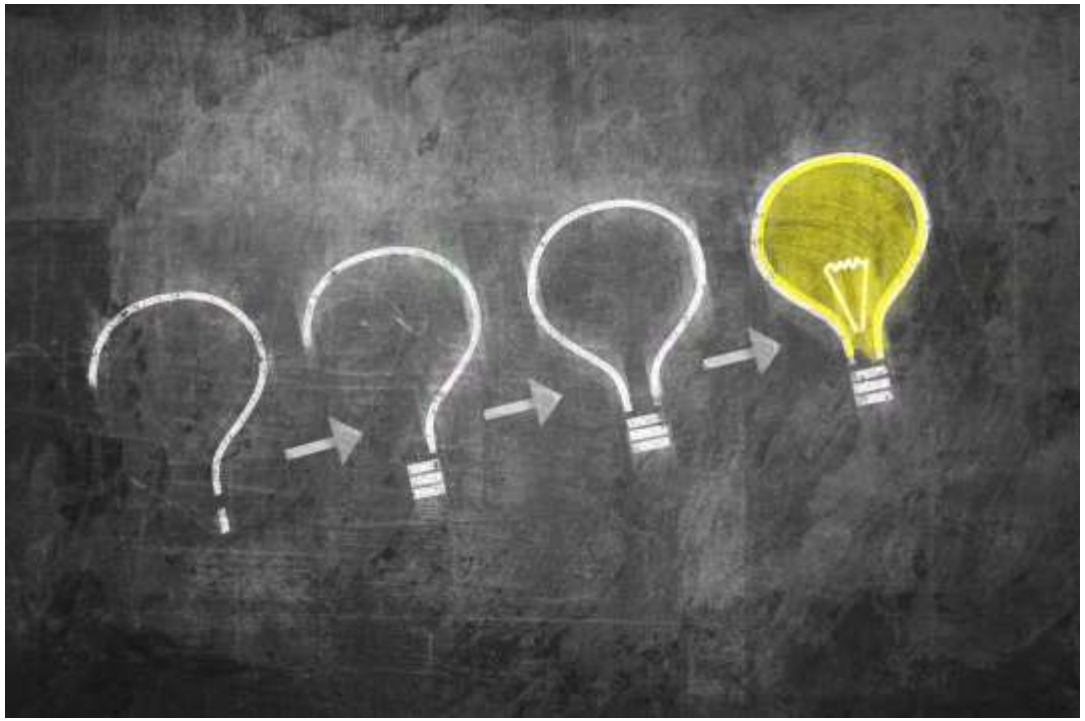
- 进行攻击实验时，芯片内部产生的瞬态感应电压毛刺的极性和晶体管的类型均会导致芯片内部晶体管状态（导通或截止）的变化
- 感应电压毛刺只会将一对 P 沟道和 N 沟道晶体管中的一个晶体管切换到导通状态，而不会导致芯片内电源线和地线的短路

- 红气球安全公司创始人崔昂，研究科学家里克·豪斯利，展示了黑进处理器芯片的一种新方法——**用电磁脉冲在硬件中产生特定短时脉冲波形干扰**。通过在精确的时间间隔干扰正常活动，可导致防止处理器运行不可信代码的安全启动(Secure Boot)保护失效



更高级的故障注入

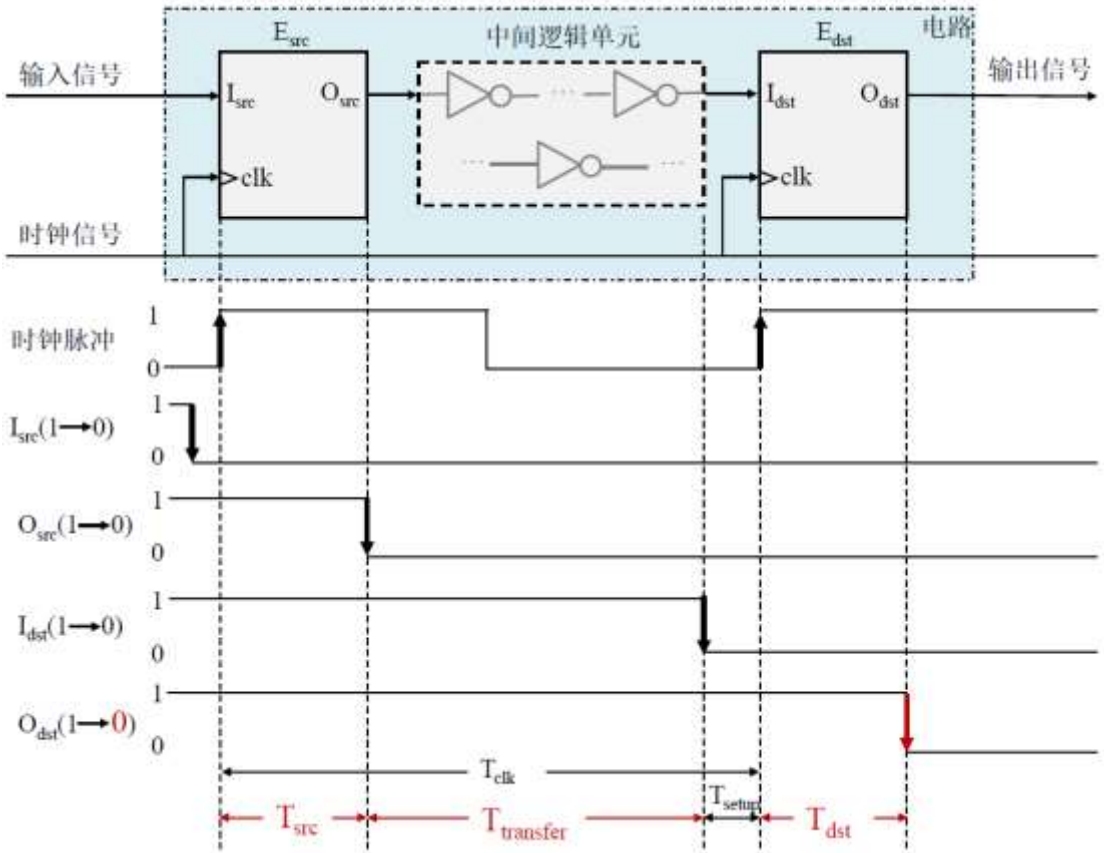




有没有更高级的故障注入方式?

集成**时序电路**的约束条件?

时序电路



输出信号受**输入信号**和**时钟脉冲**共同影响

- ***Tclk***：表示一个时钟周期，是两个时钟上升沿的间隔，也反映了电路的频率
- ***Tsetup***：表示最后一个时序电子元件在处理输入数据时输入数据必须要保持稳定的时间，也是中间逻辑单元的输出到下一个时钟上升沿需要满足的间隔
- ***Tsrc***：表示第一个时序电子元件的输入和输出之间的延时，也即收到时钟上升沿到给出稳定输出之间的时间
- ***Ttransfer***：表示第一个时序电子元件的输出到中间逻辑单元的输出之间的间隔，也即中间逻辑单元的处理时间

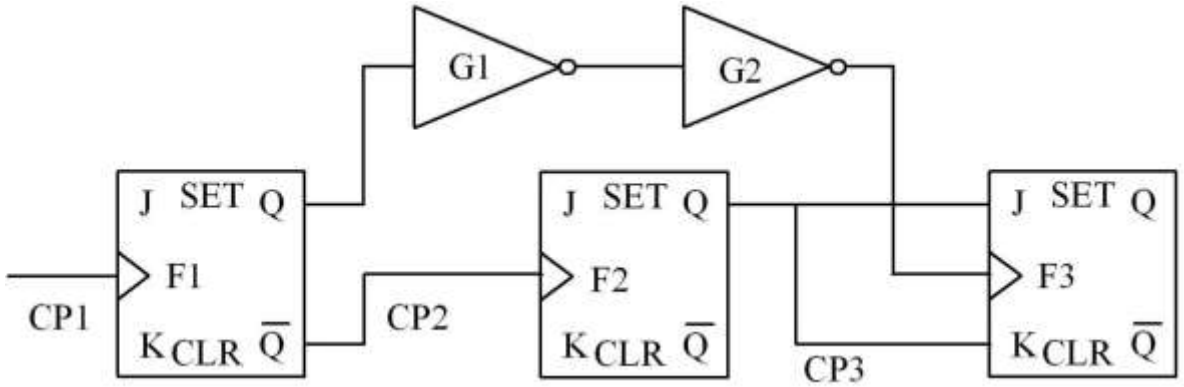
时序电路的约束

- 一个时序电路通常包括**多个**电子元件，这些电子元件在统一的时钟脉冲控制下运行，为了使电子元件稳定运行，每个电子元件需要在输入信号稳定后再开始处理输入数据
- 电子元件的输入和输出之间存在**延时**。因此，时序电路需要满足一定的时间约束才能保证各个电子元件的协调一致运行，**调试时间约束**也是设计时序电路时重要的一步

$$T_{src} + T_{transfer} \leqslant T_{clk} - T_{setup} - T_{\epsilon}$$

竞争冒险

- **竞争：**由于信号在传输和处理过程中经过不同的逻辑门、触发器或逻辑单元时产生时差，造成信号的原变量和反变量状态改变的时刻不一致，这种现象称为竞争
- **冒险：**由于竞争而引起电路输出信号中出现了非预期信号，产生瞬间错误的现象称为冒险



