Cryptography Overview (Part 2)

jedimaestro@asu.edu

This lecture...

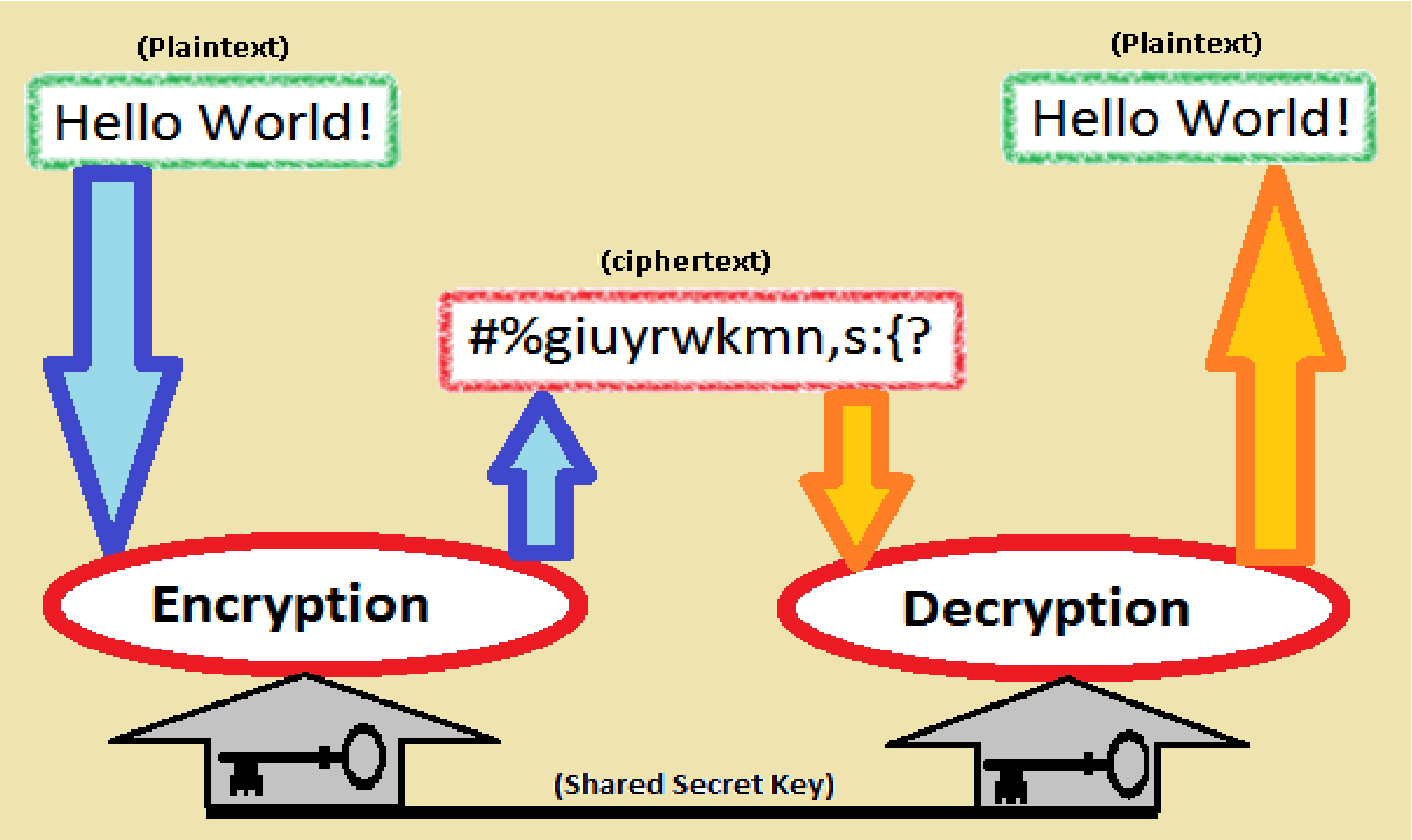
- What can’t symmetric crypto do?
- Asymmetric crypto introduction (review?)
- Intro to secure hash functions and message authentication

This should be review if you took, *e.g.*, CSE 365. If you need more review:

https://www.youtube.com/watch?v=KqqOXndnvic
https://www.youtube.com/watch?v=SkJcmCaHqS0
https://www.youtube.com/watch?v=QgHnr8-h0xI
https://www.youtube.com/watch?v=-dsKYoqwjT0

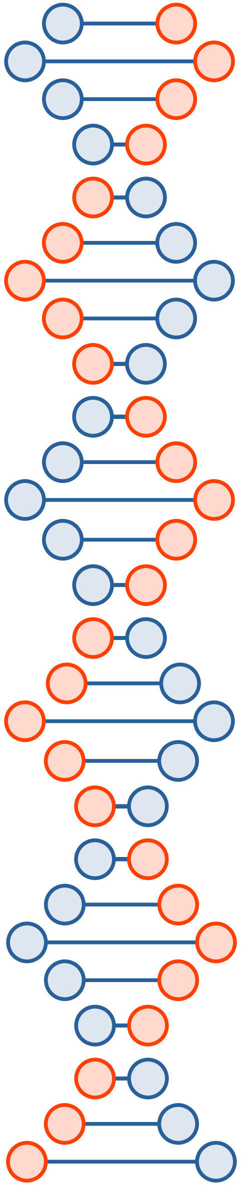
Symmetric Crypto

- Confidentiality
- Integrity
- Authentication
- ~~Non-repudiation~~
- ~~A way to distribute the shared secret keys~~

