Uses and Misuses of Key
密钥视角下的密码学安全

李卷孺
上海交通大学密码与计算机安全实验室
GoSSIP @ LoCCS

大纲

- **Crypto 101**
- **密钥管理：最佳实践与攻击**
- **密钥安全审计**
- **现实软件（二进制）中的密钥安全分析**

21小时精通密码学？！

CRYPTO 101

GoSSIP

传统的密码学教程

误区

- 软件开发人员面对的是密码学API和密码学库

Chapter 1. OpenSSL

Getting Started
- Determine OpenSSL Version and Configuration
- Building OpenSSL
- Examine Available Commands
- Building a Trust Store

Key and Certificate Management
- Key Generation
- Creating Certificate Signing Requests
- Creating CSRs from Existing Certificates
- Unattended CSR Generation
- Signing Your Own Certificates
- Creating Certificates Valid for Multiple Hostnames
- Examining Certificates
- Key and Certificate Conversion

Configuration
- Cipher suite selection
- Performance

OpenSSL
Cryptography and SSL/TLS Toolkit

Table of Contents

Sodium
crypto library

- 密码学教材和密码学软件之间存在巨大的鸿沟

密码学实践需求

- 密码算法的正确使用
  - 理解算法的适用范围
  - 理解密码算法接口的调用方式
  - 正确生成密码算法需要的各项参数（密钥、IV、Nonce等）

- 安全管理机密信息
  - 密钥生成材料
  - 密钥存储格式
  - 密钥生命周期控制

如何正确使用密码学

- 一个密钥为中心的密码学安全分析思路

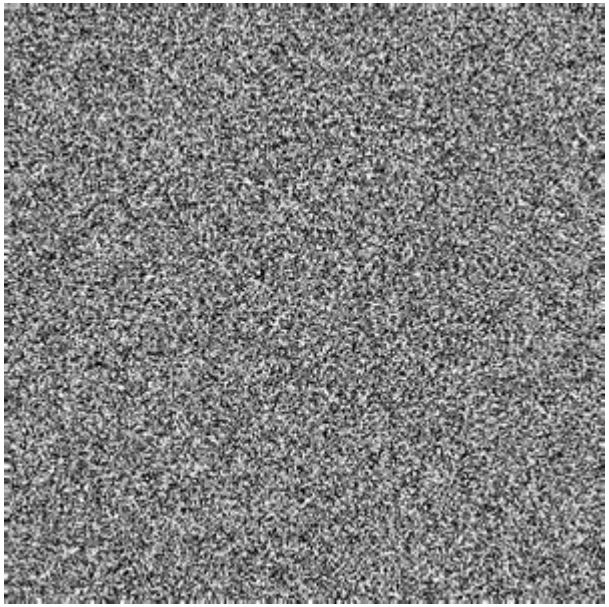
1. 了解基本密码学概念
2. 学习使用常见密码库
3. 安全管理和使用密钥
4. 分析软件中的密钥误用

随机性、单向函数与单向陷门函数

密码学理论核心概念

随机性

- 随机性是密码学安全的基石
  - 信息论与概率
  - 不可区分性
  - 复杂性
  - 困难问题



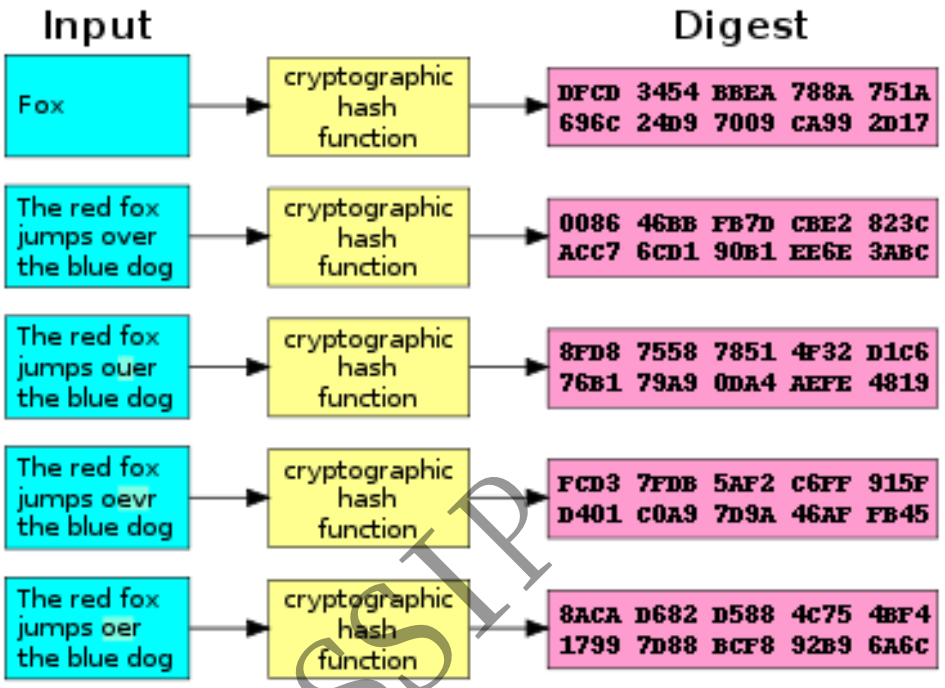
- pseudorandom generator
  - 将短随机数生成长的伪随机序列

基本概念：伪随机置换

- 通过将一个集合（伪随机地）映射到另一个集合而产生的安全变换
- 将一个集合通过某种特定的规则，映射到一个新的集合上去
- 只有掌握特定规则，才能计算出每一个映射，否则只能通过查询
- 从观察者的角度，这个置换仿佛是随机的