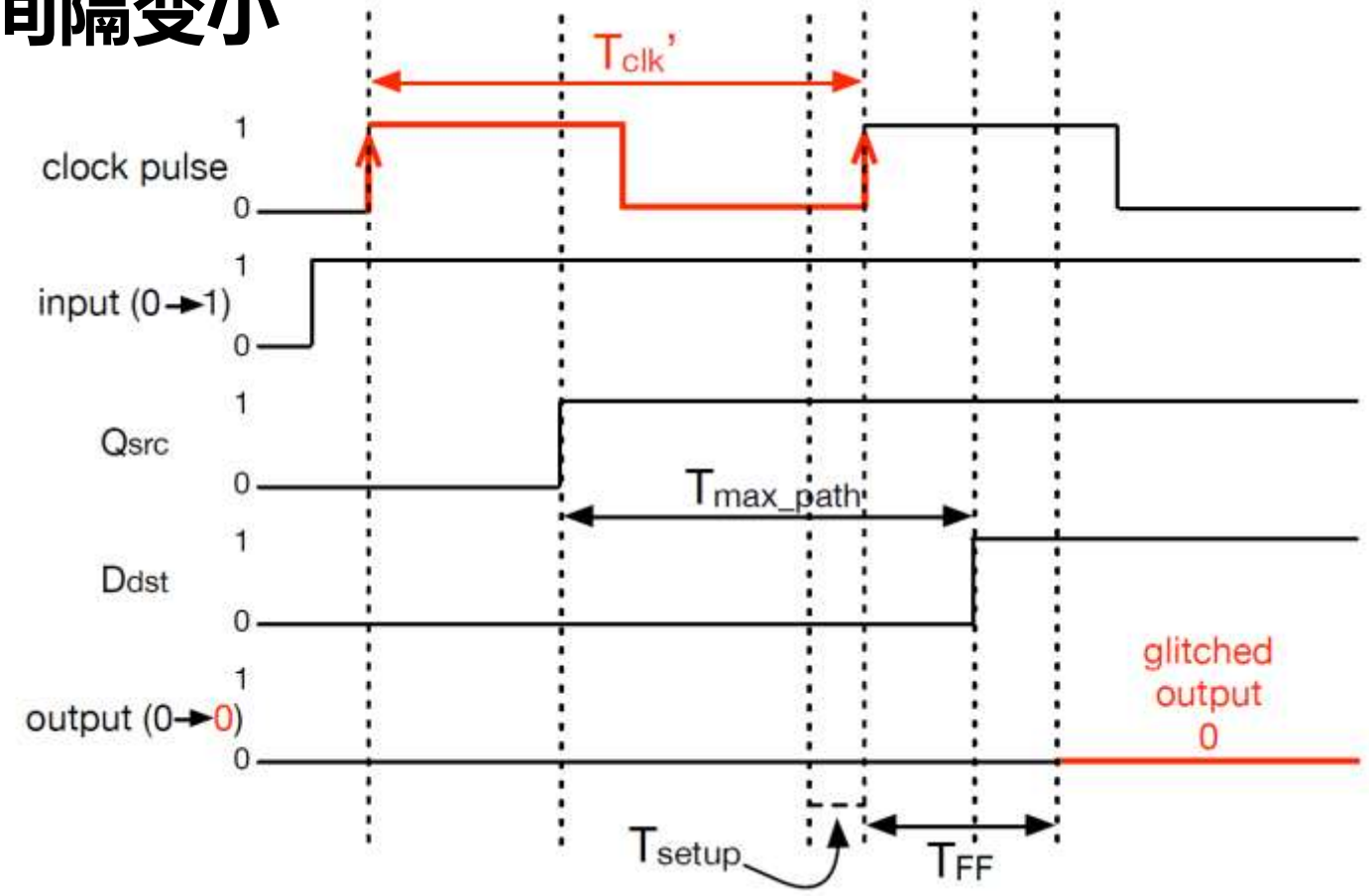
F3和CP3状态改变的现后顺序不确定 → 电路功能不稳定

异步时序电路中，当输入信号和时钟信号发生同时改变时，而且是通过不同路径到达同一触发器时，就有可能导致竞争冒险

同步时序电路不存在竞争现象，除非**恶意引入**

破坏时序电路约束：提高频率

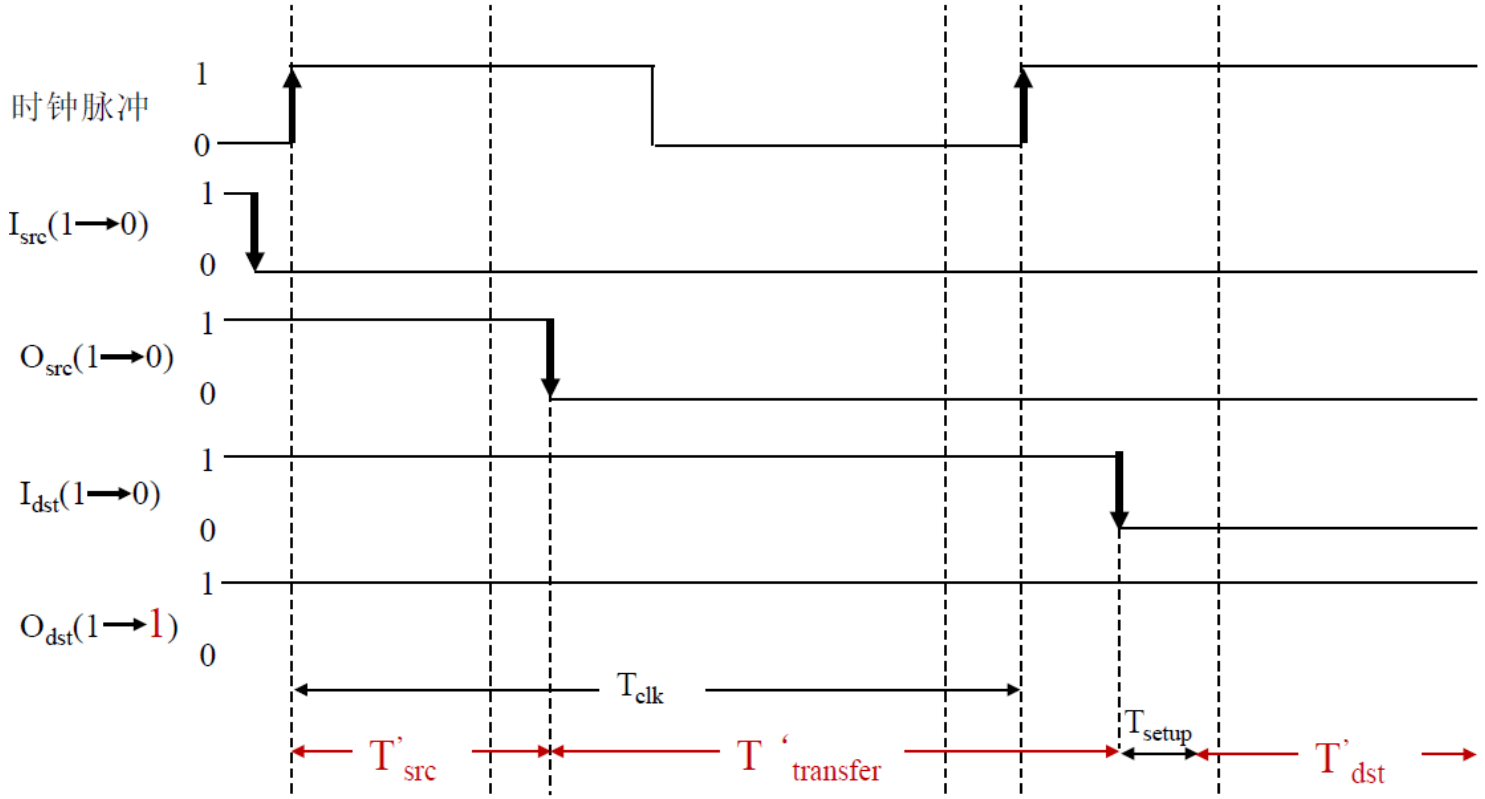
脉冲间隔变小



- 约束右侧变小
- 存储电路来不及充放电
- 输出依然保留上次输出值

破坏时序电路约束：降低电压

电路延时变大

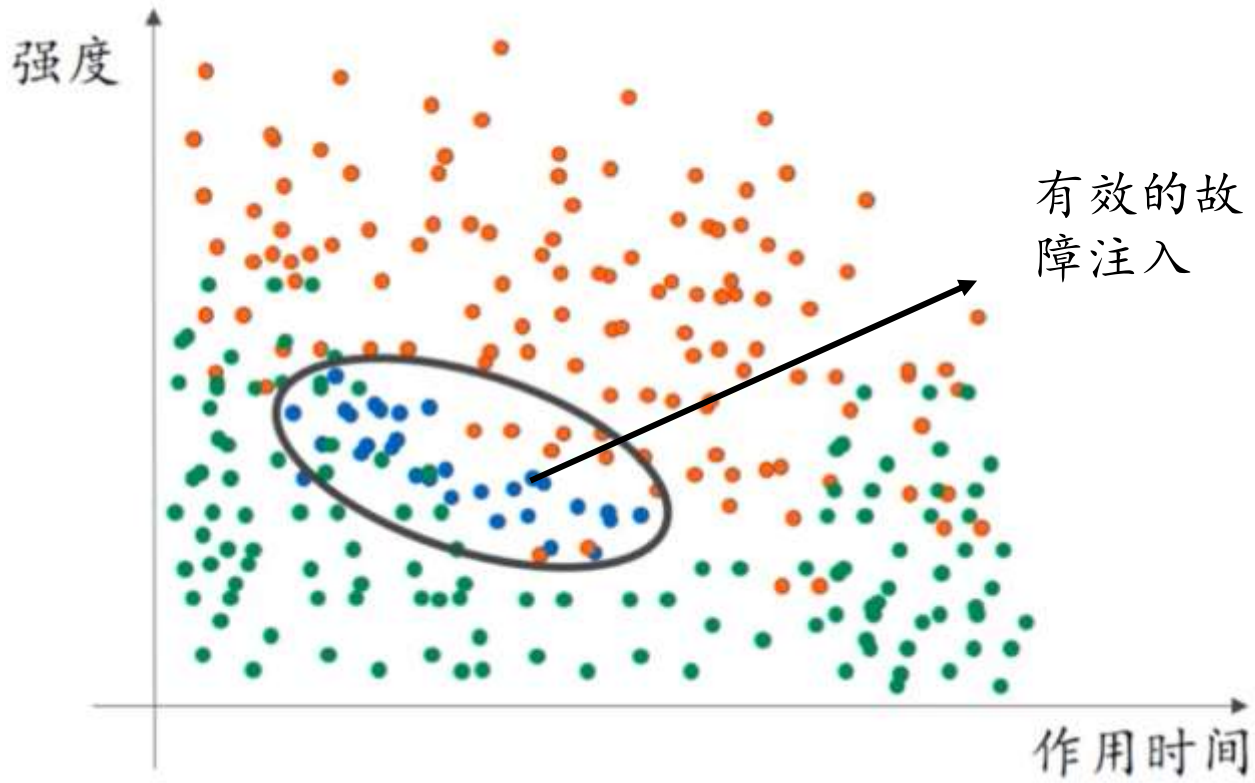


- 存储电路充放电速度下降
- 电压变低→延时变大→约束左侧变大
- 输出依然保留上次输出值

影响因素

故障注入攻击因素

- 攻击时刻
- 攻击强度
- 作用时间
- 空间位置



威胁四: 旁路侧信道

- 故障注入攻击
- 侧信道攻击

侧信道攻击：利用硬件系统的旁路信息实施攻击

侧信道——不接触攻击

安全吗

真的安全吗

你确定安全吗

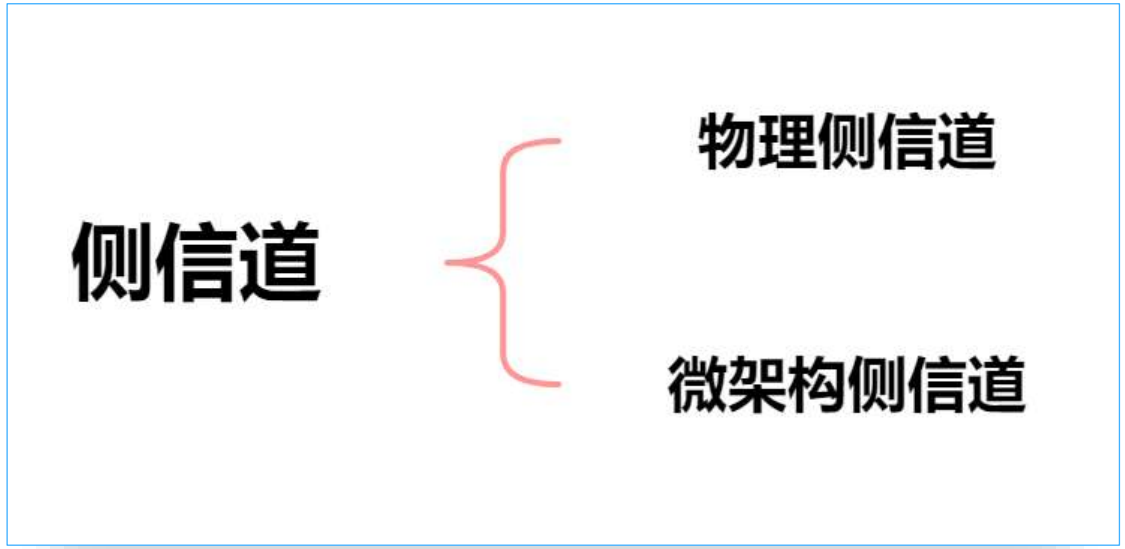
你真的确定安全吗

硬件隔离能保证安全吗

侧信道攻击

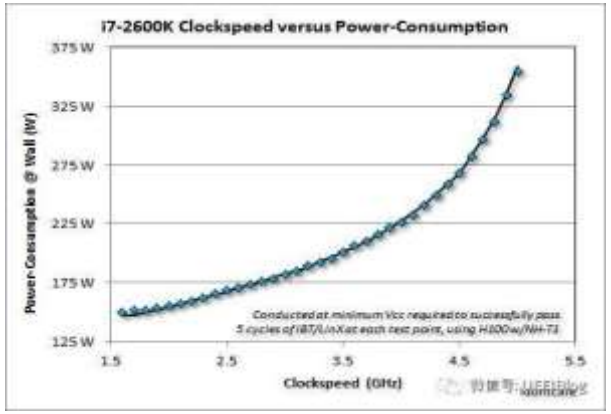
侧信道攻击

- **侧信道攻击**（SCA，Side Channel Attack）：基于从密码系统的物理实现中获取的信息，例如时间信息、功率消耗、缓存使用等
- 侧信道攻击需要的**设备成本低**、**攻击效果显著**，严重威胁了密码设备的安全性

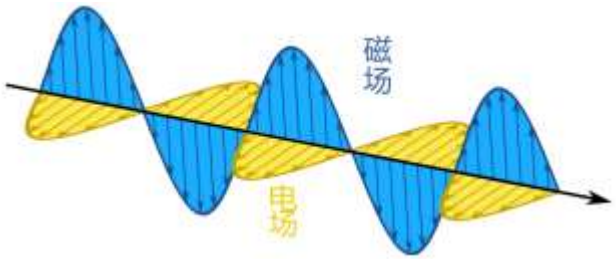


物理侧信道

时间侧信道



功耗侧信道



电磁侧信道

声波侧信道

时间侧信道示意

时间侧信道通过系统在不同行为流程下所表现出的执行时间差异，推断有用的信息，以达成或辅助攻击

- 假如一个网站使用如下代码

```
$username = $this->input->post("username")
$password = $this->input->post("password")
if ($this->is_valid_user($username) == false) {
        echo "Invalid login";
        return;
}
if ($this->verify_password($username, password) == false) {
        echo "Invalid login";
        return;
}

echo "Hello, " . $username;
```

该段代码的逻辑如下：

1. 若用户名不存在，则登录失败
2. 若用户名存在但密码不正确，则登录失败
3. 只有当用户名和密码都正确，才返回成功

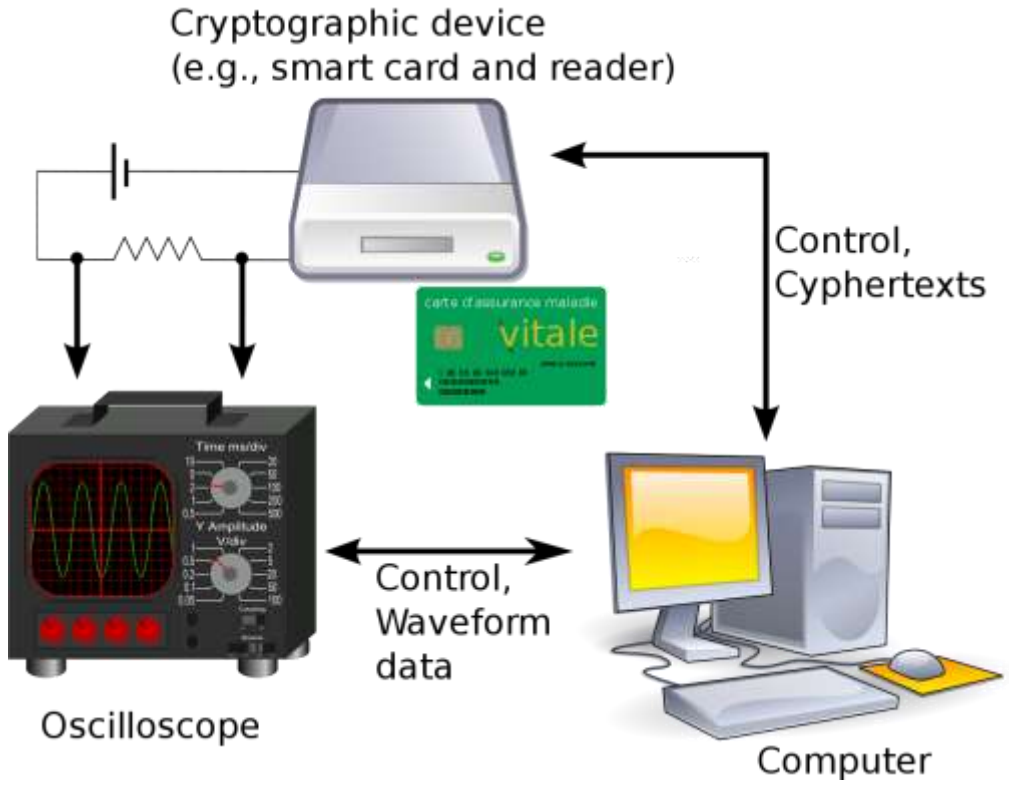
时间侧信道攻击

- 以上代码的逻辑没有问题，并且两次错误的返回的字符串信息一致，避免了攻击者根据该信息判断出用户名是否存在，而进行后续的针对性的攻击
- **真的没有问题了吗**？考虑下面的攻击方式：

① 攻击者从本地提交post请求，其中，变化username，而保持password为任意固定值，例如“666”
② 攻击者统计每次从发出请求起到收到服务器回执所耗费的时间，对于同一组用户名和密码，也可以通过测量多次取平均值、中值等方法消除网络延迟引入的时间抖动
③ 对于统计时间较长的请求，认为其所对应的用户名是存在的
④ 基于存在的用户名，进一步发动针对性的攻击

功耗侧信道攻击

- 功耗侧信道分析最早是由Kocher等提出
- CPU 执行不同指令的时候会有不同的功耗特征：不同的指令触发的半导体数量不同，有的指令还会访问内存、缓存等等，各种各样的因素会导致不同的指令执行的时候会产生不同的功耗特征
- 利用不同功耗特征可以推测出CPU执行过程中的某些信息



功耗侧信道攻击可分为以下三种：

简单功耗分析 → **差分功耗分析** → **相关功耗分析**

简单功耗分析

- **简单功耗分析（simple power analysis, SPA）：**攻击者通过直接分析设备随时间的功耗曲线，以及曲线特征和分析者经验可直观得出CPU执行的顺序或指令

受害者芯片

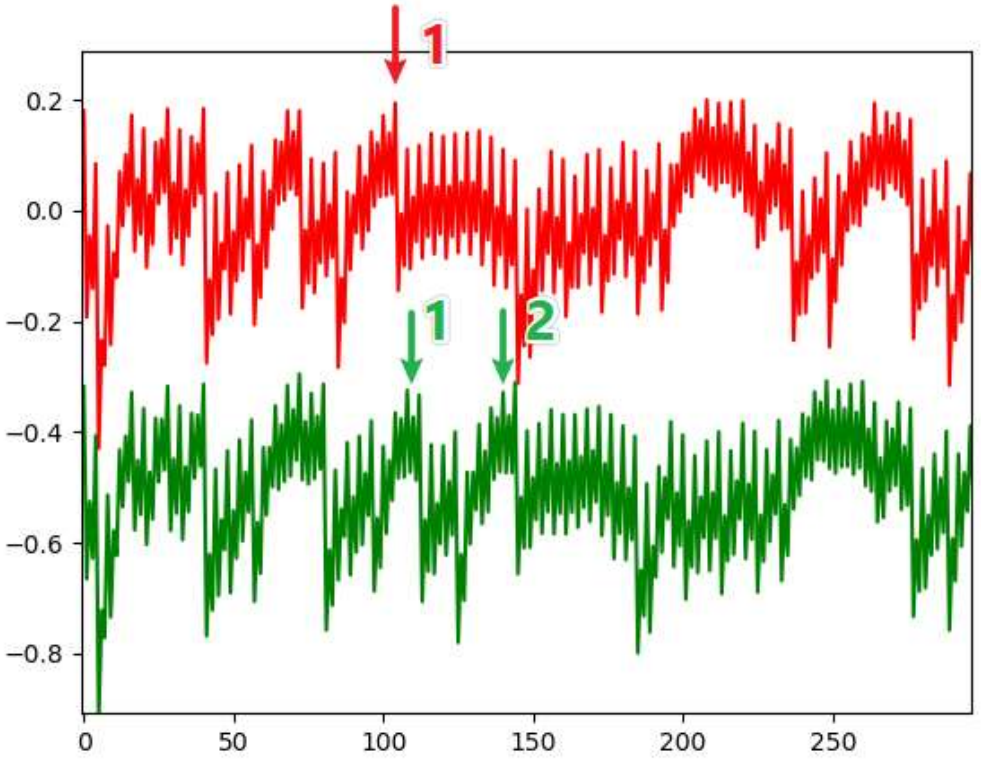
1. 从串口读取一行用户输入的密码
2. 跟固件中一个写死的密码 h0px3 进行比较
3. 向串口输出匹配结果

攻击者

通过硬件接口监测受害者芯片电压（功耗）变化情况，依次输入两个字符串“a\n”和“h\n”

观察两次功耗曲线的差异

简单功耗分析示意



正确密码：h0px3

两次功耗曲线不一样?

a

- 一次字符对比a≠h，就退出循环

h

- 两次字符对比，第一次h=h；第二次输入不是0，退出循环

密码爆破

密码位数：5位

密码字符：36个（字母+数字）

爆破复杂度：$O(36^5)$

功耗分析

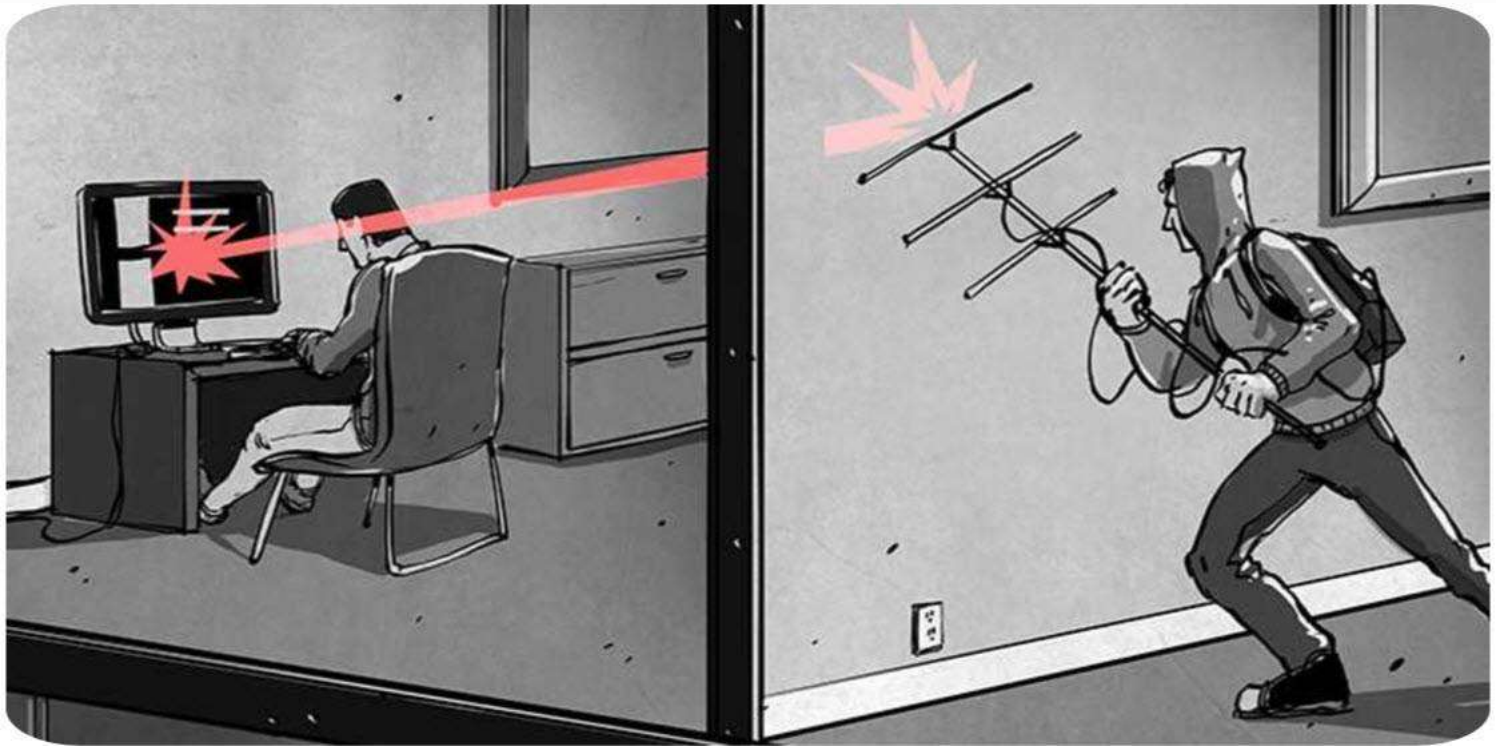
密码位数：5个1位

密码字符：36个（字母+数字）

爆破复杂度：O(36)

电磁侧信道

- 无论是JavaScript亦或是C语言编写的甚至定制硬件电路也适用
- 任何程序都要靠硬件电路来运行实现，由此，也带来一些有趣的副作用：运动电荷激发磁场——詹姆斯·克拉克·马克斯韦尔

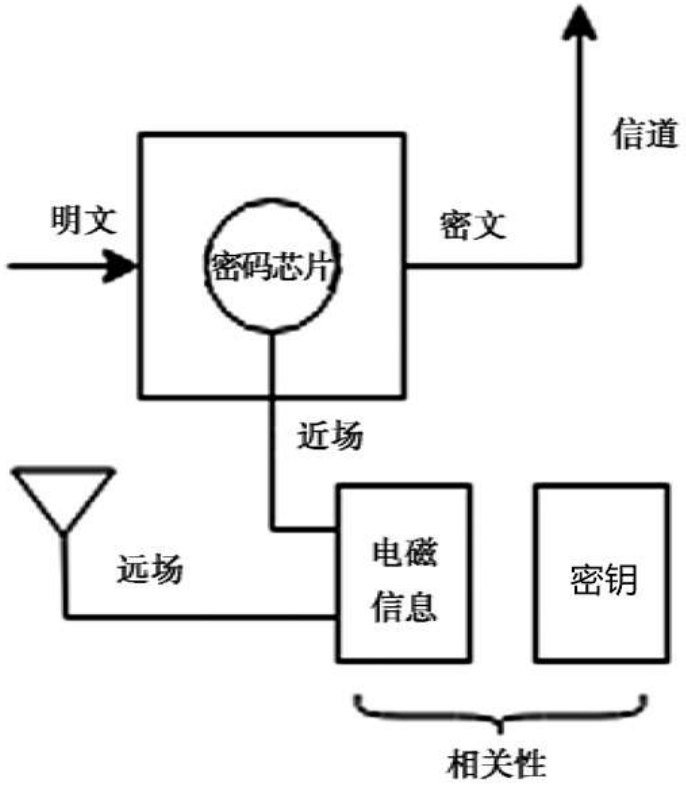


利用电磁场窃取信息

电磁侧信道攻击

- **电磁分析（electromagnetic analysis）攻击**：由比利时学者Quisquater和Samyde于2000年首先提出，与数据相关的电流在处理器中的变化可以导致磁场的变化，攻击者分析磁场进而分析出数据，电磁分析与功耗分析方法类似——非接触式功耗分析

$$d\vec{B} = \frac{\mu I d\vec{l} \times \vec{r}}{4\pi r^2}, \vec{B} = \int d\vec{B}$$