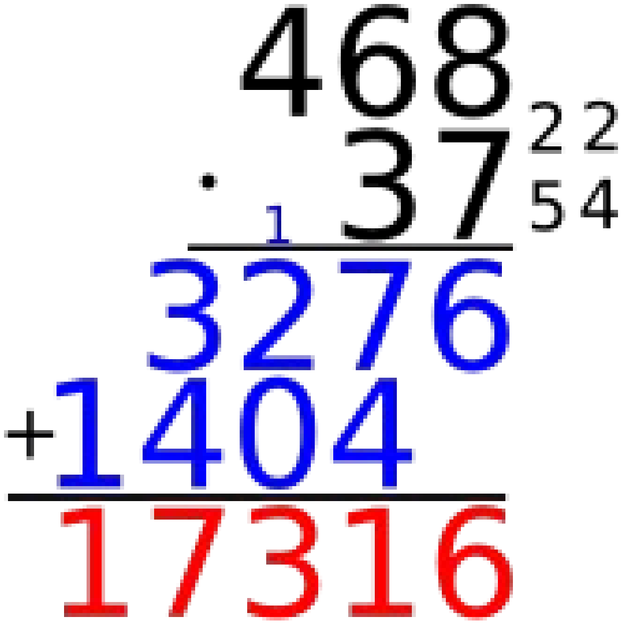


Source: Wikipedia

How computers handle big numbers…



Multiplication is polynomial time in number of digits (O(n^2) or O(n log n))



Modular exponentiation

$153^{189} \pmod{251}$

Naive way: multiply 153 times itself 189 times.
Won't work for, *e.g.*, 2048-bit numbers,
especially for the exponent

Better way (all mod 251)

