密码Hash函数

清华大学计算机系
于红波
2023-04-12

提纲

- Hash函数基础
- Hash函数算法
- Hash函数安全性分析
- Hash函数碰撞攻击引发的安全问题

第一部分

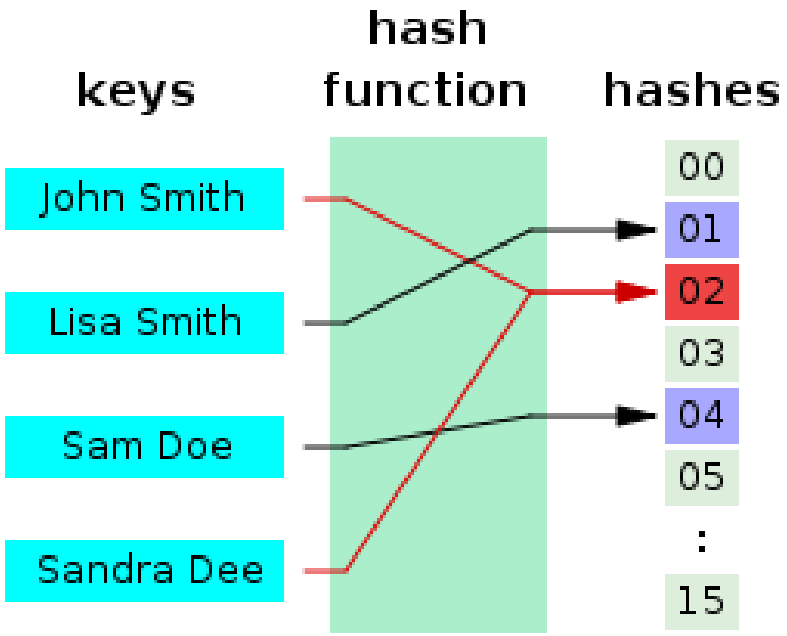
Hash函数基础

Hash函数和密码Hash函数

- Hash函数又称杂凑函数、散列函数、数字指纹等
- 早期的Hash函数不用于密码学， 1953年IBM
- 1979年，Merkle给出了实质性定义，包括抗原像和第二原像性
- 1980年，Davies和Price将Hash函数用于电子签名，用于抵抗RSA等数字签名中的存在性伪造攻击，标志着密码Hash函数的开始
- 1987年， Damgård首次给出抗碰撞的Hash函数的正式定义，指出了抗碰撞性和抗第二原像性

Hash函数定义

- Hash函数： Y=H(M)，可以将任意长的消息压缩为一个固定长度的摘要 $H(M): M \in \{0,1\}^* \rightarrow \{0,1\}^l$
- 压缩函数: 将固定长度的消息M压缩成一个固定长度的输出 $f(M): \{0,1\}^{m+l} \rightarrow \{0,1\}^l$



Hash函数属性

- 有效性(Efficiency)：已知消息M，计算消息指纹Y=H(M)是容易的
- 抗原像攻击(preimage resistance, one-way)：给定任意消息指纹Y=H(M)，恢复消息M是计算不可行的。理想的复杂度是搜索攻击2^n次计算
- 抗第二原像攻击(Second-preimage resistance, Weak collision resistance)：给定任意消息M_1，找到另一个消息M_2具有相同电子指纹 $H(M_1)=H(M_2)$是计算不可行的。理想的复杂度是2^n
- 无碰撞性(Collision-resistance, Strong Collision resistance)：找到不同的消息(M_1, M_2)有相同的指纹（杂凑值）$H(M_1)=H(M_2)$是计算不可行的。生日攻击复杂度$2^{n/2}$

MD迭代结构

- 大多数Hash函数的设计采用迭代结构，每次处理一个固定程度的消息分组。基于压缩函数的MD设计准则。
- 压缩函数：一个压缩函数f是一个映射

$$f:\{0,1\}^m \times \{0,1\}^n \to \{0,1\}^m$$

其中 $n > m \geq 1$ ，f 计算有效

- Merkle- Damgård Construction
  - Mekle’s meta-method for hashing
  - Damgård的级联方法

Mekle's meta-迭代算法

- 输入：无碰撞的压缩函数 f
- 输出：无碰撞的Hash函数h

1. 假设f 把 $(n+r)$-比特的输入映成 n比特的输出(例如，n=128, r=512)。由 f 构造一个具有 n比特杂凑值的Hash函数如下。
2. 把长度为 b的输入x 分解成 t个长度为 r比特的分组$x_1x_2\ldots x_t$，假如 b不是r 的倍数，则在x_t 的后面添充上若干个 0，使其成为一个完整的消息分组。
3. 添加最后的一个分组 x_{t+1}，它是 b的长度 的二进制表示(假设 $b<2^r$)
4. 设0^j 代表j 比特的零字符串。定义 n比特的散列值 计算如下:

$$H_0 = 0^n;$$

$$H_i = f(H_{i-1} \| x_i), \quad 1 \le i \le t+1.$$

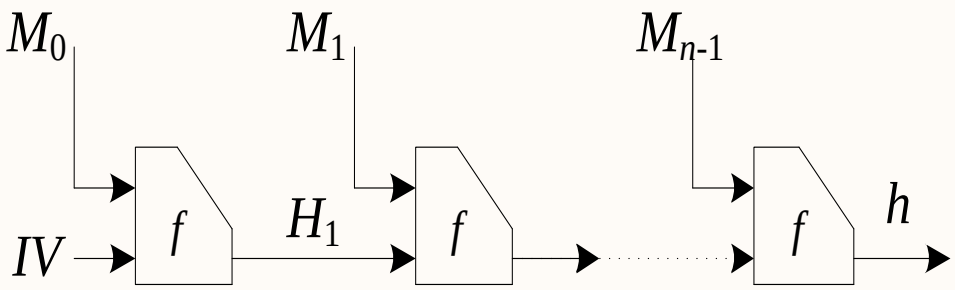
MD迭代结构

$M=(M_0, M_1, ..., M_{n-1})$为填充后的消息，摘要$h$计算如下：

$$H_0=IV$$

$$H_i=f(H_{i-1}, M_{i-1}),\ 0<i<n+1$$

$$h = H_n$$

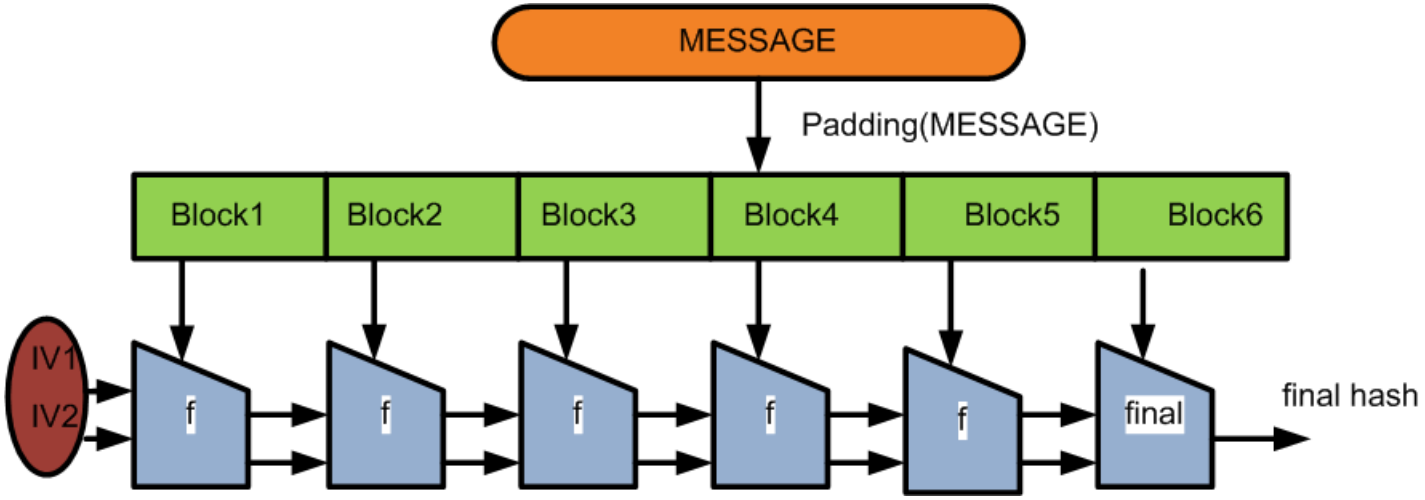


Hash函数的延展定理

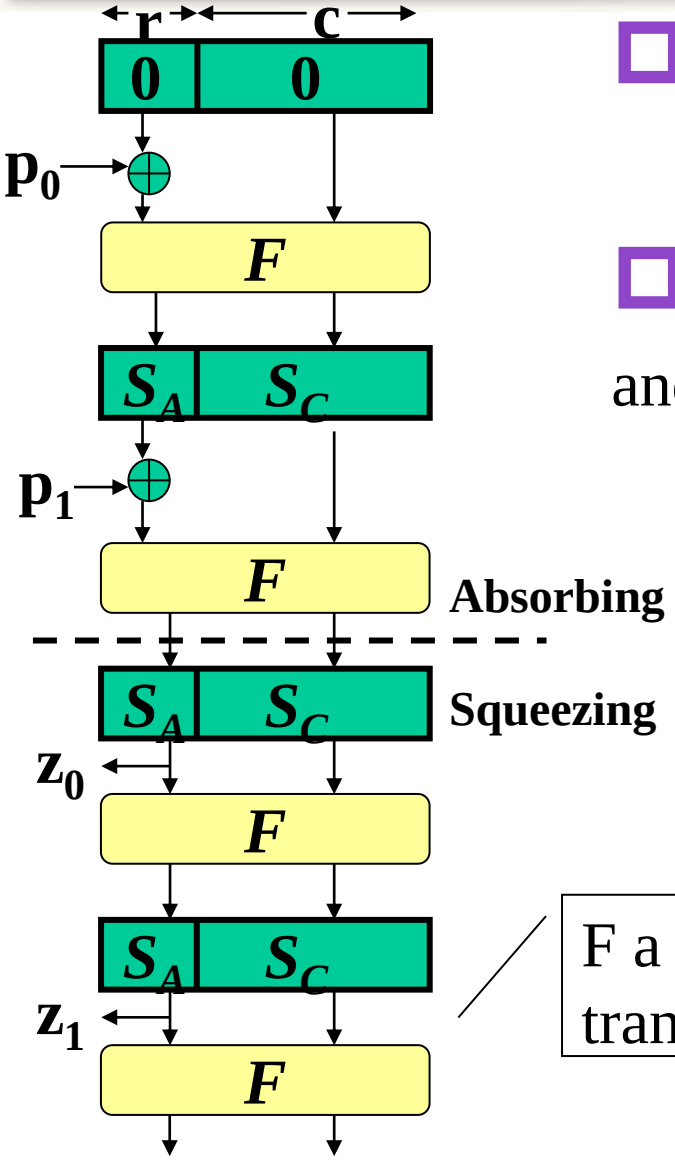
- 假设 $f:\{0,1\}^{n+r}\to\{0,1\}^{n}$ 是一个无碰撞压缩函数，则由MD迭代结构构造的Hash函数h是无碰撞的。
- 证明采用反证法。假设我们能够找到 $x\neq x'$使得$h(x)=h(x')$。则可以利用这一对碰撞消息在多项式时间内找到f的碰撞，这与f无碰撞矛盾。

Hash函数迭代结构

- Wide-pipe结构



Sponge 结构



- **Sponge Function:** a new iterative hash function (or stream cipher) framework.
- **Introduced** by Bertoni, Daemen, Peeters and Assche in ECRYPT 2007.

The last block absorbed shall not be zero.

F a unique fixed length round transformation (or a permutation).

Hash函数应用

- 完整性鉴别
- 数字签名
- 口令认证
- 消息认证码（MAC）
- 身份认证
- 数据库保护
- 零知识证明

通用Hash函数

- MD5
  - 1992年由Rivest设计
  - 输出长度128比特
- SHA-1
  - 1995年由NIST提出
  - 输出长度160比特
- SHA-2
  - 2002年由NIST提出
  - 输出长度256，384，512比特
- Whirlpool
  - 2000年由Rijmen等设计
  - 输出长度512比特
- SHA-3标准Keccak
  - 2007年由Daemen设计
  - 输出长度256,384,512比特
- 我国Hash函数标准SM3
  - 2010年发布，由王小云等设计
  - 输出长度256比特

实例一：Hash在登陆认证中的应用

- 用户提供用户名和密码，服务器在数据库中查找用户名，获取salt值，计算Hash(salt+password)与数据库中比对，相同则通过认证

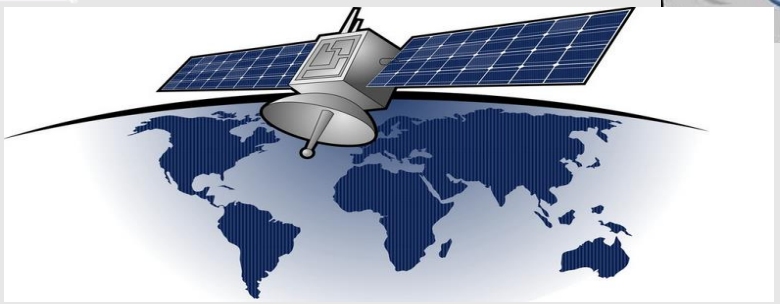
避免黑客/管理员通过据库获取用户密码

user account	salt	Hash(salt + password)
john@hotmail.com	2dc7fcc...	1a74404cb136dd60041dbf694e5c2ec0e7d15b42
betty@gmail.com	afadb2f...	e33ab75f29a9cf3f70d3fd14a7f47cd752e9c550
...	...	...

实例二：Hash在密钥衍生中应用

- Hash函数可以作为密钥衍生函数

同步计算下一结果



- New pwd = Hash(pwd)
- 广泛应用于RFID、卫星通讯等密码系统中

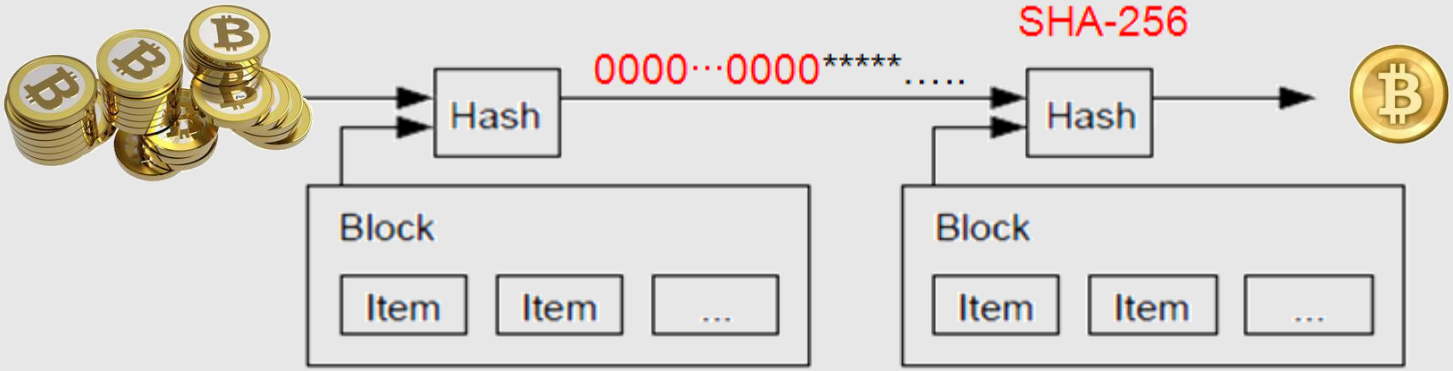
实例三：数字（电子）签名

- 电子签名

- 应用实例：金融安全

实例四：电子货币

- 比特币：通过搜索满足特定输出条件的SHA-256的原像分组生成比特币和绑定交易



  - 创建SHA-256的原像分组要求输出值的前60比特为0，获得原像的复杂度为2^{60}次运算
  - **CPU, GPU, FPGA, ASIC和cloud mining等方式计算**
- 电子货币：一种代替货币的电子签名，通过用户的公钥（数字证书）可验证货币的合法性

第二部分

Hash函数算法

Hash函数ISO标准

- **ISO/IEC 10118-3:2018**

SHA-1

RIPEMD-128/160

SHA-2(SHA-256,SHA-384,SHA-512,SHA-224,
SHA-512/224, SHA-512/256)

WHIRLPOOL

STREEBOG（STREEBOG-512,STREEBOG-256）

SHA3 （SHA3-224, SHA3-256, SHA3-384, SHA3-512 ）

SM3

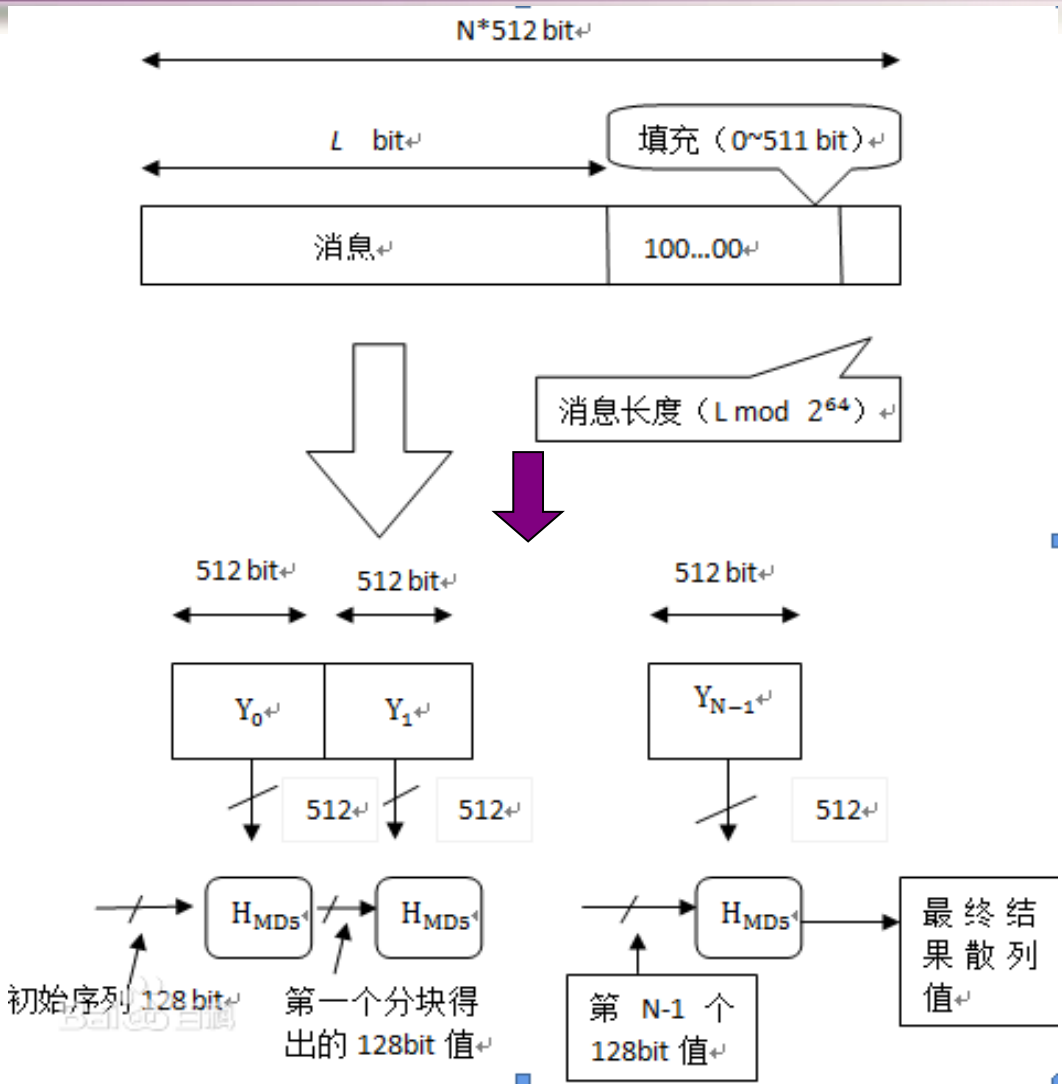
MD5算法

- MD5消息摘要算法(RFC 1321)是由MIT的Ron Rivest 提出的。
- MD5的输入是任意长度的消息，对输入按照512位的分组为单位进行处理，算法的输出是128位的消息摘要。