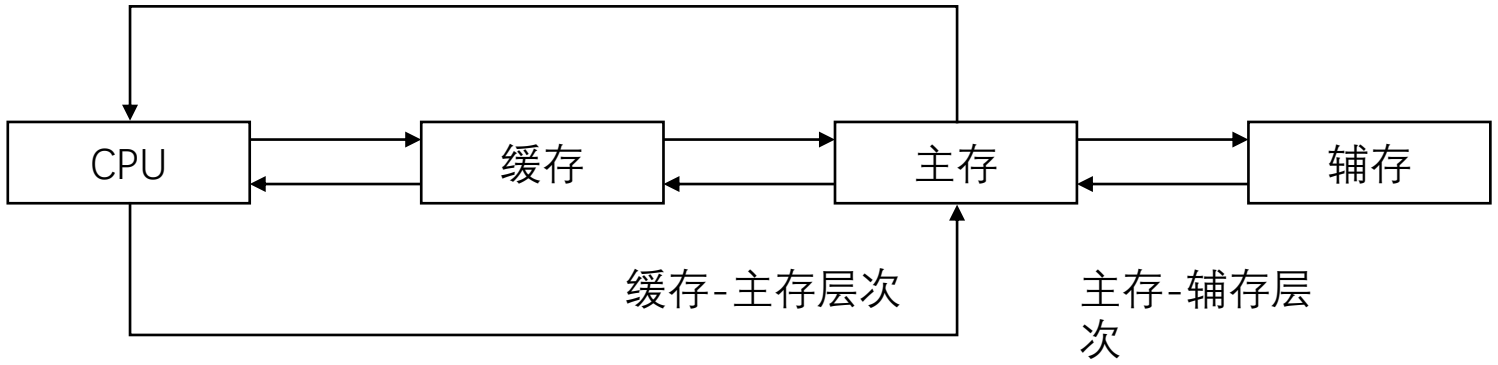
- 密码芯片在工作过程中会**不可避免**的对外产生**电磁辐射**，辐射的信息和芯片内部的数据存在一定的相关性

微架构侧信道

- 现代处理器的优化技术，包括乱序执行和推测机制等，对性能至关重要
- 随着处理器性能的不断提升，与之相关的漏洞也越来越多
- 以Meltdown和Spectre为代表的侧信道攻击表明：由于异常延迟处理和推测错误而执行的指令结果虽然在架构级别上未显示，但仍可能在处理器**微架构状态**中留下痕迹，通过隐蔽信道可将微架构状态的变化传输到架构层，进而恢复出秘密数据

Cache

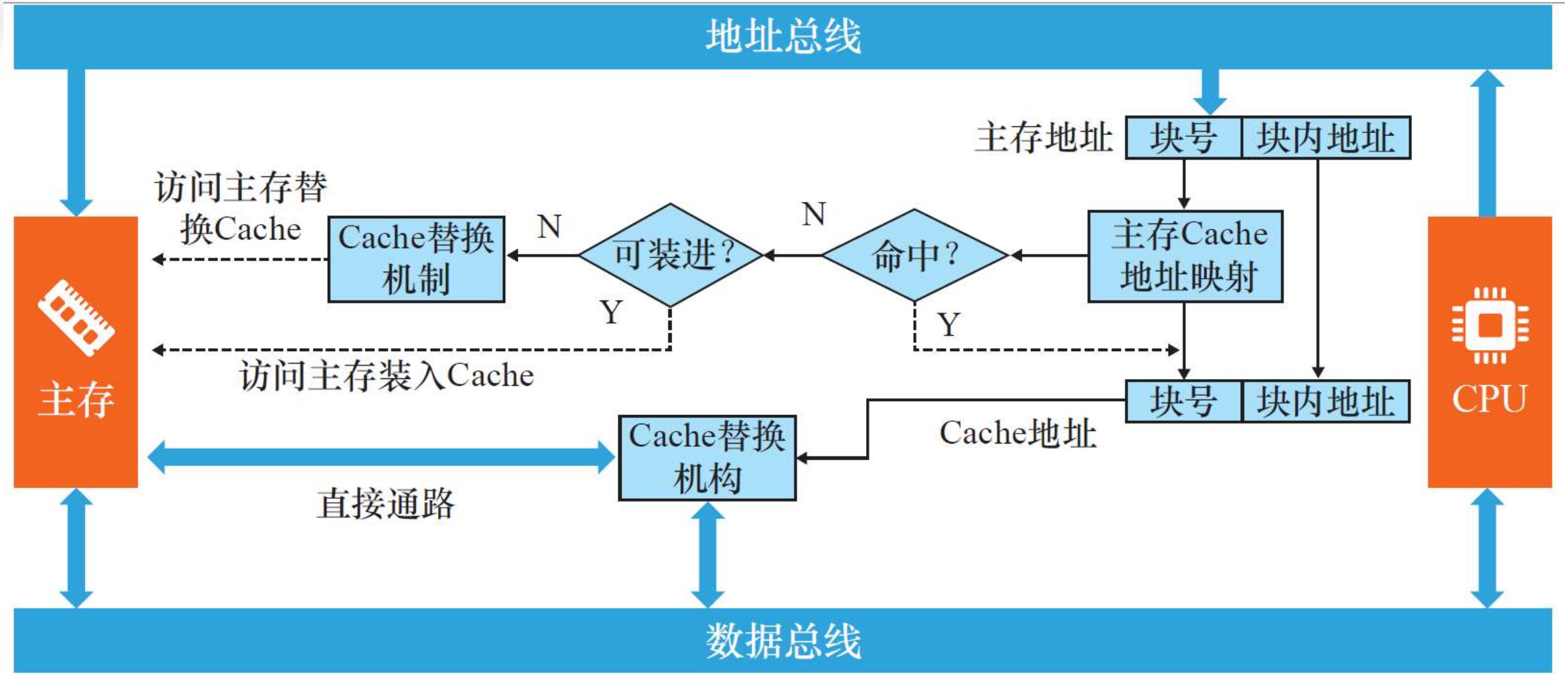
存储系统层次结构



Cache组成

Cache存储体	**存放由主存调入的指令与数据块**
地址转换部件	**建立目录表以实现主存地址到缓存地址的转换**
替换部件	**在缓存已满时按一定策略进行数据块替换，并修改地址转换部件**

Cache工作原理

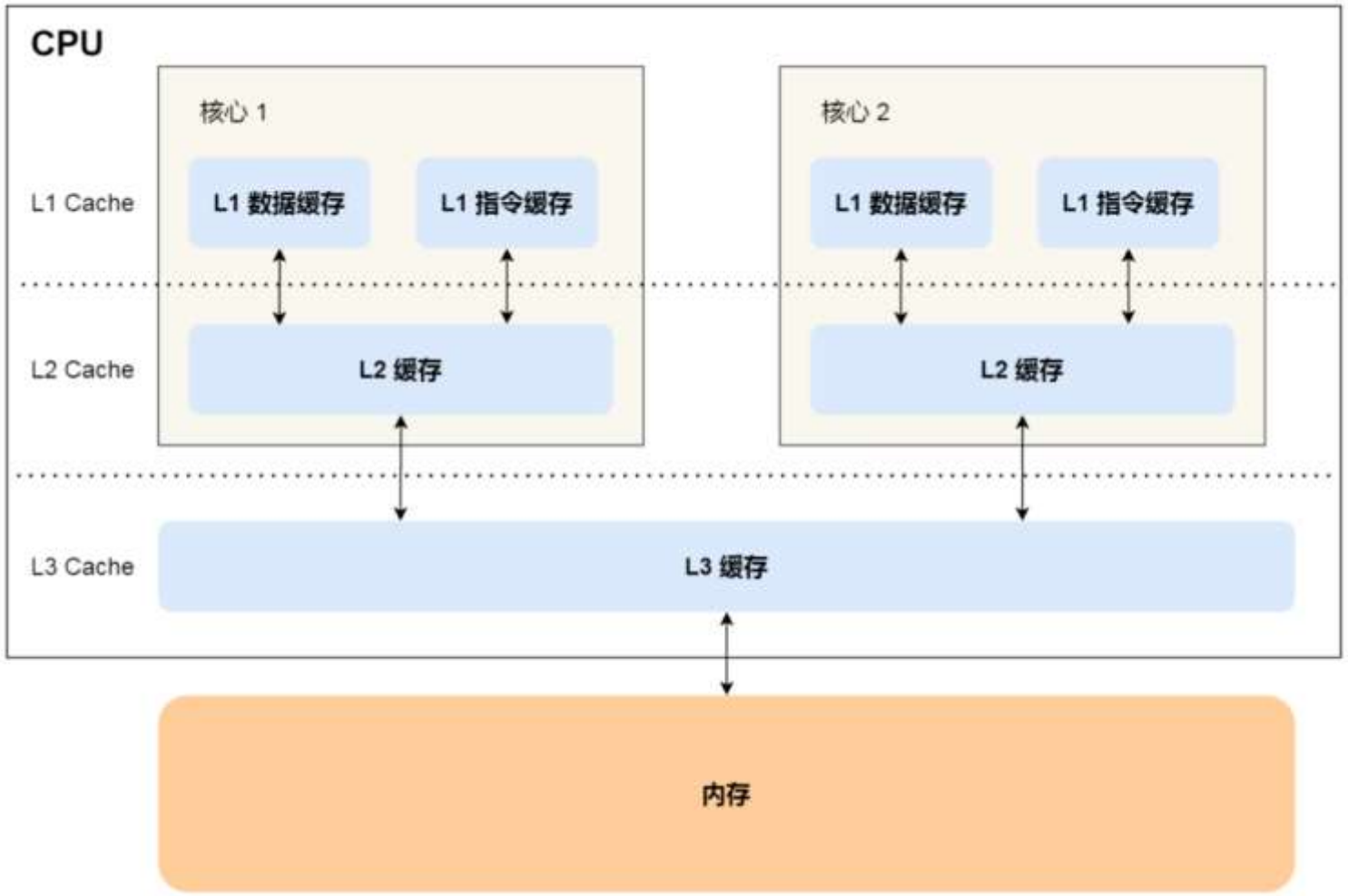


Cache基本结构

Cache结构

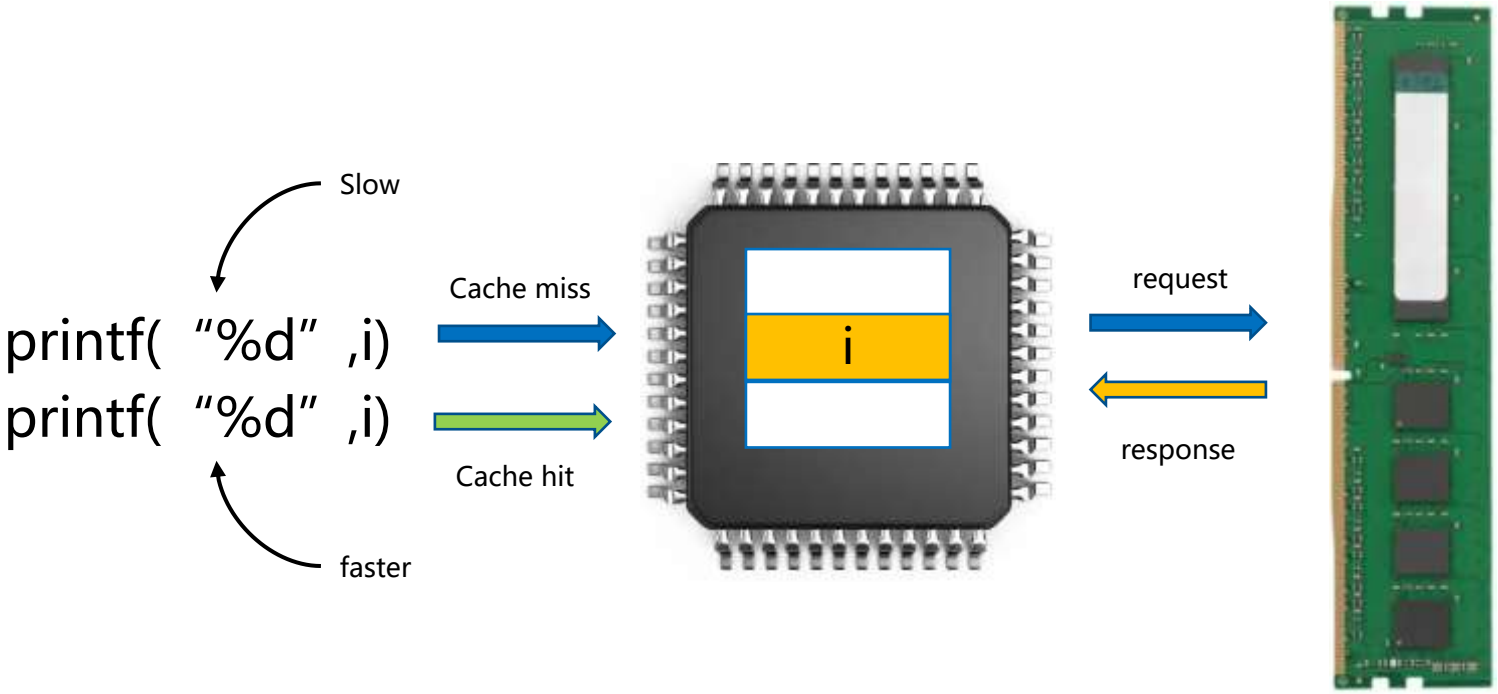
部件	CPU 访问所需时间
L1 Cache	2~4 个时钟周期
L2 Cache	10~20 个时钟周期
L3 Cache	20~60 个时钟周期
内存	200~300 个时钟周期

- 数据与指令分离
- L1、L2单核独有
- L3多核共享

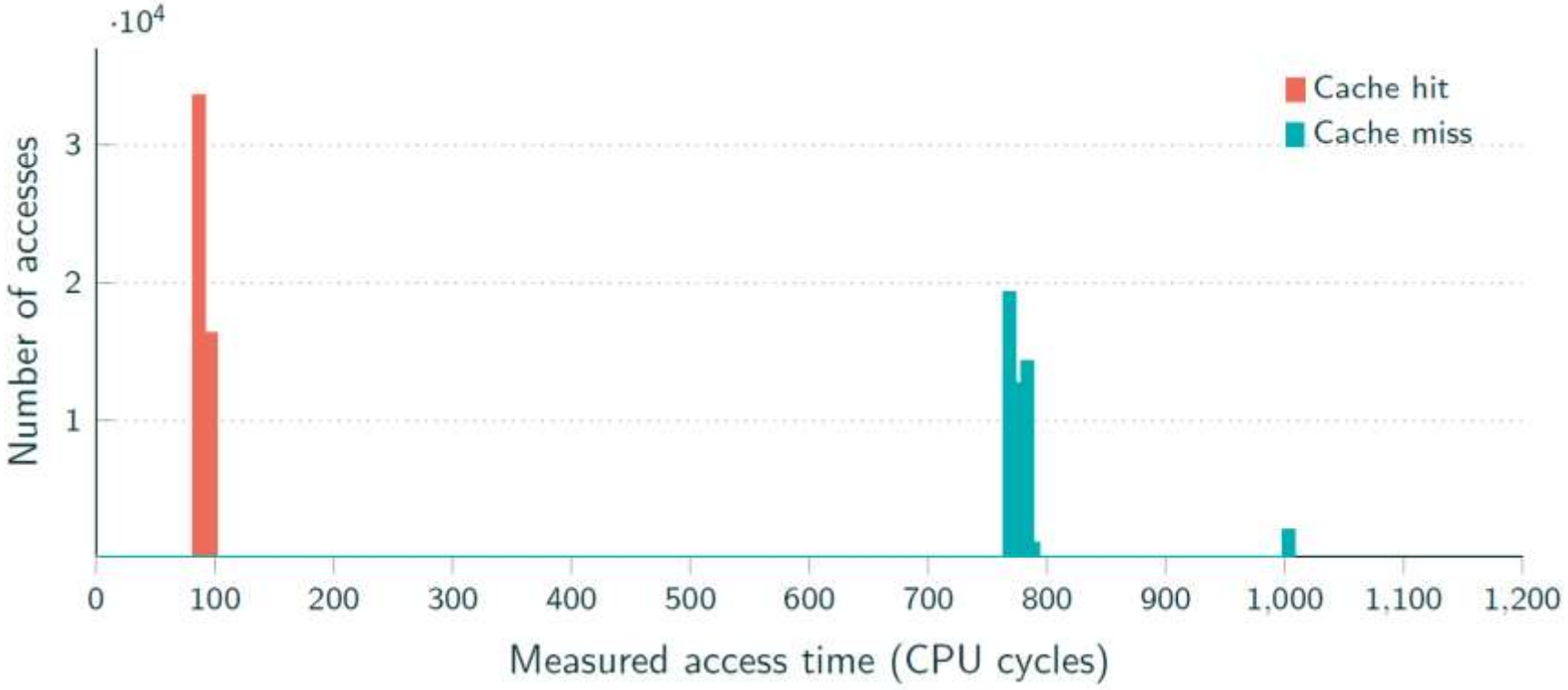


Cache侧信道

攻击原理：多核之间**cache数据共享**，而cache命中和失效对应**响应时间有差别**，攻击者可以通过访问时间的差异，推测cache中的信息，从而获得隐私数据



