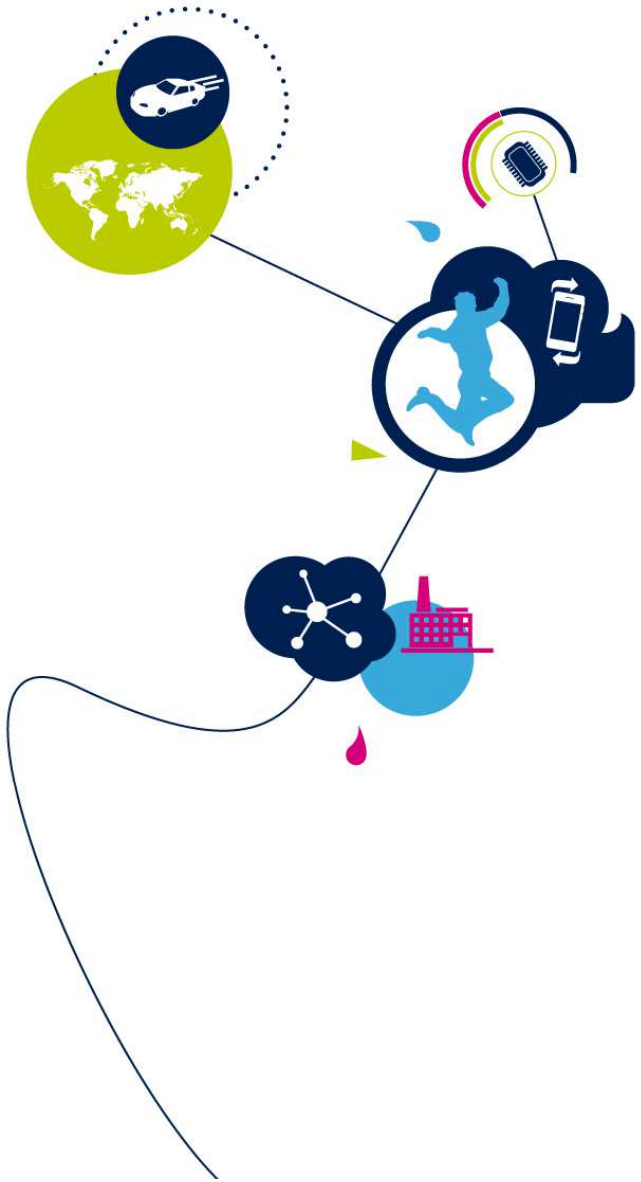
STM32加解密技术

STM32 Security

MCU China 2019



life.augmented

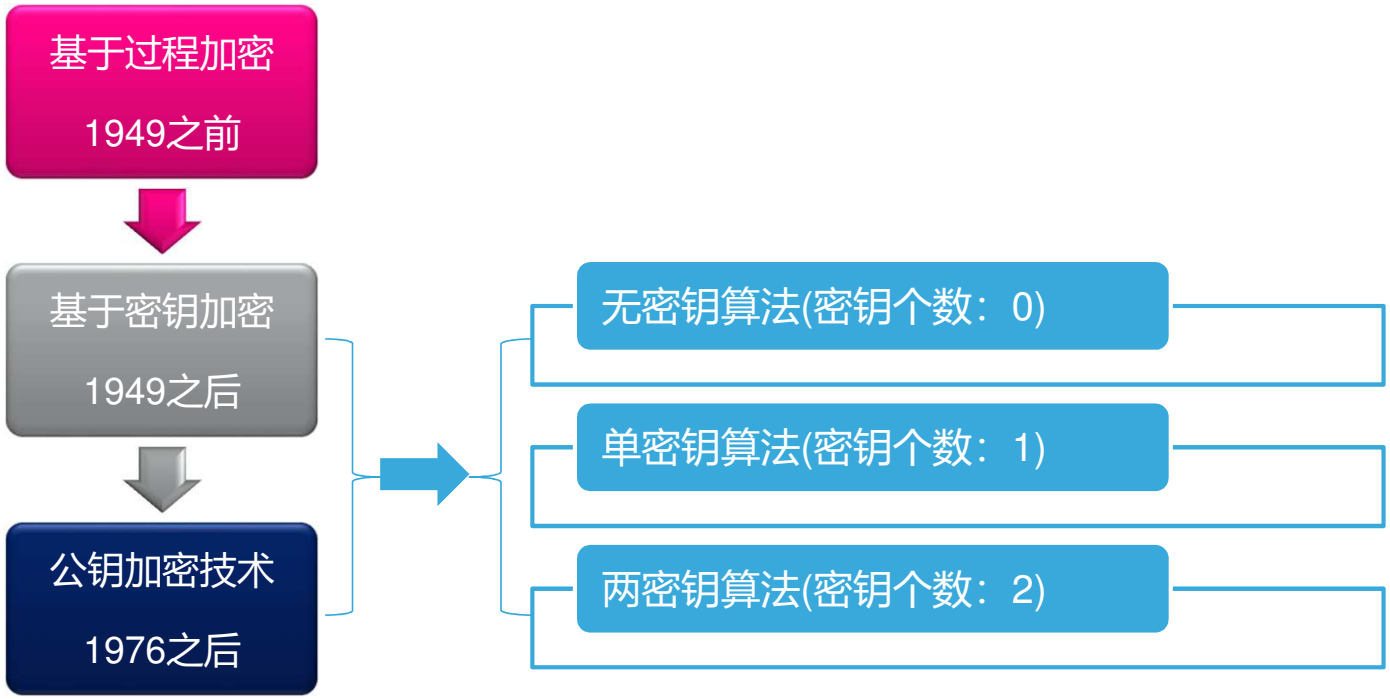
加解密技术是一个数学工具，提供构建系统的关键安全服务

life.augmented

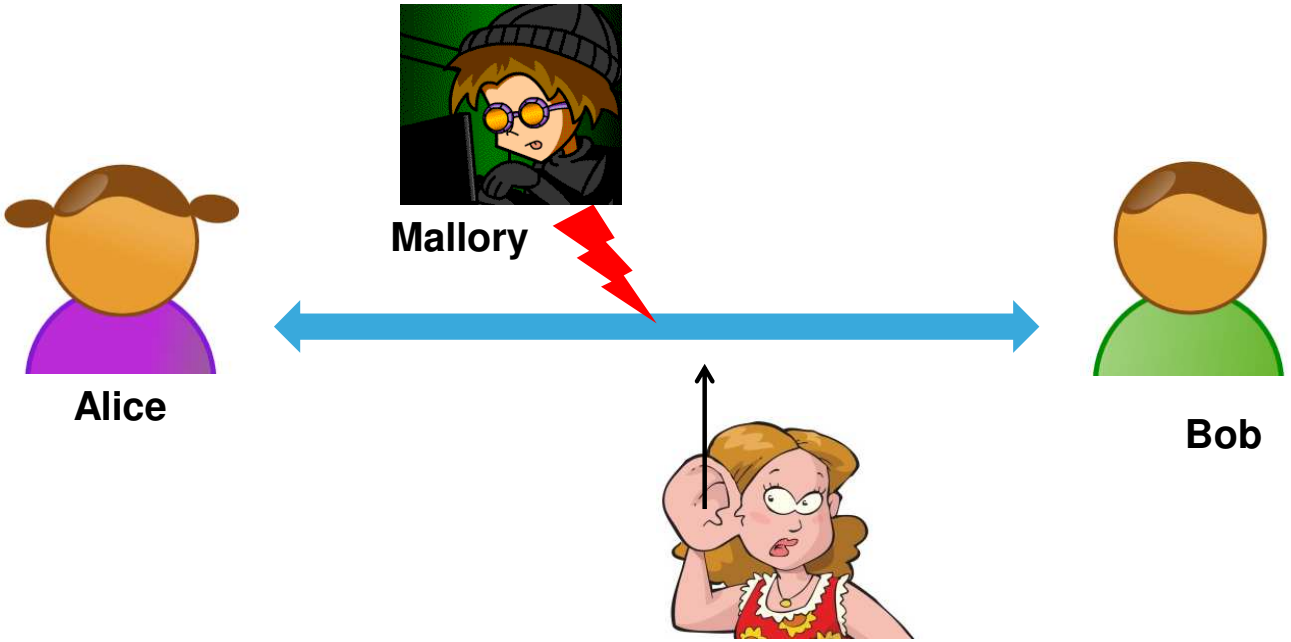
加解密技术的地位

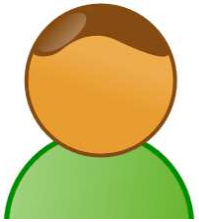
- 没有加解密技术是万万不能的
  - 通讯安全
  - 平台安全
  - *例外：STM32 RDP保护知识产权
- 加解密技术不是万能的
  - 只是工具
  - 无法代替其它STM32安全技术

加解密技术的发展

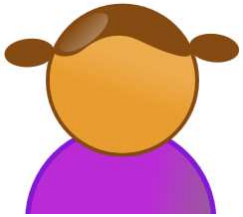