$153^0 = 1$

$153^1 = 153$

$153^2 = 66$

$153^4 = 89$

$153^8 = 140$

$153^{16} = 22$

$153^{32} = 233$

$153^{64} = 73$

$153^{128} = 58$

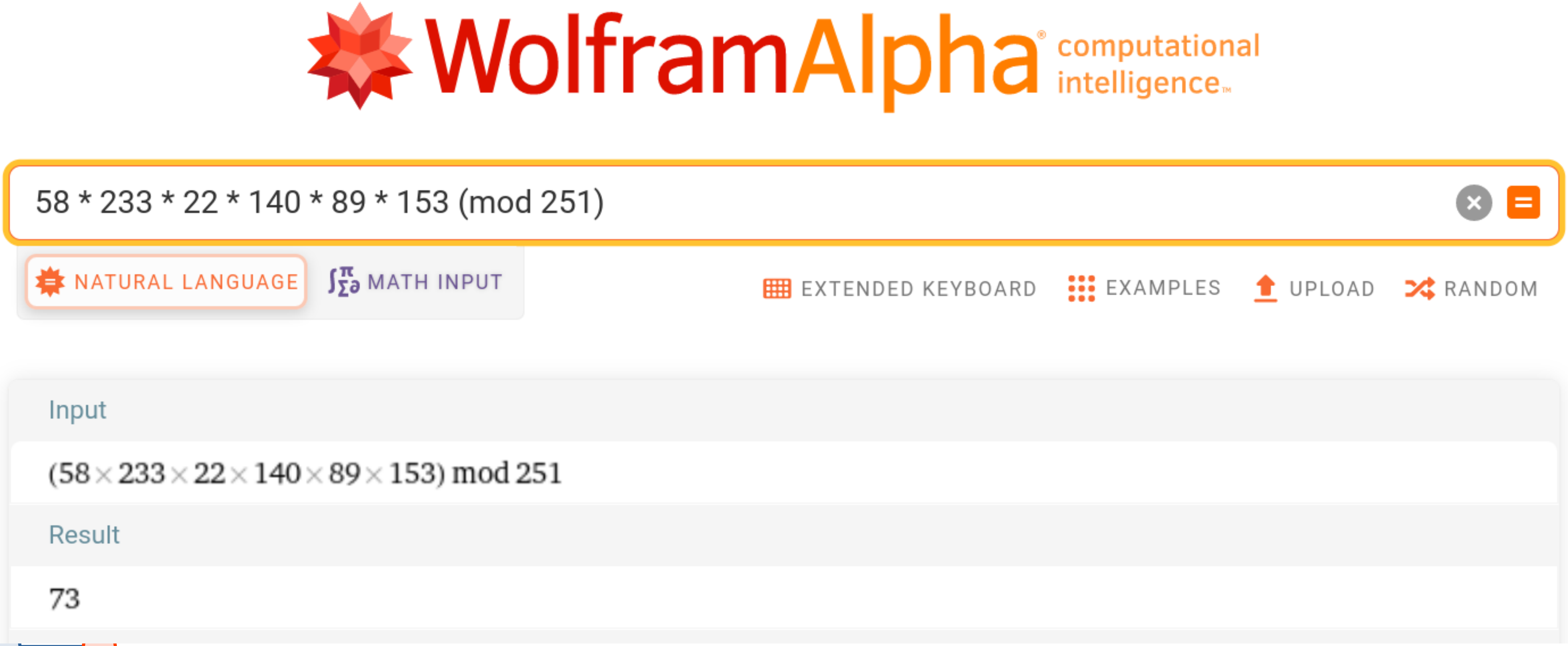
Better way

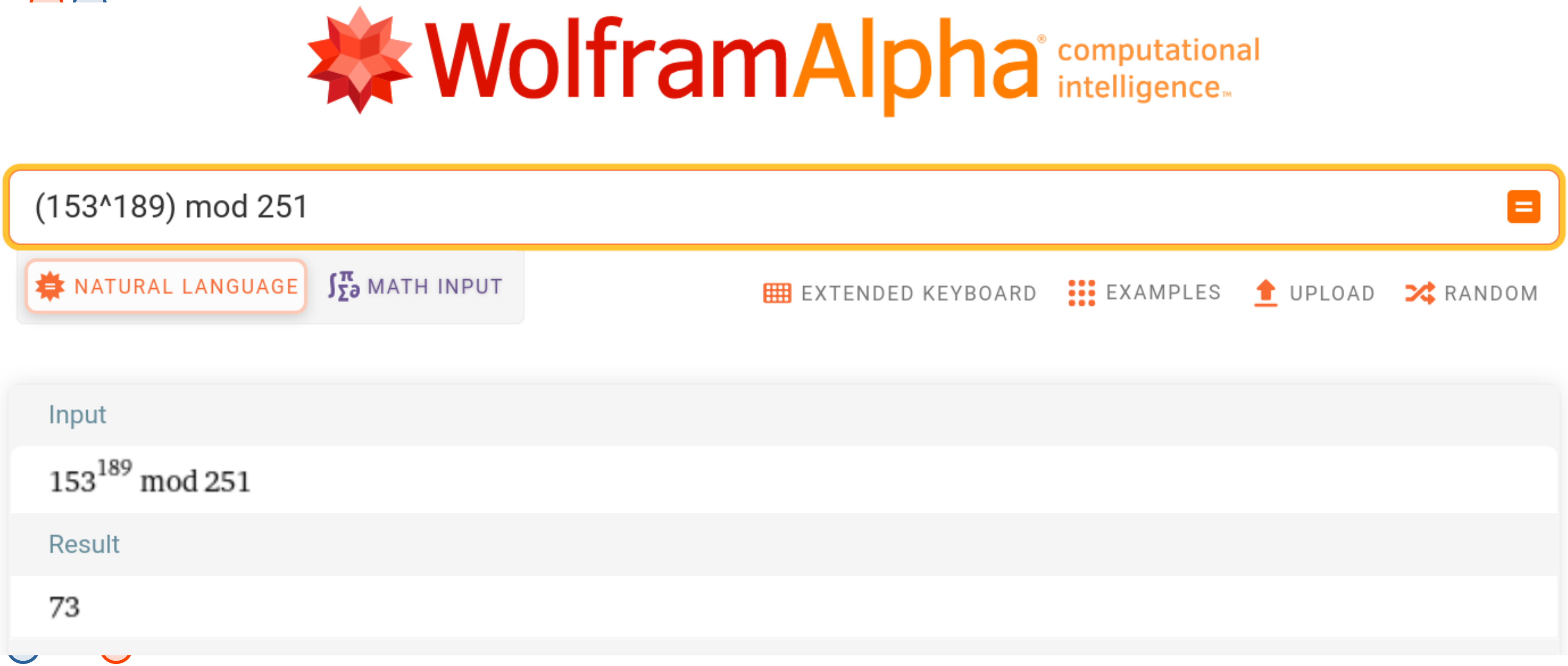
- 189 in binary is 0b10111101
- $189 = 1*2^7 + 0*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0$
- $153^{189} \pmod{251} = 153^{(128+0+32+16+8+4+0+1)} \pmod{251}$

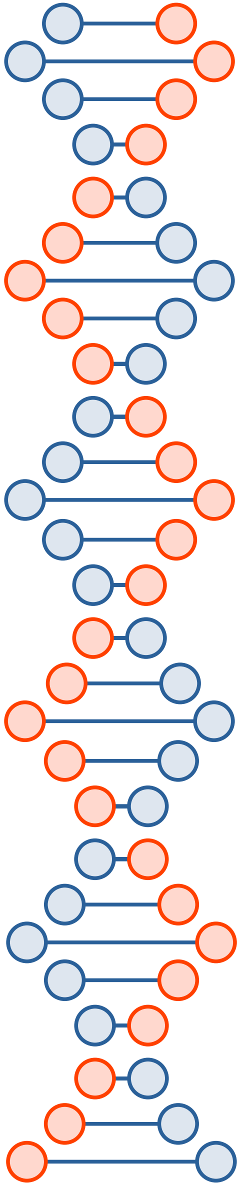
$= 153^{128} * 153^{32} * 153^{16} * 153^{8} * 153^{4} * 153^{1} \pmod{251}$

$= 58 * 233 * 22 * 140 * 89 * 153 \pmod{251}$

$= 73$


WolframAlpha computational intelligence.
58 * 233 * 22 * 140 * 89 * 153 (mod 251)
NATURAL LANGUAGE
MATH INPUT
EXTENDED KEYBOARD
EXAMPLES
UPLOAD
RANDOM
Input
(58 × 233 × 22 × 140 × 89 × 153) mod 251
Result
73