MD5 算法的设计目标

- 安全性
  - 找到两个摘要相同的消息在计算上是不可行的。
- 速度
  - 算法应该有利于快速的软件实现。特别是，算法的快速实现是针对32位机的，因此算法基于的是字长为32位的基本操作。
- 简单和简洁性
  - 算法应易于描述且易于编程，不需要使用大程序或者置换表。
- 有利于低位处理器

MD5 算法过程


N*512 bit
L bit
填充（0~511 bit）
消息
100...00
消息长度（L mod 2^64）
512 bit
512 bit
512 bit
Y0
Y1
YN−1
512
512
512
HMD5
HMD5
HMD5
初始序列 128 bit
第一个分块得出的 128bit 值
第 N-1 个 128bit 值
最终结果散列值

MD5 算法过程

- Step1：增加填充位。
  - 填充消息使之与448模512同余。
    即填充后的消息比512的整数倍少64位。
  - 即使消息本身满足上述长度要求，依然需要填充。
    所以，填充位数在1-512位之间。
- Step2：填充长度。
  - 用64位表示填充前的报文长度，附加在填充的结果后面。
    - 所得消息的总长度是512的整数倍。
  - 填充后的消息，每个512位分组用Y0 ,Y1 ,Y2 , … Yl-1表示。
    - 消息的总长度可以表示为L*512位。
  - 如果用32位表示，消息的总长度N*32位，其中N=L*16。

MD5填充

- 例如：假设消息为“abcde”，它们的比特串表示为：

  01100001 01100010 01100011 01100100 01100101

  其16进制表示为：61 62 63 64 65
- 其长度为42比特（其16进制表视为28），在Little-Endian结构的处理器上存储时，它的字节序列表示为（从低字节到高字节）：28000000 00000000。
- 填充完消息为：

61 62 63 64 65 80 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 28 00 00 00 00 00 00 00

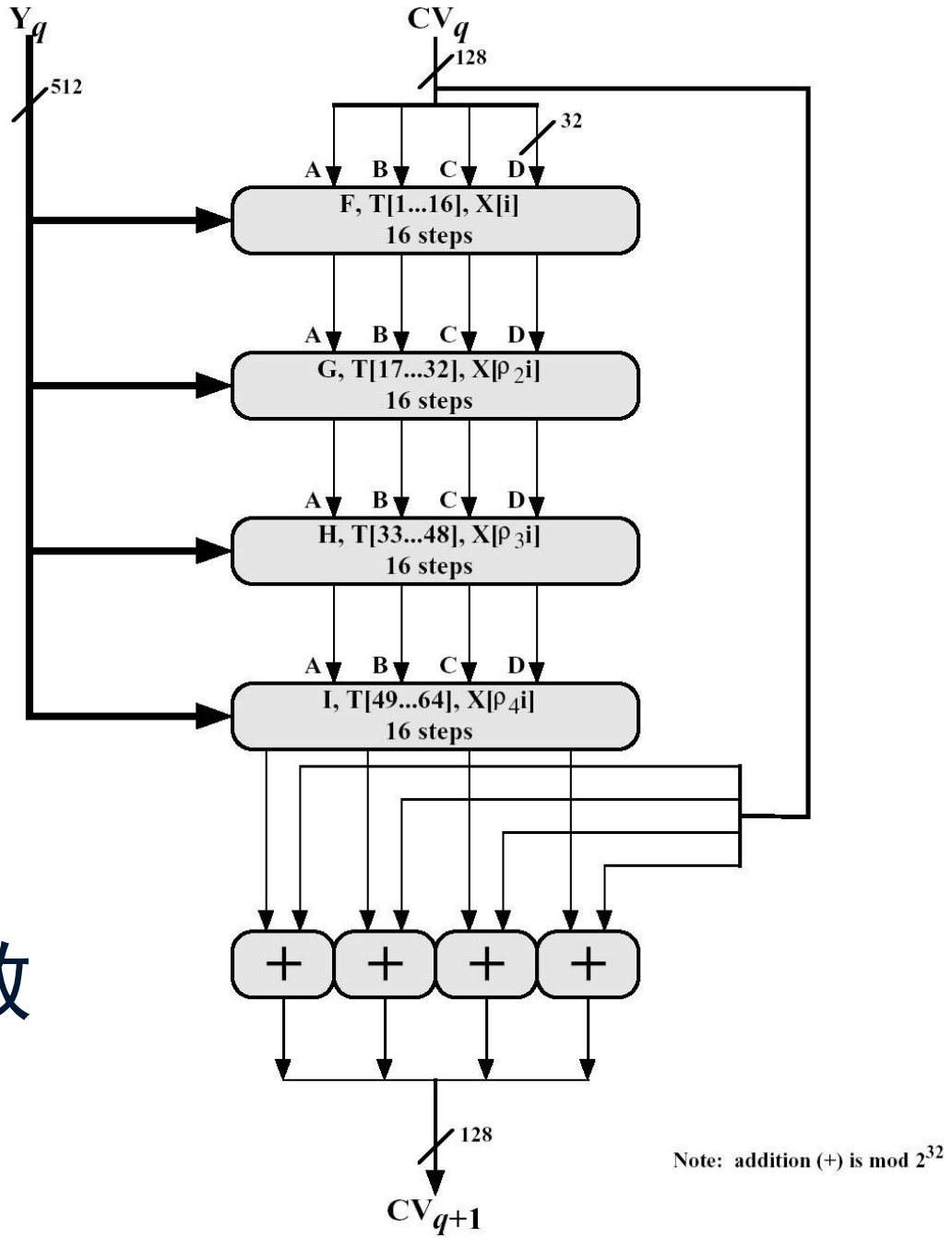
MD5 算法过程

- Step3：初始化MD 缓存。
  - Hash函数的中间结果和最终结果都保存于128位的缓冲区中，缓冲区用4个32位的寄存器(A,B,C,D)表示，并将这些寄存器初始化为下列32位的整数。
  - 初始值以低端格式存储。
    - A= 01 23 45 67
    - B= 89 AB CD EF
    - C=FE DC BA 98
    - D=76 54 32 10

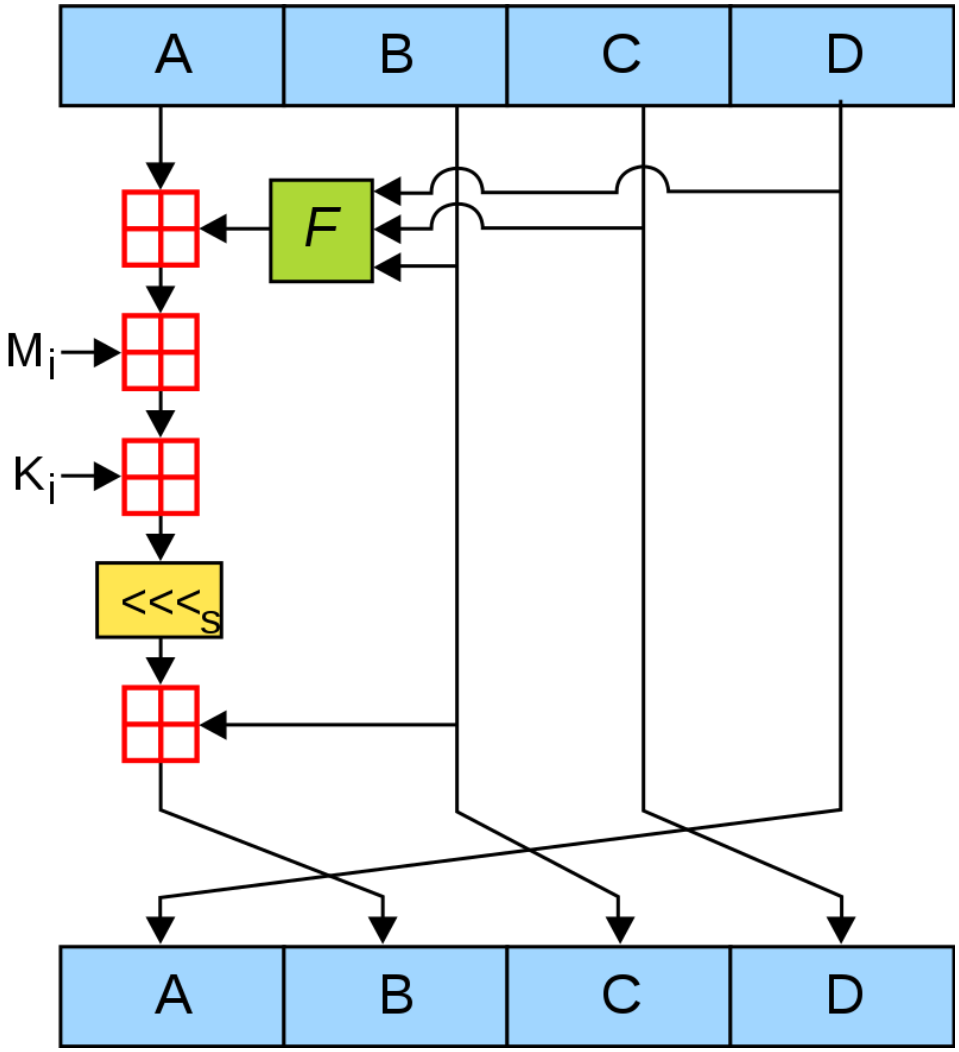
MD5 算法过程

- Step4：以512位的分组(16个字)位单位处理消息。
  - 由四轮(64步)运算组成的压缩函数是算法的核心。压缩函数标记为HMD5。
- Step5：输出。
  - 所有的L个512位的分组处理完之后，第L个分组的输出即是128位的消息摘要。

MD5 的压缩函数


Y_q
512
CV_q
128
32
A B C D
F, T[1...16], X[i]
16 steps
A B C D
G, T[17...32], X[ρ 2i]
16 steps
A B C D
H, T[33...48], X[ρ 3i]
16 steps
A B C D
I, T[49...64], X[ρ 4i]
16 steps
+ + + +
128
CV_{q+1}
Note: addition (+) is mod 2^{32}

步操作



$$F(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$$

$$G(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$$

$$H(B,C,D) = B \oplus C \oplus D$$

$$I(B,C,D) = C \oplus (B \vee \neg D)$$

从正弦函数构造的表T

- 表T是通过正弦函数构造的，T的第i个元素记为T[i]。
- T的每个元素都可以用32位表示，堆积化的32位输入数据，消除了输入数据的规律性。
- $T[i] = 2^{32} \text{ abs } (\sin(i))$ $i=1,2,..64$

T[1] = D76AA478	T[17] = F61E2562	T[33] = FFFA3942	T[49] = F4292244
T[2] = E8C7B756	T[18] = C040B340	T[34] = 8771F681	T[50] = 432AFF97
T[3] = 242070DB	T[19] = 265E5A51	T[35] = 699D6122	T[51] = AB9423A7
T[4] = C1BDCEEE	T[20] = E9B6C7AA	T[36] = FDE5380C	T[52] = FC93A039
T[5] = F57C0FAF	T[21] = D62F105D	T[37] = A4BEEA44	T[53] = 655B59C3
T[6] = 4787C62A	T[22] = 02441453	T[38] = 4BDECFA9	T[54] = 8F0CCC92
T[7] = A8304613	T[23] = D8A1E681	T[39] = F6BB4B60	T[55] = FFEFF47D
T[8] = FD469501	T[24] = E7D3FBC8	T[40] = BEBFBC70	T[56] = 85845DD1
T[9] = 698098D8	T[25] = 21E1CDE6	T[41] = 289B7EC6	T[57] = 6FA87E4F
T[10] = 8B44F7AF	T[26] = C33707D6	T[42] = EAA127FA	T[58] = FE2CE6E0
T[11] = FFFF5BB1	T[27] = F4D50D87	T[43] = D4EF3085	T[59] = A3014314
T[12] = 895CD7BE	T[28] = 455A14ED	T[44] = 04881D05	T[60] = 4E0811A1
T[13] = 6B901122	T[29] = A9E3E905	T[45] = D9D4D039	T[61] = F753FE82
T[14] = FD987193	T[30] = FCEFA3F8	T[46] = E6DB99E5	T[62] = BD3AF235
T[15] = A679438E	T[31] = 676F02D9	T[47] = 1FA27CF8	T[63] = 2AD7D2BB
T[16] = 49B40821	T[32] = 8D2A4C8A	T[48] = C4AC5665	T[64] = EB86D391

MD5压缩函数

- MD5中每轮对缓冲区ABCD进行16步迭代，每步迭代为：

  $a \leftarrow b + ((a + g(b,c,d) + X[k] + T[i]) <<< s)$

  - a,b,c,d：缓冲区的四个字，它按照一定的次序随迭代步变化
  - g：基本逻辑函数F/G/H/I之一
  - <<< s：32位的变量循环左移s位
  - X[k] = M[q*16+k]：消息第q个512位分组的第k个32位字
  - T[i]：矩阵T中的第i个32位字
  - +：模2^{32}加法

安全hash算法：SHA

- 安全hash算法是NIST设计，于1993作为联邦信息处理标准FIPS180发布的，修订版于1995年发布FIPS 180-1，也称之为SHA-1。
- SHA算法建立在MD4之上，基本框架与MD4类似。
- SHA-1算法输入是长度小于2^{64}位的消息，输出是160位的消息摘要，输入消息以512位的分组为单位进行处理。

SHA算法步骤

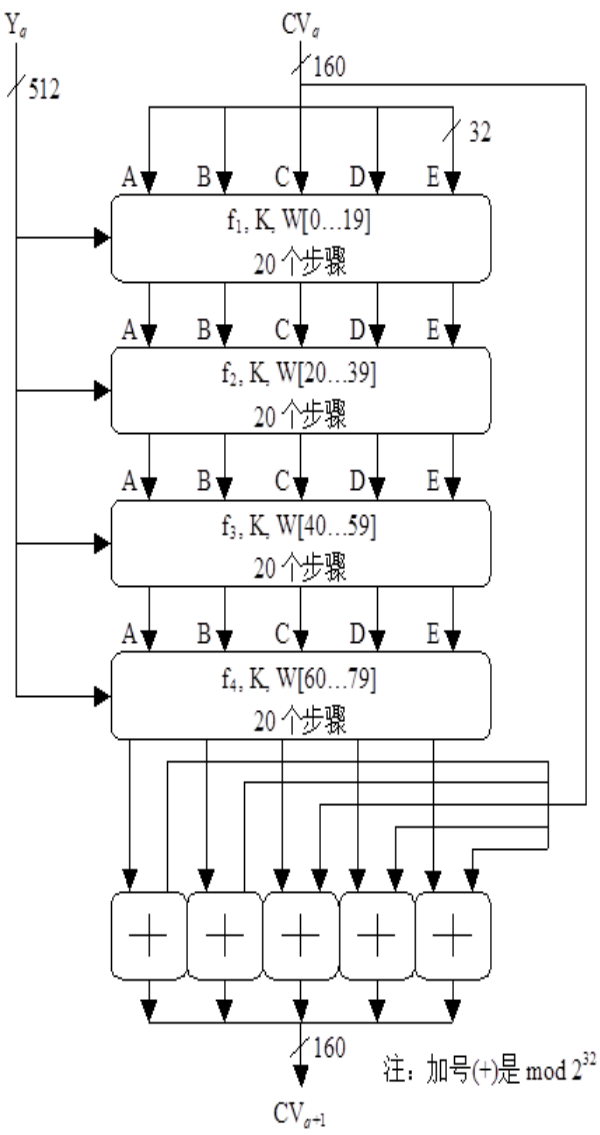
- SHA-1算法也将消息按照512位分组，但hash值和连接变量长为160位。
- Step1：增加填充位。
  - 填充消息使之与448模512同余。即填充后的消息比512的整数倍少64位。
- Step2：填充长度。
  - 用64位表示填充前的报文长度，附加在填充后的结果后面。

SHA-1算法步骤

- Step3：初始化MD 缓存。
  - Hash函数的中间结果和最终结果都保存于160位的缓冲区中，缓冲区用5个32位的寄存器(A,B,C,D,E)表示，并将这些寄存器初始化。
  - 采用高位优先的结构。
    - 字A：67 45 23 01
    - 字B：EF CD AB 89
    - 字C：98 BA DC FE
    - 字D：10 32 54 76
    - 字E：C3 D2 E1 F0

SHA-1算法步骤

- Step4：以512位的分组(16个字)位单位处理消息，包含四个具有相似结构的“循环”，但每循环使用不同的原始逻辑函数。
  - 每一循环都以当前正在处理的512 bit(Yq)和160 bit的缓存值ABCDE为输入，然后更新缓存的内容。每循环也使用一个额外的常数值Kt，其中0≤t≤79说明四循环80步中的一步。
  - 第四循环(第80步)的输出加到第一循环的输入(CVq)产生CVq+l。相加是缓存中5个字分别与CVq中对应的5个字以模232相加。

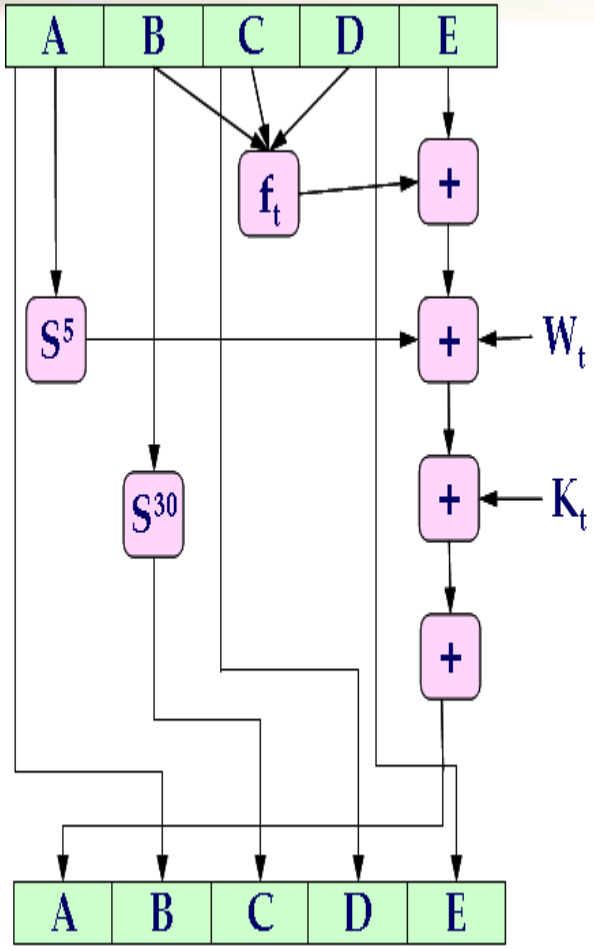


SHA-1算法步骤

- Step5：输出。
  - 所有的L个512位的分组处理完之后，第L个分组的输出即是160位的消息摘要。

SHA-1压缩函数

- 处理一个512位的分组要执行80步。
  每步的处理过程是一样的。
  - A,B,C,D,E：缓冲区的5个字
  - t：步骤编号，0<=t<=79
  - f(t,B,C,D)：第t步使用的基本逻辑函数
  - Sk：32位的变量循环左移k位
  - Wt：从当前512位输入分组导出的32位字
  - Kt：加法常量。
    共使用了四个不同的加法常量。
  - +：模2^{32}加法



系列SHA算法

- NIST已经在FIPS 180-4中颁布了SHA标准：SHA-1, SHA-224, SHA-256, SHA-384, SHA-512

	SHA-1	SHA-256	SHA-384	SHA-512
Message digest size	160	256	384	512
Message size	$<2^{64}$	$<2^{64}$	$<2^{128}$	$<2^{128}$
Block size	512	512	1024	1024
Word size	32	32	64	64
Number of steps	80	80	80	80
Security	80	128	192	256

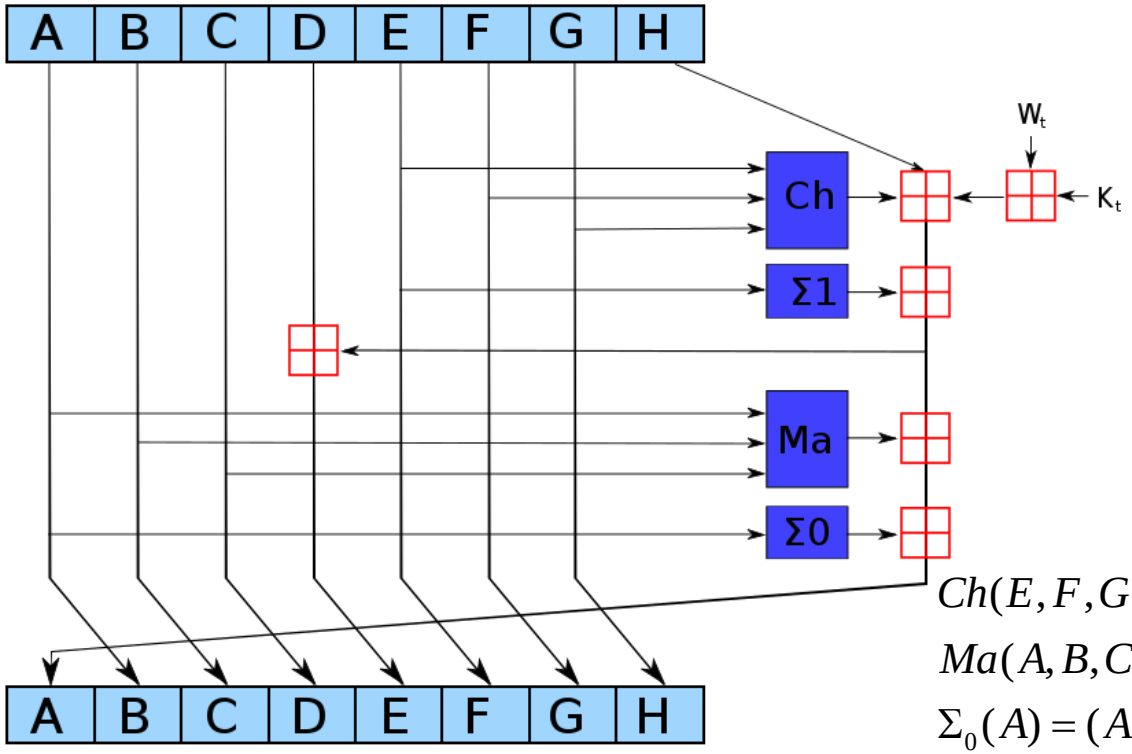
Notes: 1. All sizes are measured in bits.
2. Security refers to the fact that a birthday attack on a message digest of size n produces a collision with a workfactor of approximately $2^{n/2}$.

