# 密码软件引擎 vs. 密钥安全技术

**林璟锵**

中国科学院信息工程研究所 信息安全国家重点实验室
中国科学院数据与通信保护研究教育中心

# 国产密码技术的应用推广

- 《密码法》征求意见稿
  - 国家对关键信息基础设施的**密码应用安全性进行分级分类评估**，按照国家安全审查的要求对影响或者可能影响国家安全的密码产品、密码相关服务和密码保障系统进行安全性审查。
- 公安部《网络安全等级保护条例（征求意见稿）》
  - 非涉密网络应当按照国家密码管理法律法规和标准的要求，**使用密码技术、产品和服务。第三级以上网络应当采用密码保护**，并使用国家密码管理部门认可的密码技术、产品和服务。
  - 第三级以上网络运营者应在**网络规划、建设和运行阶段，按照密码应用安全性评估管理办法和相关标准，委托密码应用安全性测评机构开展密码应用安全性评估**。网络通过评估后，方可上线运行，并在投入运行后，每年至少组织一次评估。密码应用安全性评估结果应当报受理备案的公安机关和所在地设区市的密码管理部门备案。
- 网信办《关键信息基础设施安全保护条例（征求意见稿）》
  - 关键信息基础设施中的**密码使用和管理，还应当遵守密码法律、行政法规的规定**。

# 国产密码技术的应用推广

- 密码算法公开
  - SM2/3/4
  - ZUC、SM9
- 应用标准配套
  - 算法工作模式、随机数发生器
  - 数字证书、SSL、安全电子邮件
  - PKCS#7、PKCS#12
  - ……
- 产品实现多元
  - 软件、硬件、固件
  - 混合形式

# 密码模块——软件

- Cryptographic Engine, Cryptographic Module
- Cryptographic Implementation, Cryptographic Software

- 最容易实施的密码模块
- 适用性最好的密码模块

- 安全挑战最大的密码模块实现
  - 密钥安全
  - 随机数安全
    - 国际标准经常使用带密钥的DRBG来实施

# 密钥安全——面临的威胁

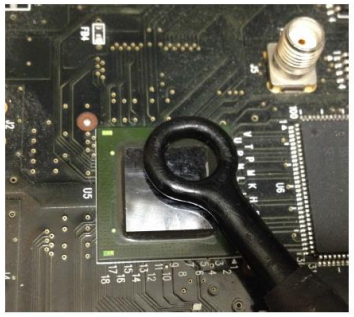
- 侧信道攻击
  - 密码学的传统研究领域之一
- 物理攻击
  - 近年来新型攻击类型
  - 移动终端、物联网终端
- 系统软件攻击
  - 系统安全的传统研究领域之一
  - 软件安全、内存安全
- 密码引擎自身实现漏洞
  - 逻辑漏洞

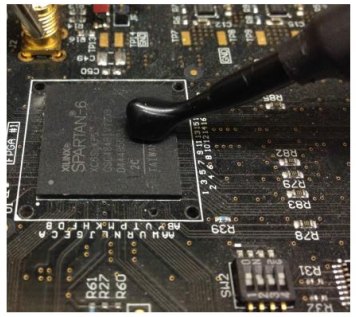
# 密钥安全——面临的威胁

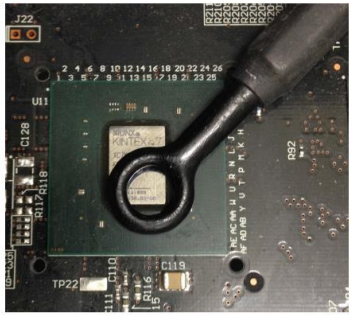
- 侧信道攻击
  - 密码学的传统研究领域之一
- 物理攻击
  - 近年来新型攻击类型
  - 移动终端、物联网终端
- 系统软件攻击
  - 系统安全的传统研究领域之一
  - 软件安全、内存安全
- 密码引擎自身实现漏洞
  - 逻辑漏洞

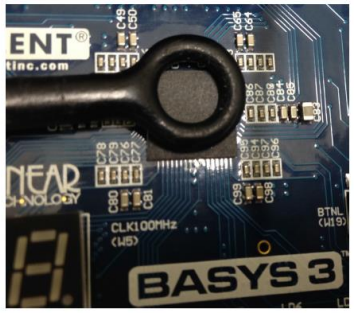
# 侧信道攻击

- 各种形式的侧信道攻击
  - 能量
  - 计时
  - 电磁
  - 声音
  - Shared Cache
  - ……
- 涵盖几乎所有形式的密码模块
  - 硬件
  - 固件
  - 软件