WolframAlpha computational intelligence.
(153^189) mod 251
NATURAL LANGUAGE
MATH INPUT
EXTENDED KEYBOARD
EXAMPLES
UPLOAD
RANDOM
Input
153^189 mod 251
Result
73



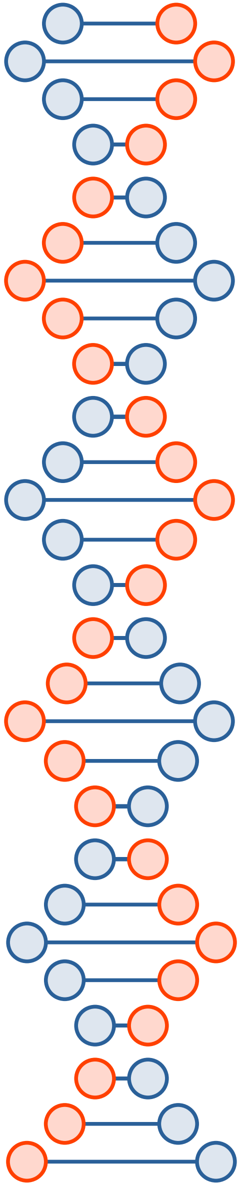
$$153^{189} = 73 \pmod{251}$$
$$189 = \log_{153} 73 \pmod{251}$$

$$153^{???} = 73 \pmod{251}$$

$$??? = \log_{153} 73 \pmod{251}$$

This is called the discrete logarithm, and there is no known algorithm for solving it in the general case that is polynomial in the number of digits.

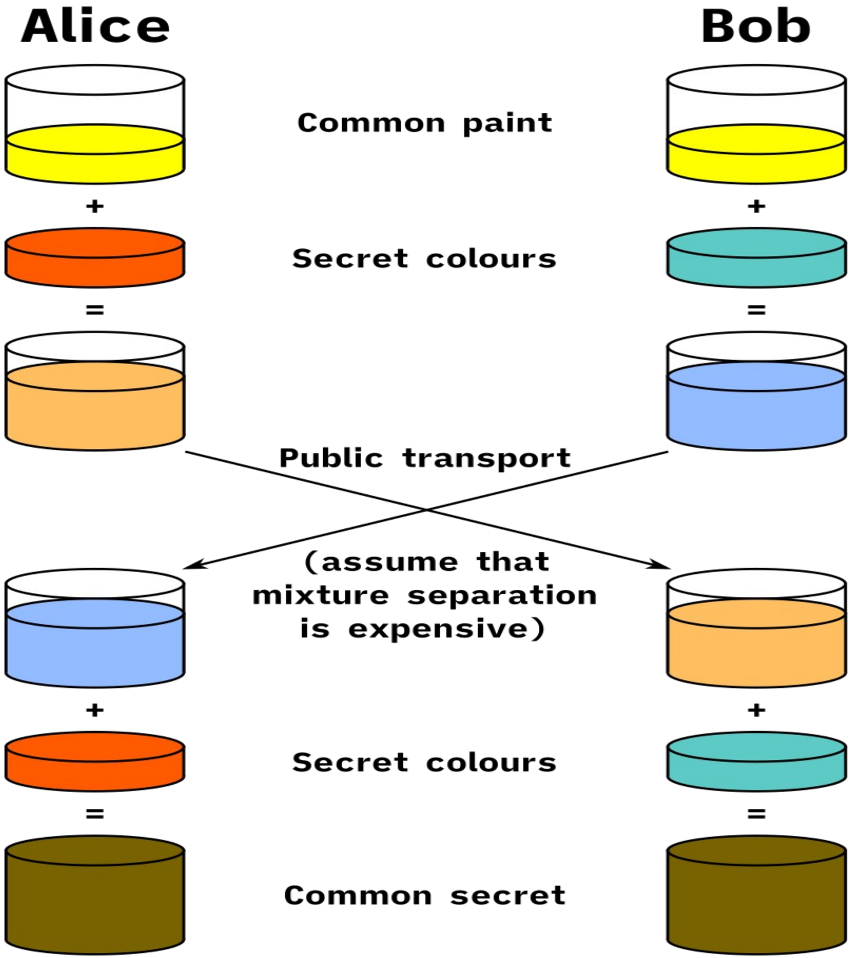
$$153^{189} = 73 \pmod{251}$$
$$153^{64} = 73 \pmod{251}$$



$$153^{189} \equiv 73 \pmod{251}$$
$$153^{64} \equiv 73 \pmod{251}$$

$$153^{189} \equiv 153^{64} \equiv 73 \pmod{251}$$

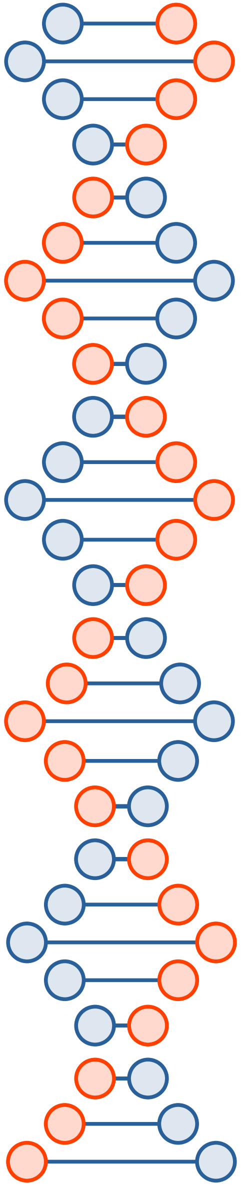
Diffie-Hellman (1976)...