场景





和

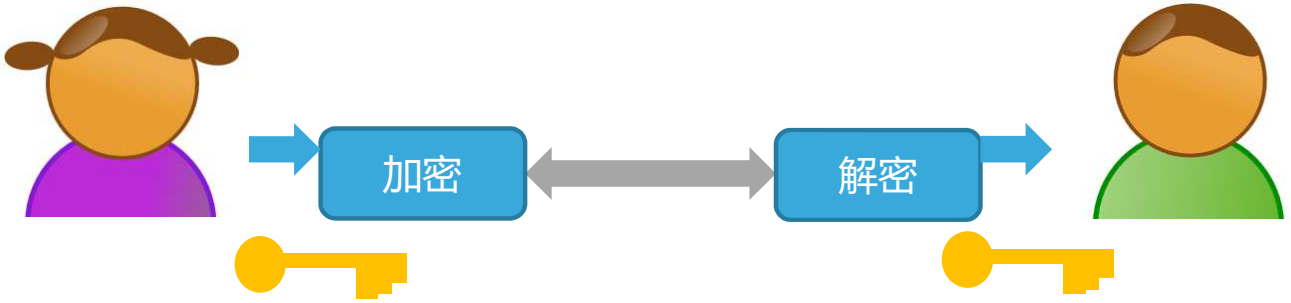


- 需要交换保密信息
- 确保消息的完整性(没有修改)
- 认证: Bob需要检查Alice确实是Alice

交换保密数据

对称密钥

- Alice 和 Bob 共享一个共同的密钥
- 使用该密钥基于对称密钥算法进行加密与解密



- 对称密钥系统算法简单，速度快
- 主要缺点：要通过安全通道共享密钥

life.augmented

高级加密标准(AES)

- **AES是今天使用最为广泛的对称密钥算法**
  - AES在2001年从15个算法中历时4年的竞争中脱颖而出,被US NIST选择取代DES
  - 输入块大小: 128 bit
  - 输出块大小: 128 bit
  - 密钥长度
    - 128
    - 192
    - 256
- 支持操作模式ECB, CBC, CFB, OFB, CTR, **GCM** 等.