# 侧信道攻防 vs. 密码模块

- 防御措施
  - 对称算法：双轨、查表、……
  - 非对称算法：冗余计算、……

- 对称算法
  - 直接有效的、适合商业产品的方式，也许是：
    - Constant能量、Constant时间、Constant缓存、Constant电磁辐射……
- 非对称算法
  - 直接有效的、适合商业产品的方式，也许是：
    - 随机化：RSA Blinding、ECC Randomized

# 密钥安全——面临的威胁

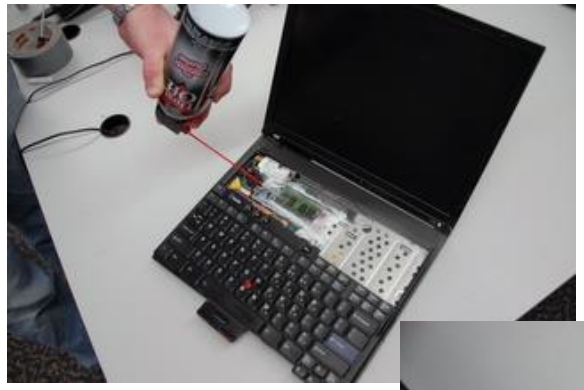
- 侧信道攻击
  - 密码学的传统研究领域之一
- 物理攻击
  - 近年来新型攻击类型
  - 移动终端、物联网终端
- 系统软件攻击
  - 系统安全的传统研究领域之一
  - 软件安全、内存安全
- 密码引擎自身实现漏洞
  - 逻辑漏洞

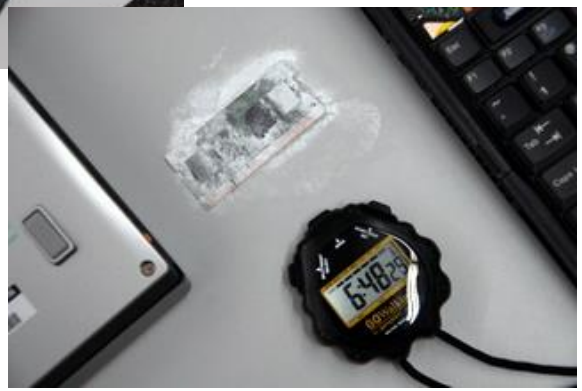
# 物理攻击

- 攻击者与设备有物理接触
  - 结合特定的物理条件，发起攻击

- 冷启动攻击
  - 利用内存芯片的数据延迟消失
- DMA攻击
  - 利用内存数据的DMA传输特性

# 冷启动攻击





- 内存条上的数据，断电之后仍然存在
  - 数分钟～数小时[低温]
- PC、移动设备、DDR3内存
  - 在智能手机上，也有类似效果

Photo from https://citp.princeton.edu/research/memory/media/和
http://www1.informatik.uni-erlangen.de/frost

# 冷启动攻击——防御

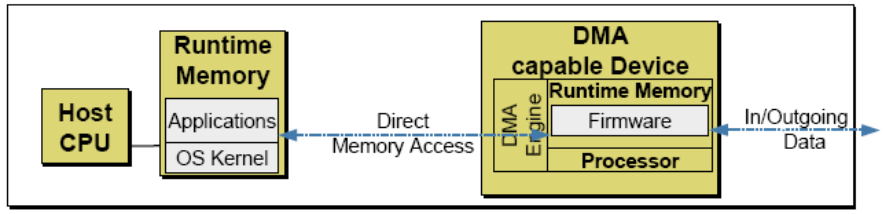
- 基本思路
  - 密钥等敏感信息，不应该出现在内存芯片中
  - 密钥、随机数、中间计算结果……

- 解决方案——密码软件引擎
  - 寄存器
  - 高速缓存Cache

# DMA攻击

- 恶意外设发起DMA请求，读取内存数据
  - 直接访问物理地址的内存空间，绕过操作系统的权限检查

- 防御
  - 需要在外设的访问权限上实施控制
  - BIOS配置




*Direct Memory Access (DMA) capable Computer Platform*

# 密钥安全——面临的威胁

- 侧信道攻击
  - 密码学的传统研究领域之一
- 物理攻击
  - 近年来新型攻击类型
  - 移动终端、物联网终端
- 系统软件攻击
  - 系统安全的传统研究领域之一
  - 软件安全、内存安全
- 密码引擎自身实现漏洞
  - 逻辑漏洞