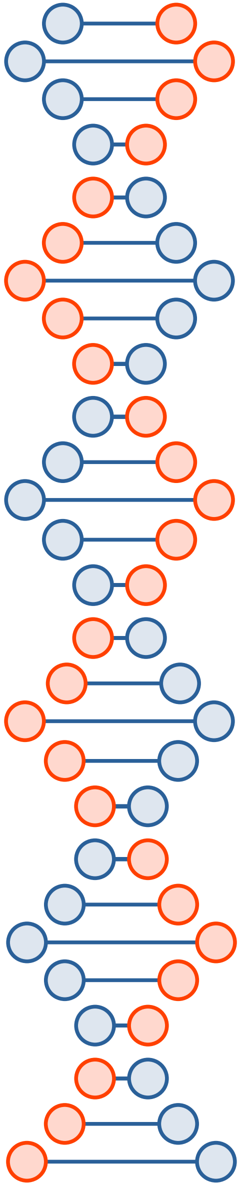
https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange#/media/File:Diffie-Hellman_Key_Exchange.svg

Diffie-Hellman

Alice		Bob		Eve	
Known	**Unknown**	**Known**	**Unknown**	**Known**	**Unknown**
$p = 23$		$p = 23$		$p = 23$	
$g = 5$		$g = 5$		$g = 5$	
$\mathbf{a} = \mathbf{6}$	b	$b = \mathbf{15}$	a		a, b
$A = 5^{a}$ mod 23		$B = 5^{b}$ mod 23			
$A = 5^{6}$ mod 23 = 8		$B = 5^{15}$ mod 23 = 19			
$\mathbf{B} = \mathbf{19}$		$\mathbf{A} = \mathbf{8}$		$A = 8$, $B = 19$	
$s = B^{a}$ mod 23		$s = A^{b}$ mod 23			
$s = 19^{6}$ mod 23 = **2**		$s = 8^{15}$ mod 23 = **2**			s



In the food coloring or paint demos, it is assumed that mixing colors is cheap, but *un-mixing* them is prohibitively expensive.



Modular arithmetic

$$5 + 7 = 2 \pmod{10}$$

$$7^2 = 9 \pmod{10}$$

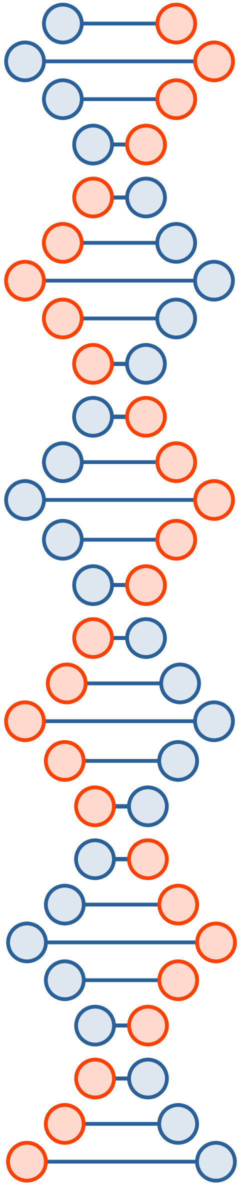
$$8 + 8 = 6 \pmod{10}$$

Modular arithmetic

$$8 + 9 = ?\ (\text{mod } 10)$$

$$4^3 = ?\ (\text{mod } 10)$$

$$1 + 1 = ?\ (\text{mod } 10)$$



Modular arithmetic

$$8 + 9 = 7 \pmod{10}$$

$$4^3 = 4 \pmod{10}$$

$$1 + 1 = 2 \pmod{10}$$

RSA (1977)

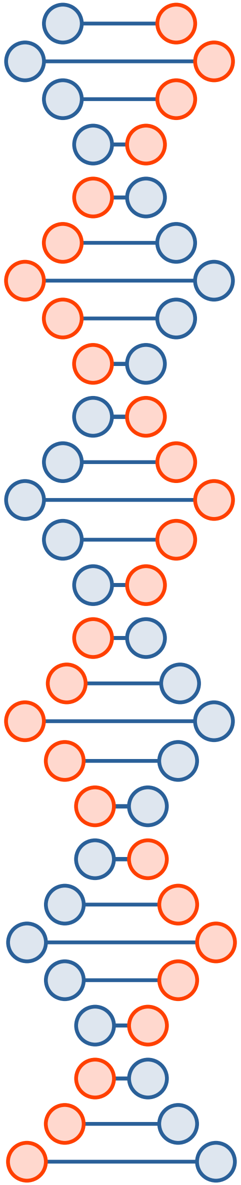
Encryption:

$$c \equiv m^e \bmod n$$

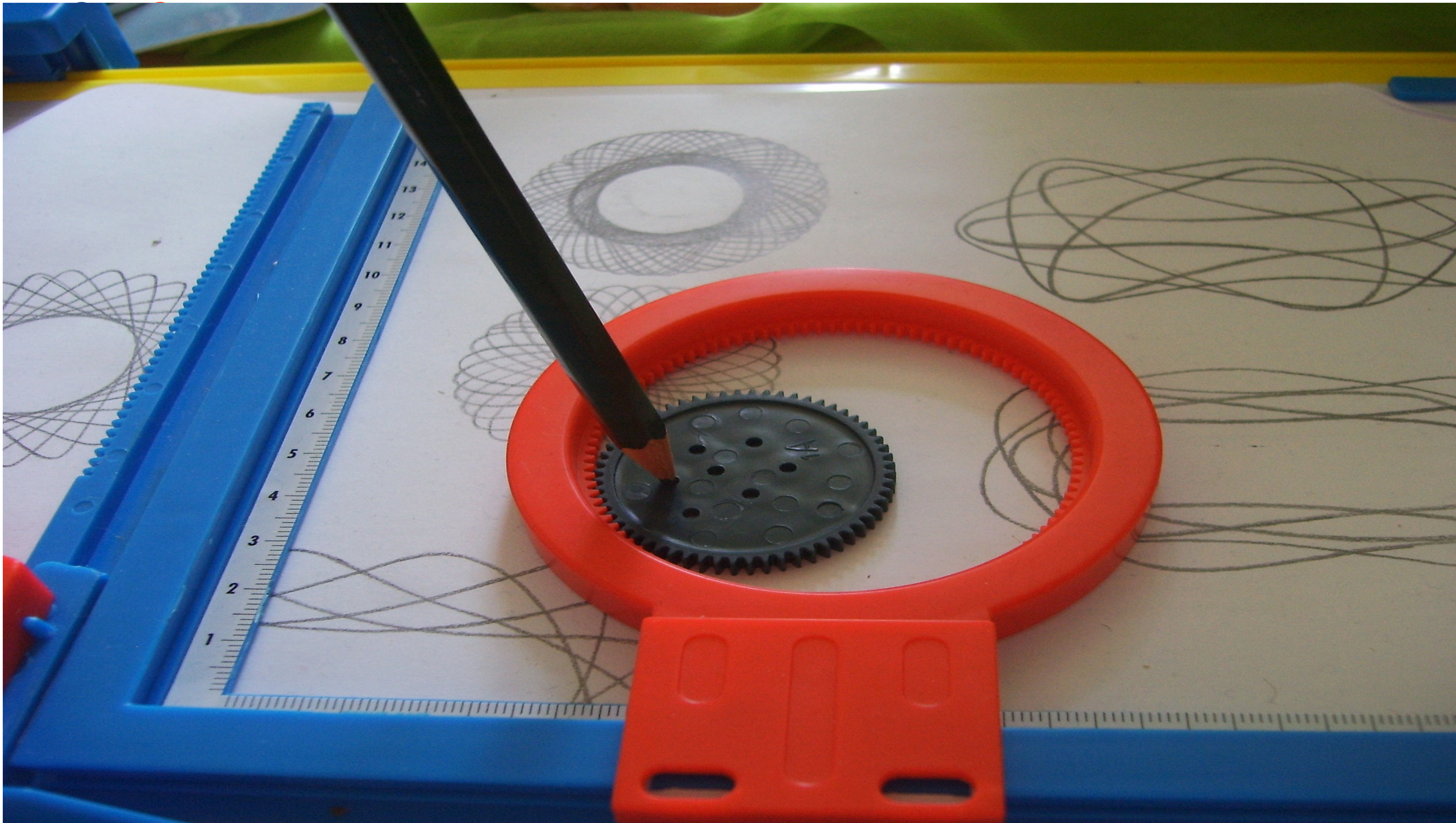
Decryption:

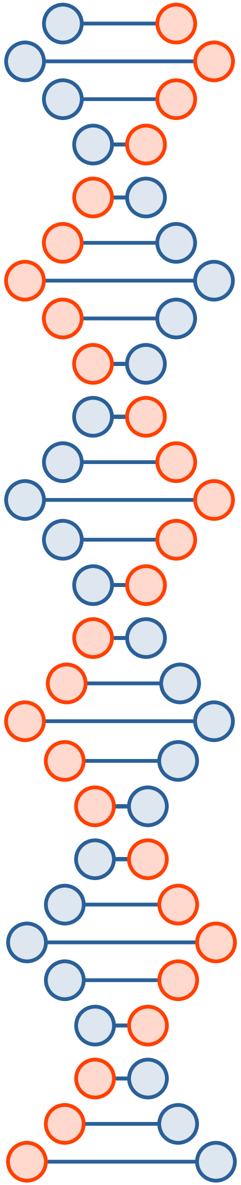
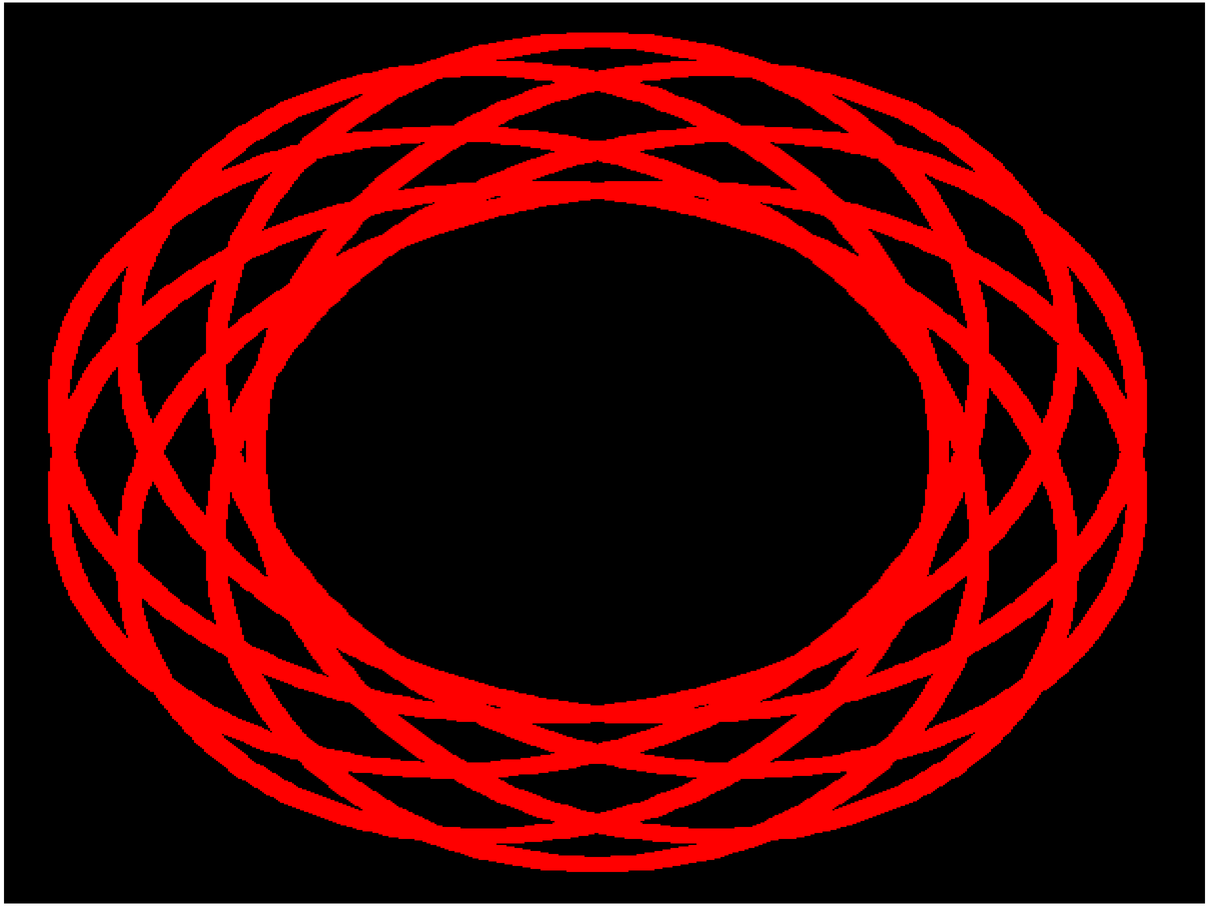
$$c^d \equiv (m^e)^d \bmod n$$

RSA provides encryption, authentication, and non-repudiation



A very loose analogy to ring theory...


1
2
3
4
5
6
7
8
9
10
11
12
13
14

RSA

- Security is based on the hardness of integer factorization

n = pq

- p and q are primes, suppose p = 61, q = 53
- n = 3233
- Euler's totient counts the positive integers up to n that are relatively prime to n
- totient(n) = lcm(p – 1, q – 1) = 780
  - 52,104,156,208,260,312,364,416,468,520,572,624,676,728,780
  - 60,120,180,240,300,360,420,480,540,600,660,720,780
- Choose 1 < e < 780 coprime to 780, e.g., e = 17
- d is the modular multiplicative inverse of e, d = 413
- 413 * 17 mod 780 = 1

- Public key is (n = 3233, e = 17)
- Private key is (n = 3233, d = 413)
- Encryption: $c(m = 65) = 65^{17} \bmod 3233 = 2790$
- Decryption: $m = 2790^{413} \bmod 3233 = 65$
- Could also do...
  - Signature: $s = 100^{413} \bmod 3233 = 1391$
  - Verification: $100 = 1391^{17} \bmod 3233$
- Fast modular exponentiation is the trick
- Using RSA for key exchange or encryption is often a red flag, more commonly used for signatures

jedi@route66: ~

```
jedi@route66:~$ python3
Python 3.8.2 (default, Jul 16 2020, 14:00:26)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> for i in range (52, 781, 52):
...     for j in range (60, 781, 60):
...         if (i == j):
...             print(i)
...
780
>>> print((413 * 17) % 780)
1
>>> print(pow(2790, 413, 3233))
65
>>> print(pow(65, 17, 3233))
2790
>>> print(pow(100, 413, 3233))
1391
>>> print(pow(1391, 17, 3233))
100
>>> 
```

jedi@route66: ~

```
1
>>> print(pow(2790, 413, 3233))
65
>>> print(pow(65, 17, 3233))
2790
>>> print(pow(100, 413, 3233))
1391
>>> print(pow(1391, 17, 3233))
100
>>> print(pow(7, 17, 3233))
2369
>>> print((2369*2790) % 3233)
1258
>>> print(pow(1258, 413, 3233))
455
>>> print(7*65)
455
>>> print("{0:b}".format(78913))
10011010001000001
>>> print("{0:b}".format(78913*32))
1001101000100000100000
>>> print("{0:b}".format(78913<<5))
1001101000100000100000
>>> 
```

Cryptographic hash functions...