非对称密钥原理

- Alice和 Bob各自拥有各自的一对密钥(公钥-私钥)
- 公钥可以发布，无须保密
- 私钥从不泄露

- Alice 使用Bob公钥加密消息，发送给Bob。其它人无法解密，甚至Alice自己
- Bob拥有私钥可以解密

- 但是，非对称密钥复杂，而且速度很慢

life.augmented

典型非对称密钥算法

- 基于数学难题
  - 一个方向运算简单
  - 相反方向的操作太难
- 相乘与分解 → **RSA**
  - 两个质数相乘：例如61和349 → 61*349 = 21289.
  - 试想如果知道21289，如何了解是哪两个数相乘而得？
- 指数与对数→ **椭圆曲线** (**ECC**)
  - 设想2个整数：3和6， 容易计算$3^6=729$.
  - 非常困难来求$\log_3(729)=6$

ECC与RSA比较

- ECC的主要优点: 对于同一个等级的安全
  - ECC → 密钥长度更小
  - ECC → 签名更快
  - 但是RSA验证签名更快

NIST guidelines for public key sizes for AES

ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	KEY SIZE RATIO	AES KEY SIZE (Bits)
163	1024	1 : 6	
256	3072	1 : 12	128
384	7680	1 : 20	192
512	15 360	1 : 30	256

Supplied by NIST to ANSI X9F1

AES Vs. RSA的性能对比

- 80MHZ STM32
  - AES 128
    - ~32K 时钟周期
  - RSA 2048
    - ~ 4.2M 时钟周期(e=65537 加密)
    - ~57M 时钟周期(解密)
- 提示： 性能数据可以查阅STM32 X-Cryptolib开发包中的用户文档

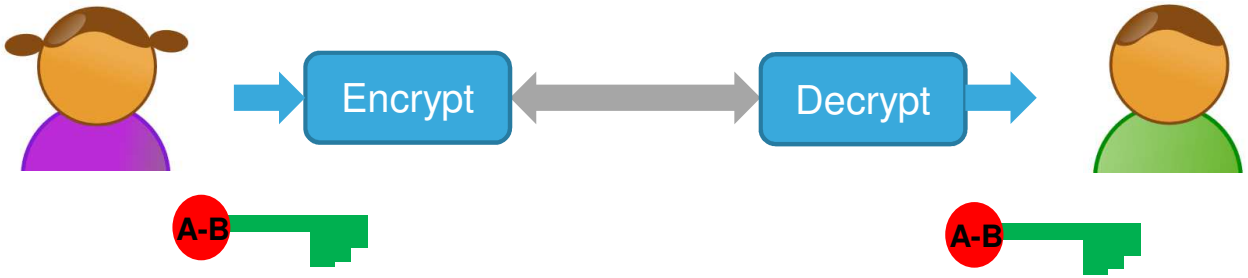
AES vs.RSA的Flash大小

- STM32
  - AES 128
    - ~5k 字节
  - RSA 2048
    - ~ 20K 字节
- 提示：Flash大小在性能优化与空间优化的情况下会有很大不同
- 提示：Flash大小数据可以查阅STM32 X-Cryptolib开发包中的用户文档

对称密钥+非对称密钥

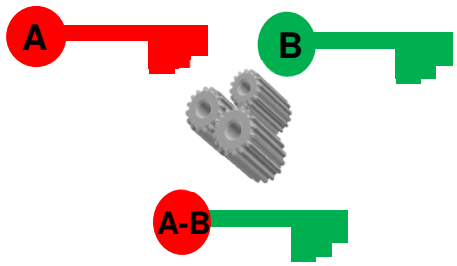
- 为了解决对称密钥的密钥分发的问题，使用非对称密钥计算出一个共享密钥（步骤一）