# 系统软件攻击——软件漏洞、防不胜防

- 操作系统的隔离漏洞
  - 恶意进程访问其它进程的内存空间
  - 恶意进程访问操作系统内核的内存空间
  - CAN-2005-0400: Information leak in the Linux kernel ext2 implementation, 2005
  - CVE-2014-0069. CVE-2014-4653
- 操作系统功能以及其它正常功能
  - Core dump, Crash report
- 内存重用/未清零
  - CWE-212: Improper cross boundary removal of sensitive data
  - CWE-226: Sensitive information uncleared before release
- ……

# 系统软件攻击

- 利用硬件漏洞
  - Meltdown
  - Spectre
  - 以及各种后续的变形

# 密钥安全——面临的威胁

- 侧信道攻击
  - 密码学的传统研究领域之一
- 物理攻击
  - 近年来新型攻击类型
  - 移动终端、物联网终端
- 系统软件攻击
  - 系统安全的传统研究领域之一
  - 软件安全、内存安全
- 密码引擎自身实现漏洞
  - 逻辑漏洞

# 密码引擎自身实现漏洞

- OpenSSL心脏出血——最典型的实例
  - 在心跳响应消息中，将进程中的内存数据发送出去
    - 随机地址的64K字节

- Attacking and Fixing PKCS#11 Security Tokens
  - ACM CCS 2010
  - PKCS#11接口实现问题，可以导出明文形式的密钥

- ……

# 思考

- 侧信道分析
  - 在计算过程中，各步骤的差异【密钥相关】
  - 外部检查到差异

  - 随机化的解决方案——差异与密钥并不直接相关/随机化因子
  - 消除外部差异
    - Constant time

- 密码引擎自身实现漏洞
  - 高素质人才
  - ……

# 思考

- 系统软件攻击和物理攻击
  - 密钥数据 vs. 普通内存数据

- 密钥——影响到大量数据的关键数据
  - 在软件实现中，没有得到专门的安全措施
  - 都是普通的数据变量
  - 与其它数据同等处理、同等对待

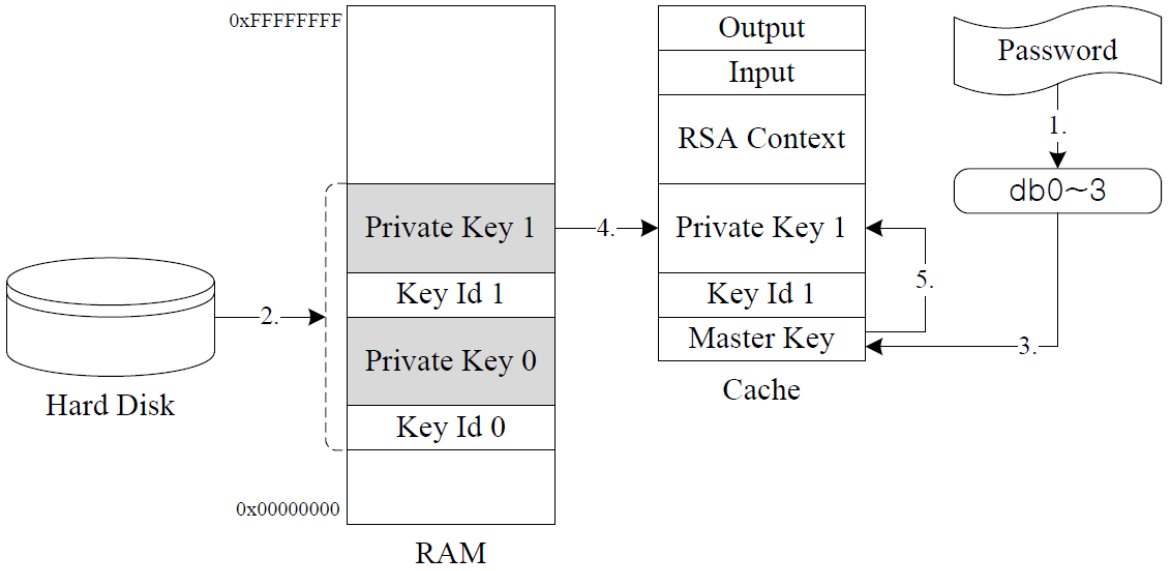
```
void Decrypt()
{
  unsigned char key[256/8];
  unsigned char *input, *output;
  ...
```