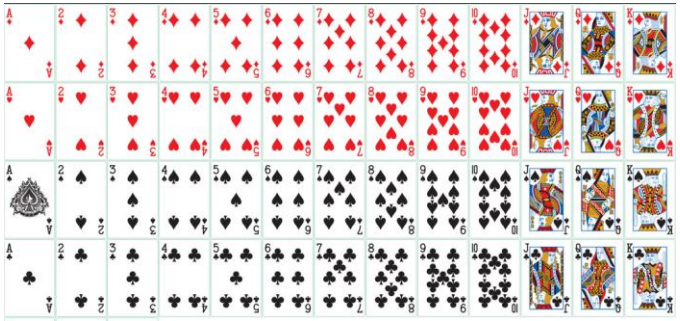
基本概念：单向函数

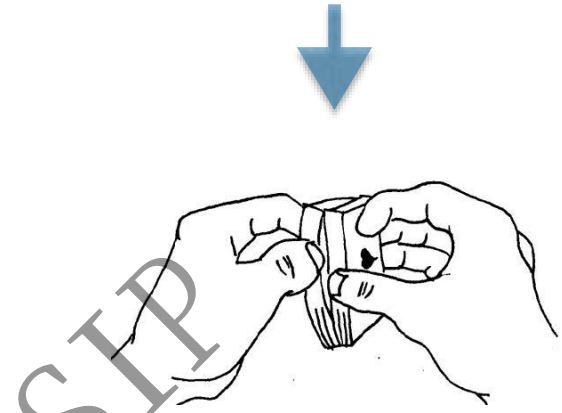
- 将一个集合根据特定规则映射到另一个集合
- 求逆困难



单向陷门函数

- 看似单向函数
- 实则伪随机置换



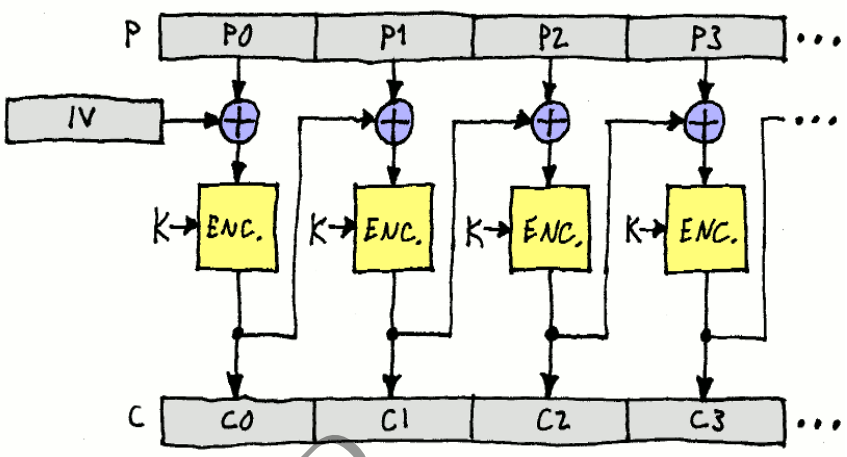


对称加密、公钥密码学和Hash函数

密码学算法：核心概念的具体化

Secret-key Cryptography

- 对称加密算法
  - 分组加密算法
  - 流加密算法
- 加密模式
- 认证加密



Public-key Cryptography

- 公钥密码算法的典型用途
  - 密钥协商
  - 签名与验证
- 椭圆曲线
  - ECDH、ECDSA
  - SM2
- 填充方式
  - RSA PKCS
  - RSA OAEP

The RSA Algorithm

P and Q are very large prime numbers.

$N = P\,Q$.

$\varphi(N) = (P - 1)\,(Q - 1)$.

$E\,D \equiv 1 \pmod{\varphi(N)}$.

Bob's public key = (E, N).

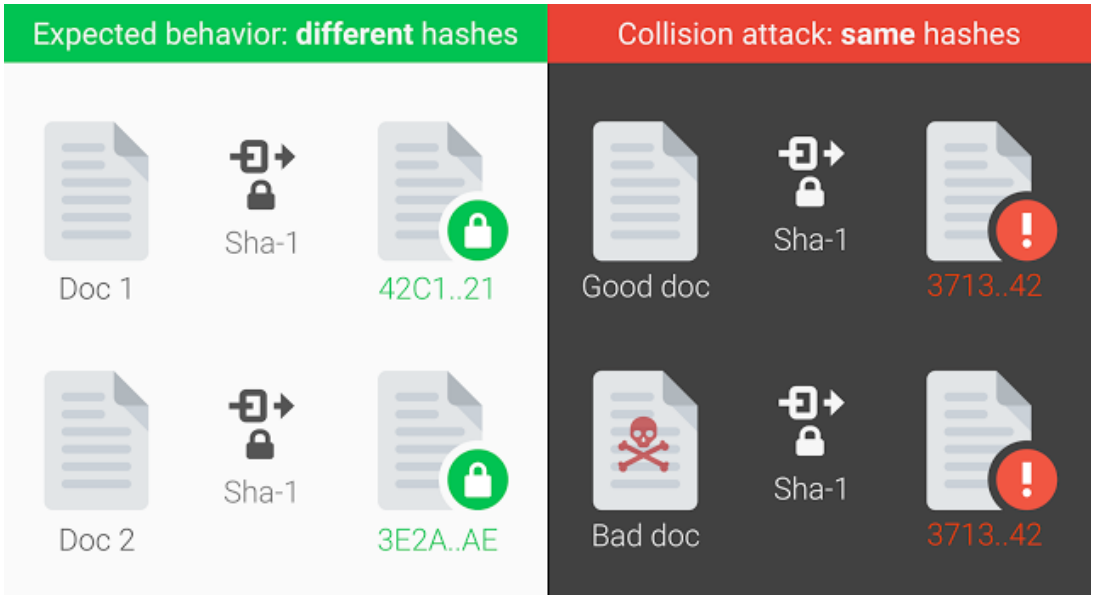
Bob's private key = (D).

The public operation: $C = M^E \pmod{N}$.

The private operation: $M = C^D \pmod{N}$.

Hashing

- Hash算法发展史
  - MD5，SHA-1
  - SHA-2
  - SHA-3
- Hash-Then-MAC
- 基于Hash函数的口令存储/密钥生成



从OpenSSL到Libsodium

逐步完善的密码算法库

密码算法库大阅兵

经典算法库	现代密码算法库	SSL库	系统API
Botan	*Cryptlib*	*OpenSSL/LibreSSL/BoringSSL*	*Microsoft CryptoAPI*
Crypto++	*Keyczar*	*XySSL/PolarSSL/TropicSSL/mBed TLS*	*Linux Crypto API*
Gcrypt	*NaCL*	*MatrixSSL*	*iOS Crypto Services*
Mcrypt	*Sodium*	*CyaSSL/WolfSSL*	*Android javax.crypto*
Nettle		*GnuTLS*	
Tomcrypt		*NSS*	
……	……	……	……