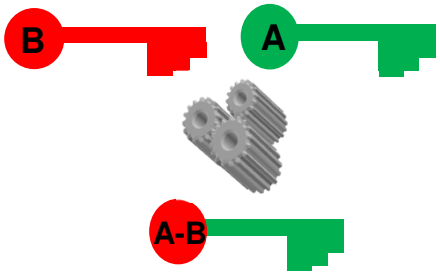
- 然后，使用共享密钥和对称密钥算法进行加密，效率高（步骤二）



Diffie-Hellmann算法

- 公钥技术允许双方在不安全通道上进行密钥协商





Diffie-Hellmann算法数学原理

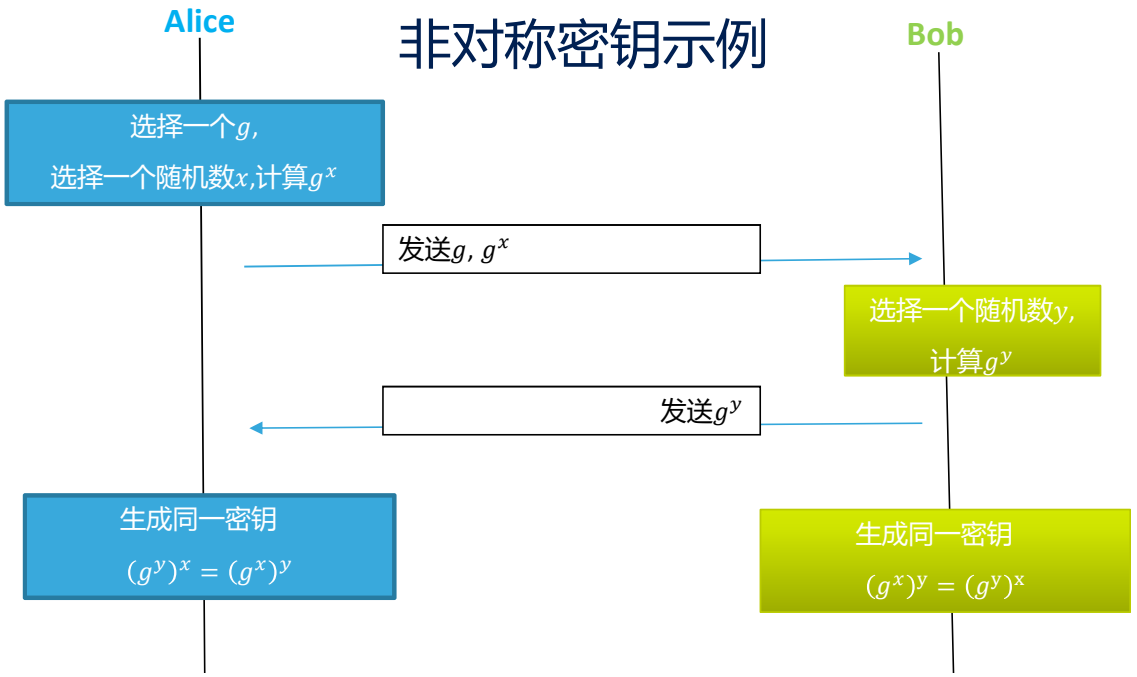
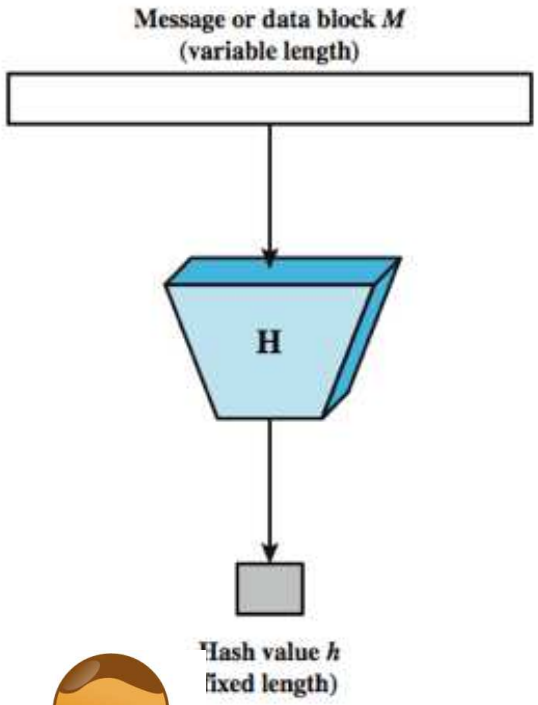


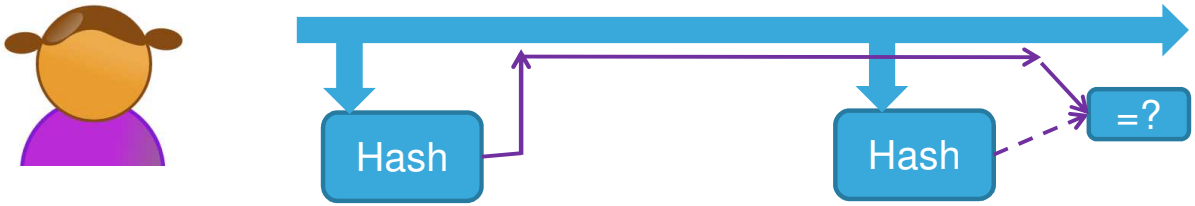
图9 DH密钥协商原理

数据完整性检查

Hash 函数

- 使用Hash函数计算消息的摘要,将其与消息一起发送
- Hash特征
  - 对消息的任何改变，摘要必然变化
  - 单向函数：无法从摘要构造出原文
- 当Bob收到消息，使用同样算法计算摘要，与收到的摘要进行**比较**





常见Hash算法

- MD5
  - 输出128 位
  - 被认为已攻破，不应使用
- SHA-1
  - 输出160位
  - 被认为在今天已经不够安全，不应使用
- SHA-256 & SHA-512 (SHA-2)
  - 相应输出为 256 & 512 位
  - SHA-224 & SHA-384 是SHA-256 & SHA-512派生而来
  - 业界认为安全
- SHA-3 (Keccak)
  - 新的 US NIST 标准 (2015)
  - 在2012年从6年竞争的 64个候选算法中被US NIST选中
  - SHA-256 & SHA-512
  - 易于硬件实现，灵活可扩展

消息认证

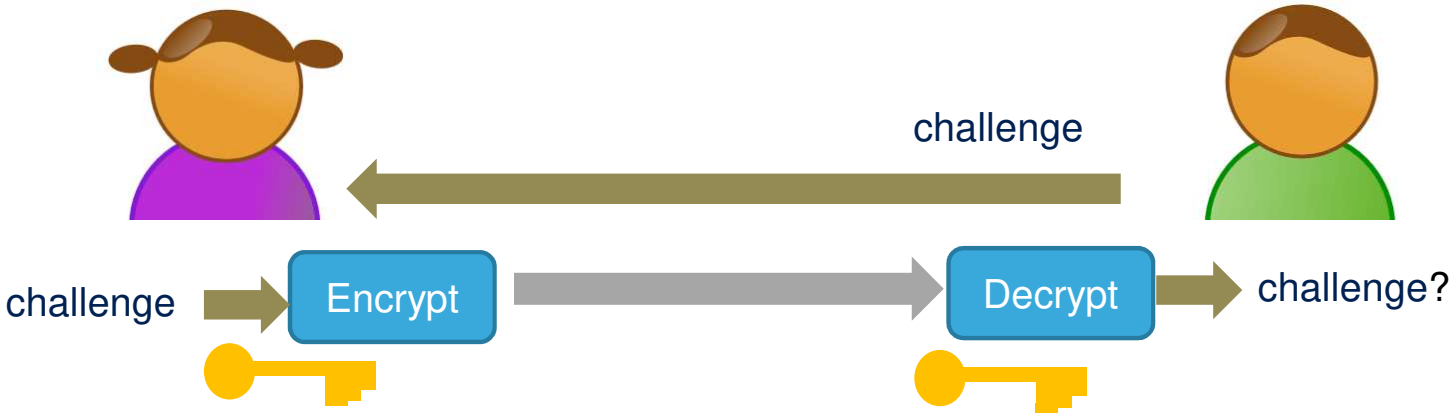
- 从安全(security)角度来说仅仅Hash没法保证完整性
- 通讯链路上的攻击者可以:
  - 改变这个消息
  - 重新计算摘要
  - 用改变后的消息 +摘要替换原来的报文
- 方案: 使用认证码(请看下一节)