# 我们的前期工作介绍

- 基于Cache的密钥安全方案——第一次在Cache上计算的公钥密码算法实现
  - 抵抗冷启动攻击
  - Copker: Computing with Private Keys without RAM, NDSS 2014
- 基于Intel TSX的密钥安全方案——第一次结合事务内存和密码应用
  - 结合Intel事务内存技术，实现密码计算，抵抗零日漏洞的内存信息泄露攻击、冷启动攻击和DMA 攻击；
  - Protecting Private Keys against Memory Disclosure Attacks using Hardware Transactional Memory, IEEE S&P 2015
- 基于门限密码的SM2拆分实现
  - 移动终端和服务器协同计算，集中式的密码计算使用控制/密钥安全
  - 中国发明专利ZL2014104375995

# Copker方案

- 在CPU的高速缓存Cache中执行公钥密码计算，抵抗冷启动攻击
- 系统架构
  - 在Debug寄存器中保存AES Master Key
    - 始终在Debug寄存器中
    - USENIX Security 2010方案
  - RSA私钥[密文]载入内存
    - Cache的WB模式
  - 计算时，将AES Master Key在内核中定义的专门内存区
  - 访问RSA私钥[密文]，在该内存区中，解密得到RSA私钥[明文]、进行计算
  - 清除该内存区
    - 保留计算结果



# Copker方案——保证密钥数据在Cache

- 使用WB模式，数据“尽可能”保持在Cache中
  - 最正常的Cache访问模式
- 在计算期间，必须保证：密钥数据不会从Cache同步到内存芯片
- 如下情况需要考虑：
  - 计算的输入/输出
  - 计算任务访问的数据
  - 任务调度和中断处理
  - 共享Cache核的内存访问

- 相应的解决方案

# 从RSA算法到ECC算法

- 随机数DRBG发生器，也需要抵抗冷启动攻击
  - 同样，也是需要在特定的内存区内进行计算
  - DRBG状态更新后，以密文形式更新到内存区
  - 熵收集，以密文状态
- Copker: A Cryptographic Engine against Cold-Boot Attacks
  - IEEE Transactions on Dependable and Secure Computing (TDSC), 2018.

**Algorithm 2.** ECDSA Signing

**Input**: $digest, d, G, n$
**Output**: $signature$

1: $k \leftarrow \text{random} \in (1, n-1]$;
2: $(x_k, y_k) = k \times G$;
3: $r = x_k \bmod n$;
4: $s = k^{-1} \cdot (Str2Int(digest) + r \cdot d) \bmod n$;
5: $signature = Int2Str(r, s)$;

# Mimosa方案

- 在Intel CPU事务内存中，执行公钥密码计算，同时抵抗：
  - 冷启动攻击、DMA攻击

- 事务内存/Intel TSX硬件特性
  - 事务内存是CPU的内存访问模式，共享数据冲突访问的一种解决方法
  - 该模式用于临界区的代码
  - 某一段代码R处于该模式，则
    - 一旦有其它代码“同时”访问了R所访问的数据、且其中一方是写操作
    - R会自动回滚到临界区的开始位置、所有的执行痕迹都被自动全部清除
      - 而且，不论其它代码是否处于Transactional Memory模式

  - 可使用硬件或软件来实现——Intel TSX硬件特性

# Mimosa方案基本原理

- 在Debug寄存器中保存AES Master Key

- 在私钥计算时，如下代码处于事务内存模式：
  - AES主密钥将RSA私钥解密为明文
  - RSA私钥签名或解密
  - 清零RSA私钥和所有中间计算变量
    - 保留计算结果

- 一旦有攻击进程试图读取内存中的RSA私钥
- 事务内存自动将上述代码回滚到初始状态
  - 即，RSA私钥仍处于密文状态
  - RSA私钥明文——“写操作”结果
- 攻击者只会读取到密文状态的RSA私钥
- 硬件机制自动完成

```
while(!success){
    int times = 0;
    /* Disable interrupts and preemption */
    get_cpu();
    local_irq_save(flags);
#ifdef TSX_ENABLE /* Switch of Mimosa_NO_TSX */
    while(1){
        int status;
        if(++times == MAX_TRIES)
            goto delay;
        status = _xbegin();
        if(status == _XBEGIN_STARTED)
            break;
    }
#endif
    mimosa_protected_compute(keyid, in, out);
    success = 1;
#ifdef TSX_ENABLE
    _xend();
#endif
delay:
    /* Enable interrupts and preemption */
    local_irq_restore(flags);
    put_cpu();
    if(!success){
        /* Delay after several aborts */
        set_current_state(TASK_INTERRUPTIBLE);
        schedule_timeout(10);
    }
}
```