SHA-1碰撞攻击

- Finding collisions on the full SHA-1 , Xiaoyun wang, Yinqun Lisa Yin, Hongbo Yu, Crypto 2005.
- The first collision for full SHA-1, Marc Stevens et.al, 2017， https://shattered.it/

SHA-2 family

- **SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256.**



$$Ch(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$

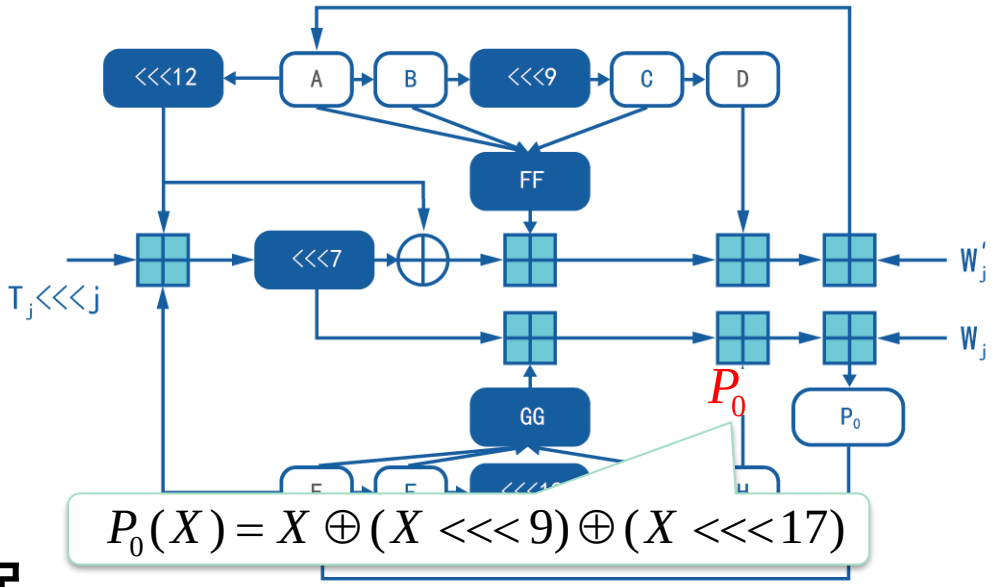
$$Ma(A,B,C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

$$\Sigma_0(A) = (A >>> 2) \oplus (A >>> 13) \oplus (A >>> 22)$$

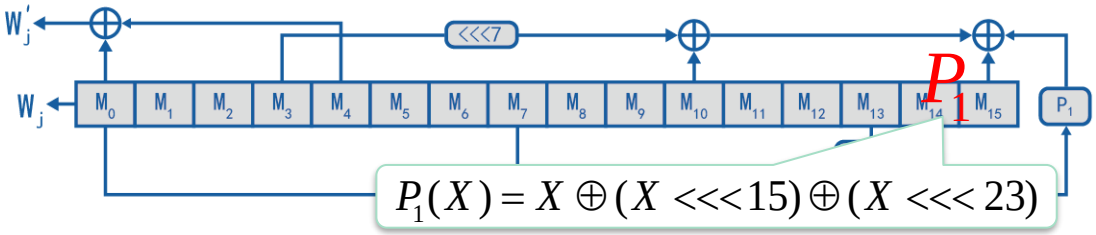
$$\Sigma_1(E) = (E >>> 6) \oplus (A >>> 11) \oplus (A >>> 25)$$

SM3 (256比特输出）

- 轮函数



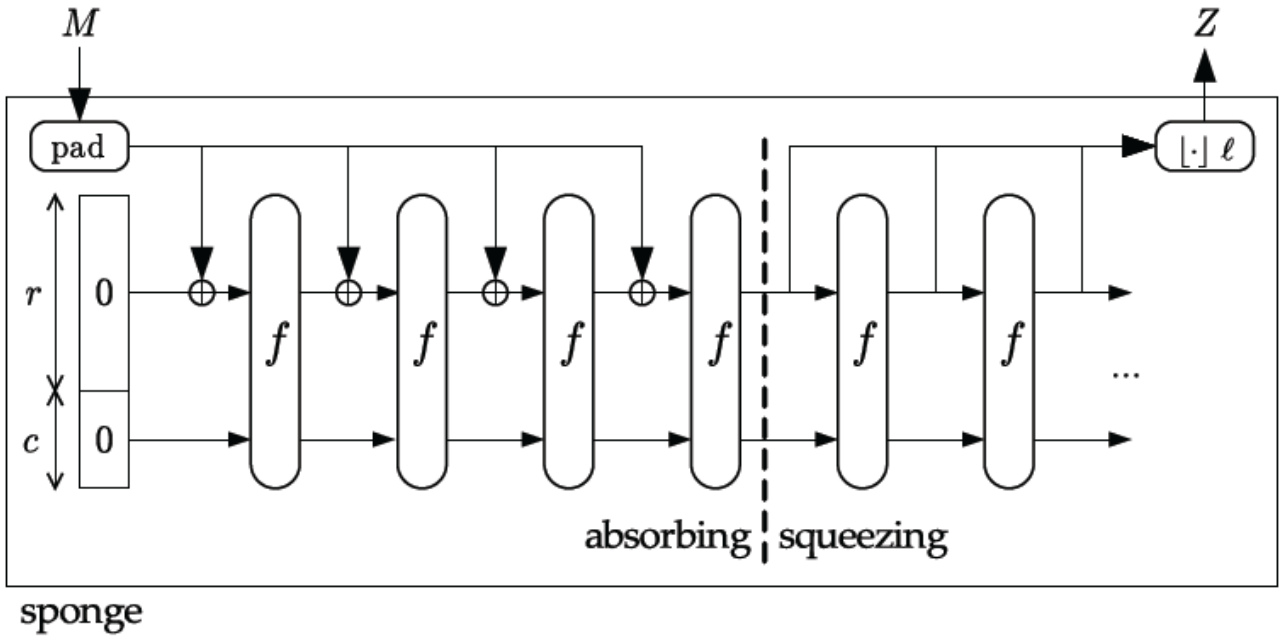
- 消息扩展



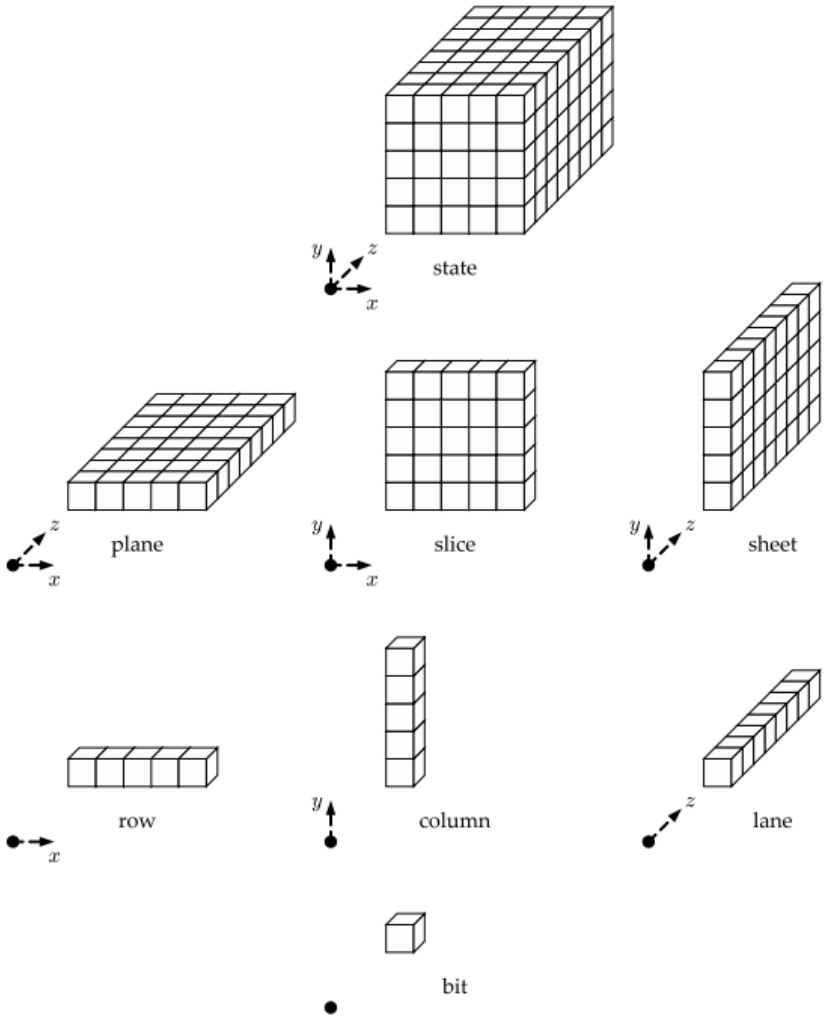
41

SHA-3

- Sponge结构



SHA-3



轮函数

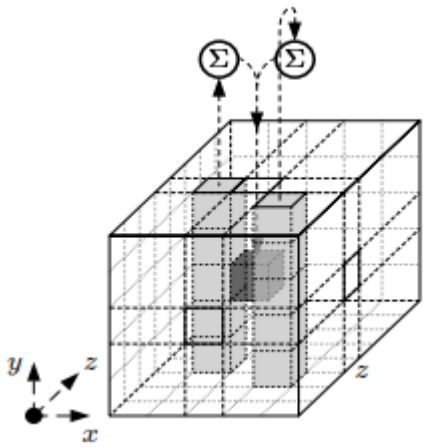
$$\theta : A_{x,y,z} = A_{x,y,z} \oplus \bigoplus_{i=0\sim4} (A_{x-1,i,z} \oplus A_{x+1,i,z-1})$$

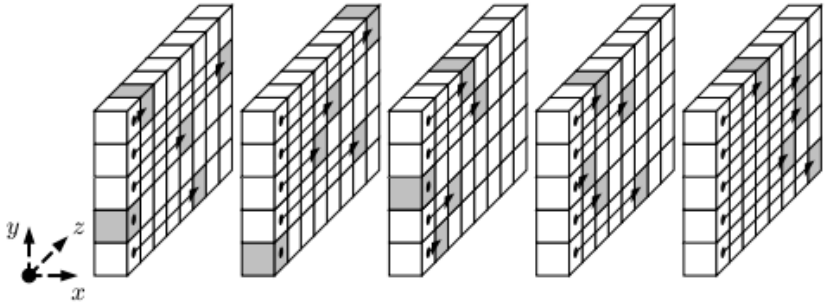
$$\rho : A_{x,y,z} = A_{x,y,(z-r_{x,y})}$$

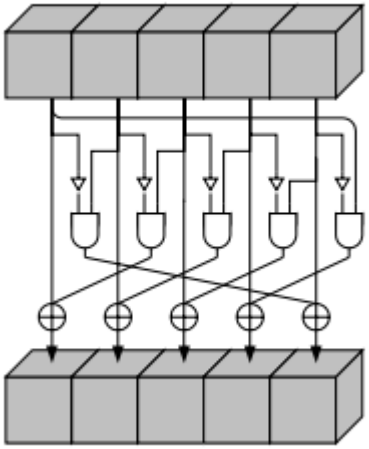
$$\pi : A_{x,y,z} = A_{x+3y,x,z}$$

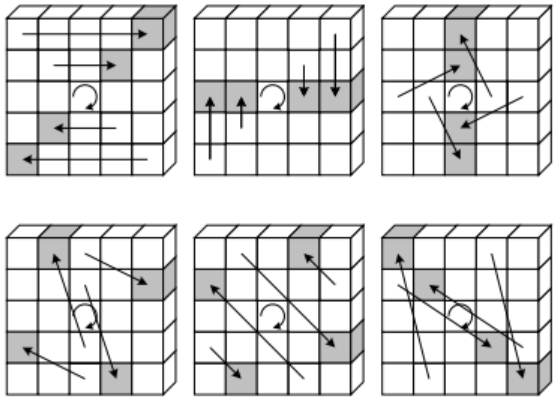
$$\chi : A_{x,y,z} = A_{x,y,z} \oplus (A_{x+1,y,z} \oplus 1) \cdot A_{x+2}$$

$$\iota : A_{0,0,z} = A_{0,0,z} \oplus RC_z$$









第三部分

Hash函数安全性分析

生日攻击

生日攻击原起源于生日悖论

问题1：房子里应有多少人才有可能使他们中至少一个人与某人生日相同？答案是365人。第二原像攻击

问题2：房间里应该有多少人才能使他们中至少两个人生日相同呢？答案出乎意料的低：23人。碰撞攻击

把寻找两个随机的具有相同生日的两个人的方法称为生日攻击（birthday attack）。由于两种答案是接近于平方倍数关系，也成为平方根攻击。

生日攻击

定理：若 $k \geq 1.18 \times 2^{n/2} \approx 2^{n/2}$，则$k$个在[1,$2^n$]中的随机数有两个相同的概率不低于0.5。

证明：设$2^n=N$，则k个随机数$y_1,y_2,\ldots,y_k$中没有碰撞的概率为

$$(1-\frac{1}{N})(1-\frac{2}{N})\cdots(1-\frac{k-1}{N})=\prod_{i=1}^{n}(1-\frac{i}{N})$$

由于 $e^{-x}=1-x+\frac{x^2}{2!}-\frac{x^3}{3!}+\cdots$ ，当x较小时 $e^{-x}\approx 1-x$

$$\prod_{i=1}^{k-1}(1-\frac{i}{N})\approx\prod_{i=1}^{k-1}e^{-\frac{i}{N}}=e^{-\frac{k(k-1)}{2N}}$$

生日攻击

则$y_1, y_2, \ldots, y_k$中至少有一个碰撞的概率为 $1-e^{-\frac{k(k-1)}{2N}}$

设 $\varepsilon \approx e^{-\frac{k(k-1)}{2N}}$ ，则$k \approx \sqrt{2\ln\frac{1}{1-\varepsilon}}\sqrt{N}$

取 $\varepsilon = 0.5$ ，则 $k \approx \ln 4\sqrt{N} = 1.17\sqrt{N} = 1.17 \times 2^{\frac{n}{2}}$

应用：当N=365，$\varepsilon = 0.5$ ， 则 k =22.3

故Hash函数摘要长度应至少为128比特。

生日攻击的应用

伪造数字签名(Yuval 生日攻击算法）

Alice如何利用生日攻击来欺骗Bob

(1) Alice准备好两个不同内容的文件（可能仅在关键字上有差别，但利害相反），一个对她有利，记为版本A，一个对Bob有利，记为版本B，显然，Bob只会对版本B签名，记为(B,sign(h(B)))；

(2) Alice想伪造Bob的签名sign(h(B))，她将文件A做一些小的改动（添加空格等），计算产生$2^{n/2}$个改动后的文件$A_i(i=1,2,\ldots,2^{n/2})$的散列值。

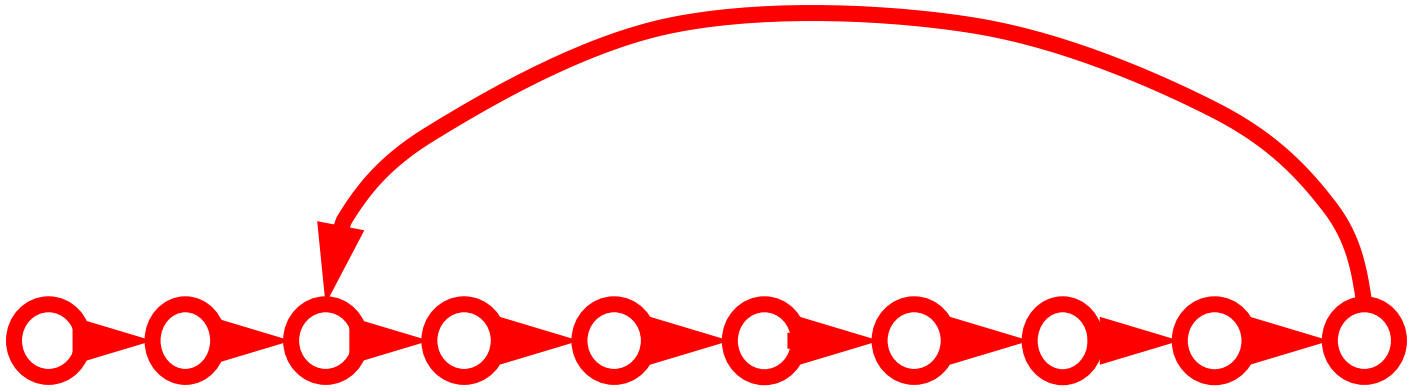
(3) 同样，Alice对文件B做一些小的改动，计算产生$2^{n/2}$个改动后的文件$B_i(i=1,2,\ldots,2^{n/2})$的散列值。

(4) 比较两个集合，寻找碰撞$h(A_i)=h(B_j)$ (复杂度2NlgN)

(5) Alice 将消息B_j送给Bob， 获得签名 sigh(h(B_j))，Alice也就获得了A_i的合法签名，欺骗成功

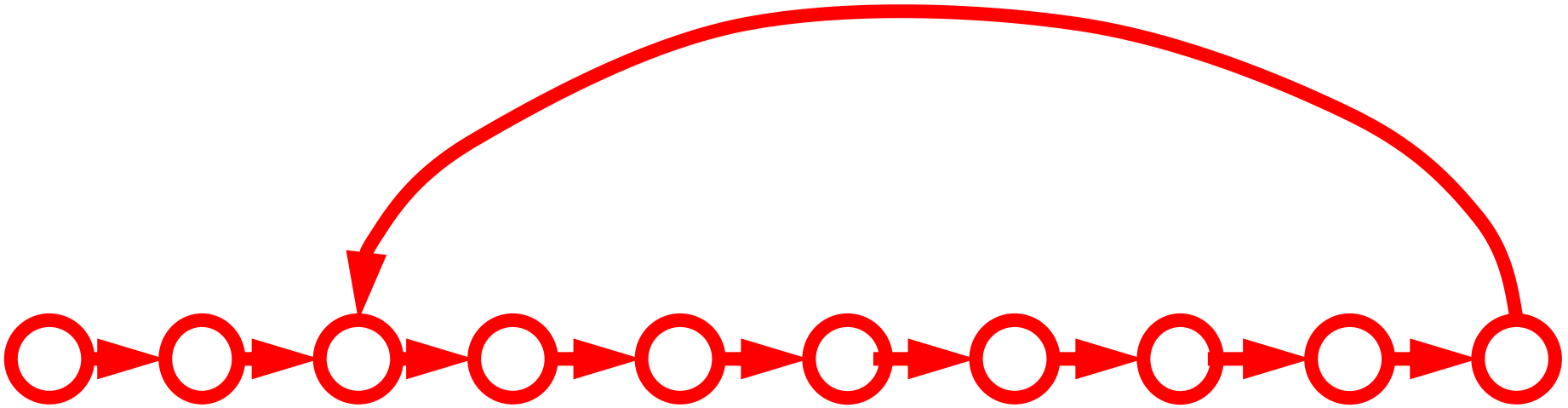
寻找碰撞的方法

- 穷举搜索: 时间 2^n ， 空间：no
- 生日攻击： 构造一个$2^{n/2}$ 随机杂凑值的表，存储该表，寻找碰撞。时间/空间 ： $2^{n/2}$
- 随机路径算法：迭代杂凑值直到找到路径中一个圈的起点(entry point). 时间$2^{n/2}$ ， 空间：可忽略