时延分析



根据时延分析构造缓存侧信道，目前，Cache侧信道攻击主要包含四种方法：

Prime-Probe | **Flush-Reload** | **Evict-Reload** | **Flush-Flush**

Prime-Probe

步骤1. Prime: 攻击者用预先准备的数据填充特定多个cache组

步骤2. Trigger: 等待目标进程响应服务请求，将cache数据更新

步骤3. Probe: 重新读取Prime阶段填充的数据，测量并记录各个cache组读取时间

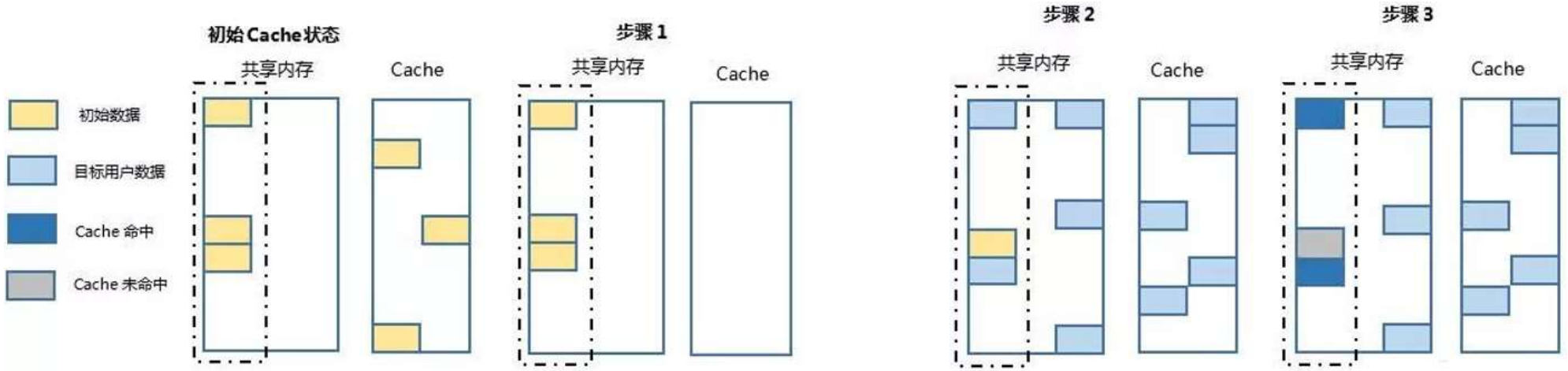
Flush-Reload

步骤1. Flush：将共享内存中特定位置映射的cache数据驱逐

步骤2. Trigger：等待目标进程响应服务请求，更新cache

步骤3. Reload：重新加载Flush阶段驱逐的内存块，测量并记录cache组的重载时间

针对共享内存进行操作，先刷



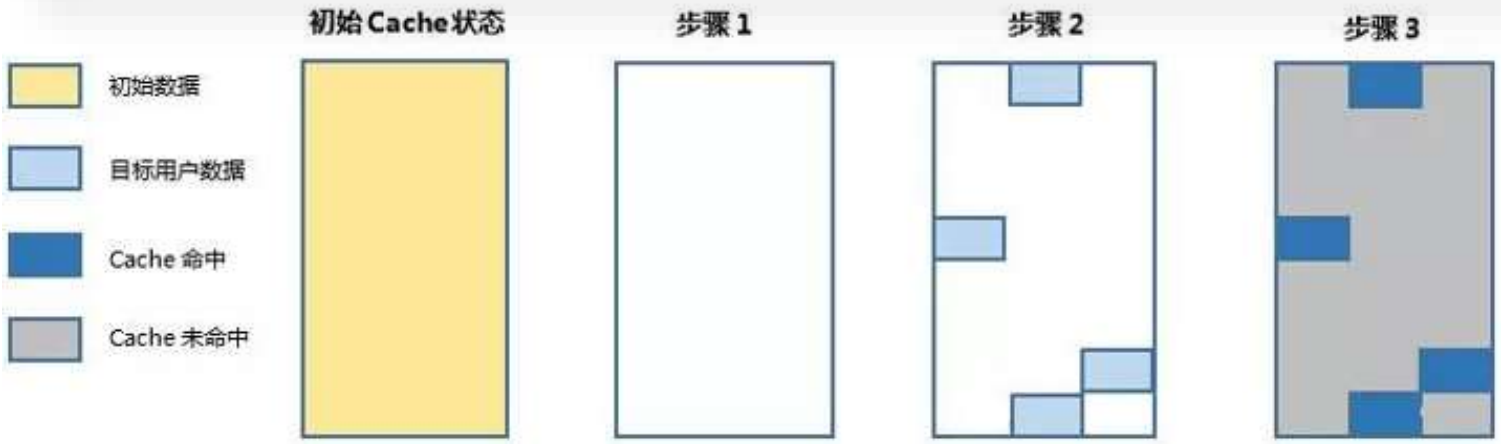
Evict+Reload和Flush+Reload的攻击流程基本一致，通过驱逐（eviction）的方式，用于替代Flush+Reload中的Flush指令缺失的情况

Flush-Flush

步骤1：通过flush清空cache原始数据

步骤2：等待目标进程运行，更新cache，并刷新共享缓存行，测量刷新时间

步骤3：根据测量时间判断原始数据是否被缓存



无需内存读取
根据clflush判断

Flush-Flush攻击是基于clflush指令执行时间的长短来实施攻击的，如果数据没在Cache中则clflush指令执行时间会比较短，反之若有数据在cache中则执行时间会比较长，与其它Cache攻击不同，Flush Flush侧信道攻击技术在整个攻击过程中是不需要对内存进行存取的，因此该攻击技术更加隐蔽

第2节 硬件防护技术

- 木马检测技术
- 隔离技术
- 密码技术
- 侧信道防护

硬件防护技术

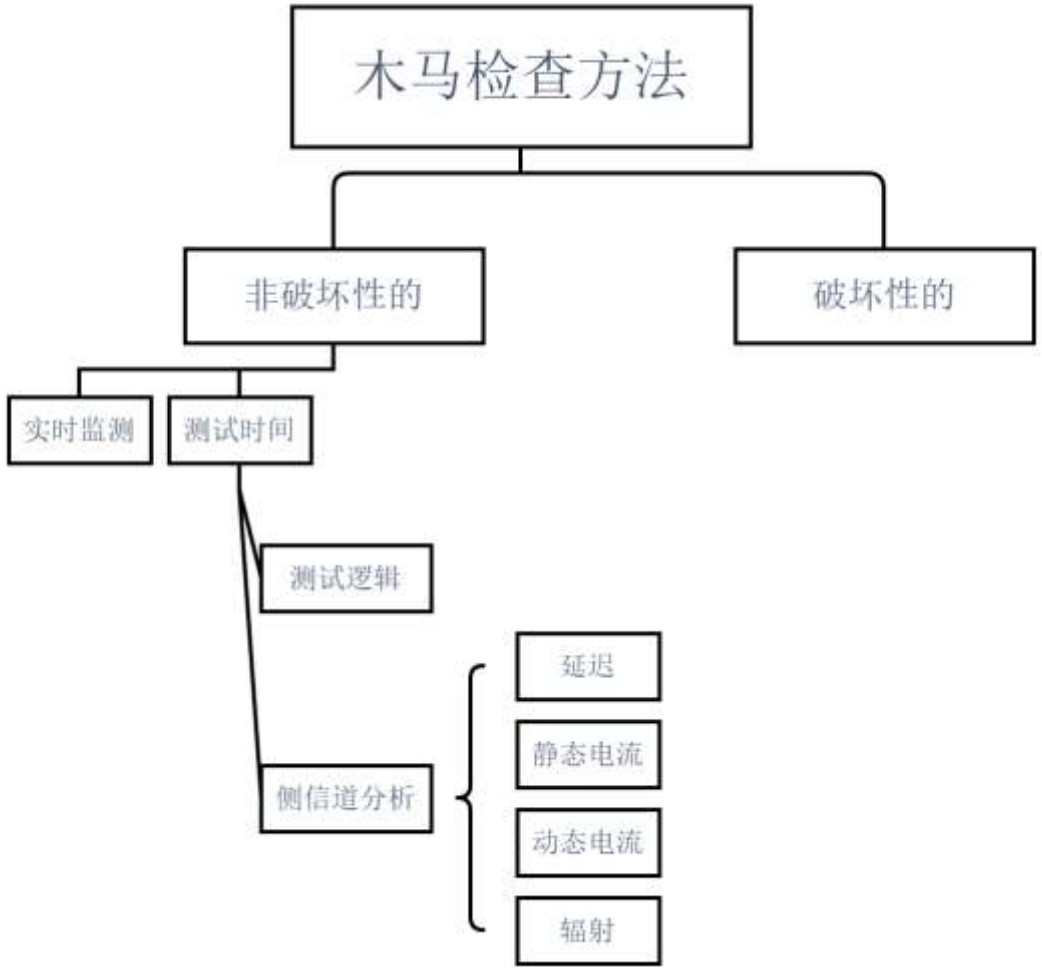

硬件防护技术
密码技术
对称密码
非对称密码
PUF技术
侧信道防护
旁路信道隐藏
掩码技术
设计分区
物理安全
隔离技术
存储器隔离
TrustZone
SGX
木马检测
破坏性检测
非破坏性检测

防护一：木马检测

- 木马检测方法大致可以分为**破坏性**和**非破坏性**两类

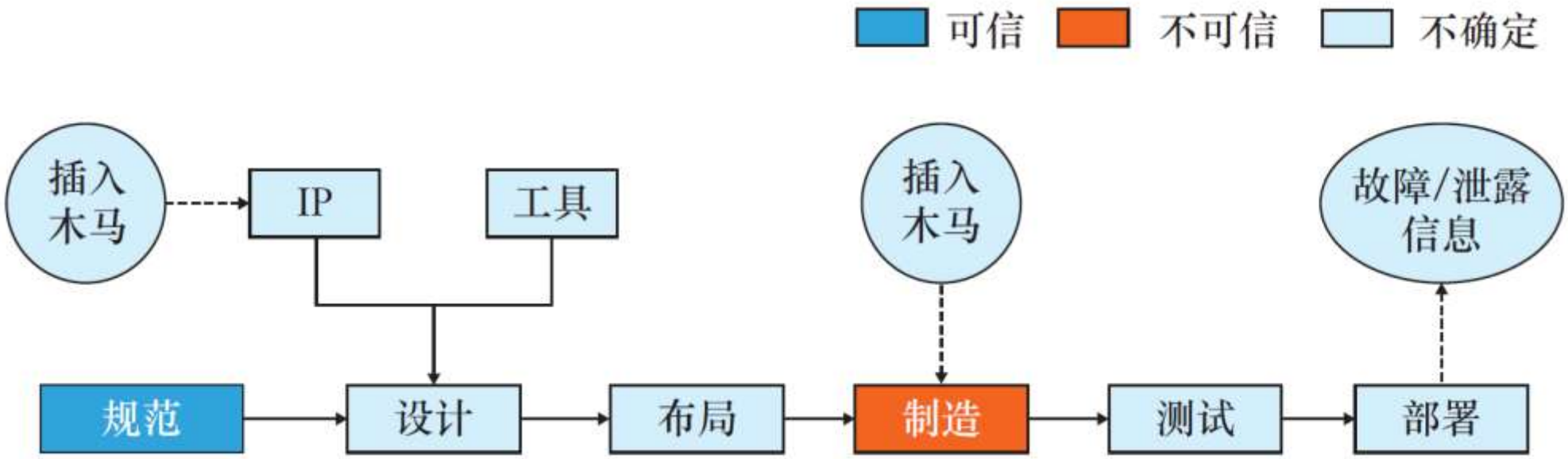
检测挑战

- 选择合适的木马模型
- 生成测试向量来激活木马或增加侧信道测量中的木马敏感度
- 消除/校准测试中的环境噪声和测量噪声



集成电路生命期

在集成电路生命期内的各个阶段都可能存在不可信的组件/人员



一般来说芯片设计阶段是可信的，因此可以获得标准的设计及测试向量进行木马检测，而制造阶段通常不可信，其他阶段则是两种都有可能

方法分类

破坏性检测

化学去封 ⇩ 电路扫描 ⇩ 逆向分析 ⇩ 芯片重构 ⇩ 模板匹配

- 费时耗力
- 不能逐个检测
- 实用性不强

非破坏性检测

芯片测试

测试向量 ⇩ 特征检测 ⇩ 结果分析

- 可以逐个测试
- 测试向量不全面
- 检测分析不完善

实时监测

- 在线监测
- 开销较大

逻辑测试和侧信道分析

- **逻辑测试法**侧重于生成并使用**测试向量**来尝试激活可能的木马电路，并观察对于主输出端有效荷载的影响
- **侧信道分析法**基于：在芯片中植入任何恶意电路都会影响某些**旁路信道**的参数值，如漏电流、静态电源电流、动态功耗轨迹、路径延迟特性和电磁辐射

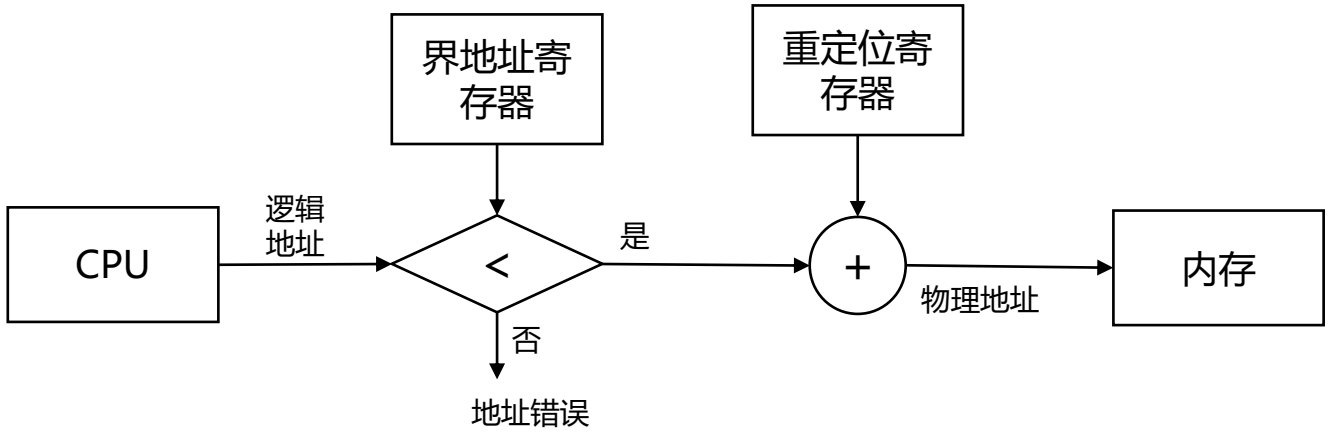
	逻辑测试	侧信道分析
优点	对小型木马有效 能抵抗制造过程噪声	对大型木马有效 测试生成简单
缺点	测试生成复杂 大型木马检测具有挑战性	对过程噪声脆弱 小型木马检测具有挑战性

防护二：隔离技术

- **隔离技术**是一种访问控制手段，用于限制用户访问非授权的计算资源。借助硬件隔离技术可以实现一些计算资源的共享（存储器隔离）、安全计算环境和不安全计算环境之间的隔离（TrustZone和SGX）

存储器隔离

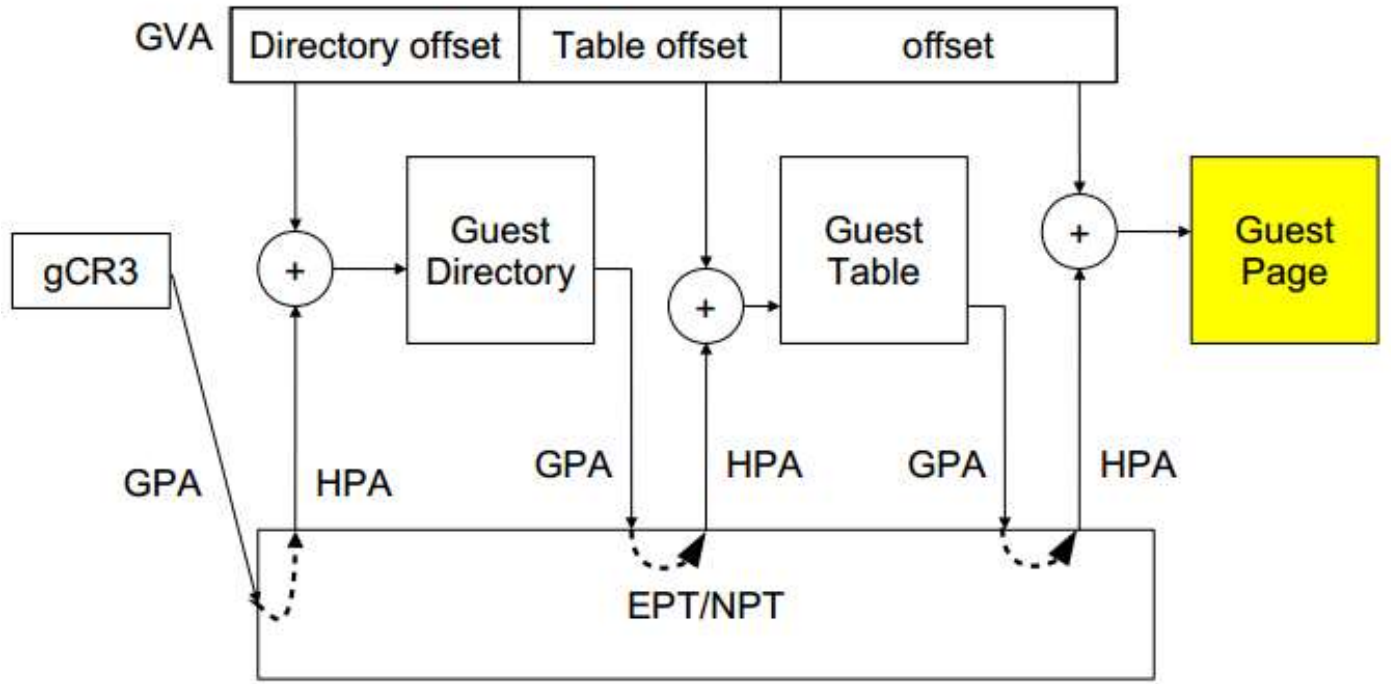
- 存储器隔离是现代处理器通常具有的一种特性
- 实现手段：存储器保护技术、EPT硬件虚拟化技术和存储加密技术



- 系统使用**IOMMU**（IO Memory Management Unit）限制设备能够访问的存储器
- 通过**重定位寄存器**和**界地址寄存器**实现IOMMU，重定位寄存器包含物理地址，界寄存器包含逻辑地址

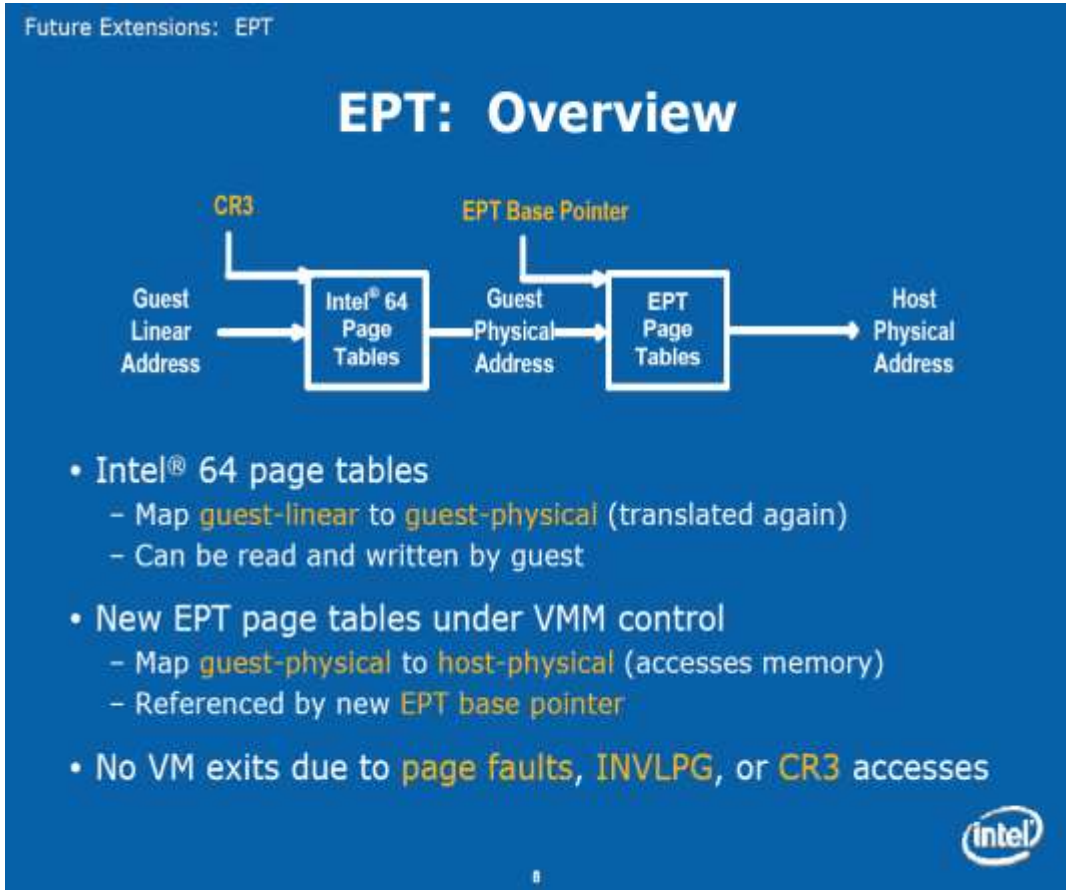
EPT硬件虚拟化技术

EPT是Intel针对软件虚拟化性能问题推出的硬件辅助虚拟化技术



- **GVA:** Guest虚拟地址
- **GPA:** Guest物理地址
- **HVA:** Host虚拟地址
- **HPA:** Host物理地址
- **gCR3:**Guest CR3