# Mimosa方案的技术挑战

- 事务内存，原本设计用于保护临界区代码
  - 极其少量的几行语句
- 能否用于完成耗时的公钥密码计算？

- 需要完成大量工作
  - 去除不兼容指令
  - 去除“非攻击性”的数据冲突
  - 禁止中断和内核抢占调度
  - 连续回滚/多次执行不成功的延迟处理

# 安全性分析

- 系统软件攻击
  - 各种不同原理的系统软件攻击，最后一步“读取内存”
  - 会导致事务回滚，攻击者只能得到RSA密钥的密文
- 侧信道攻击
  - 抵抗Cache-based侧信道
  - 所有计算都在Cache中完成，Intel TSX保证
- 物理攻击
  - 冷启动攻击
    - 所有计算都在Cache中完成，Intel TSX保证
  - DMA攻击
    - 类似于系统软件攻击

# 支持更大内存的密码计算

- 事务任务执行，通常是极小的代码
  - 例如，标记变量置位

- 将一次密码计算拆分为多个硬件事务
  - 中间计算结果，对称加密
  - 防止敏感信息泄露

- Mimosa: Protecting Private Keys against Memory Disclosure Attacks using Hardware Transactional Memory
  - IEEE Transactions on Dependable and Secure Computing (TDSC), accepted.

# SM2密钥拆分协作方案

- SM2私钥由2份组成
  - 一份位于远程的硬件密码设备
  - 一份位于移动终端
- SM2的解密、数字签名，由双方合作完成

# 门限密码算法——基本要求

- 密钥生成：
  - 双方都没有处理过完整的私钥
- 解密、签名操作
  - 私钥不合成；双方分别各自对数据进行处理，不向对方泄露任何有关私钥信息
  - 通信过程数据，不泄露任何有关私钥信息

- **不合成**
  - **不是“有私钥相关数据、而不去合成”，而是“所掌握的数据，不可能用来合成完整私钥”**
- **不泄露信息**
  - **不是合成私钥后计算，而是各自计算、再将计算结果合成**
  - **计算结果不会泄露任何有关私钥信息、不包括任何有关私钥信息**

# 用于2方协作的门限密码算法

- 扩展需求
  - 密钥生成尽可能不需要可信中心
  - 交互通信轮数少
  - 计算量增加少

# 拆分协作计算方案
# SM2 vs. RSA

- RSA
  - $\mathrm{d} = \mathrm{d}_1 + \mathrm{d}_2$
  - $\mathrm{s} = m^d = m^{d_1} m^{d_2}$
- SM2
  1. 产生秘密$D_1$，求$D_1^{-1}$，并计算$D_1^{-1}G$发送给对方
  2. 产生秘密$D_2$，计算公钥$P=D_2^{-1}(D_1^{-1}G)-G=dG$
     - $D=(1+d)^{-1}=D_1*D_2$
  - 签名r, s
    - $(\mathrm{x}, \mathrm{y}) = k\mathrm{G}$，$\mathrm{r} = \mathrm{m} + \mathrm{x}$
    - $s = (1+d)^{-1}(k - rd) = (1+d)^{-1}(k+r) - r = \mathrm{D}(\mathrm{k}+\mathrm{r}) - \mathrm{r}$
    1. 产生随机数$k_2$，计算$k_2G$发送给对方
    2. 产生随机数$k_1,k_3$，计算$k_1(k_2G)+k_3G=(x,y)$和$r=m+x$；计算$D_1k_1$和$D_1(r+k_3)$发送给对方
    3. 计算$s=D_2(D_1k_1)k_2+D_2(D_1(r+k_3))=D_1D_2(k_1k_2+r+k_3)=(1+d)^{-1}(k+r)$
       - $k=k_1k_2+k_3$

# 拆分协作计算方案
# SM2 vs. RSA

- 扩展需求
  - 密钥生成尽可能不需要可信中心
  - 交互通信轮数少
  - 计算量增加少

- SM2
  - 不需要可信中心
  - 1+2次消息
    - 其中，第1次消息可预计算
  - 约3倍计算量
    - 在线计算量2倍
- RSA
  - 需要可信中心
  - 2次消息
  - 约8倍计算量

# 更多的思考

- 密码软件引擎的密钥安全
  - 密码学+系统安全+网络安全

- 白盒密码
  - 安全算法设计？
  - Code lifting攻击
    - 系统安全解决方案？实现硬件绑定？
- 更多的CPU相关攻击？
- 如何在国产计算平台上实施？

# 谢谢！

林璟锵

linjingqiang@iie.ac.cn