以及无数软件中出现或开源的密码算法零散实现……

密码算法库演化史

- 算法和协议支持的发展
  - 算法不断更新：AES、SHA-3、Salsa20、Curve25519
  - 加密模式更新：GCM、AEAD
  - 协议支持升级：从 SSL 3.0 到 TLS 1.2(1.3 draft)

- 算法库对提供的接口进行了清晰的定义
  - 减少参数误用的发生

- 软件安全考量
  - Secure memory
  - Mitigation against timing attack

如何正确使用密码算法库

- 选择支持新标准的算法库
  - 可认证加密
  - 椭圆曲线
  - 是否淘汰过期的算法MD5、SHA1、RC4和协议（SSL v3.0）
- 算法库是否能够在内部保证计算和存储安全
  - Secure memory management
  - Constant time computation
- 使用算法库提供的高层接口（e.g., NaCL）

```
#include "crypto_secretbox.h"

std::string k;
std::string n;
std::string m;
std::string c;

c = crypto_secretbox(m,n,k);
```

“一切安全性寓于密钥之中”

密钥管理：最佳实践与攻击

GoSSIP

Kerckhoffs's principle

- *a cryptosystem should be secure even if everything about the system, **except the key**, is public knowledge.*
- *"The enemy knows the system." (by Claude Shannon)*
- 优势
  - 保证一个密钥的机密性远比保证复杂密码系统的机密性容易
  - 一旦泄密，更换密钥的成本极低
- *扩展阅读：*
  - *http://cryptography.wikia.com/wiki/Kerckhoffs%27s_Principle*

密钥生成、密钥交换和密钥存储

密码管理的典型应用场景

密钥生成

- 对称密码加密算法的密钥
  - 通过password推导出密钥
  - 通过key agreement协商得到密钥
  - 通过预共享密钥和密钥生成函数（Key Derivation Function）

- 非对称密码算法的密钥
  - 公钥：通过证书获得并验证
  - 私钥：通过passphrase解密使用

密钥交换

- SSL通讯中的Key Establishment
  - Discrete Log Key Agreement Schemes
  - RSA Key Establishment

- 私有协议中的密钥交换
  - 直接发送
  - 白盒加密发送
  - RSA公钥加密发送

How Does Your PC Talk to a Secure Web Site?

Prof. Alan Kaminsky
Rochester Institute of Technology -- Department of Computer Science

June 30, 2010

密钥存储

- 软件安全存储方式
  - key encryption key (KEK)
- 操作系统机制
  - Android KeyStore
  - iOS KeyChain
- Hardware based Storage
  - TEE：trusted storage

Best Practices

密码管理的最佳实践

Key Management

- NIST SP 800-57 关注的要点
  - 生命周期中的状态：the states in which a cryptographic key may exist during its lifetime.
  - 涉及的函数：the multitude of functions involved in key management.

NIST Special Publication 800-57 Part 1
Revision 4

Recommendation for Key Management

Part 1: General

Key Generation

- 密钥生成的基本要求
  - 密钥的熵：密钥材料的随机性
  - 密钥变换算法：防止密钥材料下毒
  - 密钥的长度
  - 密钥的验证：检查密钥的合法性

NIST Special Publication 800-133

Recommendation for Cryptographic Key Generation

Elaine Barker
Allen Roginsky

Password Hashing

- 错误的用法：直接使用消息摘要算法
- 正确的用法：
  - PBKDF2
  - Bcrypt
  - Argon2（Winner of Password Hashing Competition）

- 一些私有方法
  - WinRAR
  - 7-Zip

```
hash("hello") = 2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
hash("hbllo") = 58756879c05c68dfac9866712fad6a93f8146f337a69afe7dd238f3364946366
hash("waltz") = c0e81794384491161f1777c232bc6bd9ec38f616560b120fda8e90f383853542
```

内存中密钥安全存储与清理

- Secure Memory
  - 安全分配
  - 及时销毁!

- Key Wrapping
  - 通过特定的保护方式，隐藏内存中的密钥数据
  - 如果需要尽可能减少密钥的出现，还可以改造密码算法

健壮的密钥交换

- 安全密钥交换的几个关键点
  - 密钥交换的每一方都要参与决定密钥的内容！！！
    - 换言之，不能存在有一方能够操纵密钥内容的情况发生
  - Authentication：抵抗中间人攻击
    - 或者依赖可信第三方认证
    - 或者依赖双方共享的秘密信息
- 常见的问题在于忽视了Authentication
  - Android平台APP SSL证书校验问题

Attack Vectors

针对密钥的安全攻击

Cold-boot Attack

- Lest We Remember
  - https://citp.princeton.edu/research/memory/

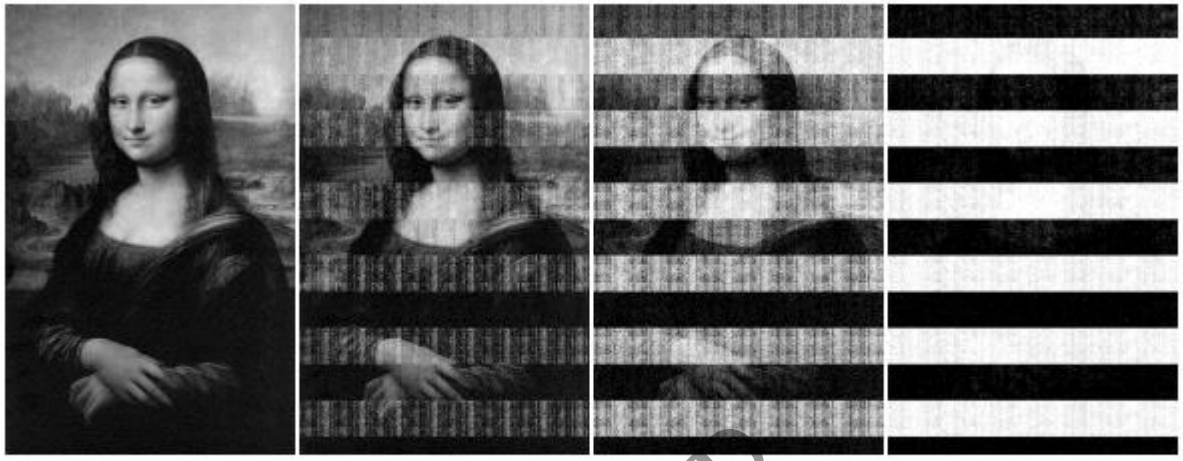


Figure 4: We loaded a bitmap image into memory on Machine A, then cut power for varying lengths of time. After 5 seconds (left), the image is indistinguishable from the original. It gradually becomes more degraded, as shown after 30 seconds, 60 seconds, and 5 minutes.