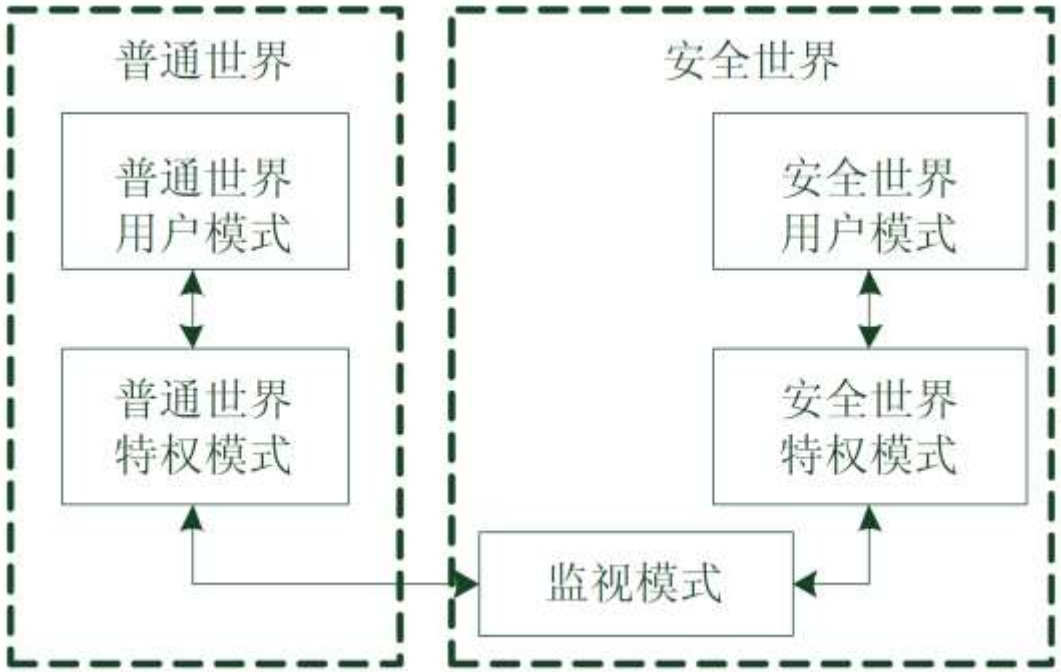
EPT地址转换



- **CR3**将客户机程序所见的**客户机虚拟地址**（GVA）转化为**客户机物理地址**（GPA），然后在通过**EPT**将**客户机物理地址**（GPA）转化为**宿主机物理地址**（HPA）。这两次转换地址转换都是由CPU硬件自动完成，其转换效率非常高

ARM TrustZone

TrustZone将CPU内核隔离成**安全**和**普通**两个区域，即单个的物理处理器包含了两个虚拟处理器核：安全处理器核和普通处理器核。从而单个处理器内核能够以时间片的方式安全有效的同时从普通区域和安全区域执行代码



- **非安全核**只能访问普通世界的系统资源，而**安全核**能访问所有资源
- **安全核**与**非安全核**之间的切换通过使用**SMC（Secure Monitor Call）指令**或者通过硬件异常机制的一个子集实现

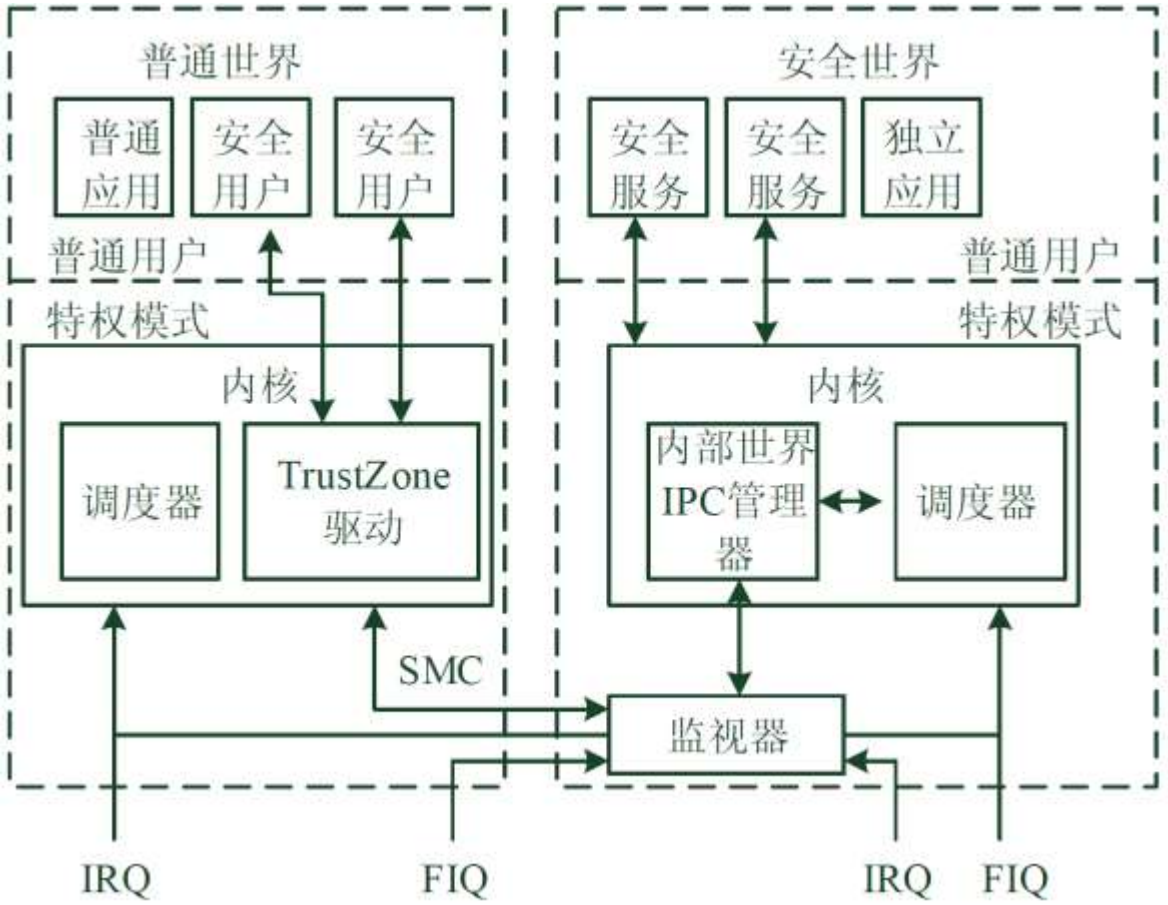
ARM TrustZone

安全世界执行

- 如指纹识别
- 密码处理
- 数据加解密
- 安全认证
- ……

普通世界执行

- 用户操作系统
- 应用程序
- ……



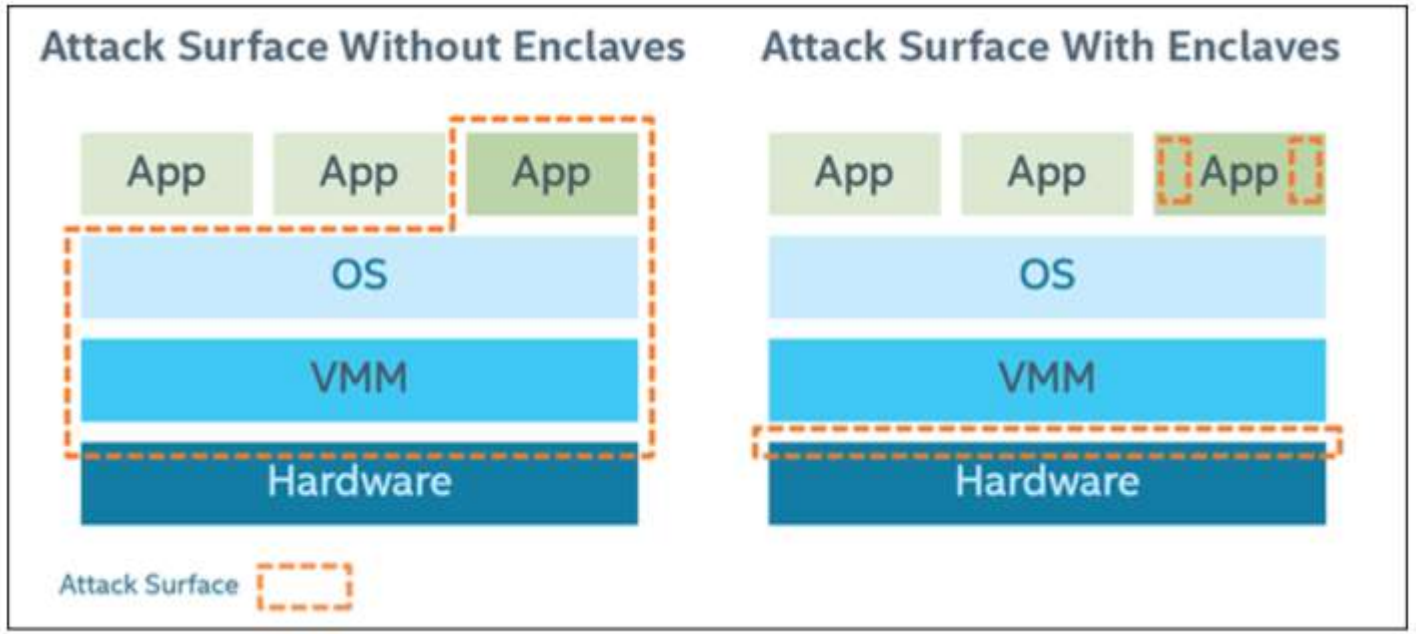
监视器

普通世界 → 安全世界

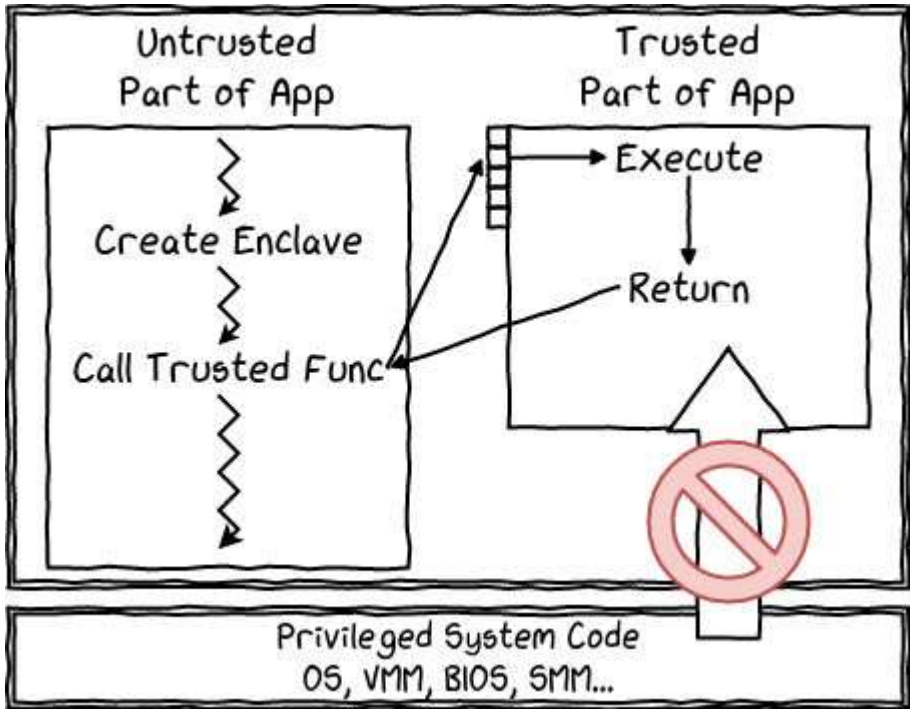
- IRQ：普通世界的中断源
- FIQ：安全世界的中断源

Intel SGX

- **Intel SGX**是一组CPU指令，可让应用程序创建安全区：应用程序地址空间中的受保护区域，即使存在特权恶意软件，也可提供机密性和完整性
- 英特尔SGX可以减少应用程序的攻击面