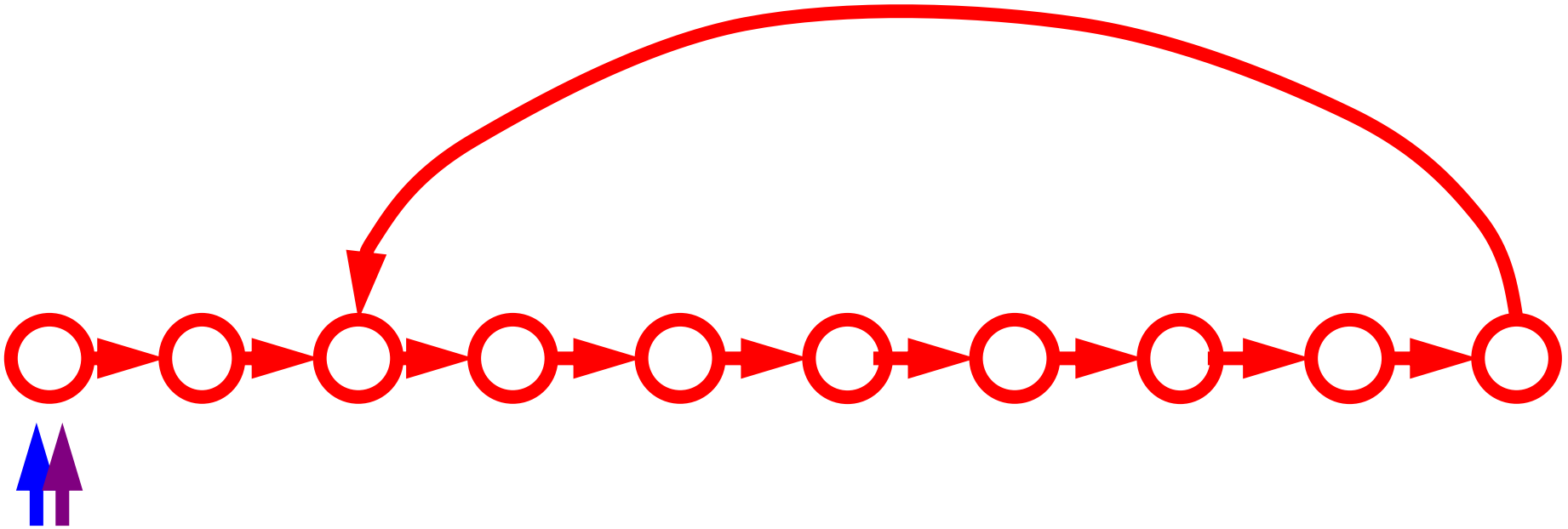
Floyd's two finger algorithm:

- 使用两个指针（pointers)
- 一个以正常速度运行，另一个以2倍速度，直到他们相遇(collide)



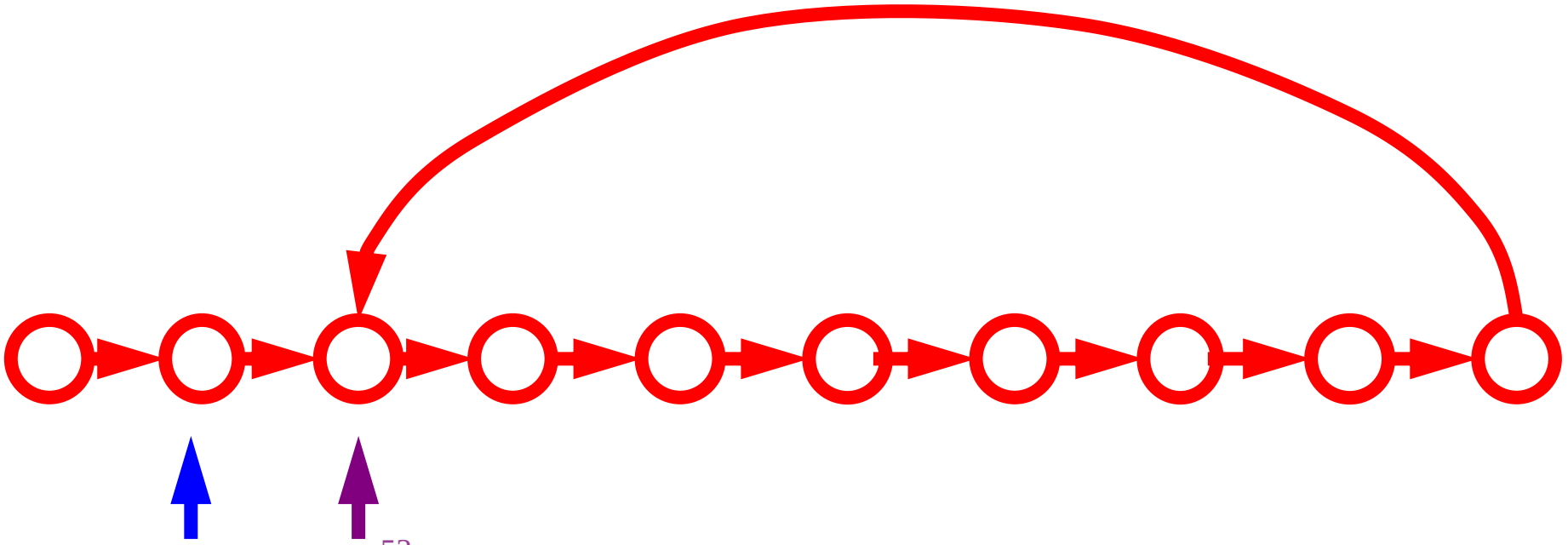
Floyd's two finger algorithm:

- 使用两个指针（pointers)
- 一个以正常速度运行，另一个以2倍速度，直到他们相遇(collide)



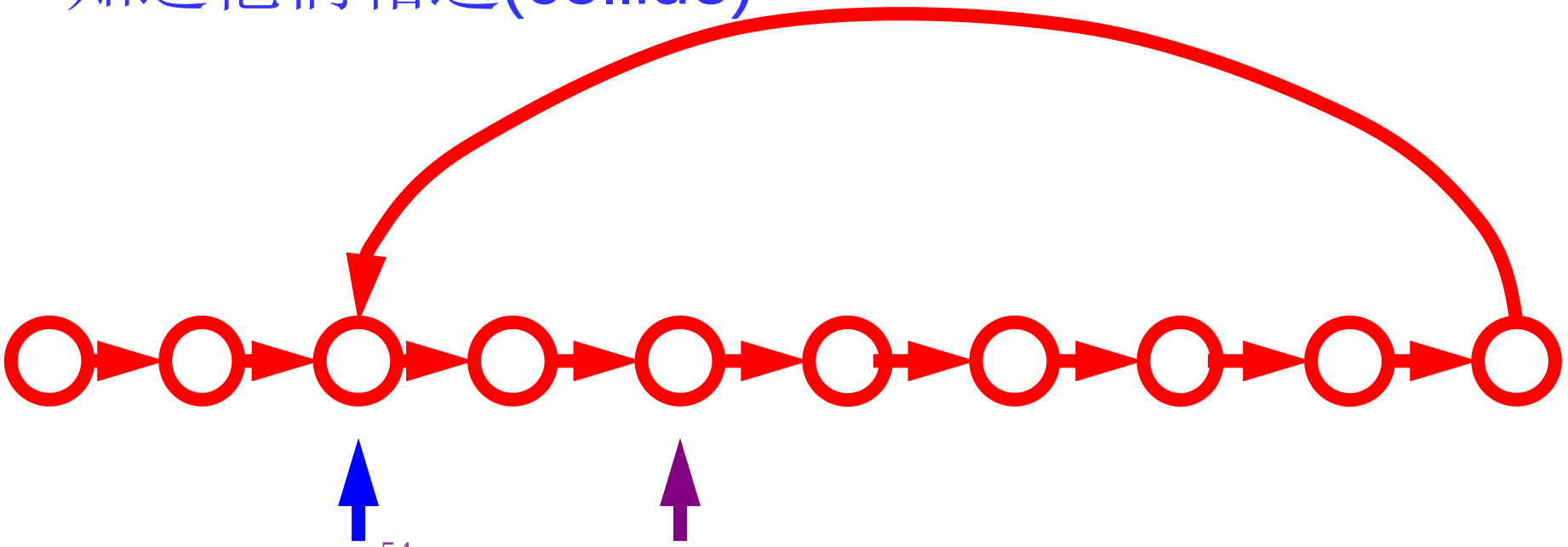
Floyd's two finger algorithm:

- 使用两个指针（pointers)
- 一个以正常速度运行，另一个以2倍速度，直到他们相遇(collide)



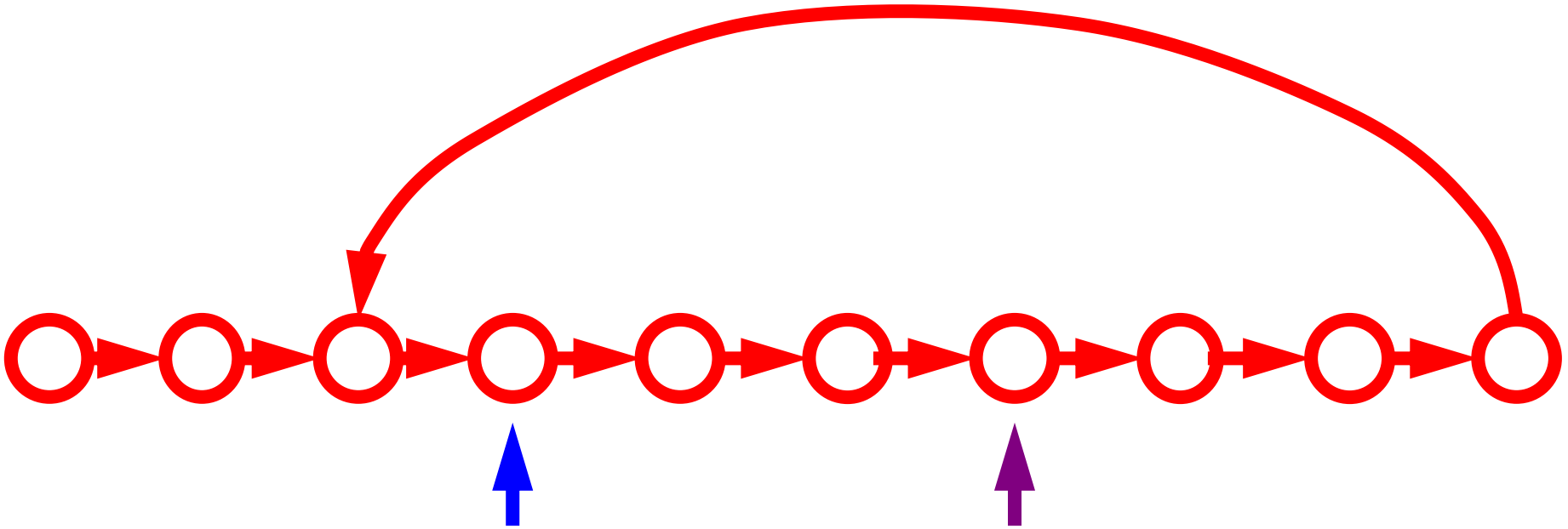
Floyd's two finger algorithm:

- 使用两个指针（pointers)
- 一个以正常速度运行，另一个以2倍速度，知道他们相遇(collide)



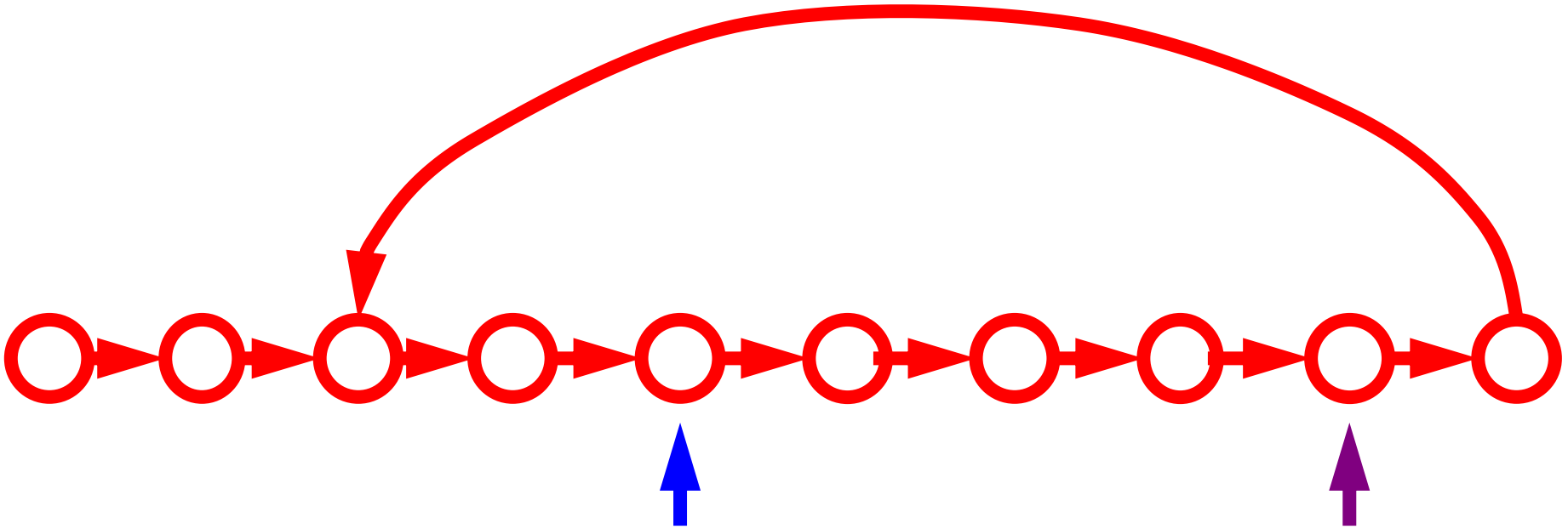
Floyd's two finger algorithm:

- 使用两个指针（pointers)
- 一个以正常速度运行，另一个以2倍速度，知道他们相遇(collide)



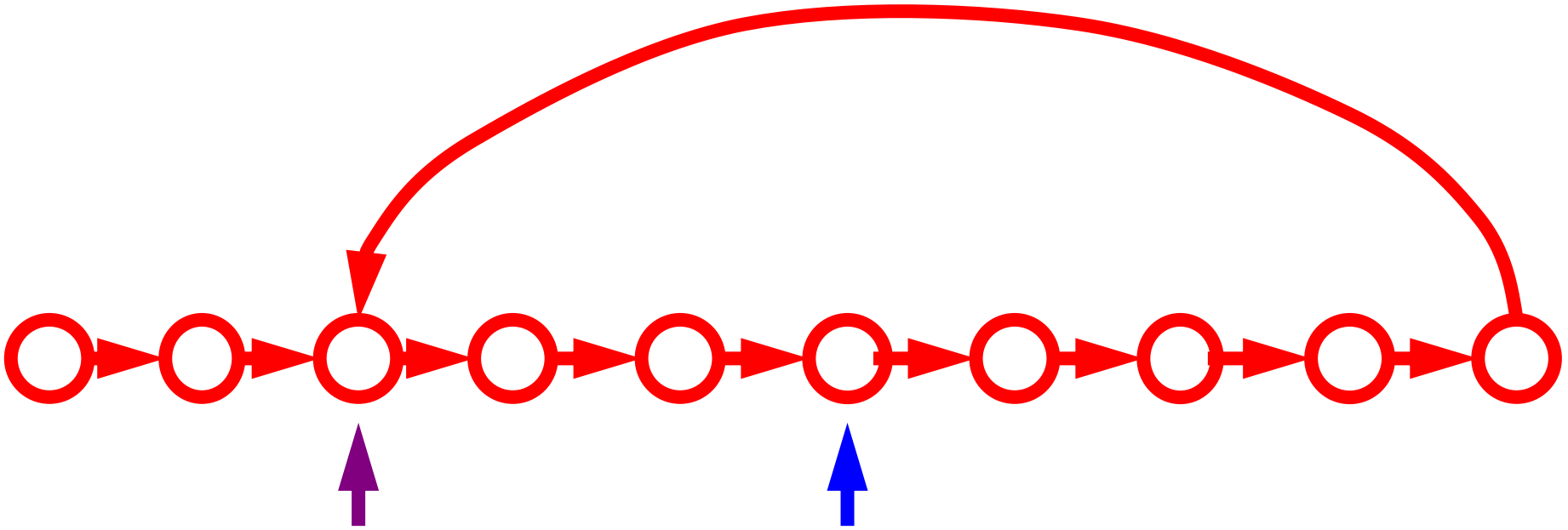
Floyd's two finger algorithm:

- 使用两个指针（pointers)
- 一个以正常速度运行，另一个以2倍速度，知道他们相遇(collide)



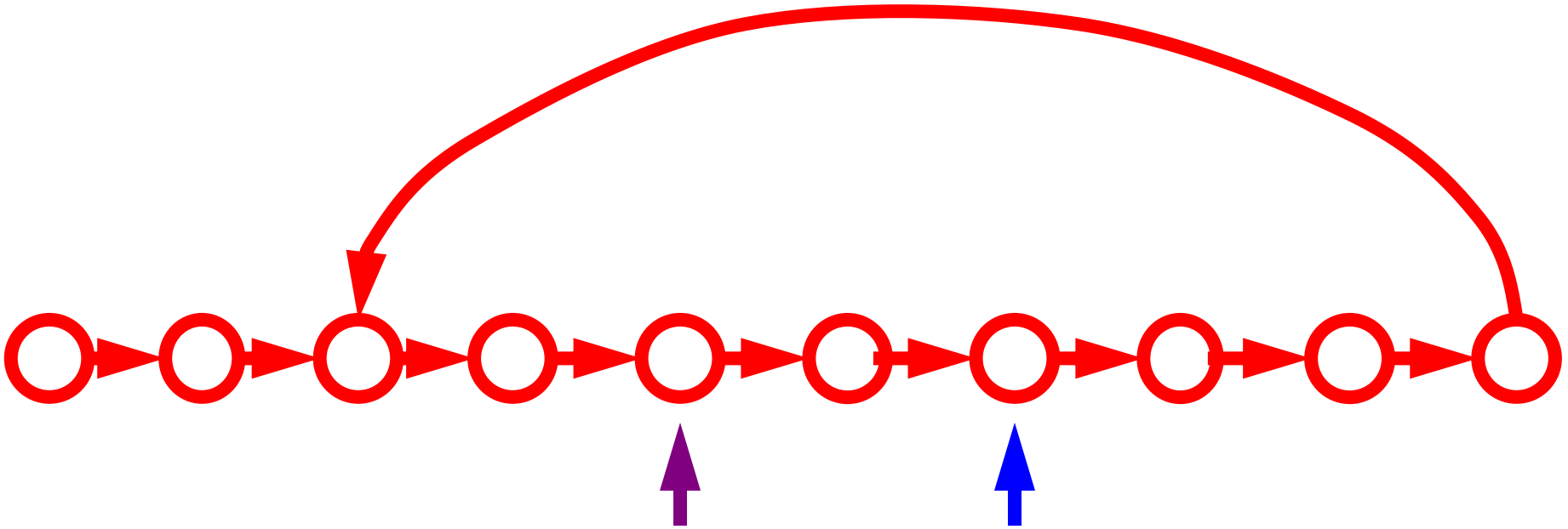
Floyd's two finger algorithm:

- 使用两个指针（pointers)
- 一个以正常速度运行，另一个以2倍速度，知道他们相遇(collide)