Fault Attack

- 真实案例
  - https://people.redhat.com/~fweimer/rsa-crt-leaks.pdf

- 故障攻击导致密钥轻微改变，即可令攻击者恢复密钥
  - RSA 故障攻击
  - 分组加密算法故障攻击

[17] Hillstone Networks, 山石网科关于第三方某型号硬件器件设计缺陷可能导致信息泄露问题的声明. July 2015. http://www.hillstone.com.cn/services/2015/0728/20150728.html (retrieved 2015-08-26)

Physical Key Extraction

- DPA竞赛
  - http://www.dpacontest.org/home/
- DPA 恢复实际密钥：IoT Goes Nuclear: Creating a ZigBee Chain Reaction)

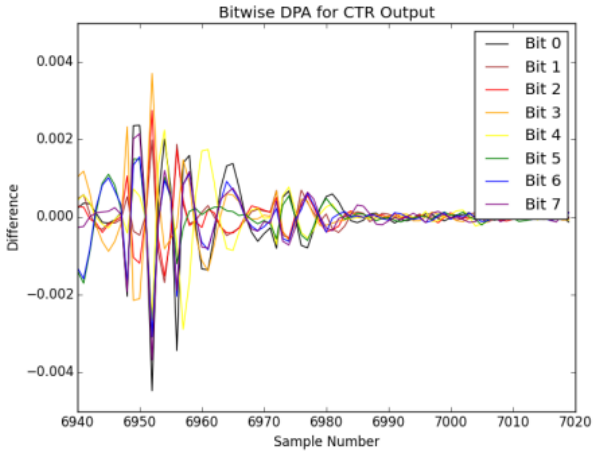
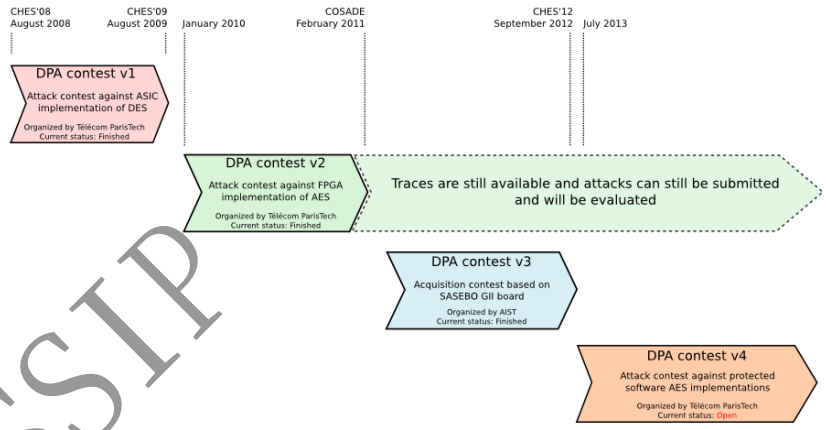


Figure 6. Bitwise DPA attack on AES-CTR 'pad', where all 8 bits are recovered.



Side-Channel Attack

- 通过运行过程的时间差异来恢复密钥的信息
  - 即使是最新的信息系统中依然存在大量可被攻击的问题
  - 学术会议上计时攻击依然是热点

Track 2

Side-Channel Attacks I

Prime+Abort: A Timer-Free High-Precision L3 Cache Attack using Intel TSX

Craig Disselkoen, David Kohlbrenner, Leo Porter, and Dean Tullsen, *University of California, San Diego*

The buoyancy of castles: Examining the effectiveness of mitigations against floating-point timing channels

David Kohlbrenner and Hovav Shacham, *UC San Diego*

Constant-Time Callees with Variable-Time Callers

Cesar Pereida García and Billy Bob Brumley, *Tampere University of Technology*

Forensics based Key Retrieving

- 当加密算法的软件实现不清理密钥时，残留在内存中的信息可被恢复！

- 应用：
  - WannaCry in-memory key recovery

Putty

```
C:\>aes-finder.exe putty.exe
Searching PID 2180 ...
[0016C904] Found AES-256 encryption key: 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
[0016C9F4] Found AES-256 decryption key: 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
[00AA4540] Found AES-256 encryption key: e2855dae921dba978ffd713e89540101144de13f468fd5da8623608255387dbc
[00AA4720] Found AES-256 decryption key: e2855dae921dba978ffd713e89540101144de13f468fd5da8623608255387dbc
[00AA5458] Found AES-256 encryption key: 892372c26c5601a28e5dd20a64976faa219907c6c9a33a9d1ff3376609bd53f7
[00AA5638] Found AES-256 decryption key: 892372c26c5601a28e5dd20a64976faa219907c6c9a33a9d1ff3376609bd53f7
Done!
```

Memory Disclosure

- OpenSSL 'Heartbleed' vulnerability (CVE-2014-0160)

- OpenSSH Vulnerability Leaks Private Crypto Keys
  - An information sharing flaw (CVE-2016-0777)
  - A less harmless buffer overflow flaw (CVE-2016-0778)

Key Cracking

- 当其他方式都失效时，还有暴力破解一条路~
- 典型案例：
  - 彩虹表攻击DES
  - RSA 512-bit 公钥分解
- 硬件加速
  - Hashcat工具

基于密钥生命周期的安全分析

密钥安全审计

GoSSIP

Enhancement

安全使用密钥的建议

密钥安全使用九规则

- 1 使用足够复杂的密钥生成算法
- 2 避免密钥长度不足
- 3 密钥协商过程中所有参与方一同决定密钥
- 4 不要在内存中明文存放密钥
- 5 使用不透明的格式存储密钥
- 6 避免密钥操作遭受旁路攻击
- 7 密钥在内存中只保留一份
- 8 “阅后即焚”策略
- 9 无关代码不得访问密钥

规则一：足够复杂的密钥生成

- 使用足够复杂的密钥生成算法
  - Bcrypt
  - Scrypt
  - Argon2
- 对抗密钥暴力搜索攻击
  - 生成算法不可并行计算
  - 占用大量内存（对抗GPU）
- 常见错误：仅仅用一次hash函数计算出密钥