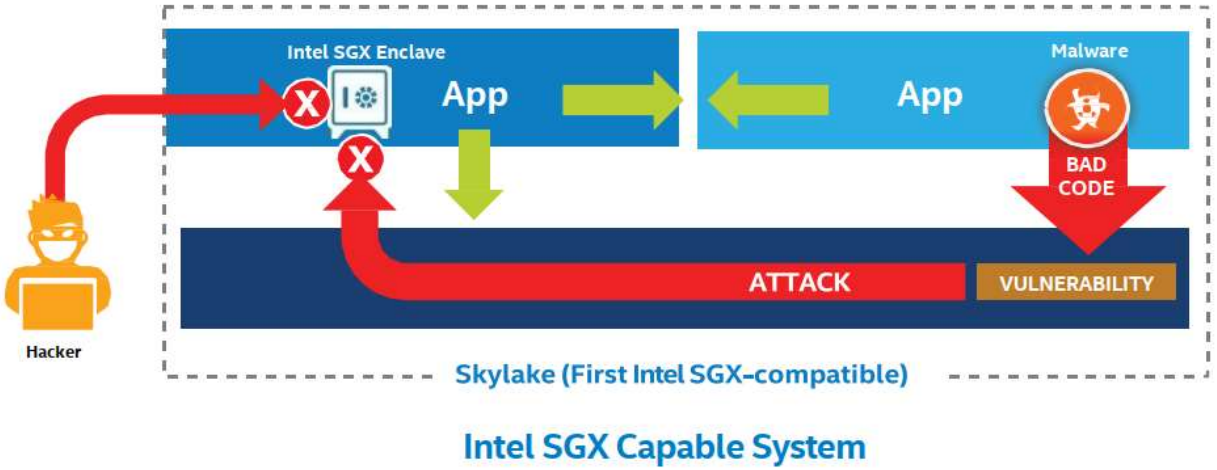
SGX概述



SGX特点

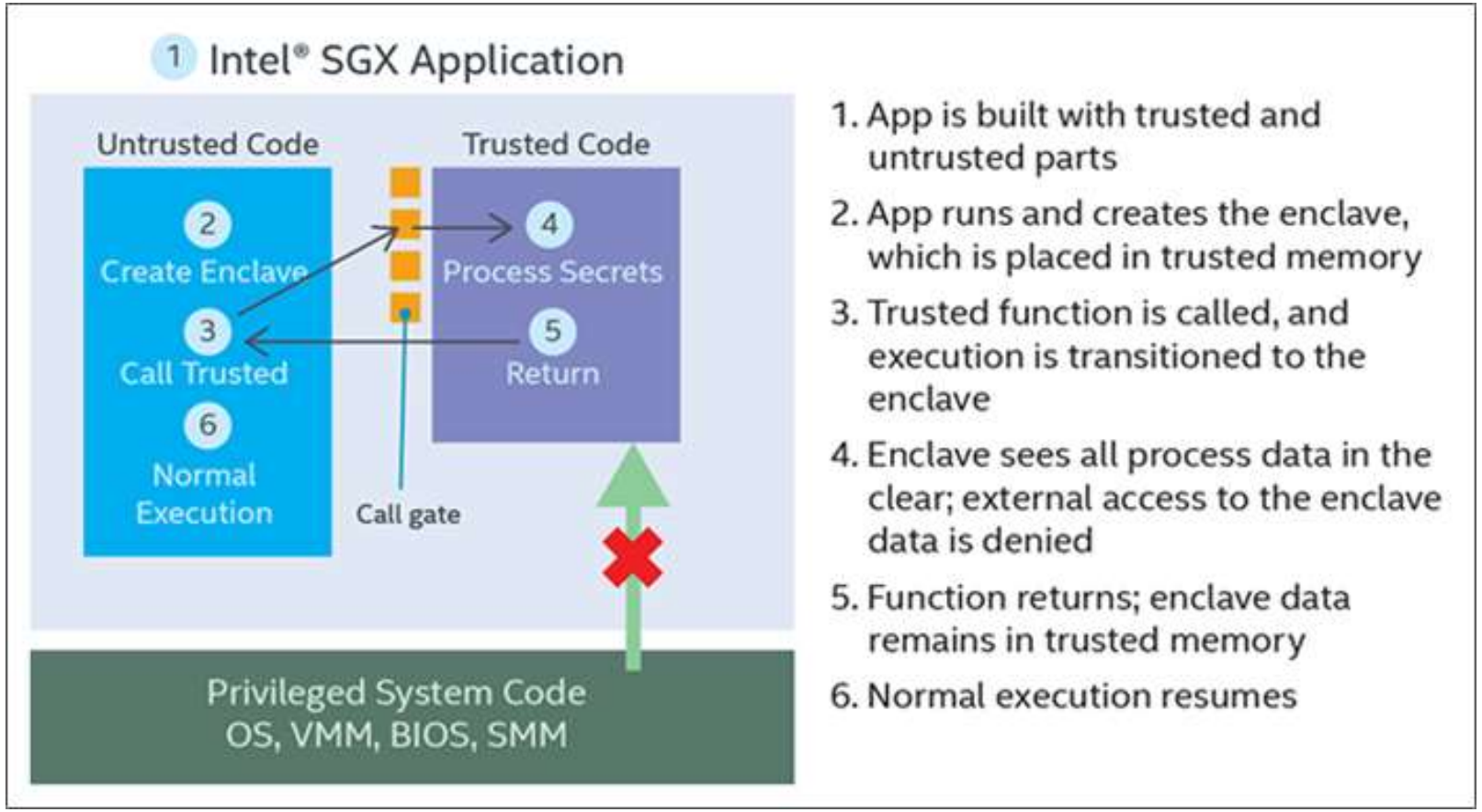
- 机密性和完整性
- 低学习曲线
- 远程认证
- 缩小攻击面

- 应用分为安全和非安全
- Enclave放置在受保护的内存中

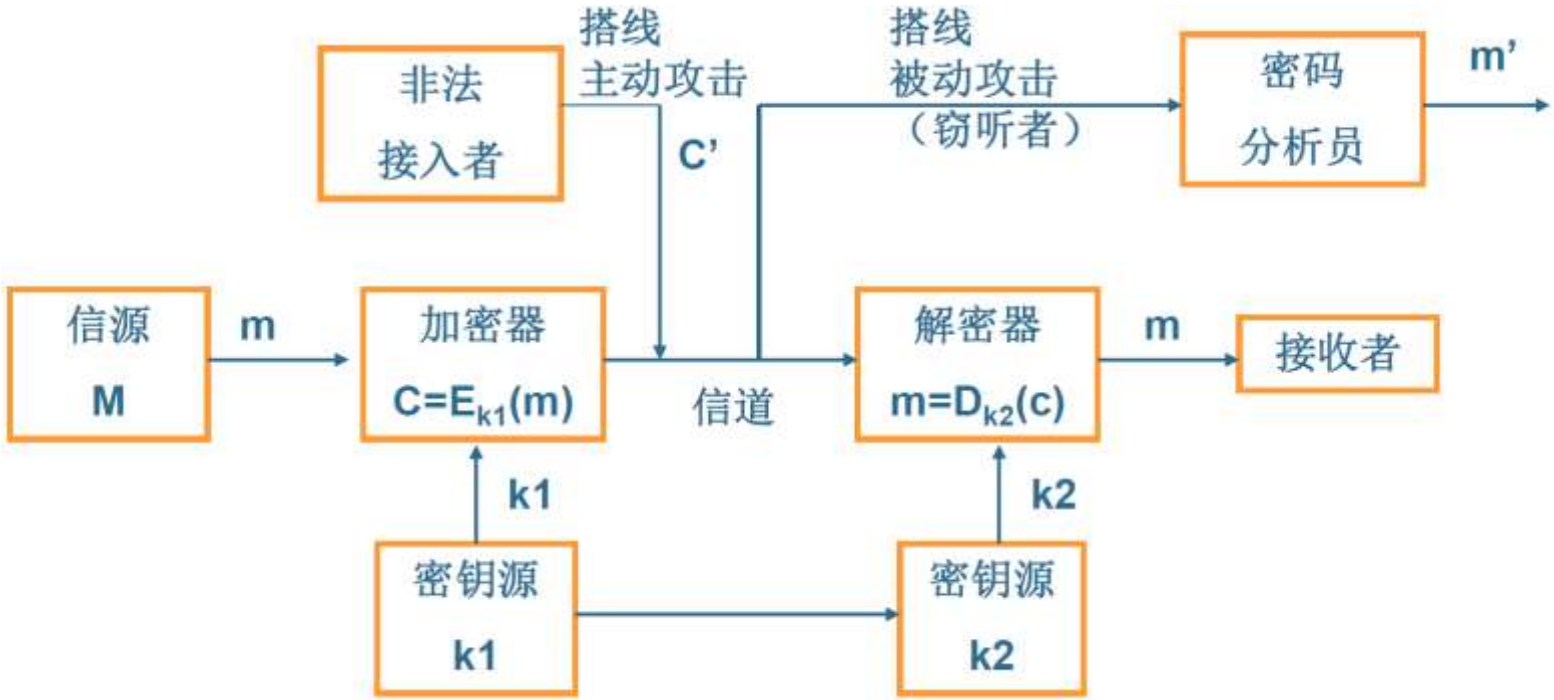


SGX应用分类

- 使用英特尔SGX进行应用程序设计时，需要将应用程序分为两个组件:**受信任的组件**和**不受信任的组件**



防护三：密码技术



- 现代密码技术可以分为对称密钥与非对称密钥两种体系
- 对称密钥常用于批量数据的加解密，非对称密钥更多用于密钥交换
- 物理不可克隆函数（PUF，Physical Unclonable Function）电路是一种新型密钥生成电路，保证密钥的唯一性和不可克隆性

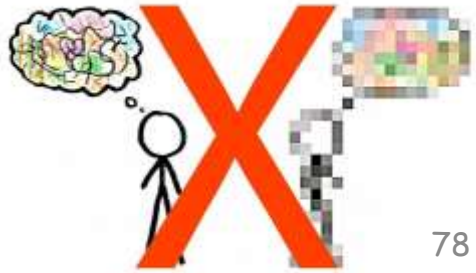
物理不可克隆函数

- **PUF**利用了在半导体生产过程中自然发生的深亚微米变化，并赋予每个晶体管些许随机的电特性

- **唯一性**
- **隐匿性**
- **稳定性**
- **随机性**

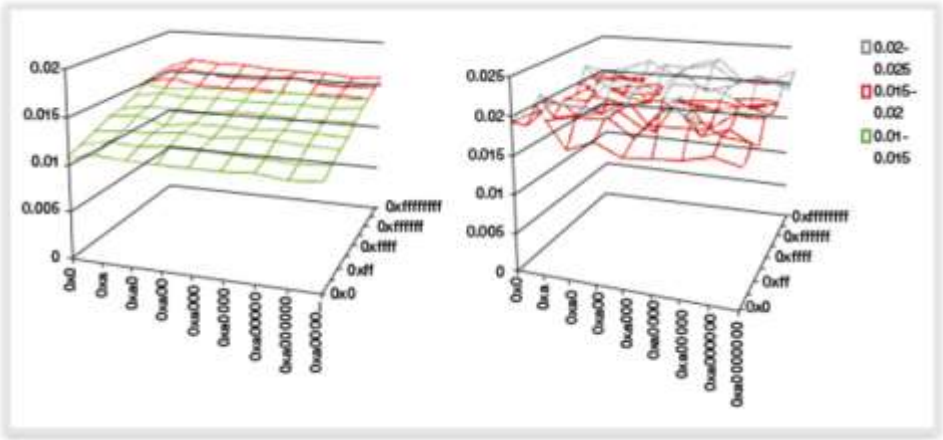
PUF分类
- 非电子PUF
- 模拟电路PUF
- 数字电路PUF

PUF应用
- 身份认证
- 密钥生成
- 创建信任根

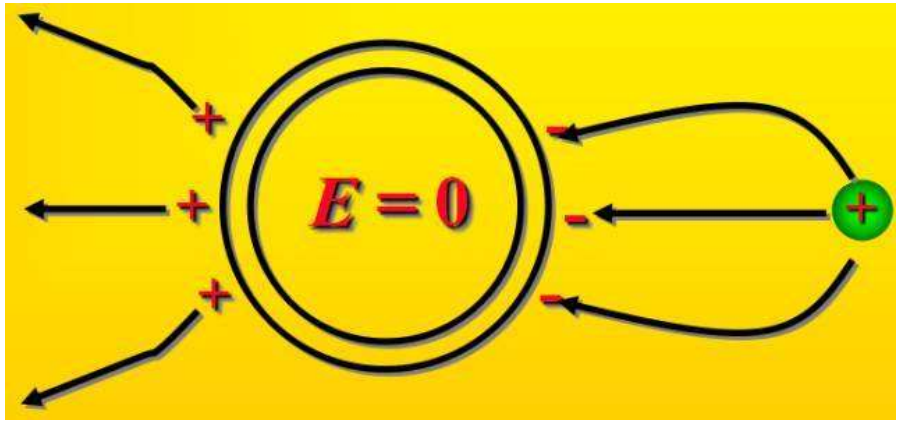


防护四：旁路信息隐藏

- 攻击者利用旁路信道的输出来获得足够的信息，以确定和芯片操作相关的敏感数据。所以他们寻求从低信噪比的旁路信道中恢复信号（试图**提高信噪比**），因此可以通过**增大噪声**或者**减小信号**来**降低信噪比**，增加攻击难度



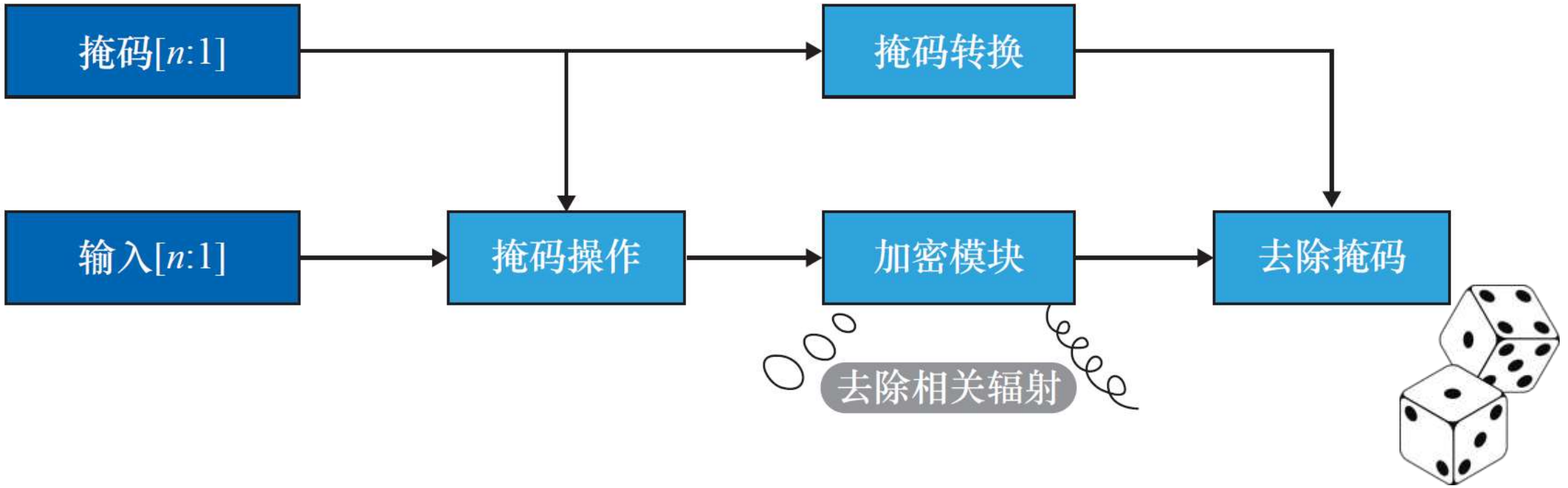
增加噪声



物理屏蔽

掩码技术

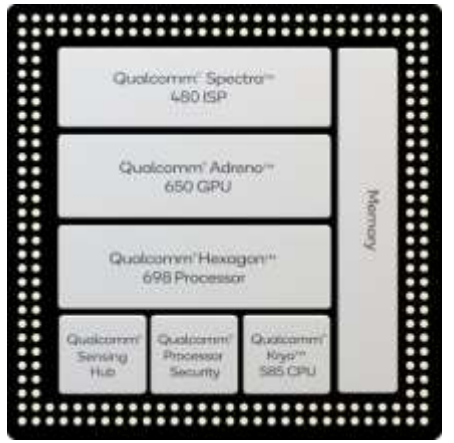
掩码技术是一种从功能模块的**中间节点**入手，移除输入数据与旁路信道相关性的措施，该技术可基于门或者基于模块来实现



模块划分及物理安全

模块划分

一些通道会泄漏信息是因为敏感信号被耦合到了其他节点上面，为了减少这种情况，设计人员可以将芯片操作中的明文操作区和密文操作区分开。除了将芯片不同区域进行物理隔离，还需要将片内共享的基础设施进行分离，包括供电基础设施、时钟基础设施、和测试基础设施



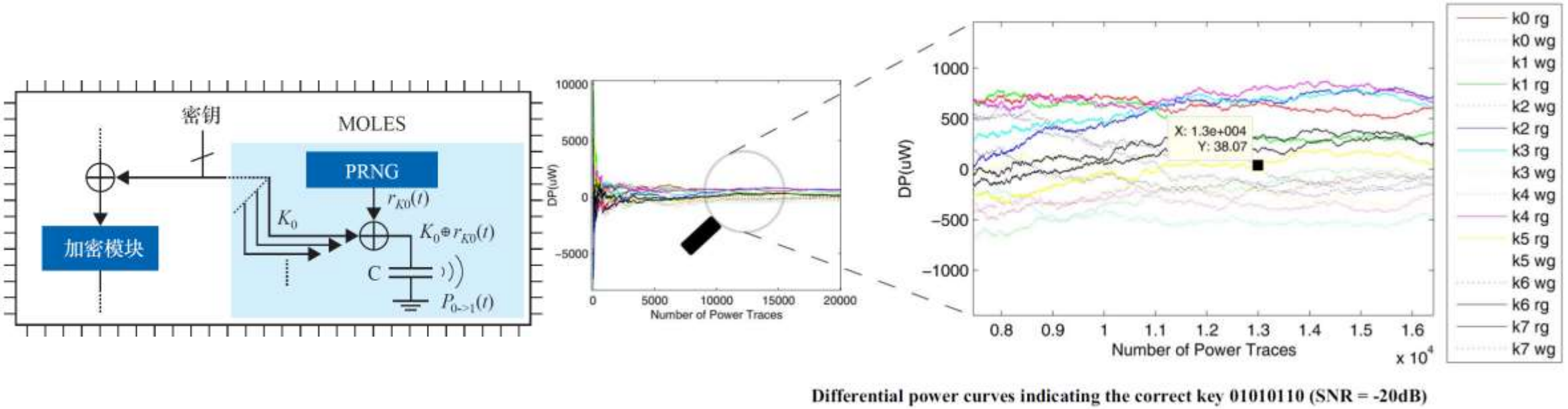
物理防护

为了感知高信噪比的旁路信息，攻击者需要对受害设备进行长时间的物理访问，甚至破坏设备。因此拒绝接近、拒绝访问和拒绝受控是降低攻击者挂载到旁路信道进行攻击的关键

第3节 典型漏洞分析

- MOLES
- Meltdown
- Spectre
- VoltJockey

漏洞一：MOLES



Differential power curves indicating the correct key 01010110 (SNR = -20dB)

MOLES(Malicious off-chip leakage enabled by side-channels)，利用木马通过侧信道泄漏芯片内部信息（如密钥），再利用差分能量分析技术提取密钥

漏洞二：Meltdown概述

1.使用乱序执行，攻击者可以在任何地址读取数据

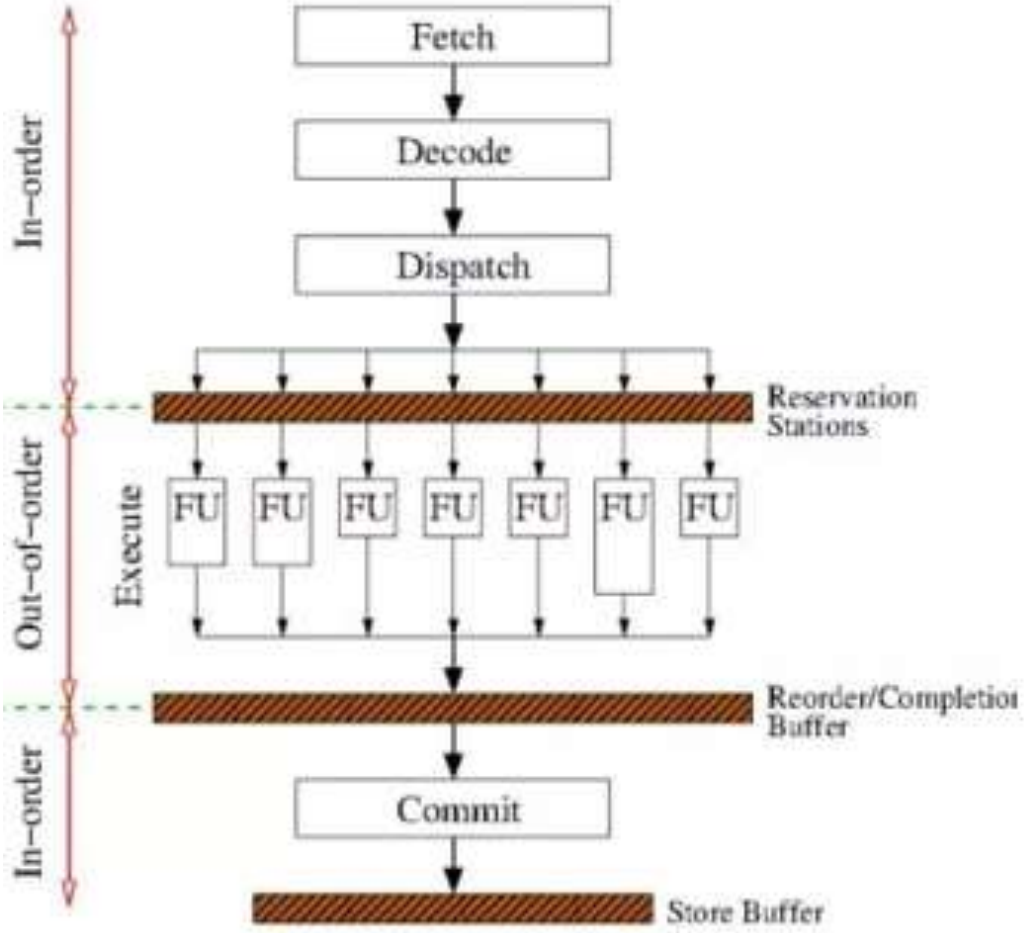
2. 泄漏内核内存

3. 影响几乎所有的Intel处理器和ARM Cortex-A75

乱序执行

乱序执行过程

1. 获取指令，解码后存放到执行缓冲区（保留站）
2. 乱序执行指令，结果保存在一个结果序列中
3. 重排，重新排列结果序列及安全检查（如地址访问的权限检查），提交结果到寄存器



攻击过程

```
1 rcx = kernel address,
   rbx = probe array
2 xor rax, rax
3 mov al, byte [rcx] //step 1
4 shl rax, 0xc //rax*4096
5 mov rbx, qword [rbx + rax]
```