认证

身份认证
消息认证

使用对称密钥

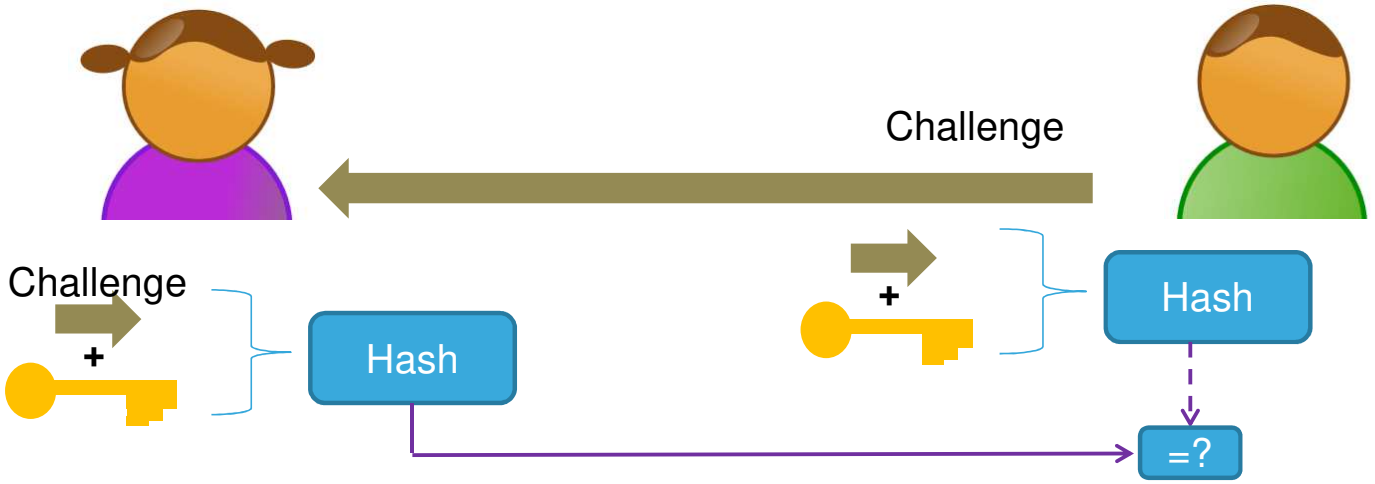
- Bob 和 Alice共享一个密钥



- Alice确实如假包换，因为质询使用只有Alice和Bob才知道的密钥加密了challenge
- 为了避免重放攻击，challenge不应重复

加上HASH

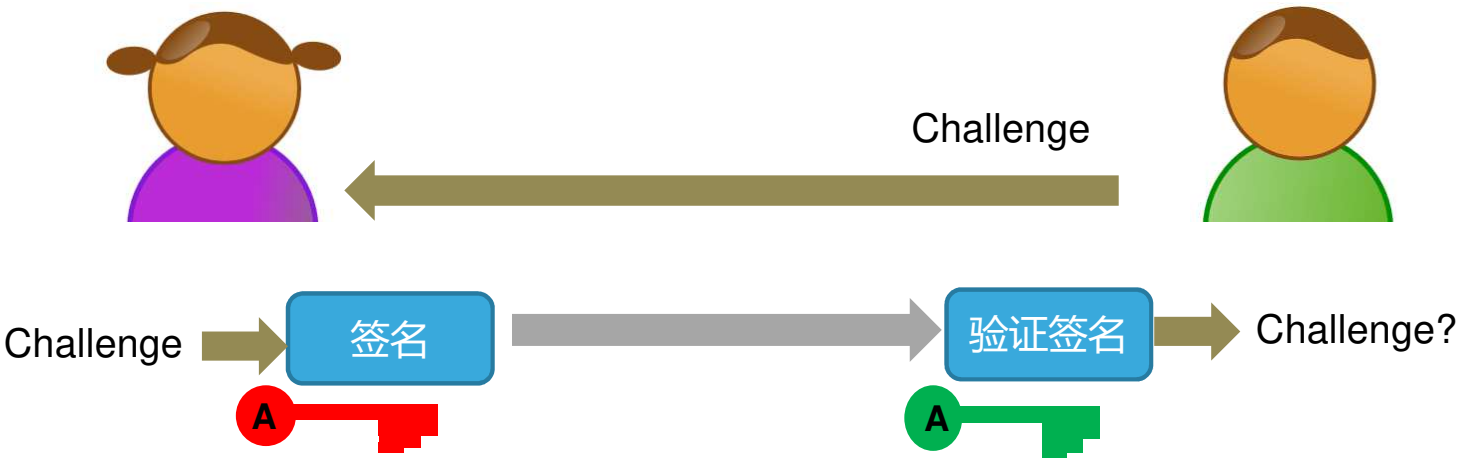
- Bob 和 Alice共享一个密钥



- Alice确实如假包换，因为使用了只有Alice和Bob才知道的密钥对challenge 计算HMAC

使用非对称密钥

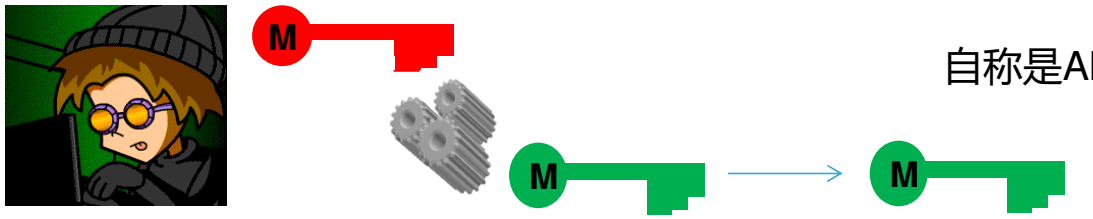