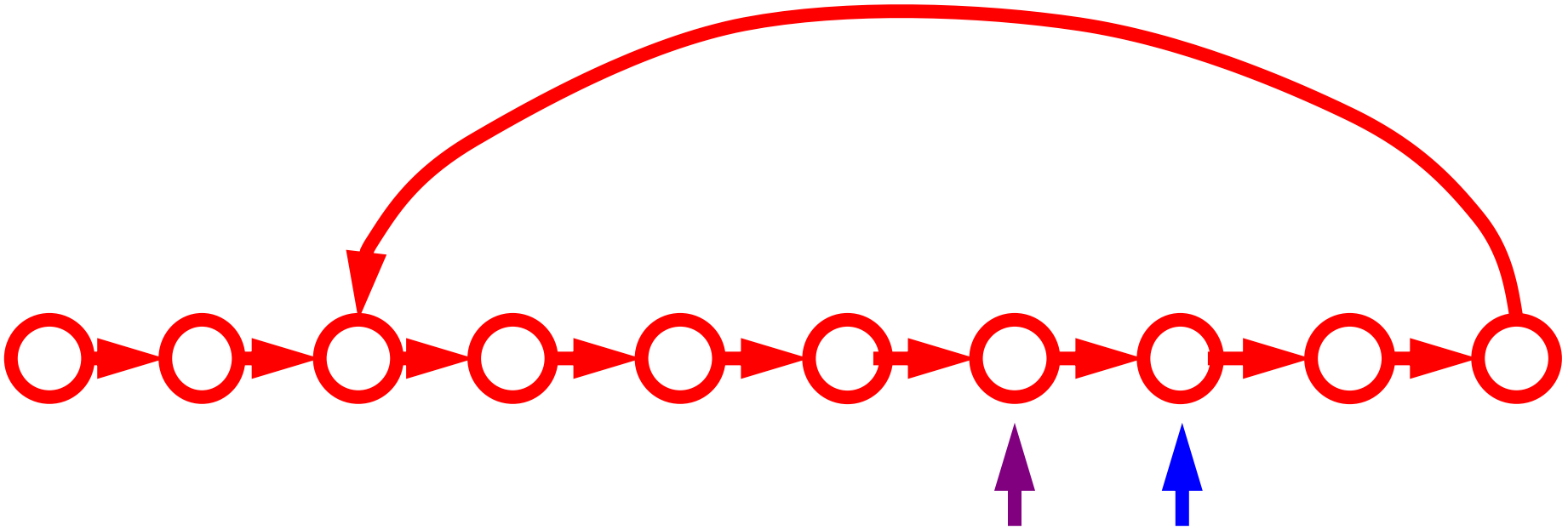
Floyd's two finger algorithm:

- 使用两个指针（pointers)
- 一个以正常速度运行，另一个以2倍速度，知道他们相遇(collide)



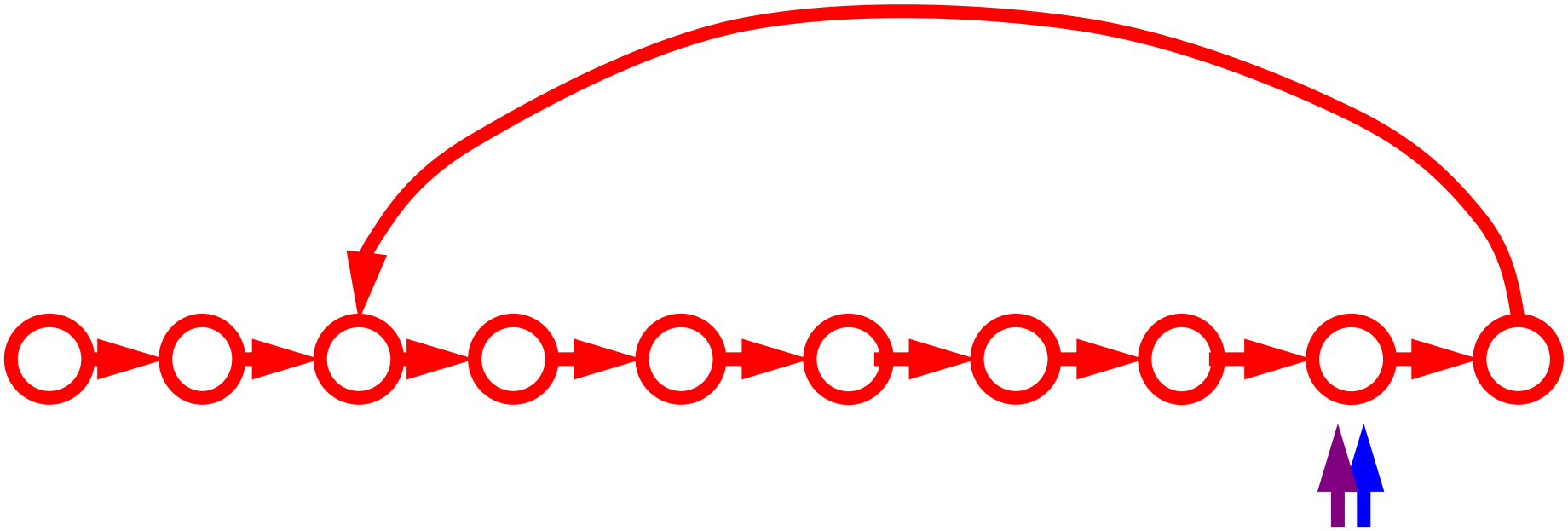
Floyd's two finger algorithm:

- 使用两个指针（pointers)
- 一个以正常速度运行，另一个以2倍速度，知道他们相遇(collide)



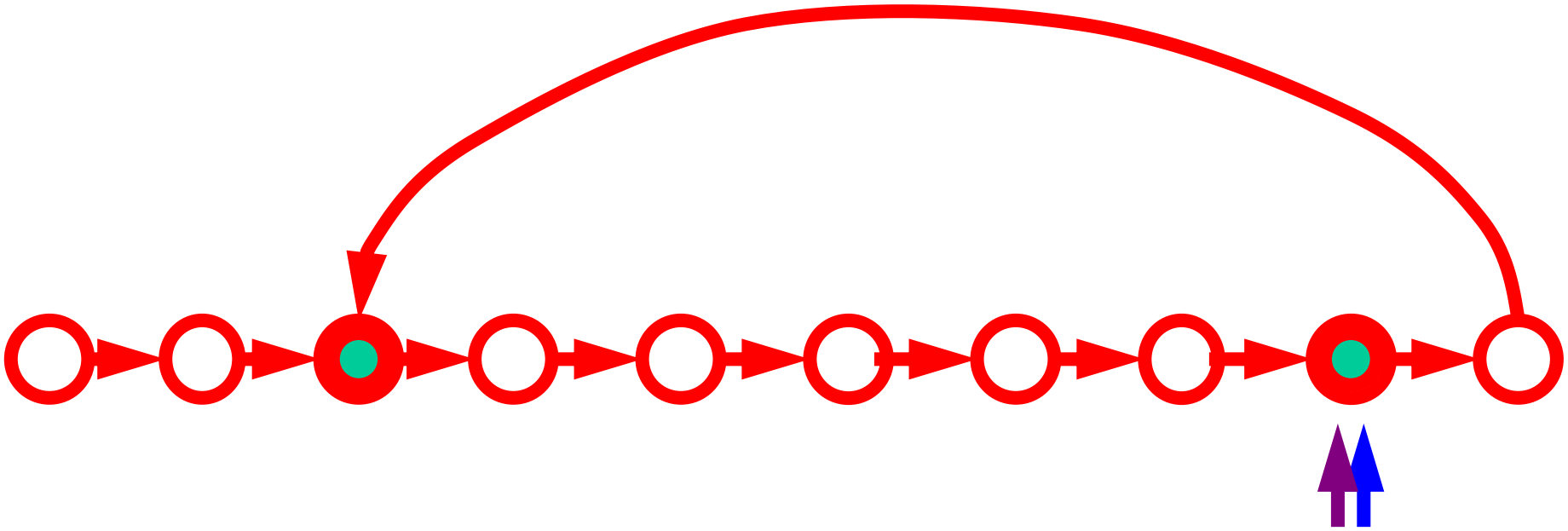
Floyd's two finger algorithm:

- 使用两个指针（pointers)
- 一个以正常速度运行，另一个以2倍速度，直到他们相遇(collide)



如何使用Floyd算法寻找碰撞?

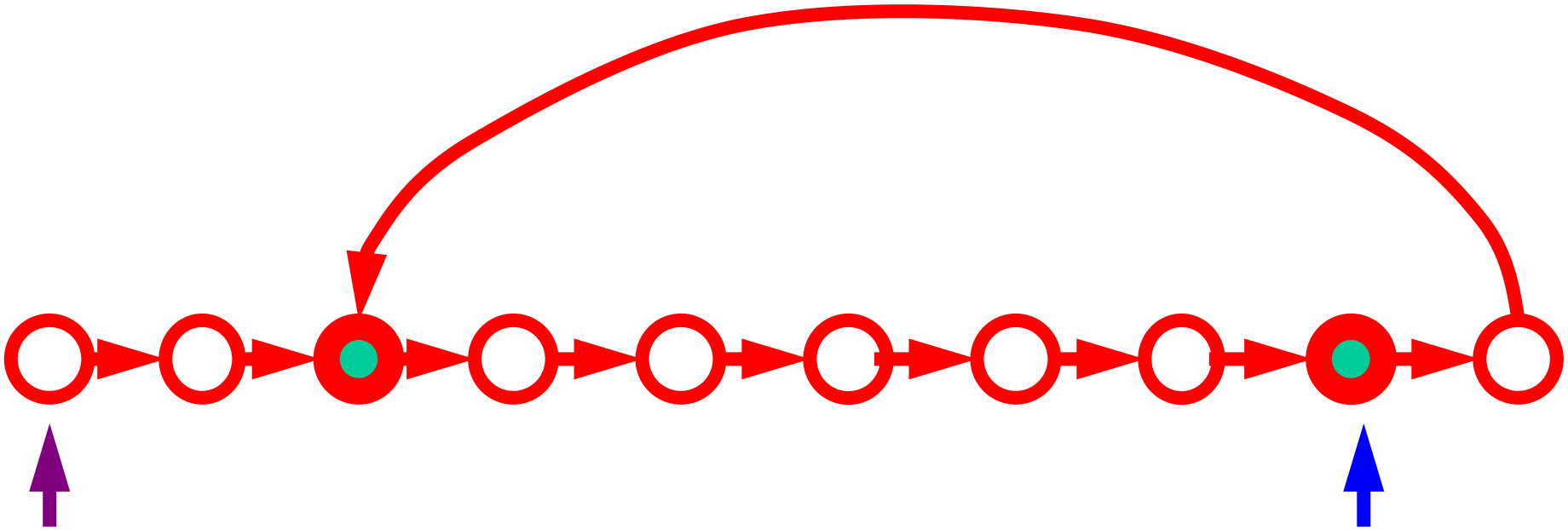
-首先寻找两个指针的相遇点



如何使用Floyd算法寻找碰撞?

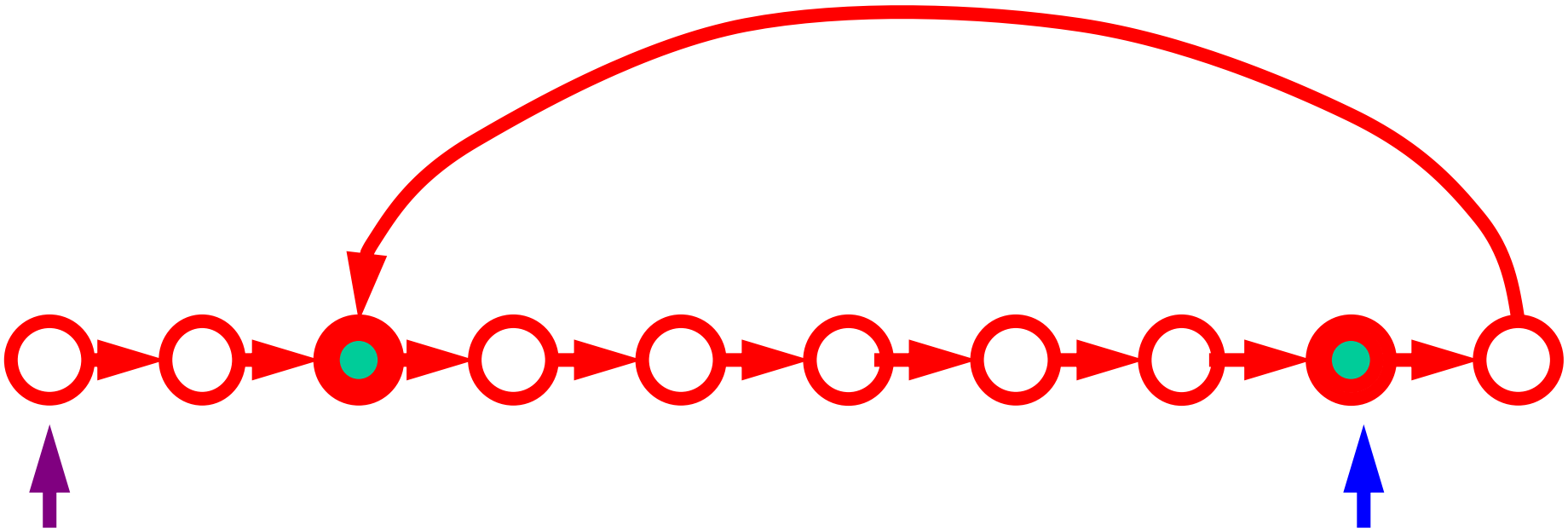
-首先寻找两个指针的相遇点

- 然后把其中一个指针移到起点



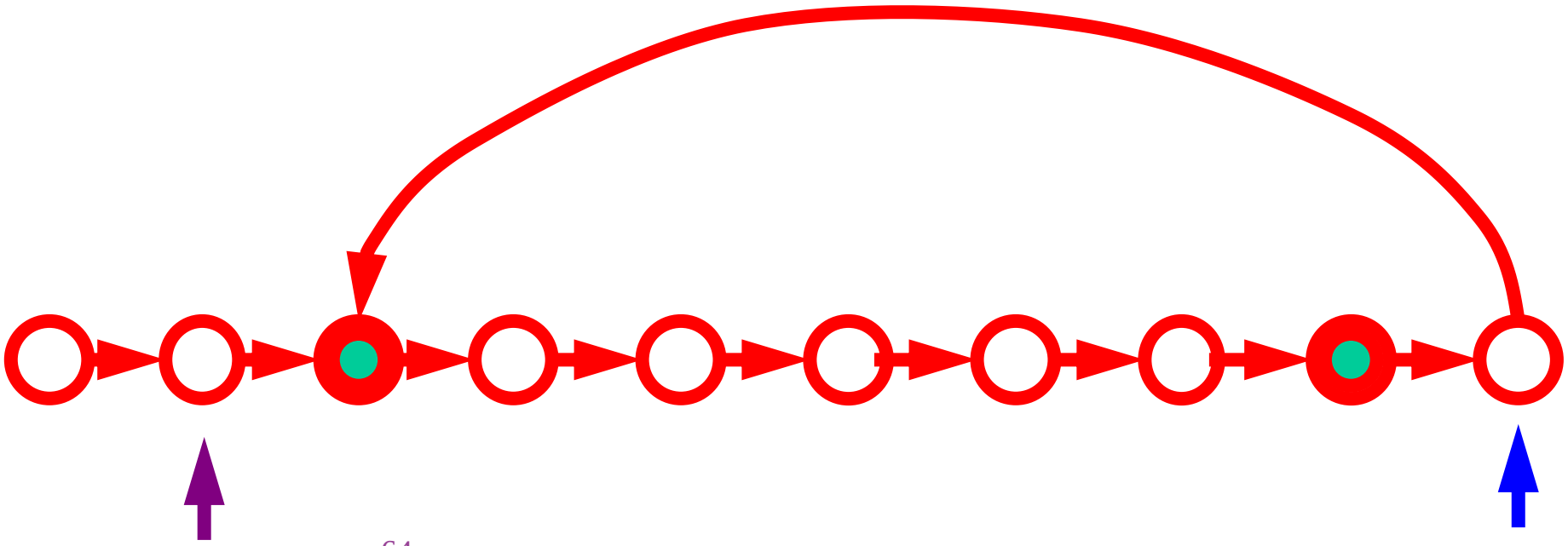
如何使用Floyd算法寻找碰撞?

- 首先寻找两个指针的相遇点
- 然后把其中一个移到起点
- 两个指针以相同的速度移动



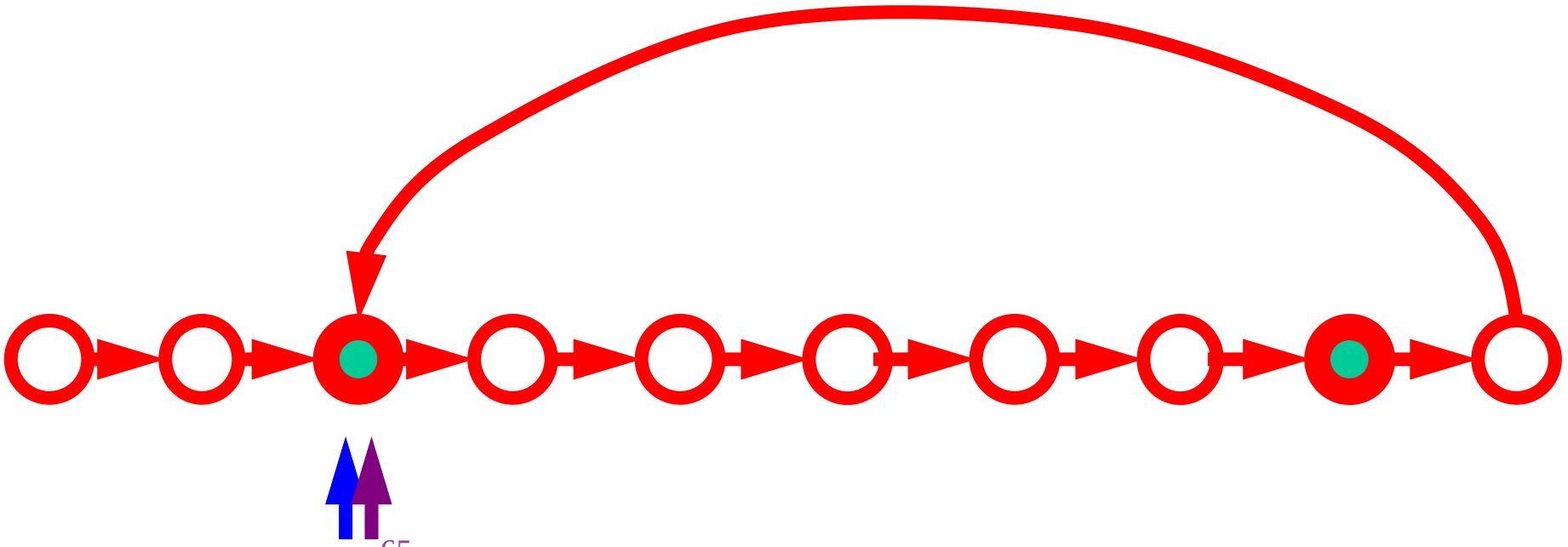
如何使用Floyd算法寻找碰撞?

- 首先寻找两个指针的相遇点
- 然后把其中一个移到起点
- 两个指针以相同的速度移动

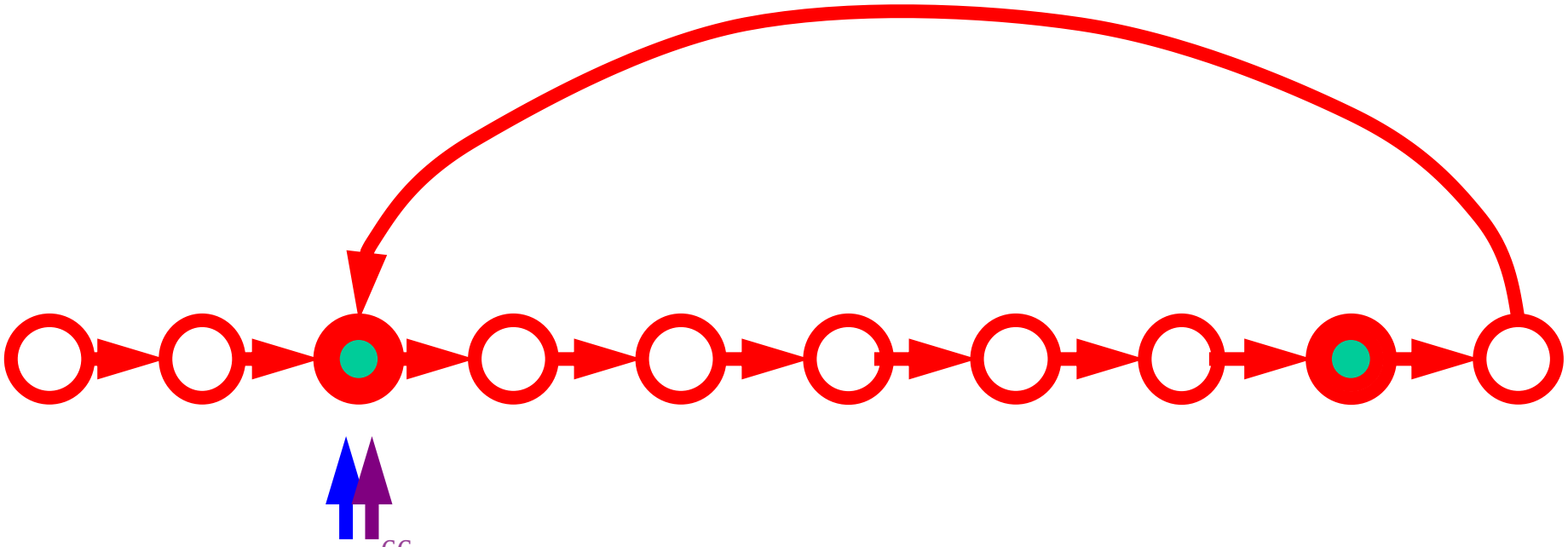


如何使用Floyd算法寻找碰撞?