目标地址

将数据映射到缓存页（4KB）

在攻击者缓存集上留下痕迹

攻击场景

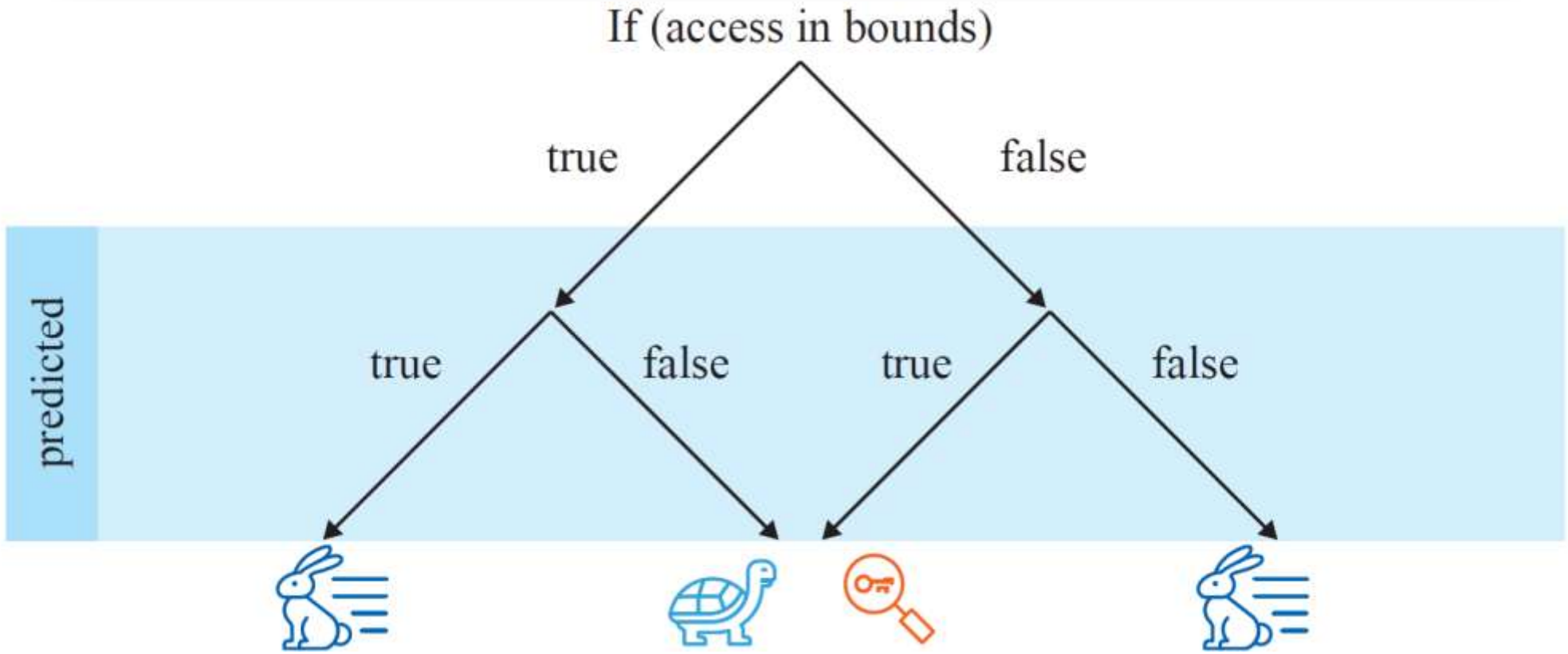
攻击者：用户级特权

目标：读取内核数据

- line3、4、5乱序执行
- 敏感数据泄漏到cache
- 安全检测丢弃违例结果
- Cache侧信道提取数据

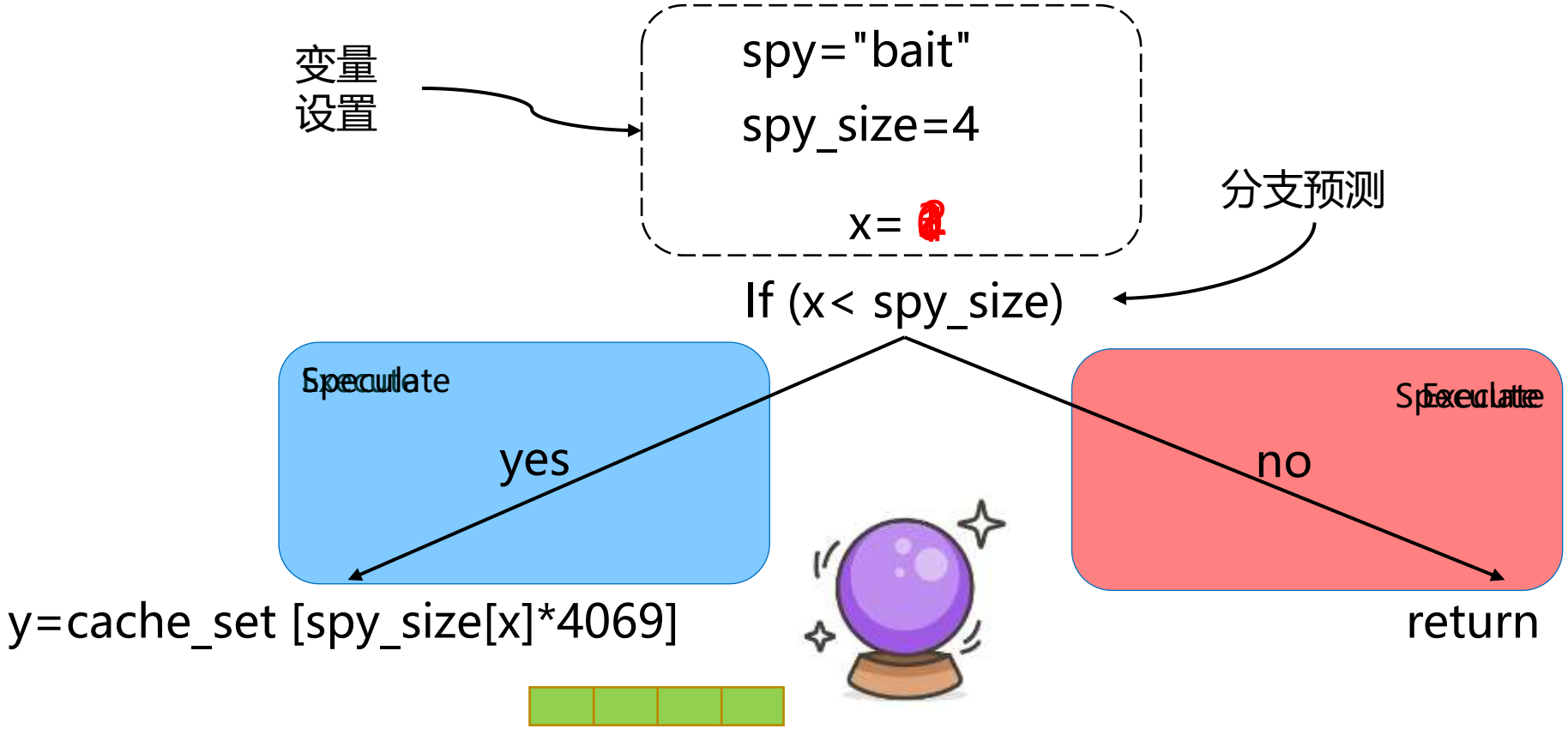
漏洞三：Spectre

分支预测起在分支指令执行结束之前**猜测**哪一路分支将会被运行，以提高处理器的指令流水线的性能



攻击过程

训练分组预测器



关键步骤

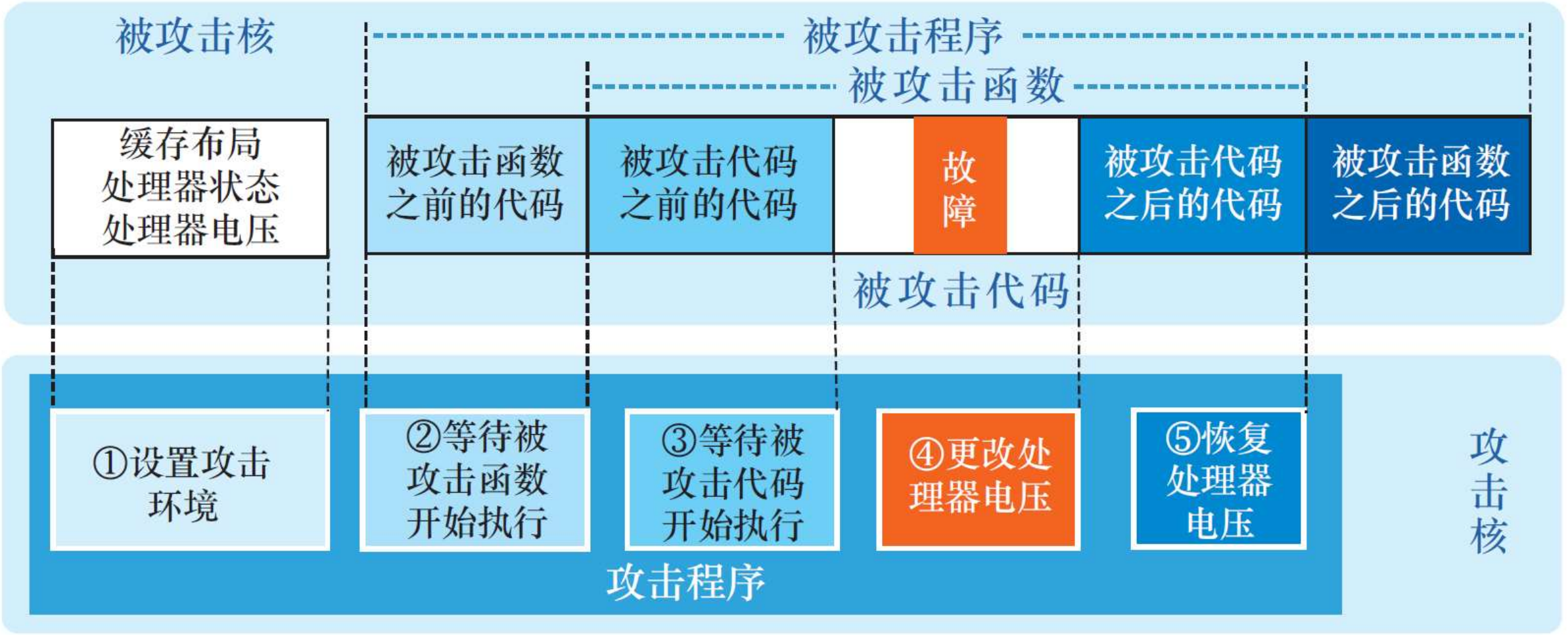
与Meltdown类似，Spectre的原理是，当CPU发现分支预测错误时会丢弃分支执行的结果，恢复CPU的状态，但是不会恢复**CPU Cache**的状态，利用这一点可以突破进程间的访问限制，从而获取其他进程的数据

```
if (x < array1_size) { y = array2[array1[x] *
4096];
// do something detectable when
// speculatively executed }
```

DEMO分析

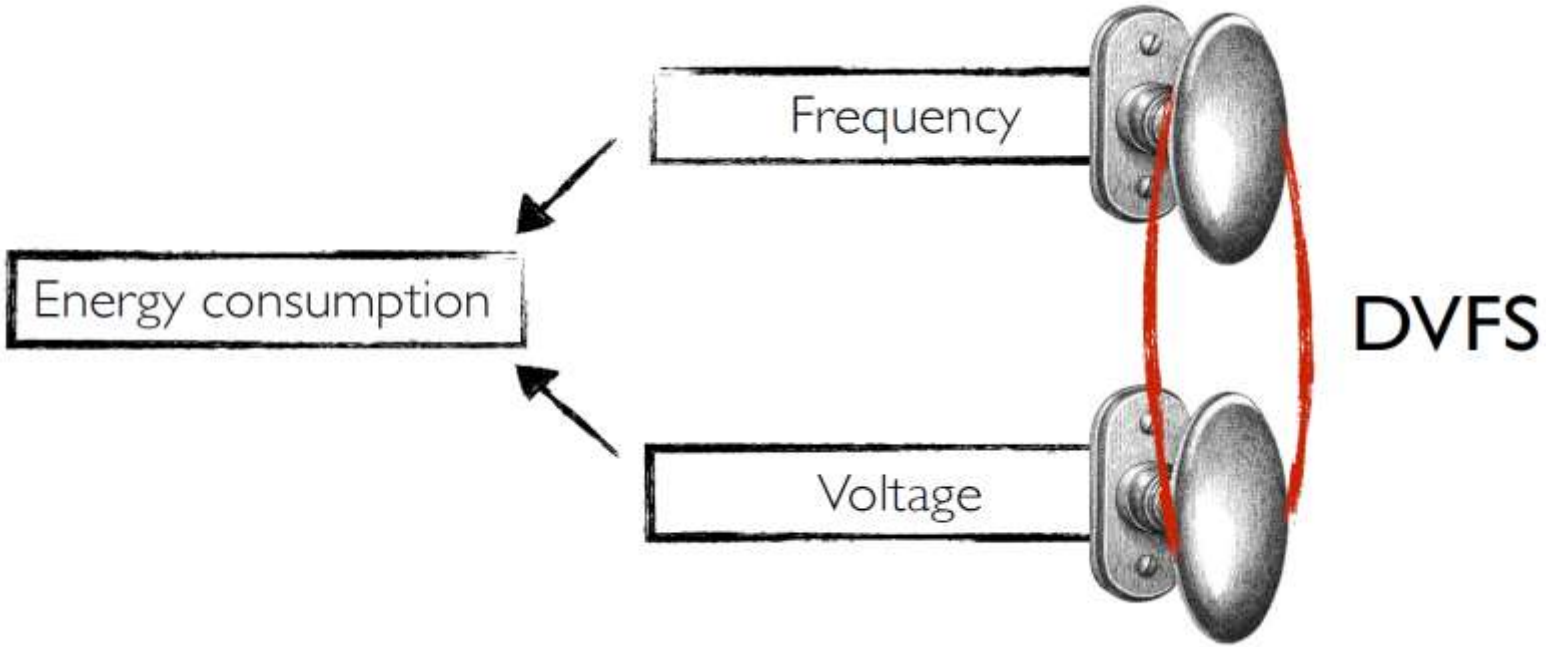
1. 训练CPU的分支预测单元使其在运行代码时执行特定的预测
2. 分支预测越权访问敏感数据并将其映射到cache中
3. 利用缓存测信道，通过cache使用情况窃取敏感数据

漏洞四： VoltJockey

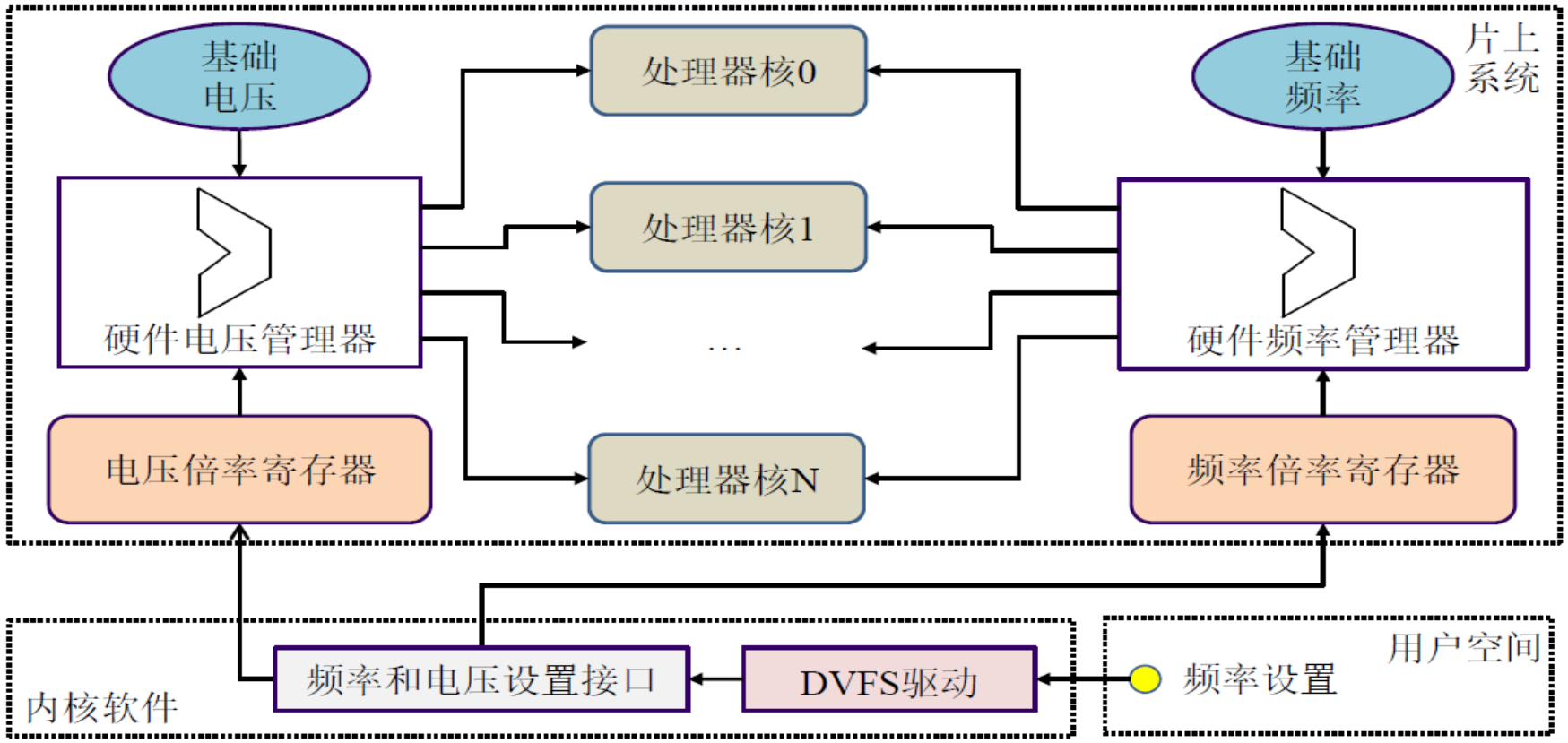


核心动态电源管理技术

- **动态电源管理技术**（Dynamic Voltage and Frequency Scaling, DVFS）在满足用户对性能的需求下根据处理器的负载状态动态改变电压和频率，实现节省能耗的目的，是一种被广泛应用在现代处理器中的低功耗技术
- 为了支持DVFS，处理器的硬件频率和电压管理器的输出被设计成是可调的