- Bob 拿到 Alice的公钥

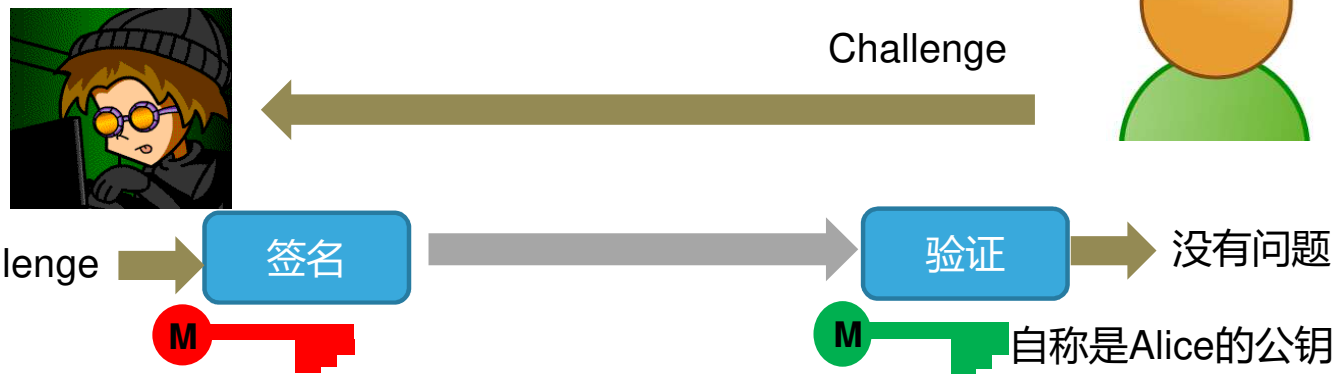


- 风险: 如何确定公钥来自Alice?
- 如果Mallory发布一个公钥自称是Alice呢?

中间人攻击

- Mallory 以Alice的名义发布她的公钥





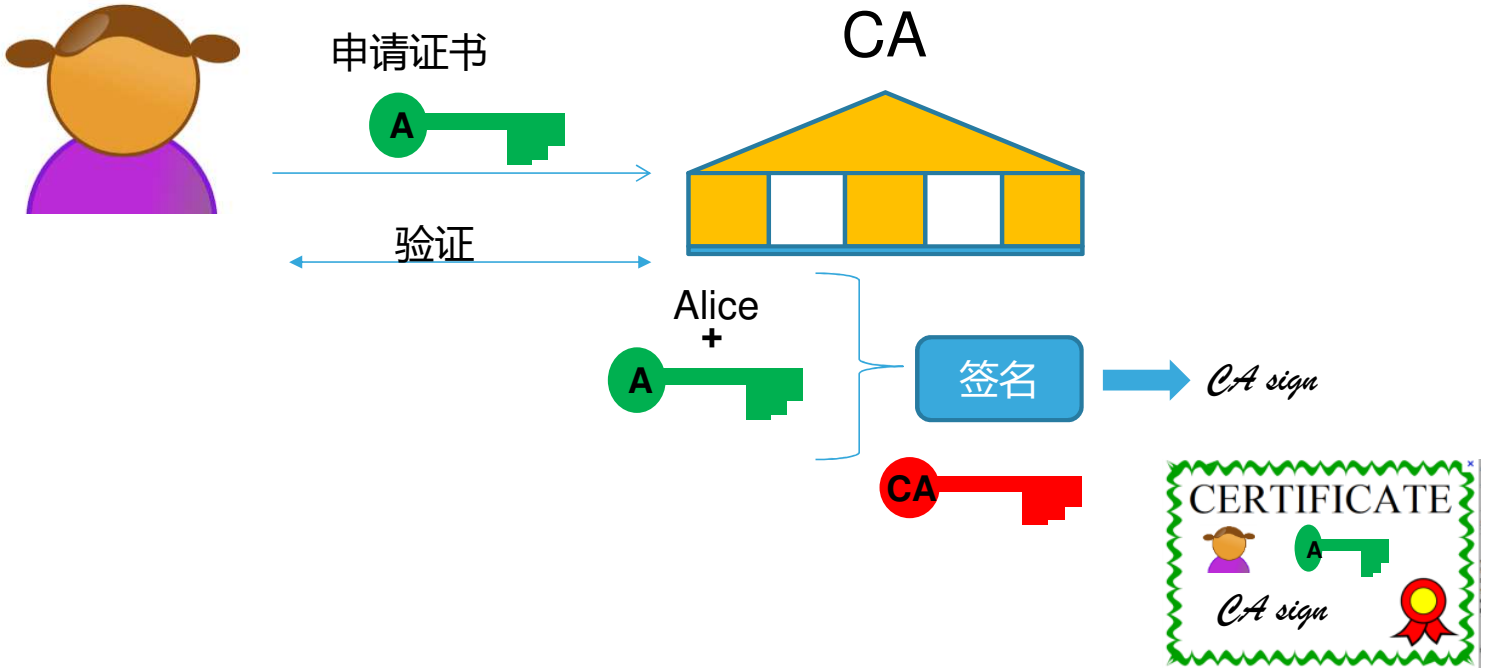
life.augmented

数字证书

- 授权中心Certification Authority (CA)是一个发布证书的机构
- 数字证书确认了一个对某个名字之下的公钥的所有权
- 授权中心CA 是一个可信第三方