- 首先寻找两个指针的相遇点
- 把其中一个移到起点
- 以相同的速度移动两个指针

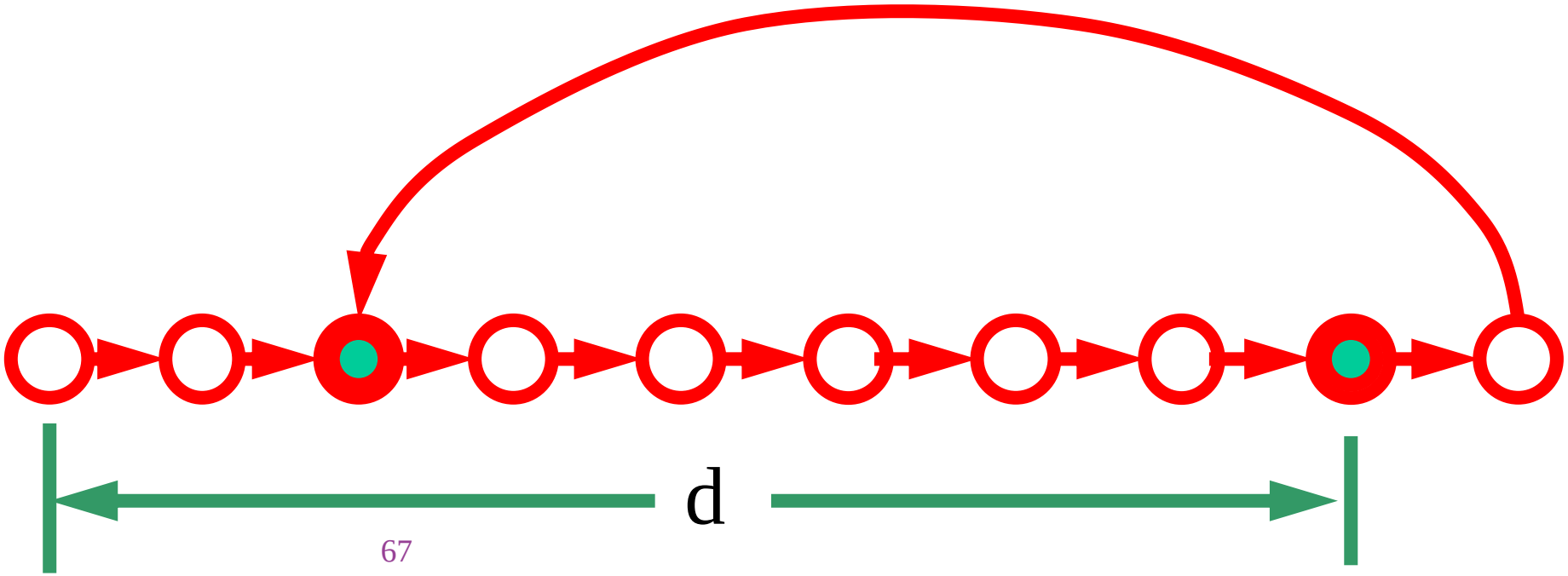


Why does it work?



Why does it work?

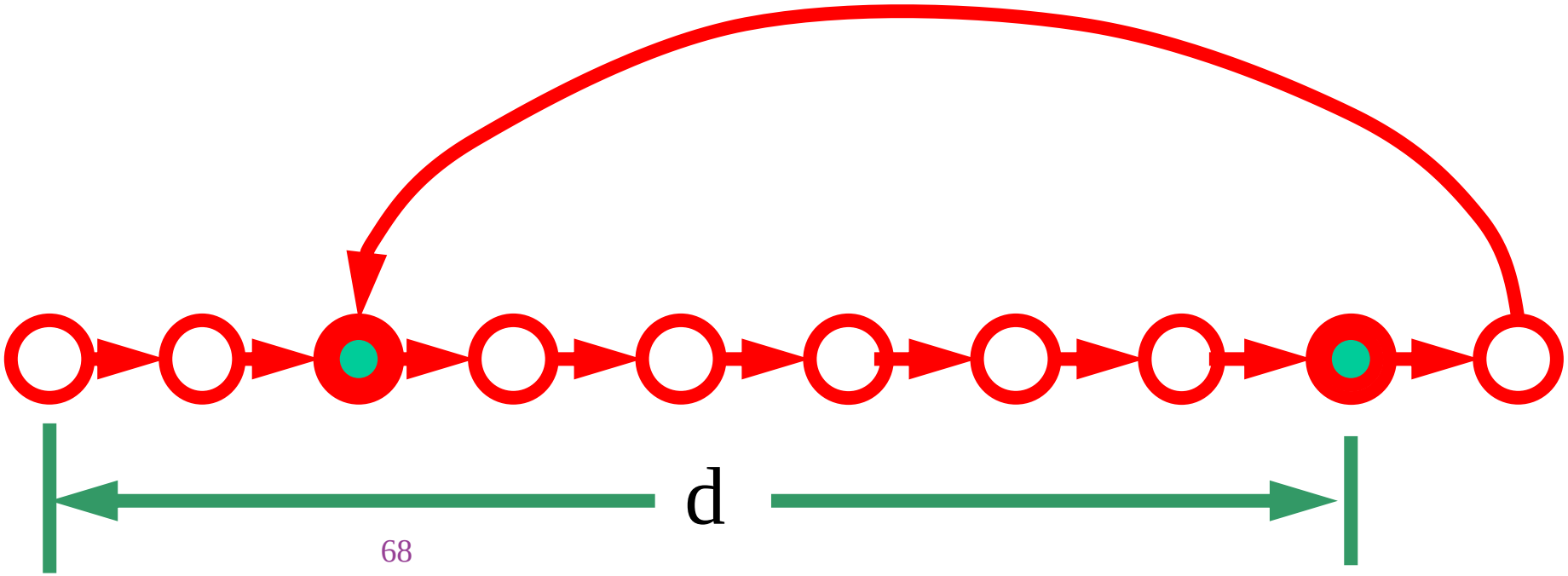
- 让 d 代表从起始点到相遇点的距离



Why does it work?

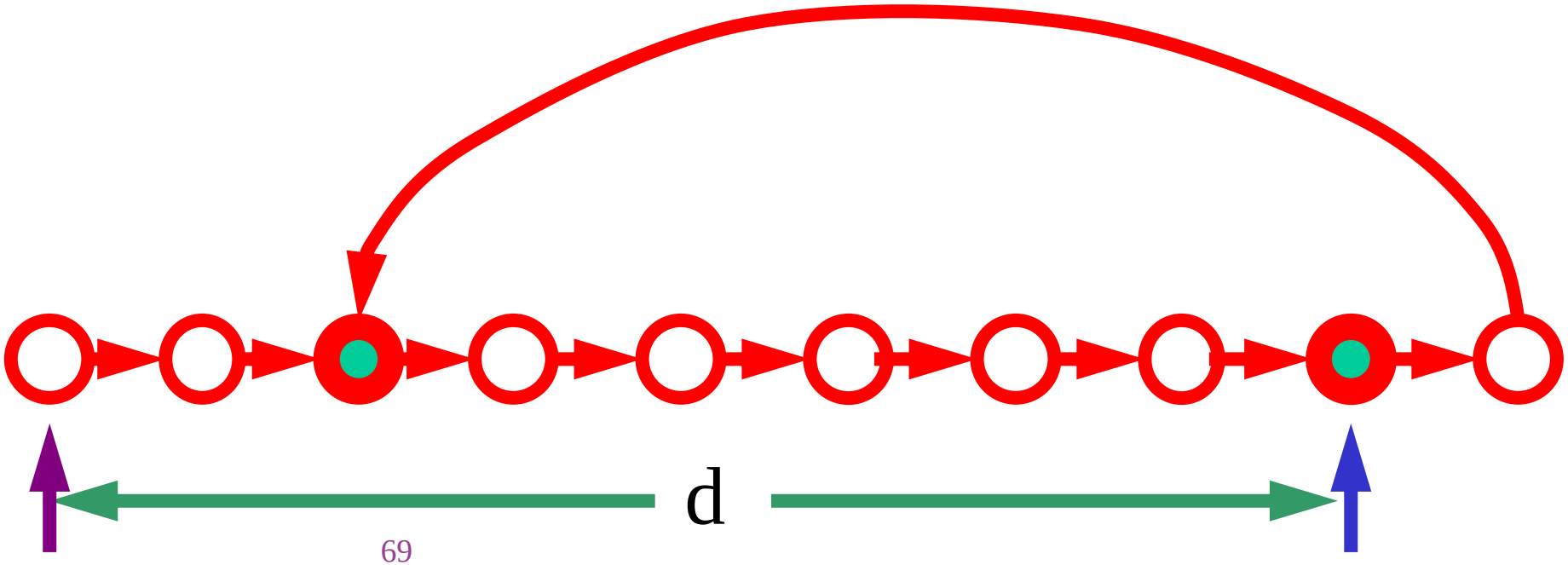
-让 d 代表从起始点到相遇点的距离

- 快的指针移动了2d到达相同的点，所以d(未知)是一个环（圈）的长度的倍数



Why does it work?

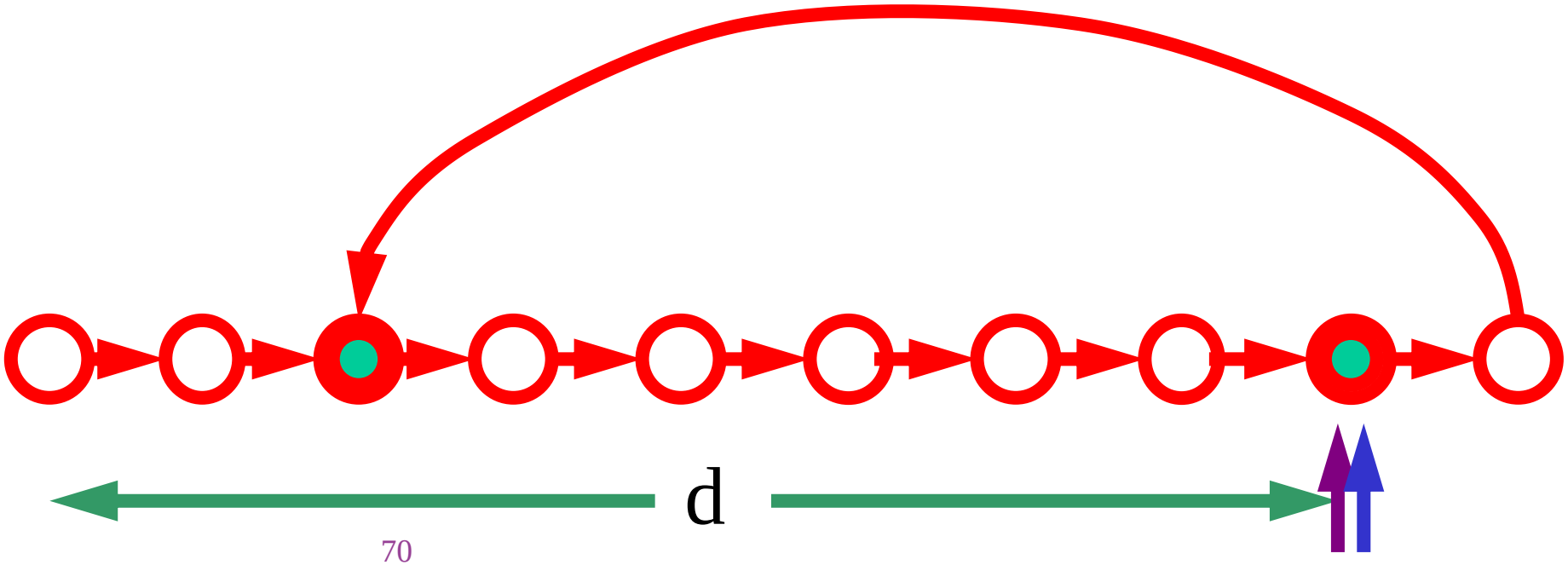
- 两个指针再移动d步就到达相同的点



Why does it work?

-两个指针再移动d步就到达相同的点

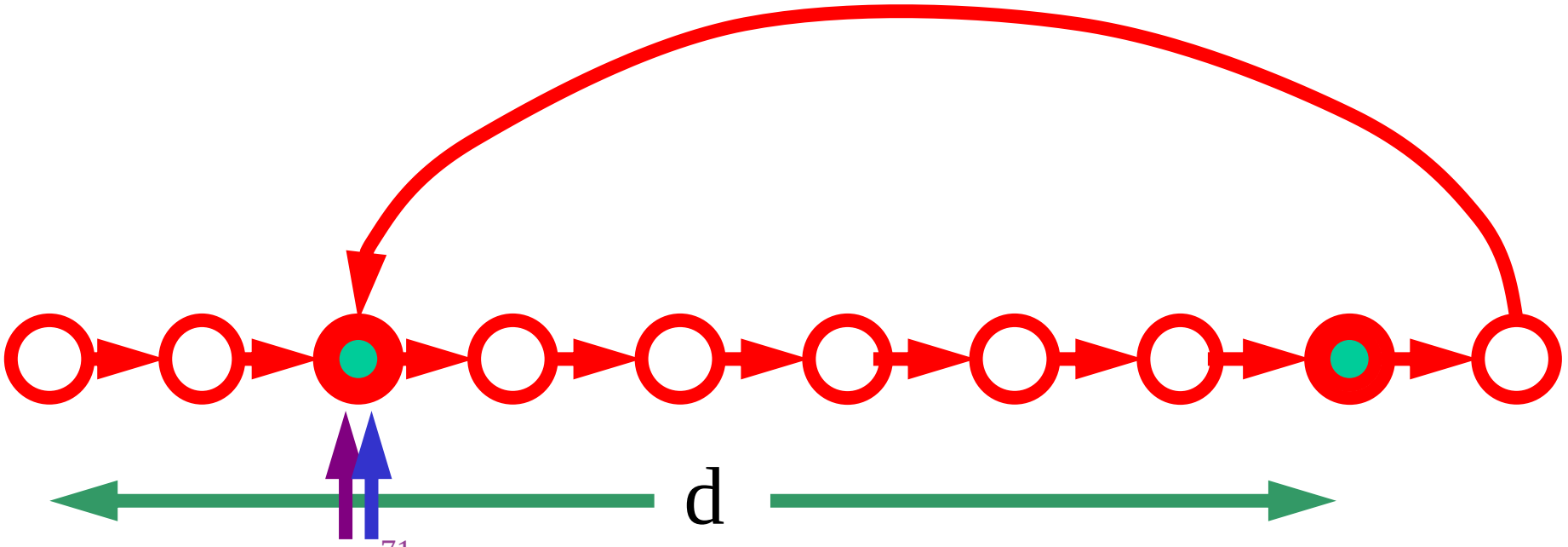
- 所以两个指针再次首先会在环的起点相遇



Why does it work?

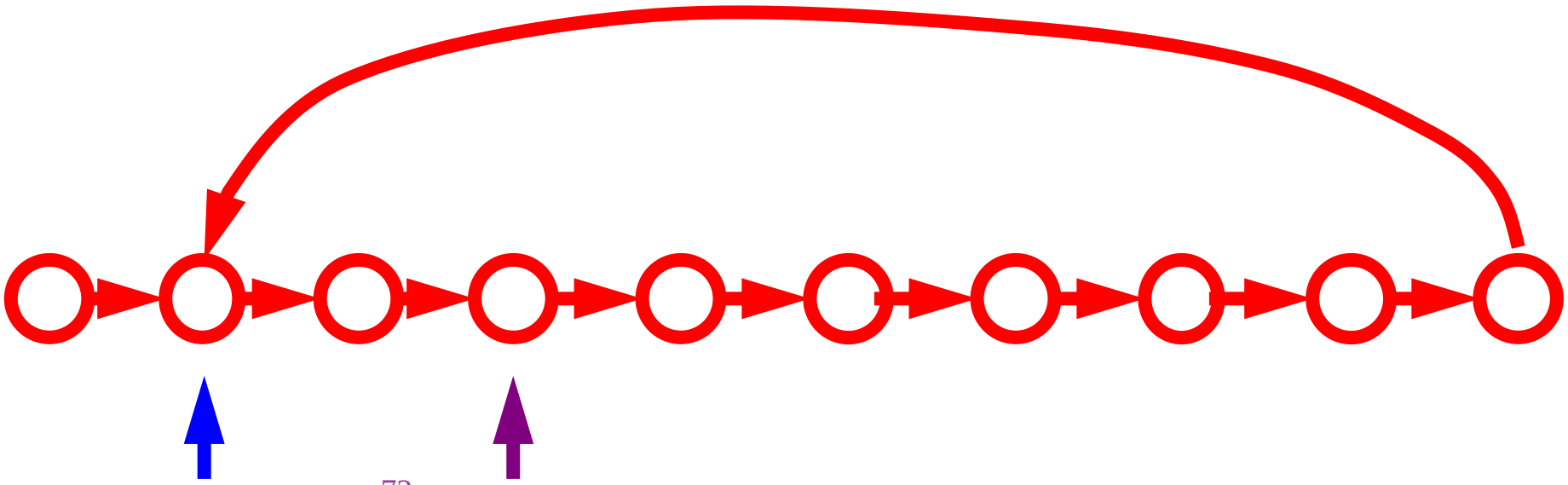
-两个指针再移动d步就到达相同的点

- 所以两个指针再次首先会在环的起点相遇



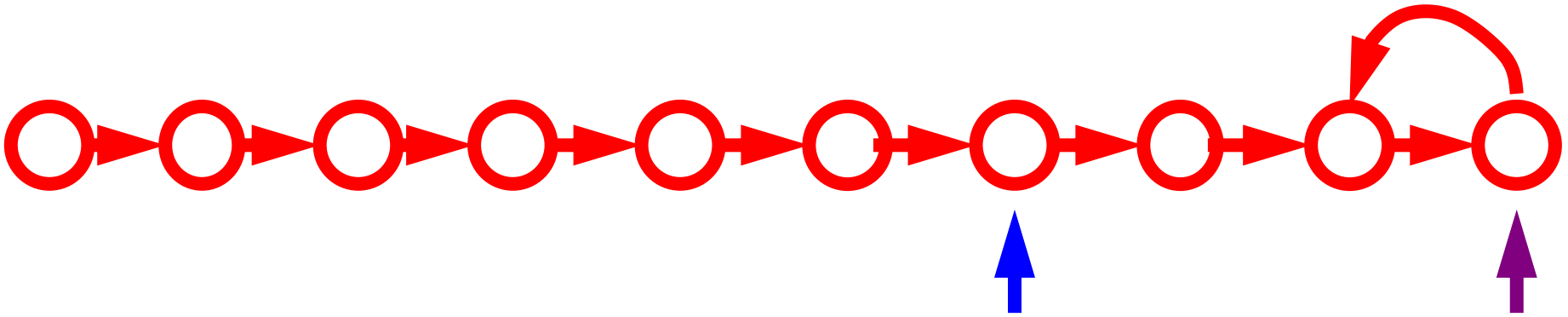
Floyd算法效率

- 设路径有n个节点，当环发生之前的路径(trail)短的时候，Floyd's 算法大约需要 3n~5n 步



Floyd算法效率?