规则二：避免密钥长度不足

- 密钥长度不足导致暴力破解可行
  - 密钥材料的信息量不足
  - 密钥生成过程中出现信息截断
  - 密钥实际使用过程中长度设定错误

- Keylength - Cryptographic Key Length Recommendation
  - https://www.keylength.com/en/

- 常见错误：
  - 密钥材料的信息量不够，生成的密钥长度无意义

规则三：多方参与密钥协商

- 密钥协商过程中所有参与方一同决定密钥
  - 每一参与方提供一定的密钥材料
  - 通过安全的计算方式生成会话密钥

- 目的
  - 假定通讯双方中有一方是恶意参与者，也无法制造一个有漏洞的密钥通讯过程

- 常见错误
  - 仅仅由client或server中一方生成密钥

规则四：在内存中少出现密钥明文

- 隐藏密钥在内存中的明文出现
- 目的
  - 抵御逆向工程
  - 抵御恶意代码注入，在内存中追踪密钥
  - 抵御基于内存dump的密钥搜索
- 常见错误
  - 密钥长期存放在内存中，使用前不保护、使用后不清理

规则五：密钥格式多样化

- 使用不透明的格式存储密钥
  - 实例：OpenSSL 1.1.0后的版本，不再支持直接访问密钥结构体
- 目的
  - 密钥格式不固定，方便支持多平台上不同的布局
  - 增加随机性和多样性，可以提供一定的安全性
- 常见错误
  - 密钥的内存布局固定

规则六：常量时间密钥操作

- 对于所有涉及密钥的敏感操作，避免使用分支操作

- 目的
  - 避免密钥操作存在时间差异，遭受旁路攻击

- 常见错误

```
bool cmp ( unsigned char * src, unsigned char * dest, size_t len )
{
    for ( size_t i = 0; i < len; ++i )
    {
        if ( src[i] != dest[i] )
            return false;
    }
    return true;
}
```

规则七：密钥不可复制原则

- 密钥在内存中只应该保留一份

- 目的
  - 避免不必要的密钥访问
  - 避免内存泄漏时密钥泄漏
  - 密钥管理时只需要考虑一份实例

- 常见错误
  - 对密钥对象进行复制传递

规则八：及时清理密钥

- “阅后即焚”策略
  - 密钥在使用完毕后应该强制清理
- 目的
  - 尽可能缩短攻击者的攻击时间窗口
- 常见错误
  - 仅仅使用free清理分配的heap memory
  - 常见的free实现并不会马上清理内存数据内容

规则九：无关代码不得访问密钥

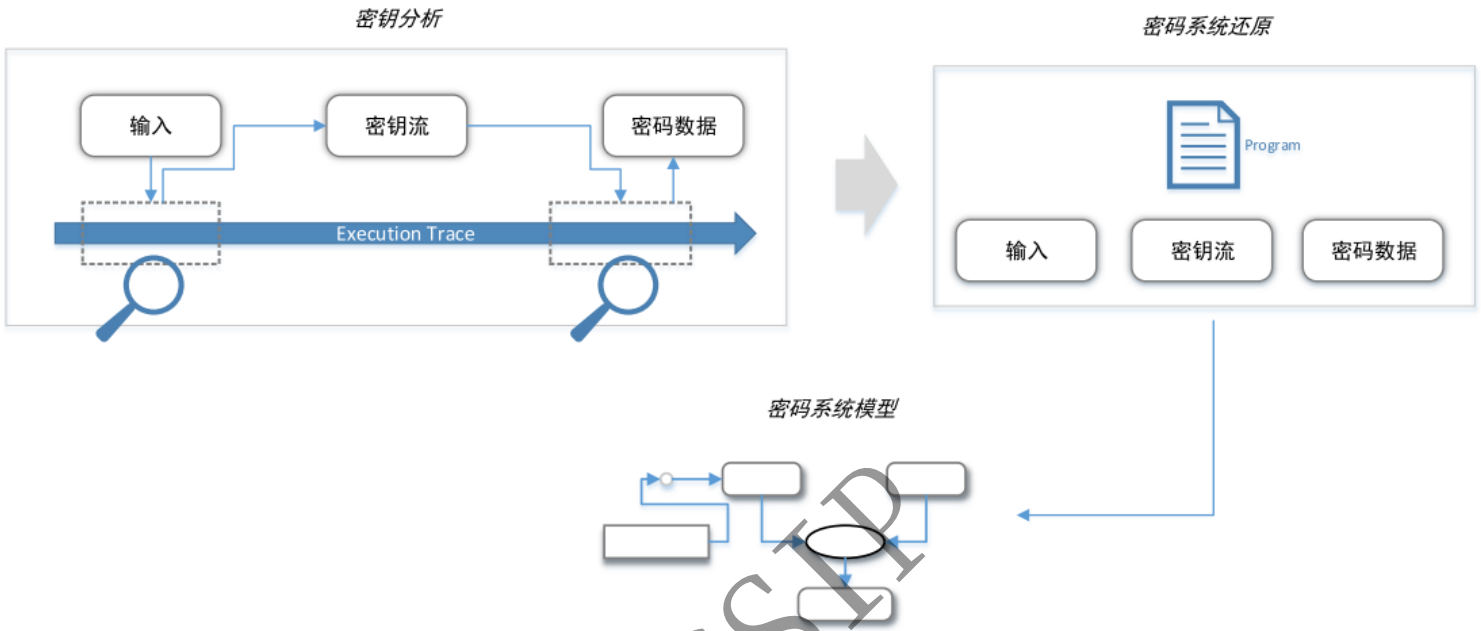
- 访问密钥的只能是密码算法
- 目的
  - 防止有意或者无意的信息流传递
- 常见错误
  - 密钥被其它无关函数访问，导致隐式信息流传递