Alice's 数字证书

- Alice从CA申请证书
- CA确认Alice确实是Alice, 生成和发布Alice的证书



数字证书

- 证明公钥的所有权
  - 数字证书包括
    - **主题Subject:** 需要鉴别的个人，实体或者对象
    - **公钥public key**
    - **数字签名digital signature:** CA对证书的签名
    - **发布者Issuer**: 验证了信息并发布证书的实体
    - **签名算法Signature Algorithm**
  - 额外数据
    - **序列号serial number**: 唯一确定该证书
    - **证书有效起点Valid-From**
    - **证书有效终点Valid-To**
    - **公钥用法Key-Usage**: 公钥的用途(e.g. 算法, 签名, 证书签名...).
    - **证书的hash以及所使用hash算法**

X.509 证书扩展名

- 常用格式
  - .pem
    - Base64编码
  - .cer, .crt, .der
    - 原始的DER二进制
- 其他格式
  - .p7b, .p7c
  - p12 – PKCS#12
  - .pfx – PFX

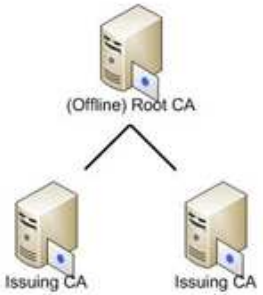
https://en.wikipedia.org/wiki/X.509

-----BEGIN CERTIFICATE-----

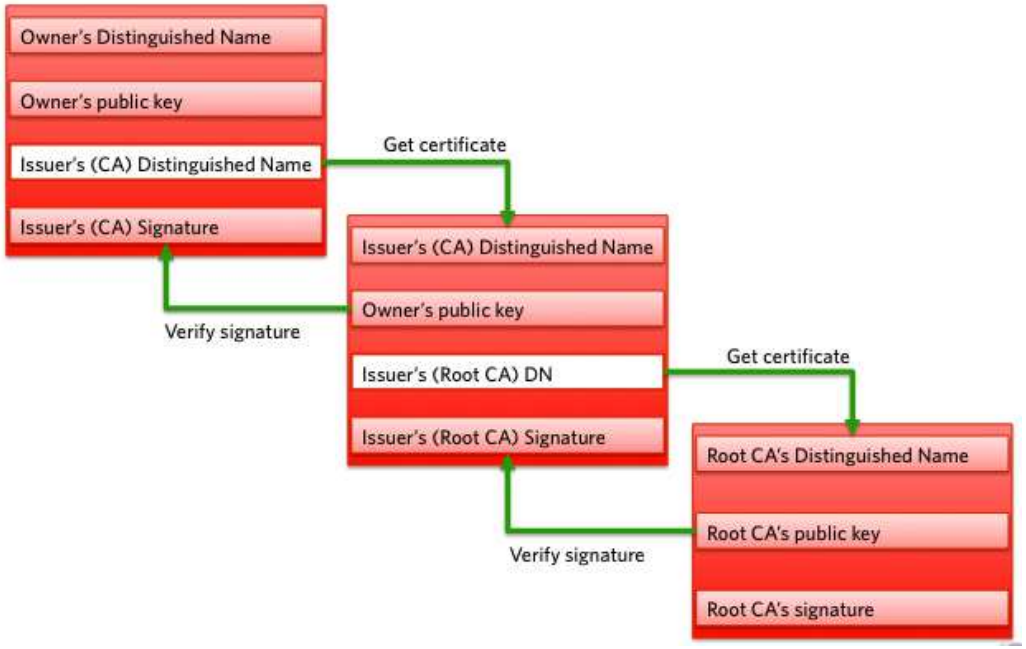
-----END CERTIFICATE-----

PEM格式

证书链

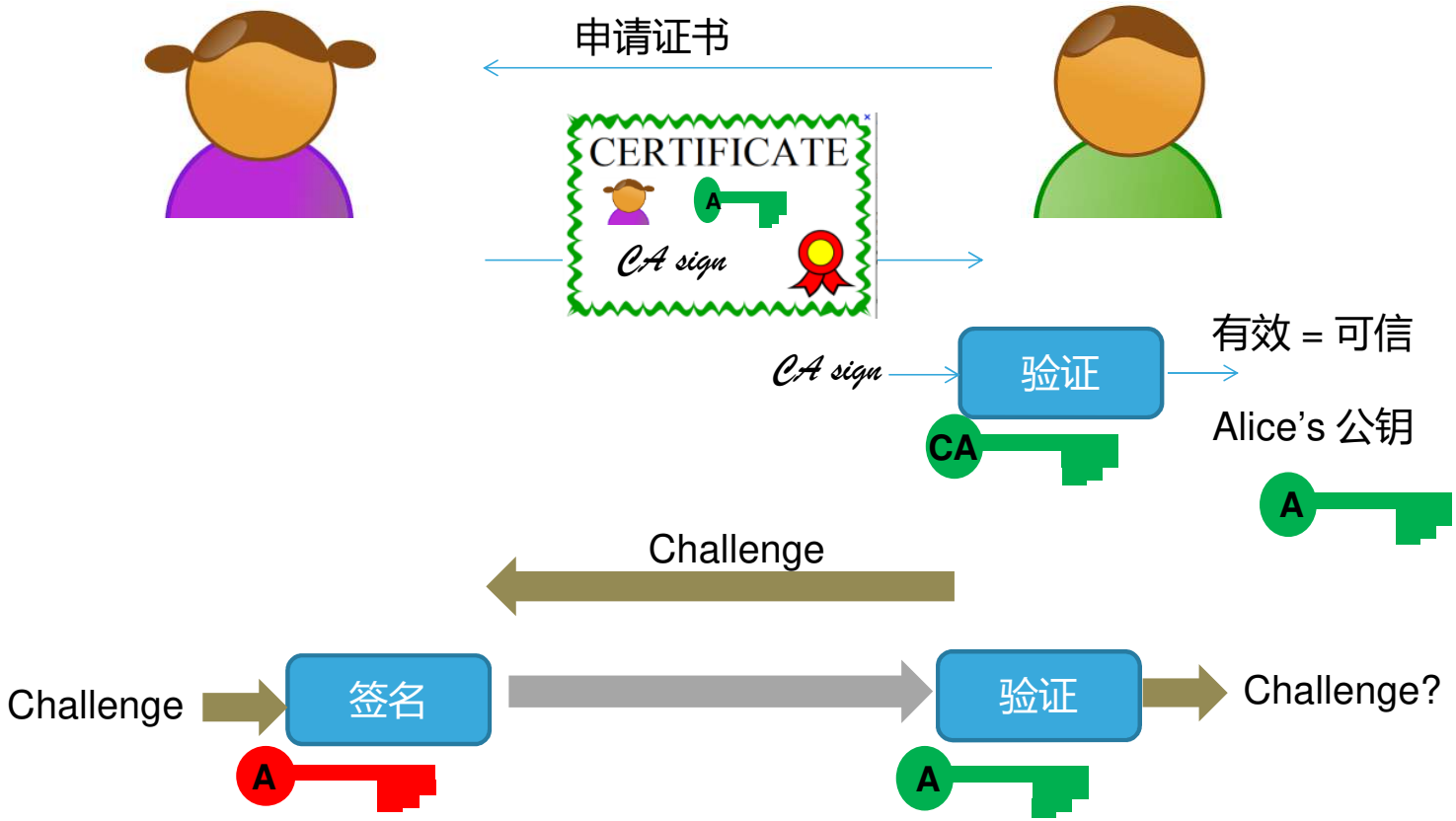


- CA是层级结构



life.augmented

使用数字证书进行认证



每天都在用,...

- https = http + TLS/SSL
- TLS (传输层安全) / SSL （安全套接字)会话