- 当环短的时候，快的指针遍历很多次



其他寻找碰撞的方法

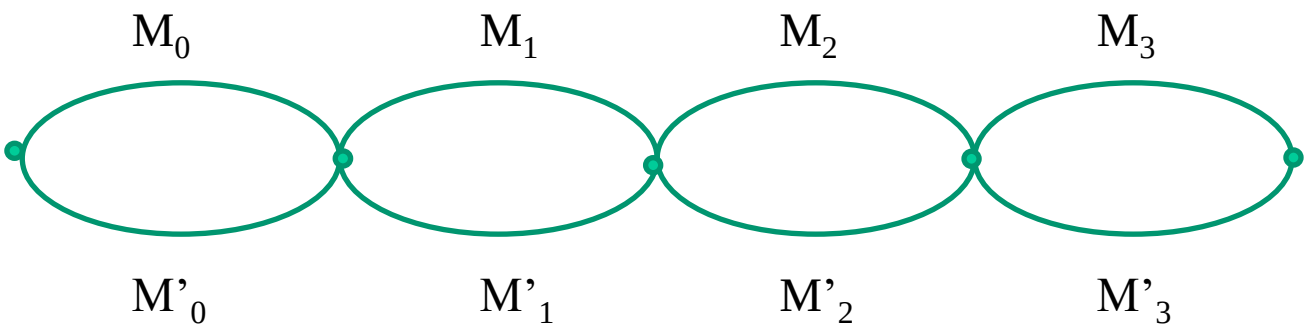
- Nivasch Algorithm(Stack Algorithm, 2004年)

  该算法无论环多长，总在第二次遍历环的时候找到碰撞。

  时间：$1.5\ 2^{n/2}$, 空间$\log(2^{n/2})$

多碰撞攻击

- 多碰撞（k-collision attack）
  - 对于k>=2，寻找 k个不同的消息， 使得 $H(M_1) = H(M_2) = H(M_k)$，寻找一个k-collision 的复杂度为 $2^{(k-1)\cdot n/k}$
  - Joux的多碰撞攻击, k-collision,复杂度为 $\log_2 k \cdot 2^{n/2}$



长消息的第二原像攻击-固定点法

- Dean’s的第二原像攻击
  - D. Dean. Formal aspects of mobile code security. *Princeton University*, 1999.
  - 关键寻找足够多的固定点 $f(y_i, x_i) = y_i$
  - 给定2^l的消息M，第二原像攻击步骤
    - 1. 寻找2t个固定点，存于$L_1=\{y_i\}$
    - 2. 从初始值寻找消息m_{i_0}，使得$z_{i_0}=f(IV, m_{i_0})= y_{j_0} \in L_1$
    - 3. 对于消息M，计算中间链接变量值并存储 $L_2 = \{h_2, h_3, \ldots, h_{|M|_{bl}}\}$

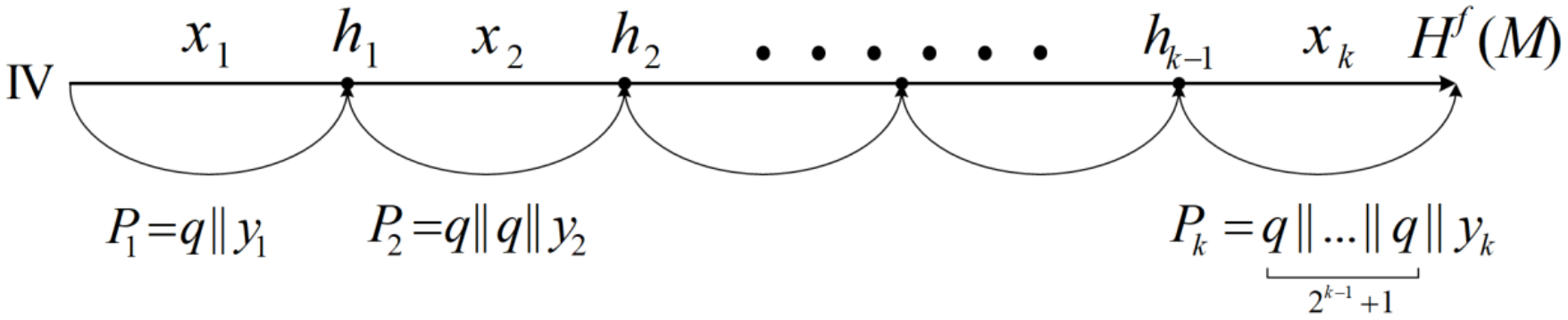
搜索消息m_{i_1},使得$f(y_{j_0}, m_{i_1}) \in L_2$，则M的第二原像M’为：

$$m_{i_0} \| \underbrace{m_{i_1} \| \cdots \| m_{i_1}}_{j-1} \| M_{j+1}$$

  - 复杂度：$2^{n/2} + 2^l$

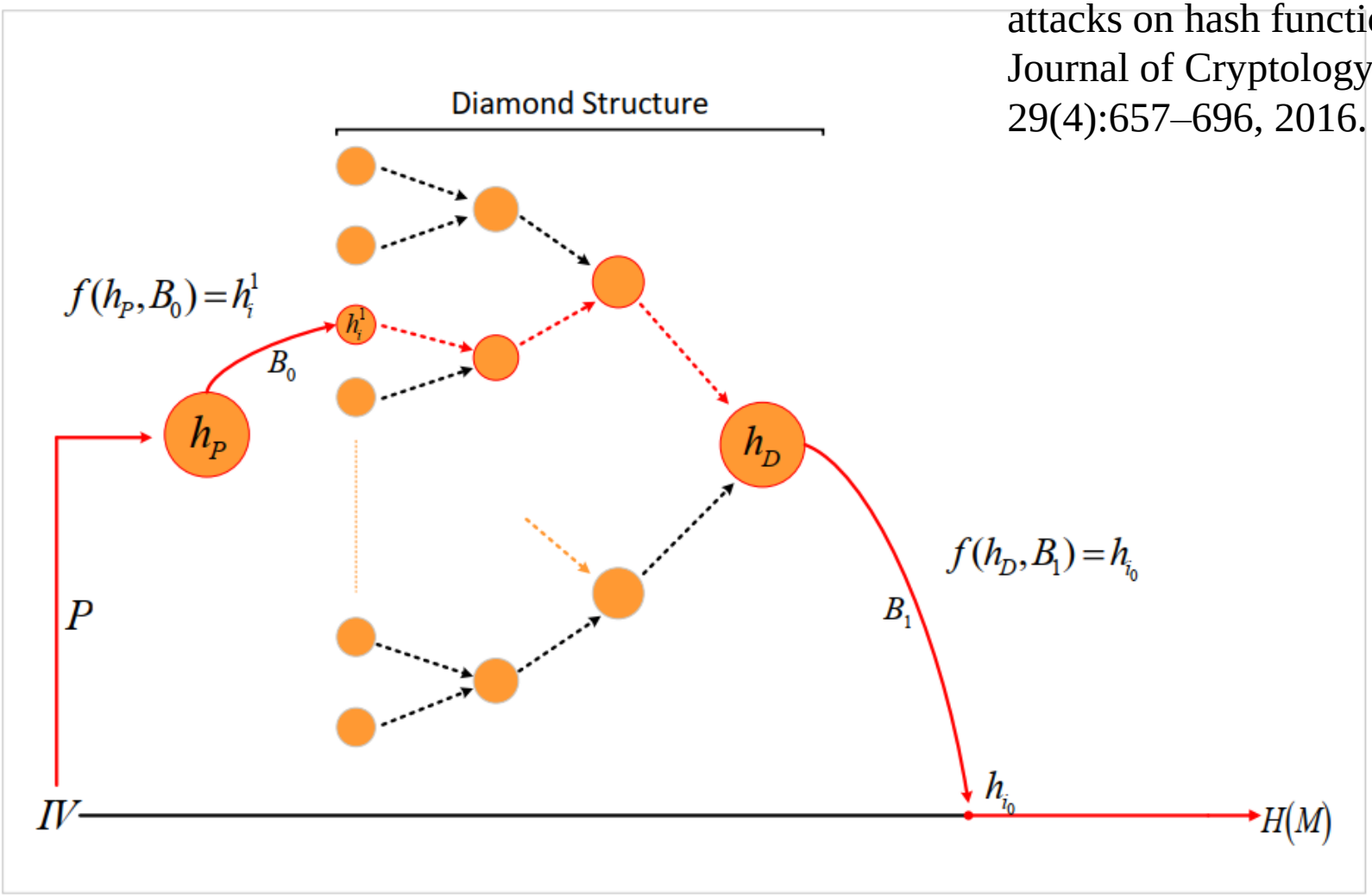
第二原像攻击-Joux方法

- Kelsey和Schneier第二原像攻击
  - J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2^n work. eurocrypt, 2005



第二原像攻击

E. Andreeva, C. Bouillaguet, et al. New second-preimage attacks on hash functions. Journal of Cryptology, 29(4):657–696, 2016.



碰撞攻击

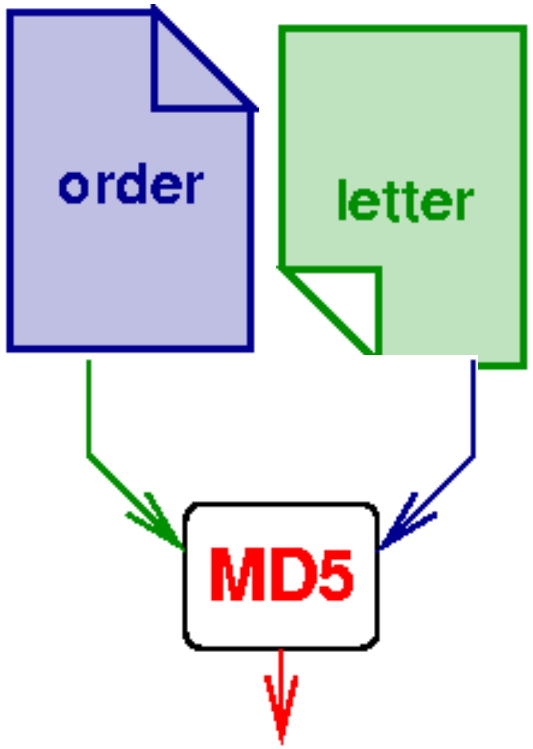
- 模差分分析方法
  - Xiaoyun Wang and Hongbo Yu. How to Break MD5 and Other Hash Functions，Eurocrypt 2005
  - Xiaoyun Wang, Hongbo Yu, Yiqun Lisa Yin: Efficient Collision Search Attacks on SHA-0. CRYPTO 2005
  - Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu. Finding Collisions in the Full SHA-1, Crypt 2005
- Rebound攻击
  - Introduced by Mendel, Rechberger, Schläffer and Thomsen, FSE 2009
  - Cryptanalysis of hash functions with AES-based design
    - Whirlpool, Grøstl, ECHO, JH, and LANE etc.

第四部分

Hash函数的碰撞攻击引发的安全问题

基于MD5随机碰撞的实际攻击

- Attacking Hash Functions by Poisoned Messages
  “The Story of Alice and her Boss“ [Magnus， Lucks， 2005]



a25f7f0b 29ee0b39 68c86073 8533a4b9

攻击原理

适用于具有IF-THEN-ELSE形式的文件格式：PS、PDF、TIFF、Word 97等，文件有冗余

1. 寻找两个随机的字符串R_1和R_2:

X_1=Preamble; put(R1)

X_2=Preamble; put(R2)

使得 MD5（X_1）=MD5（X_2）

2. 显然 MD5(X_1||S)=MD5(X_2||S)，S为任意字符串

攻击原理

The target documents are T_1 and T_2:

$$Y_1 = \overbrace{\text{preamble}; \texttt{put}(R_1);}^{X_1} \underbrace{\texttt{put}(R_1);\ \texttt{if}(=)\ \texttt{then}\ T_1\ \texttt{else}\ T_2;}_{S}$$

$$Y_2 = \underbrace{\text{preamble}; \texttt{put}(R_2);}_{X_2} \overbrace{\texttt{put}(R_1);\ \texttt{if}(=)\ \texttt{then}\ T_1\ \texttt{else}\ T_2;}^{S}$$

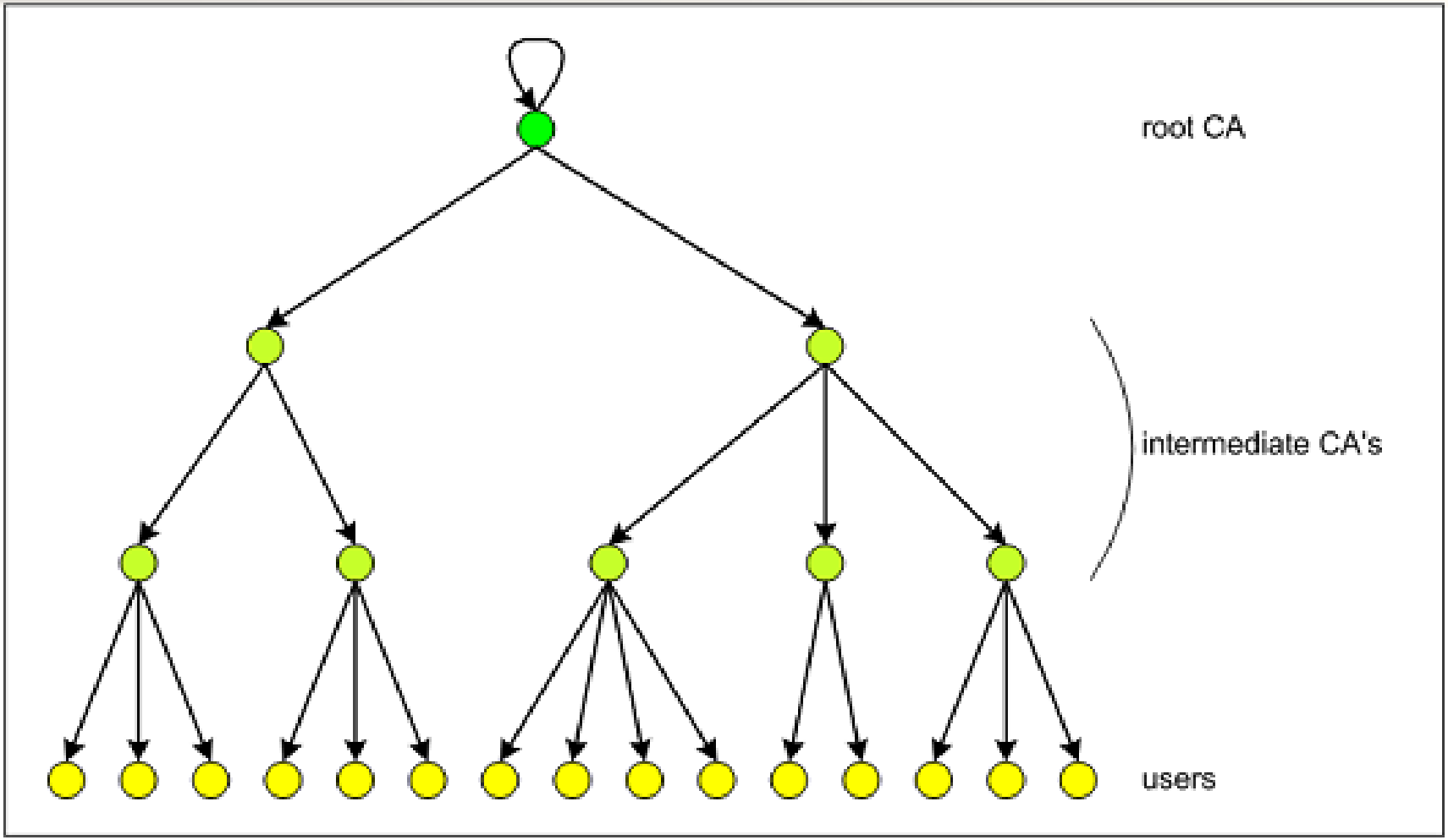
- Viewing Y_1: $R_1 = R_1$, thus T_1 is displayed.
- Viewing Y_2: $R_2 \neq R_1$, thus T_2 is displayed.

基于MD5随机碰撞的实际攻击(2)
--伪造数字证书

X. 509数字证书：目前数字证书普遍使用X.509 v3国际标准

版本
序列号
算法标识 • 算法 • 参数
发布者
有效期 • 起始日期 • 终止日期
主体
主体的公开密钥 • 算法 • 参数 • 公开密钥
签名

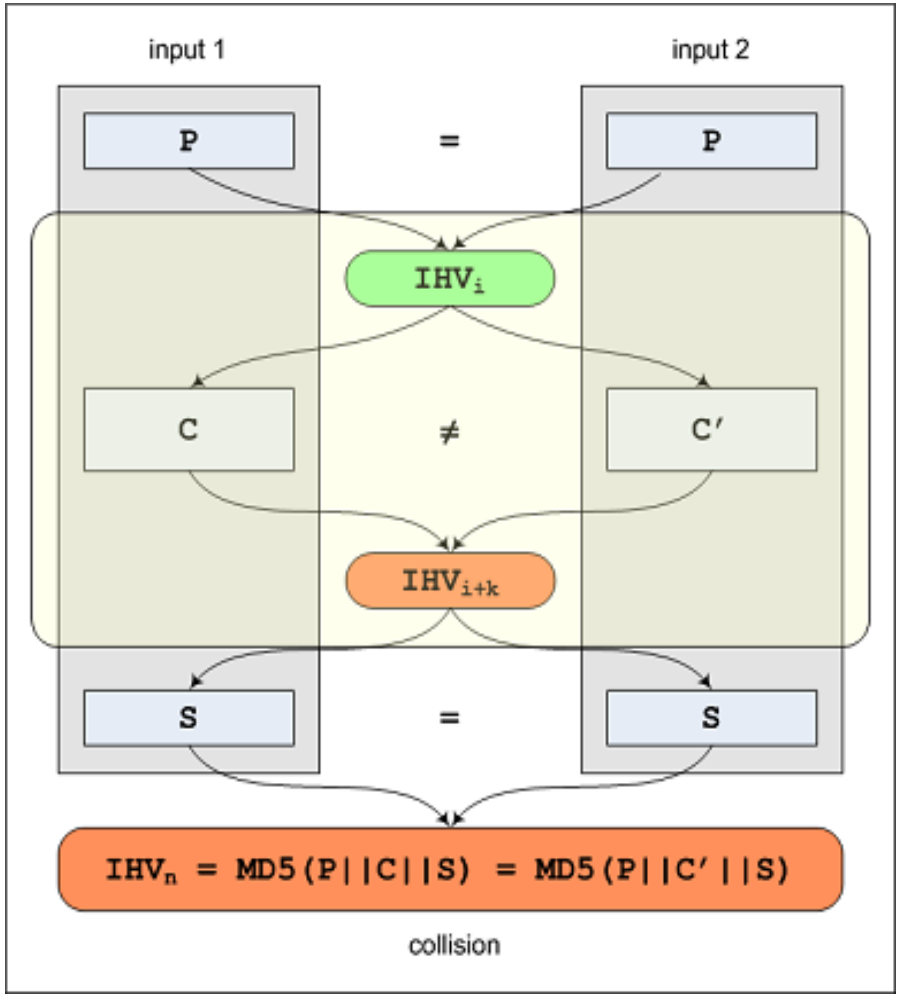
CA认证

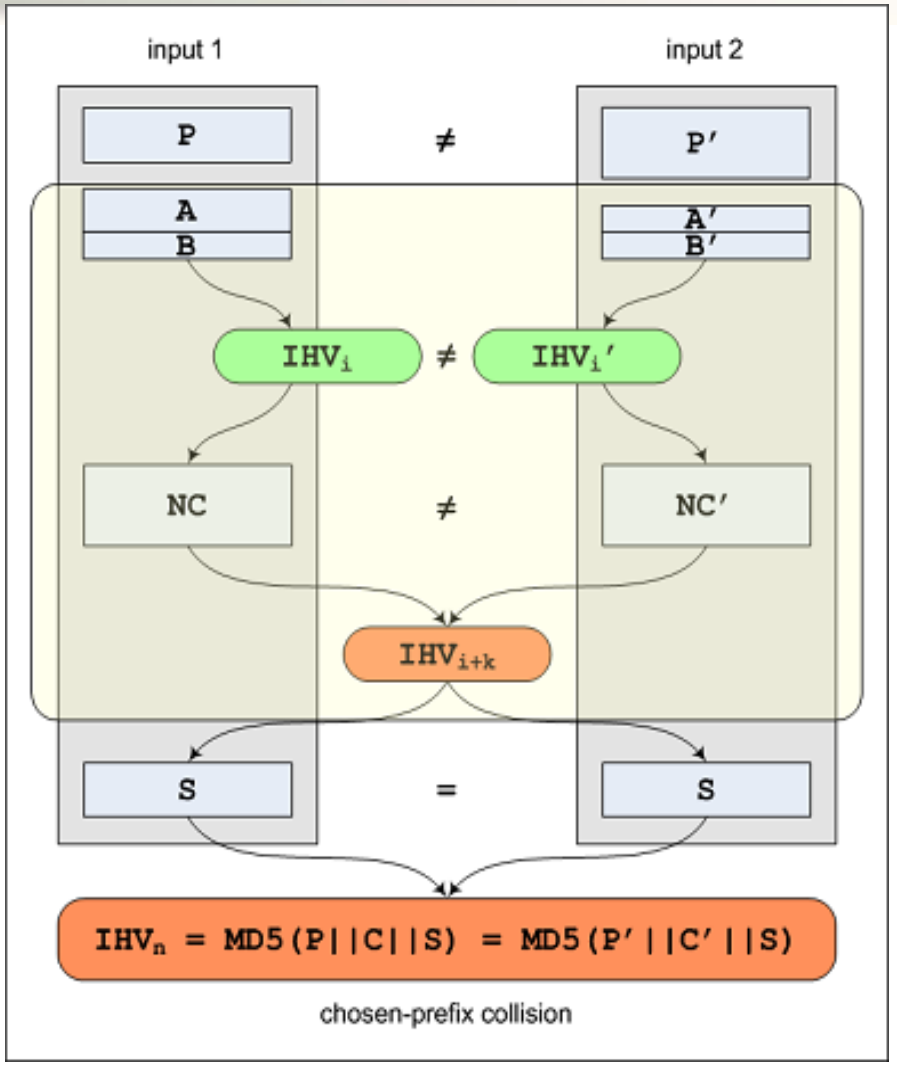


伪造X.509 数字证书

- Colliding X.509 Certificates [A.Lenstra ,X.Wang and B.Weger 2005]
  - 不同的公钥，相同的身份
- Target Collisions for MD5 and Colliding X.509 Certificates for Different Identities，[M. Stevens, A. Lenstra, and B.Weger 2007]
  - 不同的公钥，不同的身份
- Chosen-prefix Collisions for MD5 and Applications, [M. Stevens, A. Lenstra, and B.Weger 2009]
  - 伪造真实网站的数字证书
  - 2009年欧密会最佳论文奖
  - 2009年十大黑客技术之首