审查密钥生命周期

- 密钥从生成到传播到销毁的全过程

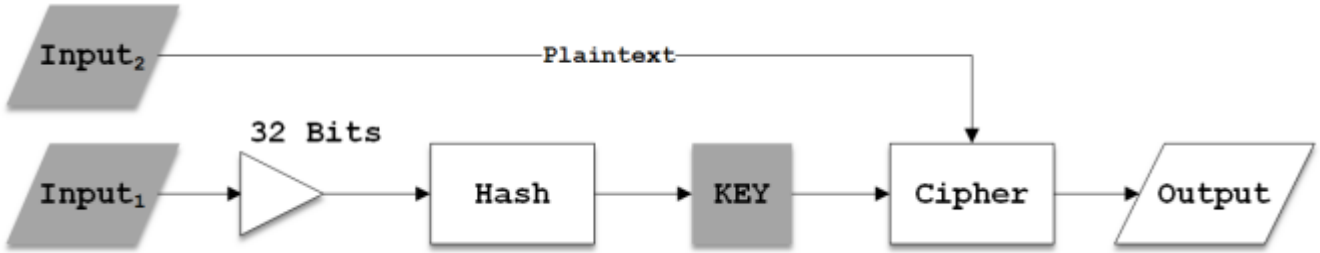
分析方法

- *从密钥分析到密码系统模型建立*



模型

- *密码系统模型*



Misuses of Crypto Key

典型的密钥误用

现实中的密码学误用

- *本课程要介绍的误用，特指在使用密码算法时，由于逻辑错误问题，导致密码系统的安全性遭到削弱甚至完全破坏的情况*
- *代码漏洞、人为操作失误、旁路信息泄漏等不在我们的讨论范围之内*
- *算法级别的安全分析不在讨论之列*

密钥生成问题

- 固定密钥
  - 腾讯移动分析SDK用一个固定的RC4加密密钥加密一段经过gzip压缩的json数据，然后将数据直接发送给服务器

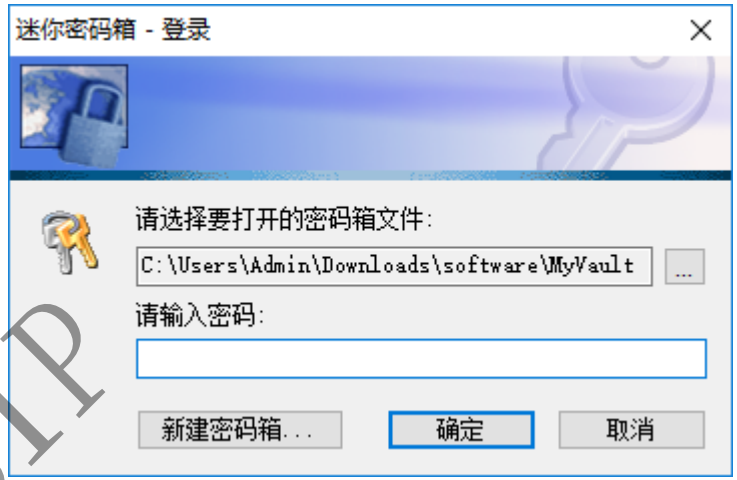
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

[{"et":2,"mid":"none","idx":12988,"mc":"02:00:00:00:00:00","ky":"IWXOPwx7c9caf15a038b0d8","ifv":"306007BC-5A39-4C68-B5E0-BEECB3B6DF94","ev":
{"cn":"WIFI","ov":"9.0.1","lg":"zh-Hans-CN","tz":"Asia\/Shanghai","os":2,"md":"iPhone5,3","sv":"1.4.13","ch":"appstore","wf":"{\"ss\":\"Random_new_storage\",\"bs\":
\"a0:63:91:d2:64:9b\"}","sr":"640x1136","pl":"iPhone
OS","mf":"iPhone","av":"5.1.1"},"jb":1,"ui":"d41d8cd98f00b204e9800998ecf8427e957e2d9d","si":1927465380,"ts":1488263549,"ut":0,"cfg":{"1":{},"2":{}}},
{"et":1000,"idx":12989,"mid":"none","mc":"02:00:00:00:00:00","ky":"IWXOPwx7c9caf15a038b0d8","ifv":"306007BC-5A39-4C68-B5E0-
BEECB3B6DF94","ui":"d41d8cd98f00b204e9800998ecf8427e957e2d9d","sv":"1.4.13","ch":"appstore","ei":"onBackground_WX","si":1927465380,"ts":1488263580,"ut":0,"av":"5.1.1"}]

密钥生成问题

- 迷你密码箱
  - 口令直接使用SHA-1算法做一次变换后作为登陆token
  - 和加密文件中的token比对

- 问题
  - 可以快速暴力搜索token



密钥生成问题

- Cryptcat

2.2 密钥初始化

`twofish2.cpp` 中的 `generateKey` 函数，功能是将用户提供的字符串password转换成twofish的128bit密钥。

```
static char key[32];
char* generateKey( char* s ) {
    int sIdx = 0;
    for ( int i = 0; i < 32; i++ ) {
        char sval = *( s + sIdx );
        if (( sval >= '0' ) && ( sval <= '9' )) {
            key[i] = sval;
        } else if (( sval >= 'a' ) && ( sval <= 'f' )) {
            key[i] = sval;
        } else {
            int q = sval%16;
            if ( q < 10 ) {
                key[i] = ('0' + q);
            } else {
                key[i] = ('a' + q - 10);
            }
        }
        sIdx++;
        if ( *( s + sIdx ) == 0 ) {
            sIdx = 0;
        }
    }
    return( &key[0] );
}
```

对password的每个字符，函数用取模的方式只取了4个bit,如果password过短的话函数就对passwrod进行循环，超过32个字符的话则会丢弃剩下的部分。造成多个字符串可以对应相同的twofish密钥，并且还存在默认密码 `metallica` 的情况。

p.s. 这个bug已经有人报告了,但开发者没有回复 https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=715415

密钥存储问题

- 本地明文密钥存储
- 弱Hash存储
- 典型案例：
  - Windows登陆口令LM HASH

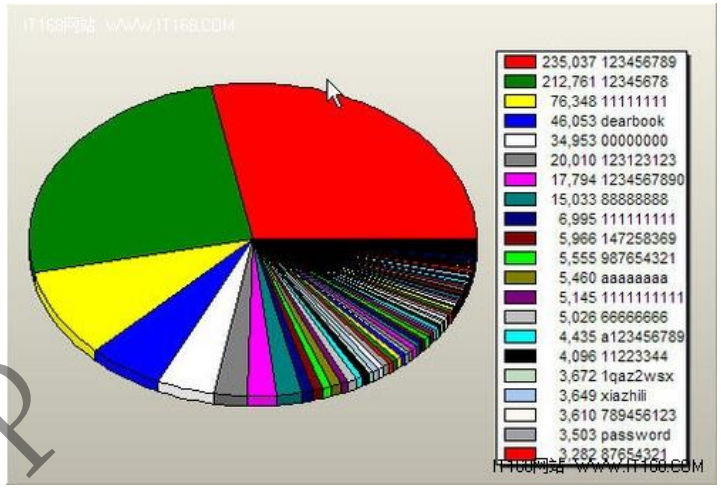
20 newest hashes cracked:

Hash	Plaintext	Cracked
c343e13c04b13884	GLOVES	2017-07-07 22:53:41
e014be2db860abea	SO!ARSY	2017-07-07 22:02:11
f589b7438db69f2e	NERULE8	2017-07-07 22:00:22
9d5a8682894f0158	SRIDHAR	2017-07-07 22:00:21
7f5876af4d21b6d7	_23	2017-07-07 22:00:21
cbb5856f2a451e95	3OOBLAC	2017-07-07 21:47:02
d9454ac676808e33	KOUT!	2017-07-07 21:47:02
73d17b616e666e6d	GUDDU12	2017-07-07 21:25:34
6080730e9dbebef1	GJ#BY1	2017-07-07 18:46:55

密钥存储问题

- 用户口令存储问题
  - Hash
  - Hash+salt
  - 安全的password hash

- 身份认证应该怎么做？
  - 公钥+私钥
  - 不可抵赖性！！



密钥清理问题

- 大部分程序都不做密钥清理，依赖于系统的Free有问题

Program	Key Buffer	Key Generation	Key Sanitization
Sumatra PDF	contiguously plaintext on heap	derived from password+IV	not sanitized
PDF XChange Viewer	contiguously plaintext on stack	derived from password+IV	not sanitized
WinRAR	contiguously plaintext	derived from password+IV	not sanitized
7-Zip	contiguously plaintext	derived from password+IV	not sanitized
CryptCat	contiguously plaintext	derived from password	not sanitized
IpMsg	contiguously plaintext	determined by one party	not sanitized

TABLE IV: Key Management of Windows Programs

密钥交换问题

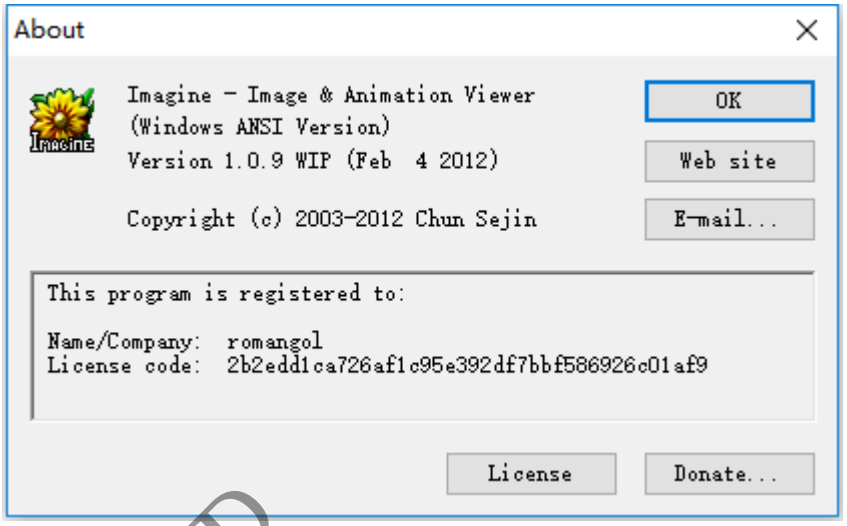
- 自定义的密钥交换协议
- 推送协议
  - 个推
- 上海移动掌上营业厅
  - http通讯
  - 3DES 固化密钥作为KEK
  - XXTEA会话密钥，可被KEK解密
  - 会话密钥使用随机数作为密钥，但是随机数生成存在问题

密钥交换问题

- RSA加密发送密钥有何不妥？
- 解开人人网登录密码的 RSA 加密 – Ring0
  - https://ring0.me/2014/05/renren-password-transfer-security/
- 招商银行RSA 512-bit 密钥登陆
  - 公钥可被分解

密钥复用问题

- DSA签名复用
  - SONY PS3攻击
  - Imagine Viewer



$$r = g^k \bmod p \bmod q$$
$$s = k^{-1}(Hash(regName) + xr) \bmod q \qquad regCode = k^{-1}(Hash(regName) + xr) \bmod q$$

审计现有软件系统中的密钥误用

现实软件中的密钥安全审查

GoSSIP

Interface-centric & Key-centric

密码代码分析

GoSSIP

密码代码分析方法学

- 基于接口的分析
  - 适用：调用系统API或者密码算法库
  - Android和iOS平台之类依赖于系统库函数的应用场景

- 基于密钥的分析
  - 适用：二进制代码，静态链接
  - Windows、Linux平台等，开发者倾向于将自己编译的密码算法库静态链接进去

密码学库的密钥管理安全统计

Crypto Component	Key Layout	Key Secrecy	Key Buffer	Key Generation	Key Singularity	Key Sanitization
Libmcrypt	raw	plaintext	OS managed	user defined	not enforced	not regulated
Libgcrypt	raw	plaintext	Lib managed	user defined	not enforced	explicit zeroisation
Nettle	raw	plaintext	OS managed	user defined	not enforced	not regulated
Tomcrypt	raw	plaintext	OS managed	user defined	not enforced	user defined
Botan	raw	plaintext	Lib managed	user defined	not enforced	secure wiping
Crypto++	raw	plaintext	Lib managed	Lib encapsulated	not enforced	secure wiping
NaCL	raw	plaintext	Lib managed	Lib encapsulated	not enforced	not regulated
libsodium	raw	plaintext	Lib managed	Lib encapsulated	not enforced	secure wiping
Cryptlib	raw	plaintext	Lib managed	Lib encapsulated	not enforced	not regulated
Keyczar	raw	support encrypted key	Lib managed	Lib encapsulated	not enforced	not regulated
Windows Crypto API	raw	plaintext	OS managed	user defined	not enforced	not regulated
Android Crypto API	raw	plaintext	OS/Language managed	user defined	not enforced	Language managed
iOS Crypto API	raw	plaintext	Language managed	user defined	not enforced	Language managed