  - 确保保密 confidentiality, 完整integrity & 认证authentication

life.augmented

随机数

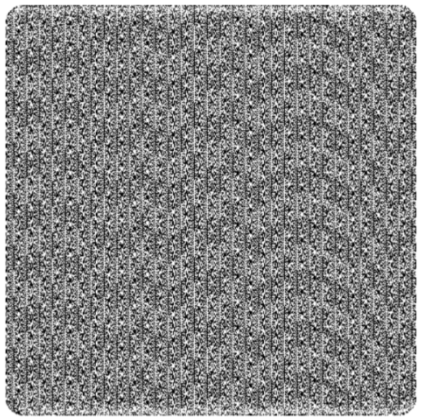
如何生成challenge,种子

随机数发生器(RNG)

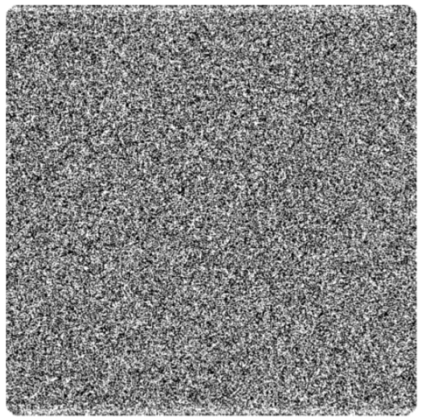
- 随机数典型用在：
  - 产生一个challenge，用在认证，密钥的种子，数字签名

- 一个熵源来进行进一步的密码学运算
- RNG的统计属性可以用来评估随机性
- 非随机或者可预测的输出则被视为一个弱点

PHP rand() on Windows



http://www.random.org/bitmaps



http://boallen.com/random-numbers.html

http://en.wikipedia.org/wiki/Random_number_generator_attack

life.augmented

随机数发生器(RNG)

- 伪随机数使用软件算法来产生随机数
- 真随机数发生器 TRNG则利用芯片的不可预测的物理属性来产生随机数
- 使用符合NIST FIPS 140-2 Annex C规范的随机数

NOTE

- 总是使用TRNG

life.augmented

总结

- 加解密技术
  - 理解基本的加解密原语以及常见的算法
    - 无密钥函数---单向函数以及随机源
    - 1密钥函数---对称加密---加解密密钥相同
    - 2密钥函数---非对称加密---加解密密钥不同
  - 理解不同加解密原语的性能区分
  - 理解哈希函数CRC与SHA的本质区别
  - 理解加解密原语的组合可以实现的扩展安全服务
    - 认证
      - 对称加密+单向函数---MAC/AEAD
      - 非对称加密+单向函数---数字签名
    - 伪随机函数
      - 对称加密+随机源
    - 密钥分发
      - 对称加密+非对称加密

- 加解密技术
  - 理解认证的形式
    - 数字证书
  - 理解证书链与CA