两种MD5碰撞


input 1
input 2
P
=
P
IHV$_i$
C
≠
C′
IHV$_{i+k}$
S
=
S
IHV$_n$ = MD5(P||C||S) = MD5(P||C′||S)
collision


input 1
input 2
P
≠
P′
A
B
A′
B′
IHV$_i$
≠
IHV$_i$′
NC
≠
NC′
IHV$_{i+k}$
S
=
S
IHV$_n$ = MD5(P||C||S) = MD5(P′||C′||S)
chosen-prefix collision

伪造X.509 数字证书2：不同的公钥，不同身份[Lenstra etc. 2007]

set by the CA			
	serial number	chosen prefix (difference)	serial number
	validity period		validity period
	real cert domain name		rogue cert domain name
	real cert RSA key	collision bits (computed)	real cert RSA key
	X.509 extensions	identical bytes (copied from real cert)	X.509 extensions
	signature		signature

2008年伪造实际的X.509证书

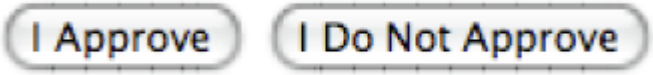
- 2005年证书：相同的身份
- 2007年证书：有效期和序列号没办法控制，8192比特RSA模太长
- 需要解决的问题
  - 寻找使用MD5的数字证书
  - 如何获得序列号和有效期？

使用MD5的数字证书

- 2008年Stevens等搜集了30,000个网站的证书
  - 9000个使用MD5和RSA进行签名
  - 97%由RapidSSL签发
- 2008年仍然使用MD5的CAs
  - RapidSSL
  - FreeSSL
  - TrustCenter
  - RSA Data Security
  - Thawte
  - Verisign.co.jp

预测有效期和序列号

- RapidSSL自动生成证书，每个生成过程只有6秒，证书签发完成后，有效[illegible]



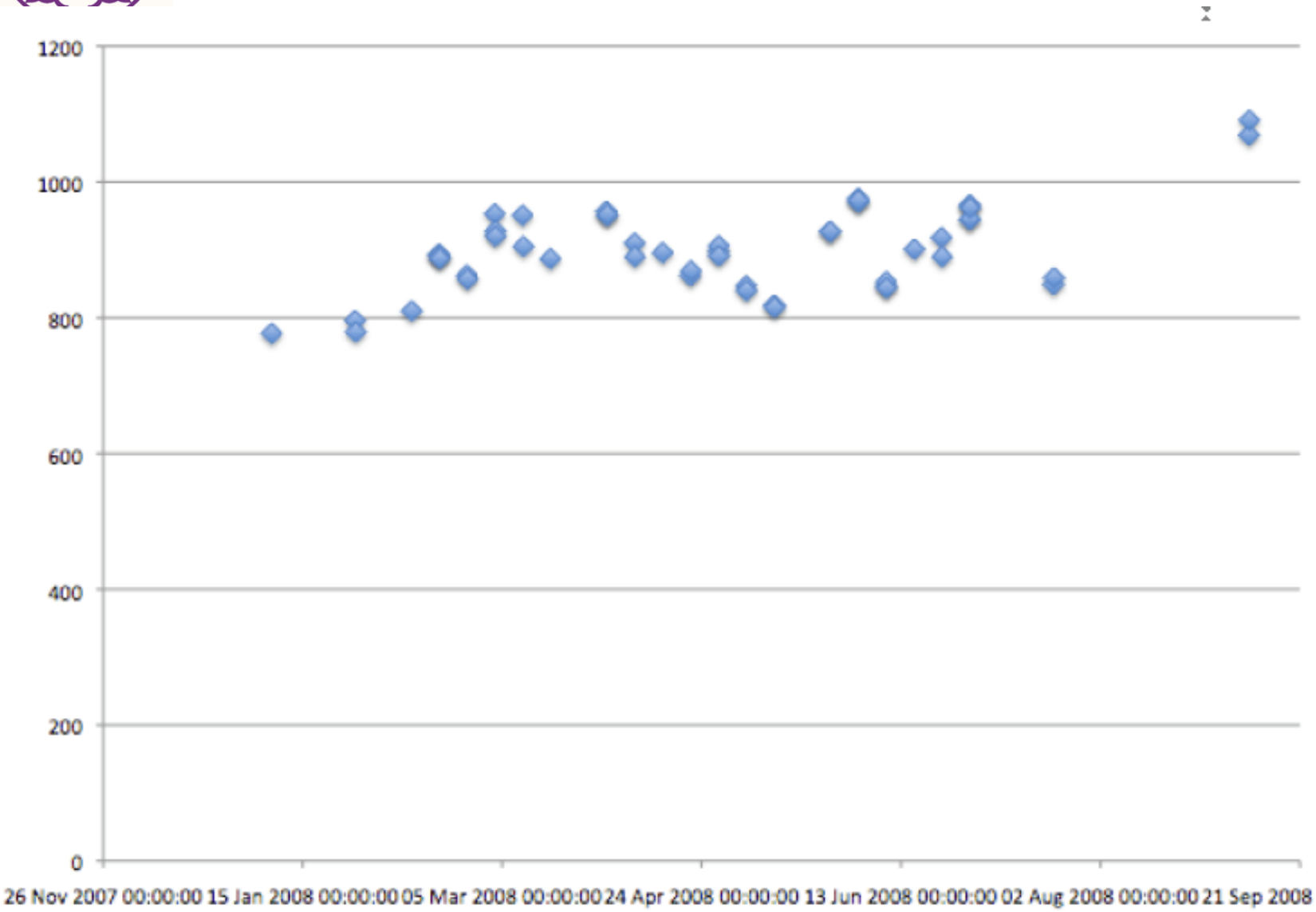
- 序列号：

Nov	3	07:42:02	2008	GMT	643004
Nov	3	07:43:02	2008	GMT	643005
Nov	3	07:44:08	2008	GMT	643006
Nov	3	07:45:02	2008	GMT	643007
Nov	3	07:46:02	2008	GMT	643008
Nov	3	07:47:03	2008	GMT	643009
Nov	3	07:48:02	2008	GMT	643010
Nov	3	07:49:02	2008	GMT	643011
Nov	3	07:50:02	2008	GMT	643012
Nov	3	07:51:12	2008	GMT	643013
Nov	3	07:51:29	2008	GMT	643014
Nov	3	07:52:02	2008	GMT	?

预测序列号

- 每购买一个证书，序列号增加1
- 费用
  - 一个新的证书$69
  - 更新一个证书$45
  - 一个证书可以免费签发20次
  - 平均增加一个序列号需要$2.25

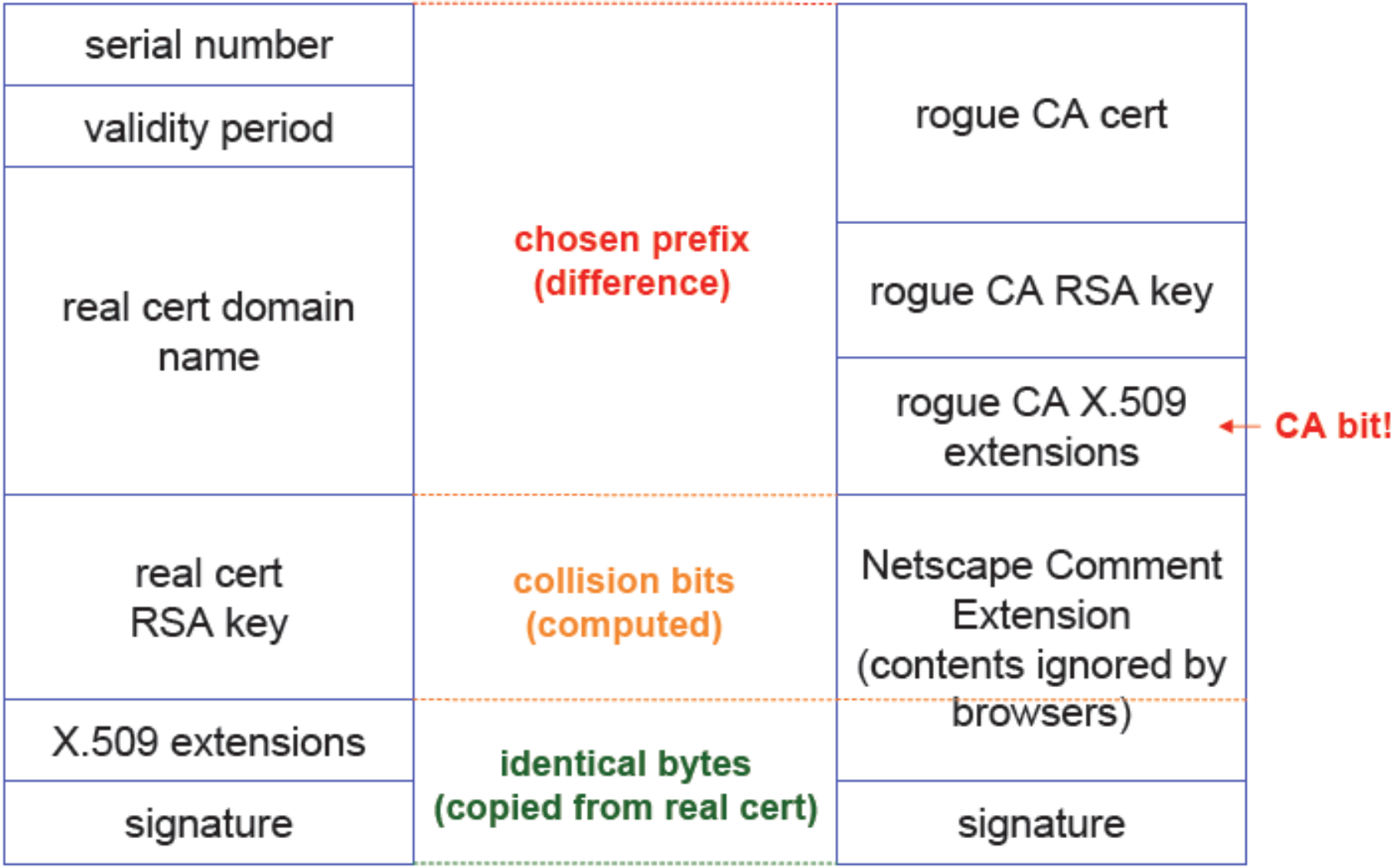
每周签发证书数量


1200
1000
800
600
400
200
0
26 Nov 2007 00:00:00
15 Jan 2008 00:00:00
05 Mar 2008 00:00:00
24 Apr 2008 00:00:00
13 Jun 2008 00:00:00
02 Aug 2008 00:00:00
21 Sep 2008

预测序列号

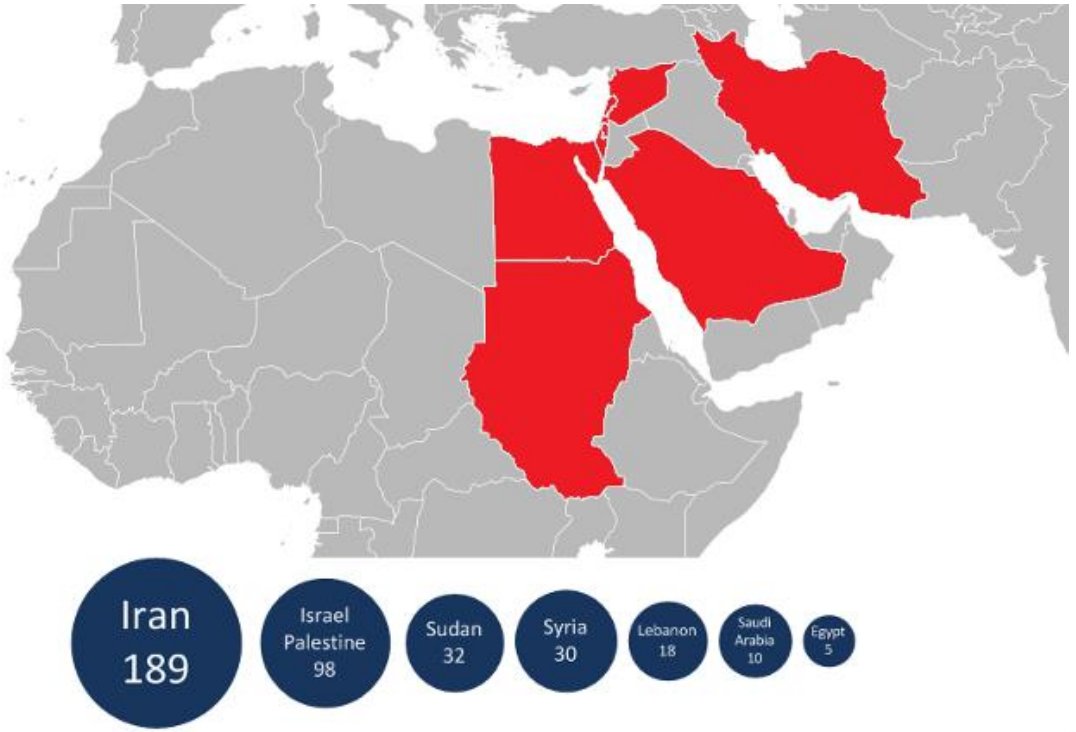
- 获得周5晚上 0:00的序列号S
- 预测周日T时刻序列号是S+1000
- 产生MD5碰撞的两个证书信息C1和C2
- 在接近T时刻的时候买足够的证书，使得证书的序列号达到S+999
- 在T时候将C1发送给RapidSSL进行签发
- 将签名复制给C2，得到一个伪造的合法的证书

伪造中间结点CA


serial number
validity period
real cert domain name
real cert RSA key
X.509 extensions
signature
chosen prefix (difference)
collision bits (computed)
identical bytes (copied from real cert)
rogue CA cert
rogue CA RSA key
rogue CA X.509 extensions
CA bit!
Netscape Comment Extension (contents ignored by browsers)
signature

火焰病毒(Worm.Win32.Flame)

- **2012年5月，被俄罗斯安全专家发现**
- **针对伊朗核武器**

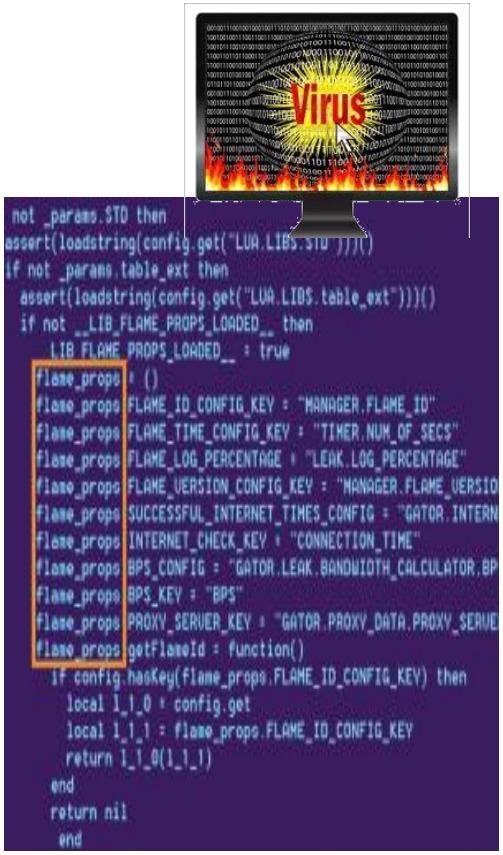


火焰病毒(Worm.Win32.Flame)

- 攻击范围

截获键盘输入
记录音频对话
获取截屏画面
监测网络流量

传送



- 搜集数据任务完成，自行毁灭，不留踪迹

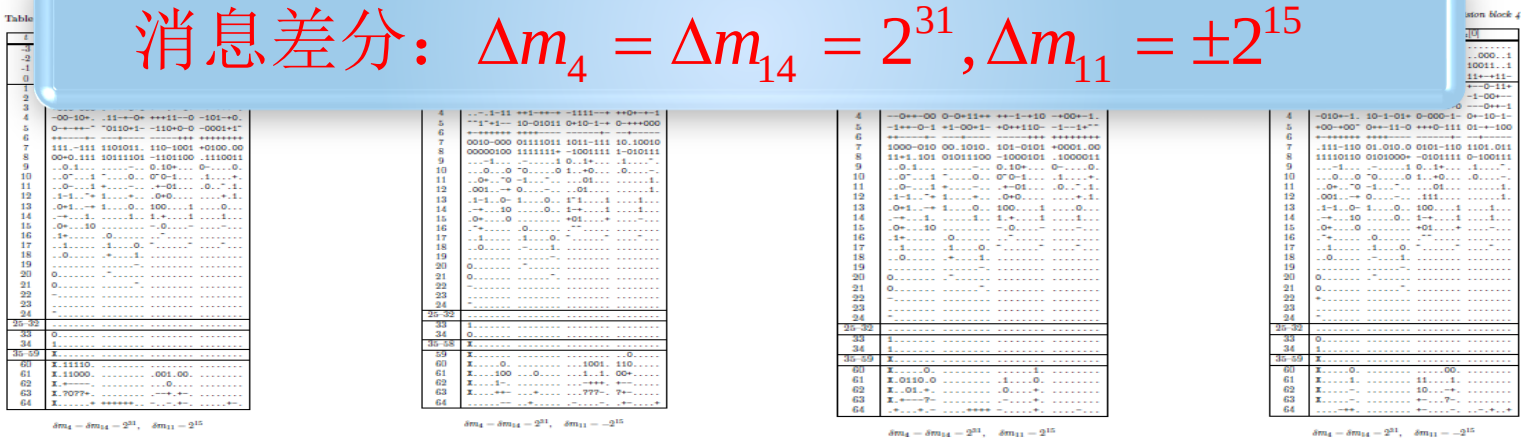
火焰病毒中的关键密码技术

微软终端服务器
许可服务证书签名的代码 (MD5)

伪造证书
签名的病毒代码 (MD5)

- **“火焰”采用选择前缀MD5碰撞攻击的新变体伪造证书，主体路线使用王小云等在Eurocrypt 05给出的路线**

消息差分：$\Delta m_4 = \Delta m_{14} = 2^{31}, \Delta m_{11} = \pm 2^{15}$



Block1 Block2 Block3 Block4

谢谢！