TABLE III: Features of key usage model in common crypto libraries and services

Android和iOS上的密钥误用

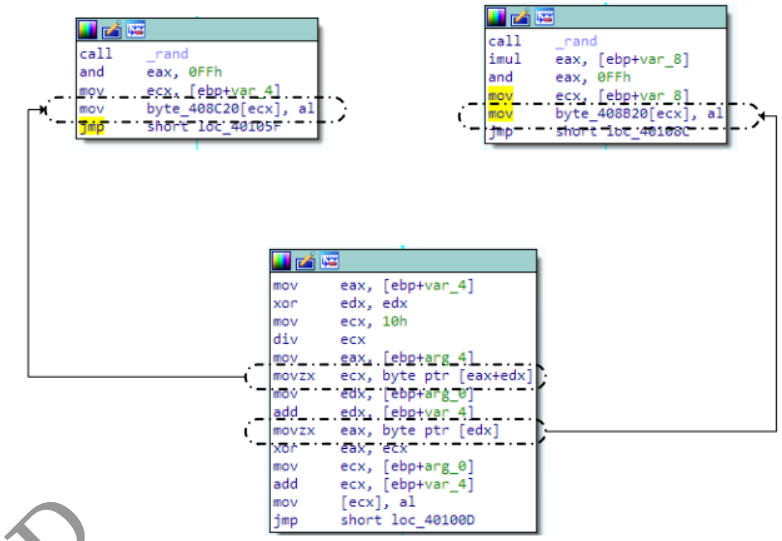
- Android
  - 100 个最广泛使用的 native libraries (.so)，来自市场上排名前10000的apps
  - 29 of them fulfill cryptographic operations.
    - 12 cases use hard-coded key
    - 11 cases use device-related information to generate key
    - only six cases use dynamically generated information
- iOS
  - 分析了 20 个银行和航空公司应用APP，其中 16 个APP包含了至少一例硬编码密钥情况，三个APP使用了固定的密钥材料来生成密钥，仅有一个APP 的密钥生成是非确定性的

二进制代码中密钥使用特征

- *典型实例*

```
1  unsigned char Key[0x10];
2  unsigned char Data[0x100];
3
4  void encrypt( unsigned char * buf, unsigned char * k )
5  {
6        for ( int i = 0; i < sizeof(Data); ++i )
7              buf[i] ^= k[i % sizeof(Key)];
8  }
9
10 int main()
11 {
12       for ( int i = 0; i < sizeof(Key); ++i )
13             Key[i] = rand() & 0xff;
14
15       for ( int i = 0; i < sizeof(Data); ++i )
16             Data[i] = i * rand() & 0xff;
17
18       encrypt( Data, Key );
19 }
```

(a) A sample crypto scheme

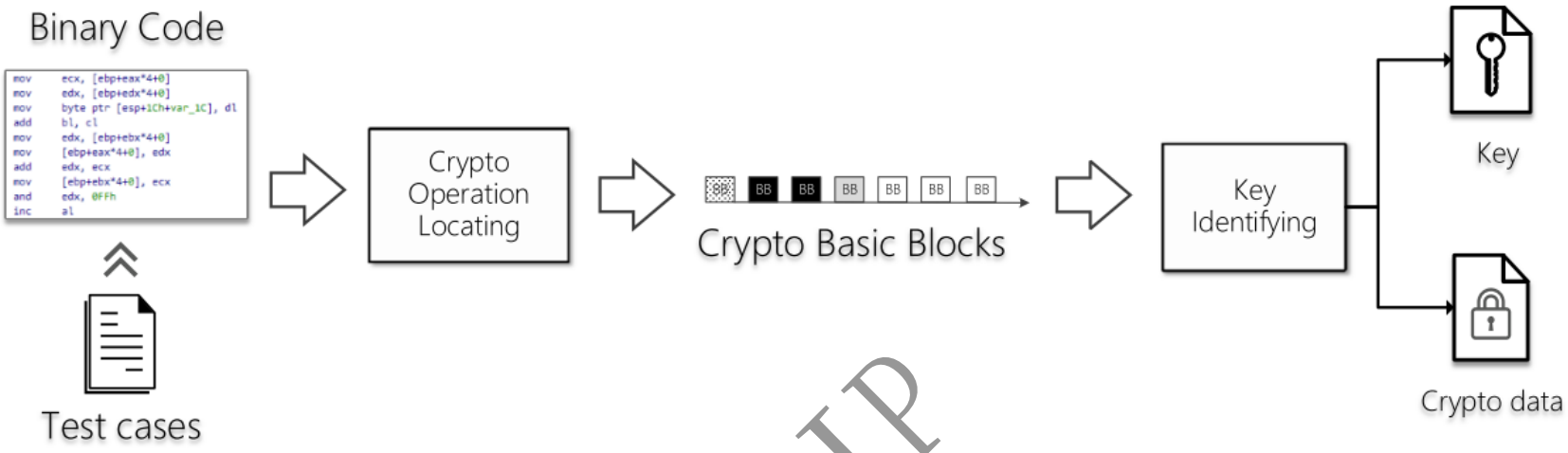


(b) Corresponding assembly code

Figure 1: An example of key usage in binary code

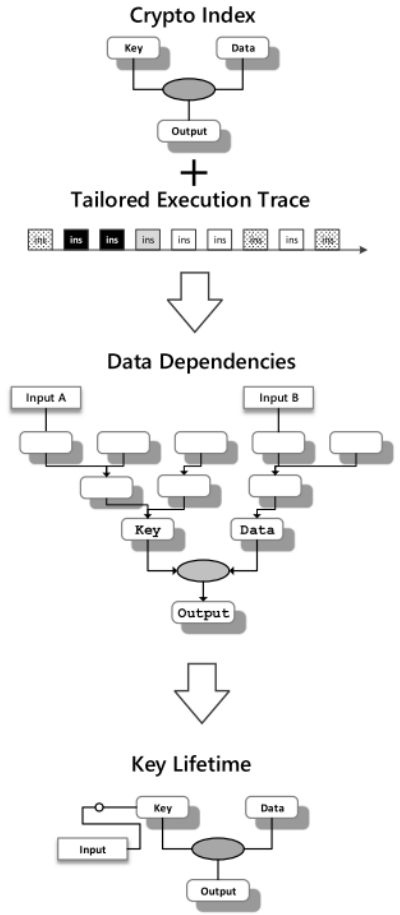
密钥和算法定位

- *工作流程*



密钥生命周期分析

- 以密钥及相关核心运算作为Index
- 记录程序执行流程
- 建立数据依赖关系
- 得到密钥生命周期



Case Study

WinRAR加密分析

谢谢！

更多内容，请访问

http://cryptomisuse.org

&

https://zhuanlan.zhihu.com/securitygossip