动态电源管理技术架构

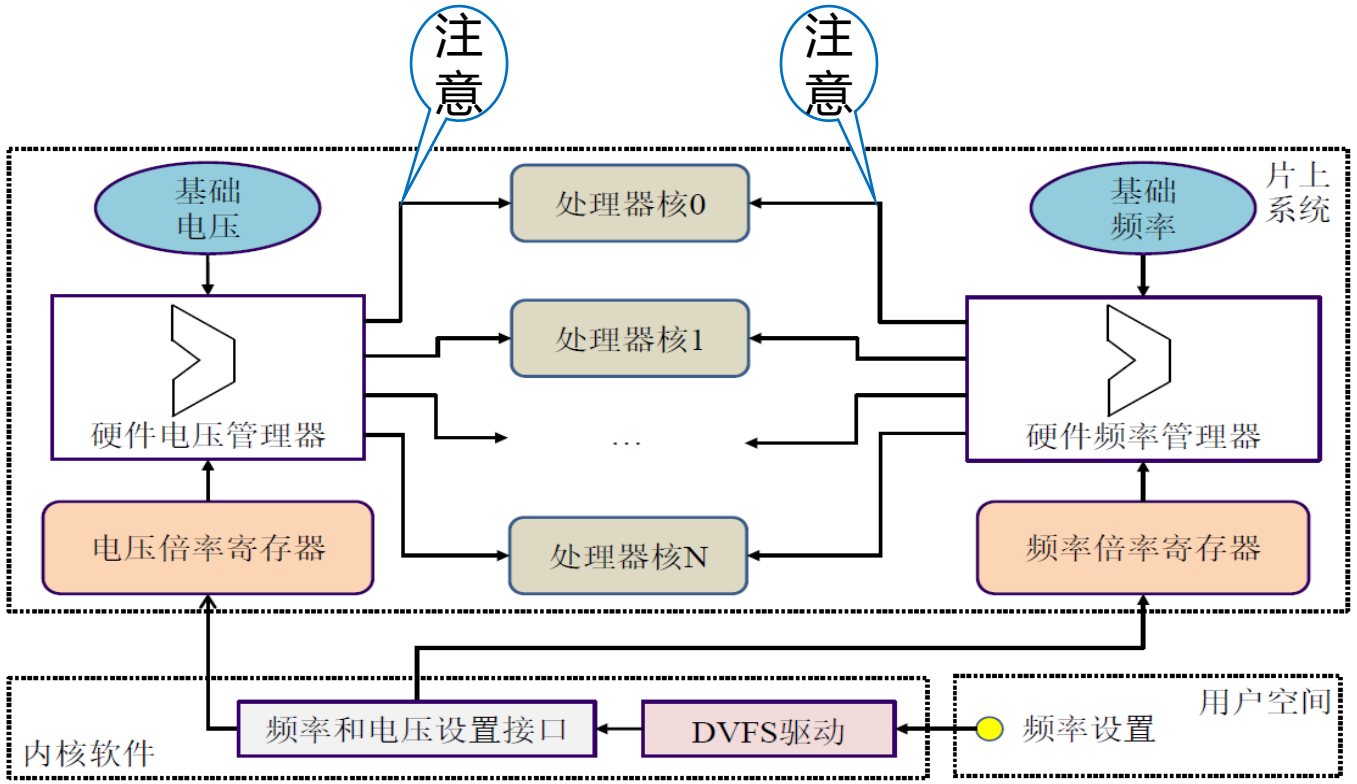


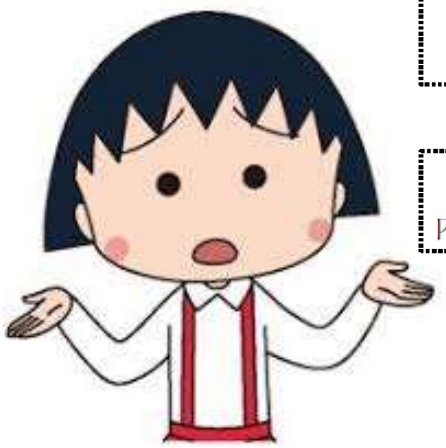
DVFS 基本架构

动态电源管理的漏洞分析

用户可以设置电压/频率管理器参数

ARM不同处理器内核的电压/频率可独立设置



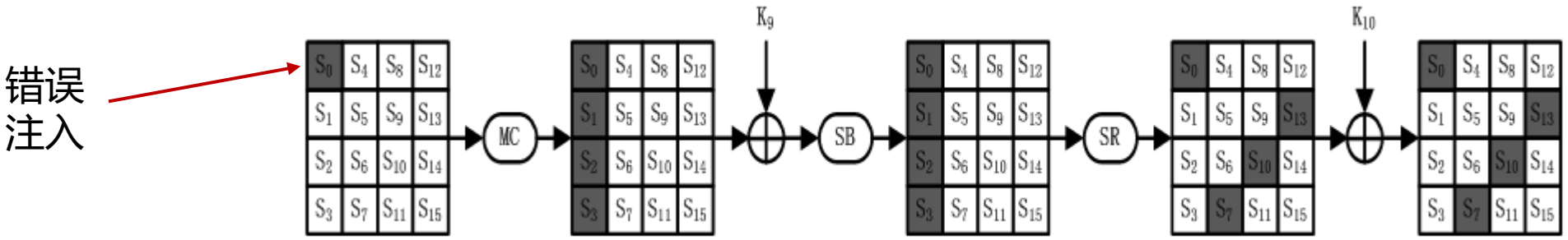


频率&电压

时序约束

漏洞四：VoltJockey

TEE（Trusted Execution Environment）可信执行环境，是CPU上的一块区域，该区域能为代码和数据提供一个隔离的、安全的执行环境。**TrustZone**是ARM设计的可信执行环境，VoltJockey能够攻击被TrustZone保护的**AES**函数

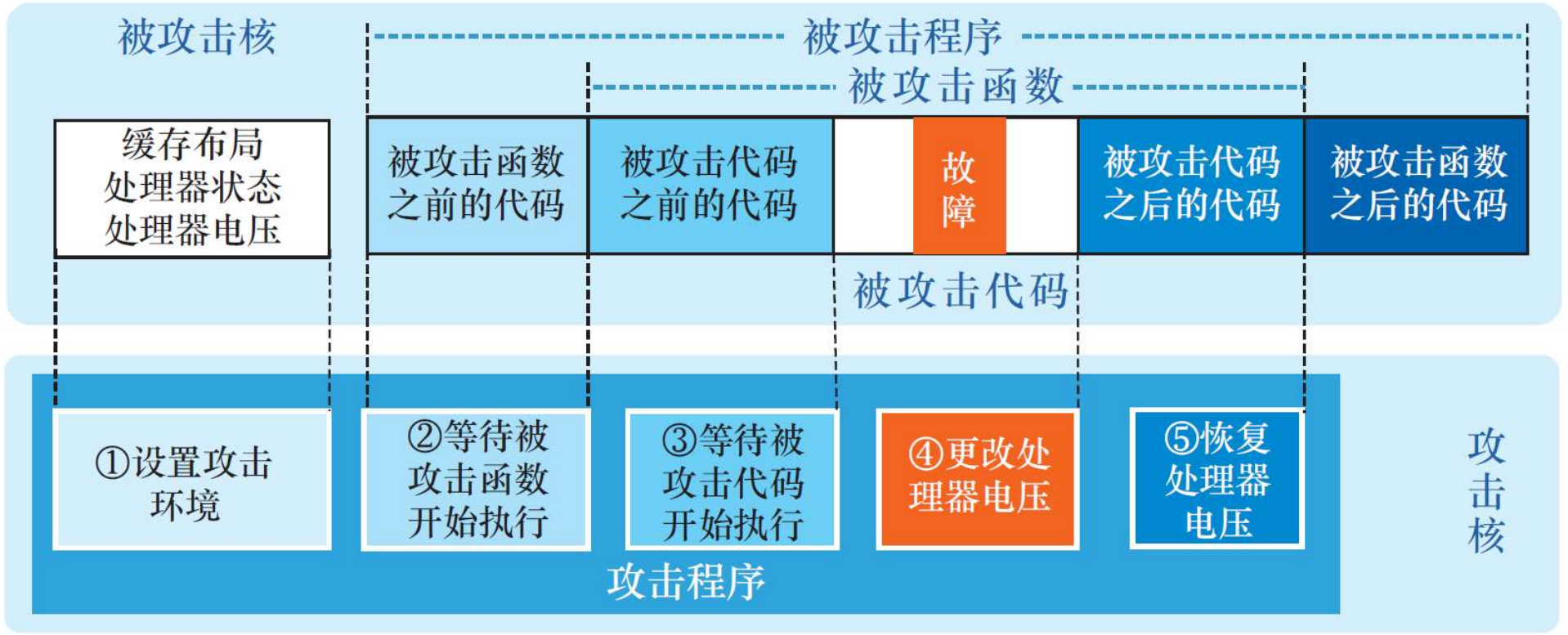


AES是一种对称加密算法，直接对AES 程序进行暴力破解以获取加密密钥是比较困难的。但是，如果AES执行时的中间状态矩阵被更改，密钥的搜索空间可以被大大减少*

攻击者调用AES算法，然后使用**电压故障攻击**将错误引入中间状态矩阵，从而破解AES的加密密钥

* Tunstall M. et al. Differential fault analysis of the advanced encryption standard using a single fault, WISTP 2011.

攻击过程



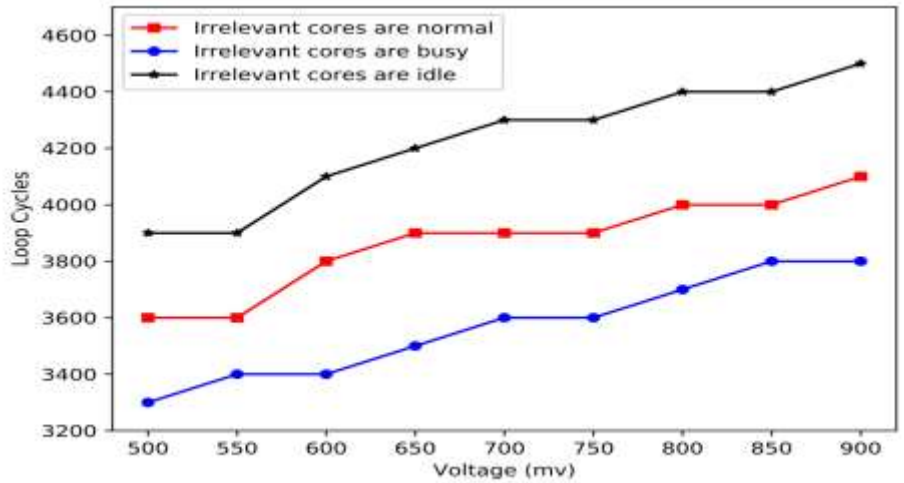
准备工作

- 分析公开的加密函数找到合适的注入点
- 分析故障注入的合适电压与时间
- 使用NOP指令精确控制时间

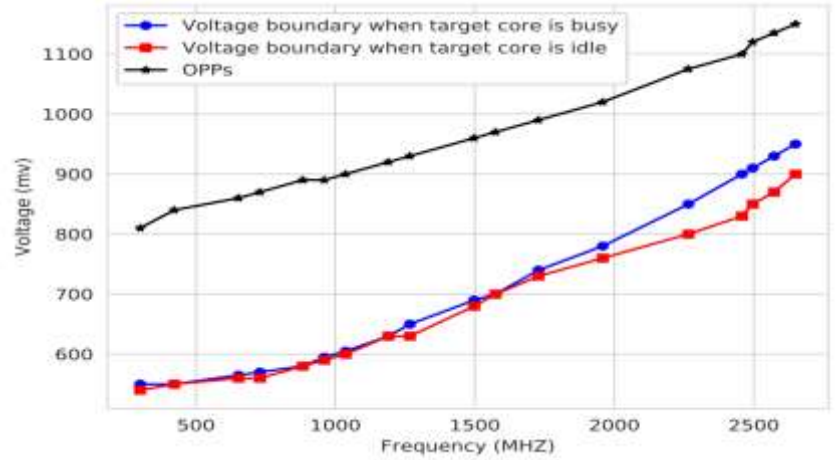
攻击要点

- 将进程绑定到特定的处理器内核
- 使用缓存侧信道监控被攻击函数
- 在注入点引入特定时间的电压故障
- 使用差分故障分析技术提取密钥

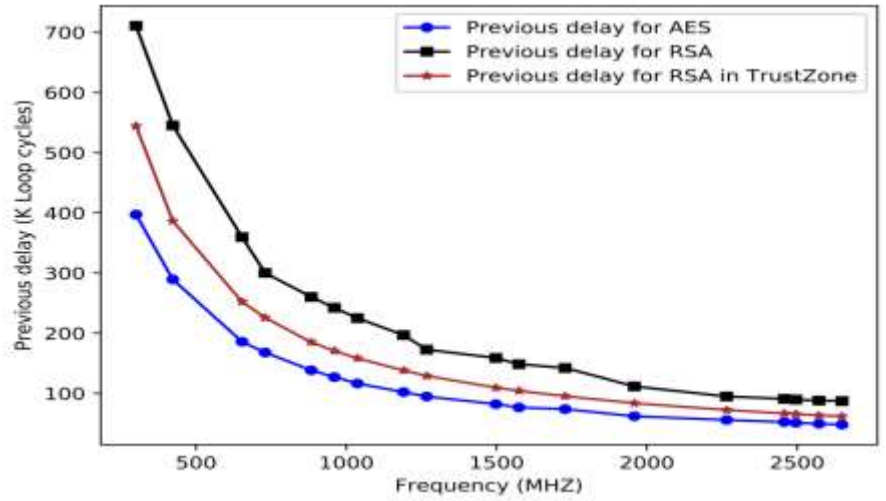
攻击参数设定



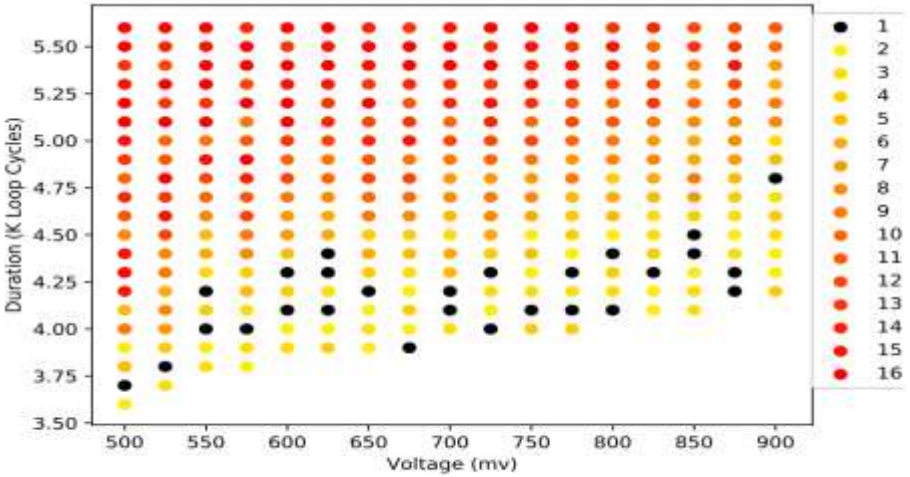
- 故障电压分析



- 确定临界电压（避免重启）



- 确定预备延迟



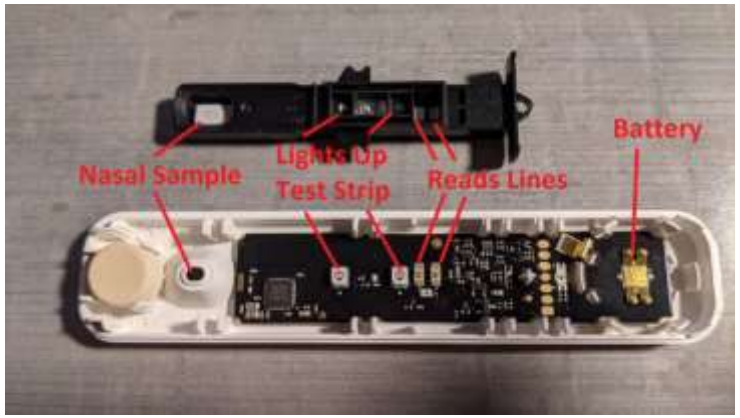
(a) Number of byte errors on the input of the eighth round of AES.

- 故障电压对AES影响

问题思考：COVID-19家庭检测套件的漏洞

美国实体药店可以购买Ellume COVID-19检测套盒，包括棉签拭子、试剂和一个分析仪

- 检测者使用棉签从鼻子或喉咙中收集粘液样本后，与测试溶液充分混合并滴在检测卡
- 用户通过Ellume COVID-19手机app发送检测请求，大约20分钟内app可接收到检测结果
- 该应用还会匿名发送到云端，方便防疫官员做统计

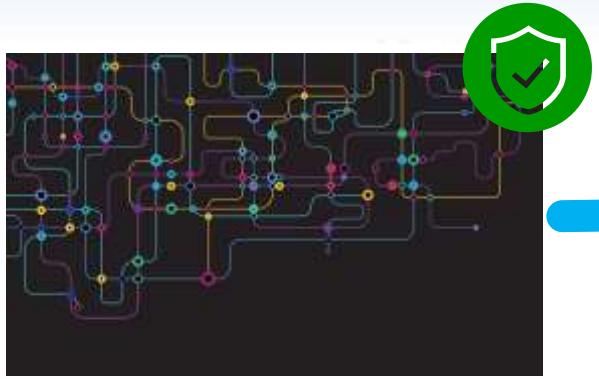


定制PCB板上的LED灯珠用于照亮测试条，一些透镜以读取测试条结果线，分析仪会通知配套的移动应用程序

第4节 总结和展望

系统硬件安全

你认为 应该实施哪些策略来保护系统硬件 ？



芯片设计	芯片制造	产品应用
规范编码	封装隔离	可信启动
安全版图设计	功能测试	电磁屏蔽
可信执行环境	木马检测	拒绝物理访问

总结

学习了常用的系统硬件攻击技术，依附于几个典型的系统硬件攻击，对真实的硬件漏洞进行溯源分析，探究了可行的系统硬件防护手段，构建了系统硬件防护体系

第一节
硬件攻击技术

- 漏洞攻击
- 硬件木马
- 硬件故障
- 侧信道攻击

系统设计缺陷、硬件制造缺陷和旁路信息泄露是造成系统硬件漏洞的主要因素

第二节
硬件防护技术

- 木马检测技术
- 隔离技术
- 密码技术
- 侧信道防护

系统硬件的产业链环境复杂，很难保证设计和生产绝对安全，需要补丁式的硬件防护技术

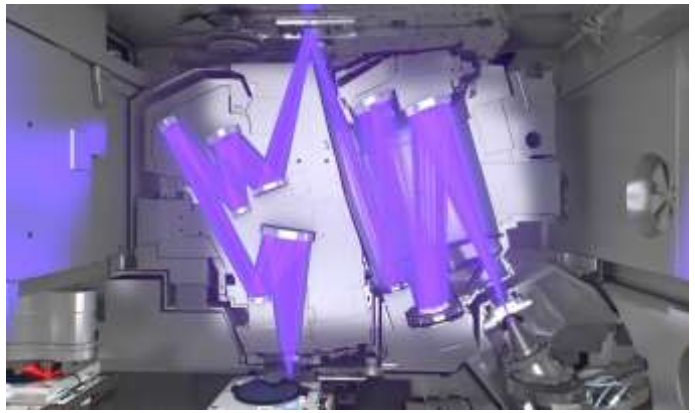
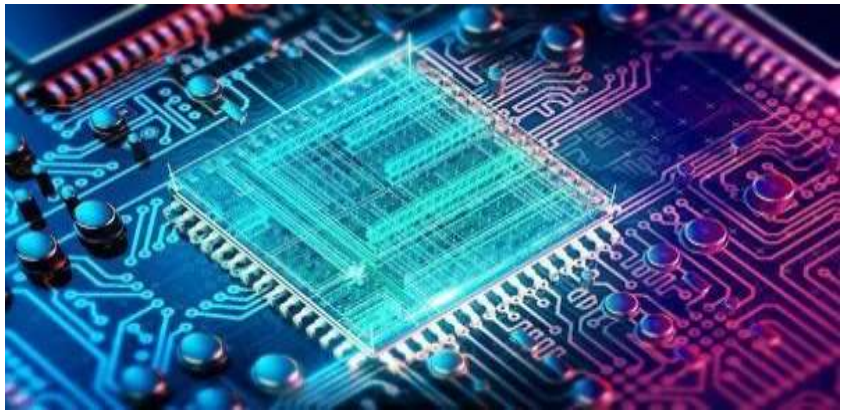
第三节
典型漏洞分析

- MOLES
- Meltdown
- Spectre
- VoltJockey

系统硬件漏洞层出不穷且影响广泛，剖析典型漏洞，探究风险根源，构建硬件防护体系

展望

建设安全无菌，自给自足的系统硬件设计及生产体系


- 突破受制于人的**半导体、光刻机**等卡脖子技术，构建**自主可控**的处理器设计和生产产业链
- **人工智能、量子安全**等新技术的发展和应用，为系统硬件的安全设计和生产提供了新的思路