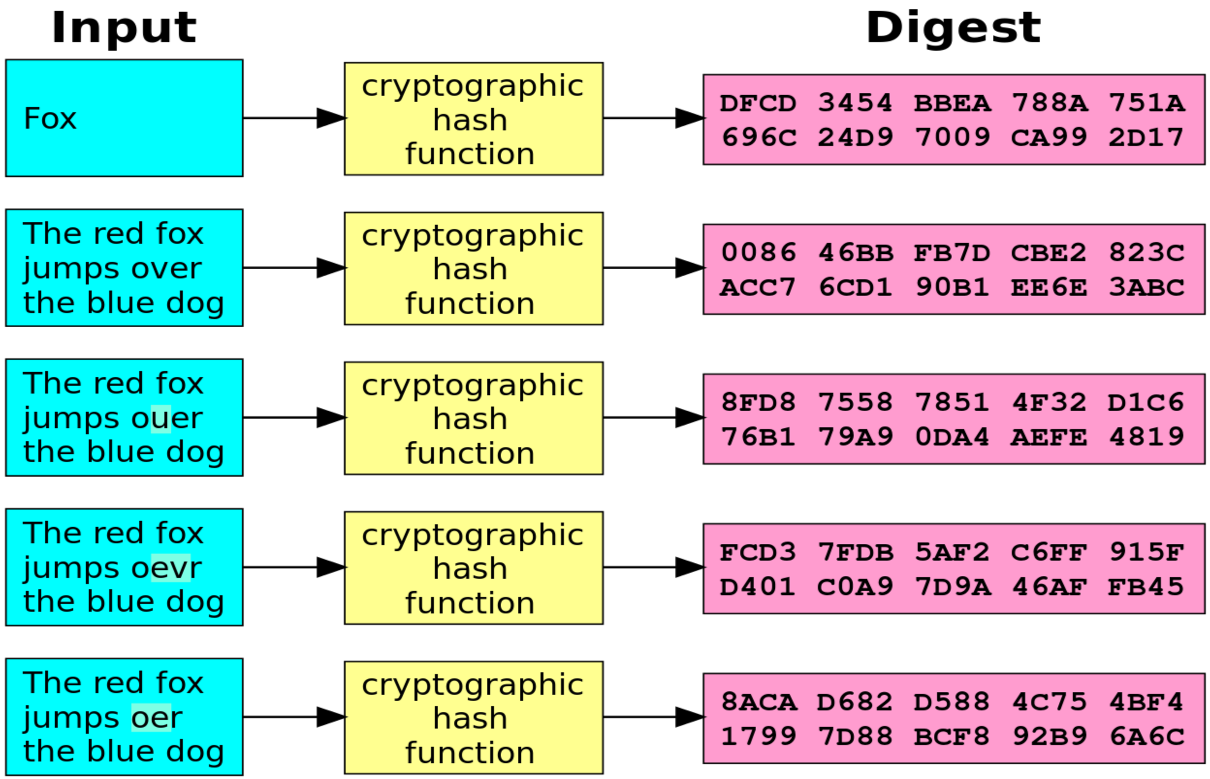
Why hash functions?

- Speed
- Error detection (*e.g.*, checksum)
- Security and privacy

Why cryptographic hash functions?

- Unique identifier for an object
- Integrity of an object
    - *E.g.*, message authentication codes
- Digital signatures
- Passwords
- Proof of work

Example



By User:Jorge Stolfi based on Image:Hash_function.svg by Helix84 - Original work for Wikipedia, Public Domain, https://commons.wikimedia.org/w/index.php?curid=5290240

What makes a hash function cryptographic?

- One-way function
- Deterministic (same input, same output)
- Infeasible to find message that digests to specific hash value
- Infeasible to find two messages that digest to the same hash
- Avalanche effect (small change in message leads to big changes in digest---digests seemingly uncorrelated)
- *Still want it to be quick*

Algorithms

- MD5: 128-bit digest, seriously broken
- SHA-1: 160-bit digest, not secure against well-funded adversaries
- SHA-3: 224 to 512 bit digest, adopted in August of 2015
- CRC32: not cryptographic, very poor choice

Property #1

- Pre-image resistance
- Given *h*, it should be infeasible to find *m* such that *h* = *hash(m)*

Property #2

- Second pre-image resistance
- Given a message m_1, it should be infeasible to find another message m_2 such that...
  $hash(m_1) = hash(m_2)$

Property #3

- Collision resistance
- It should be infeasible to find two messages, m_1 *and* m_2 such that...
  $hash(m_1) = hash(m_2)$

Wang Xiaoyun

- Tsinghua University
- Contributed a lot of ideas to cracking MD5, SHA-0, and SHA-1

Attacks

- Pre-image attack
- Collision attack
- Chosen-prefix collision attack
- Birthday attack
- Length extension attack

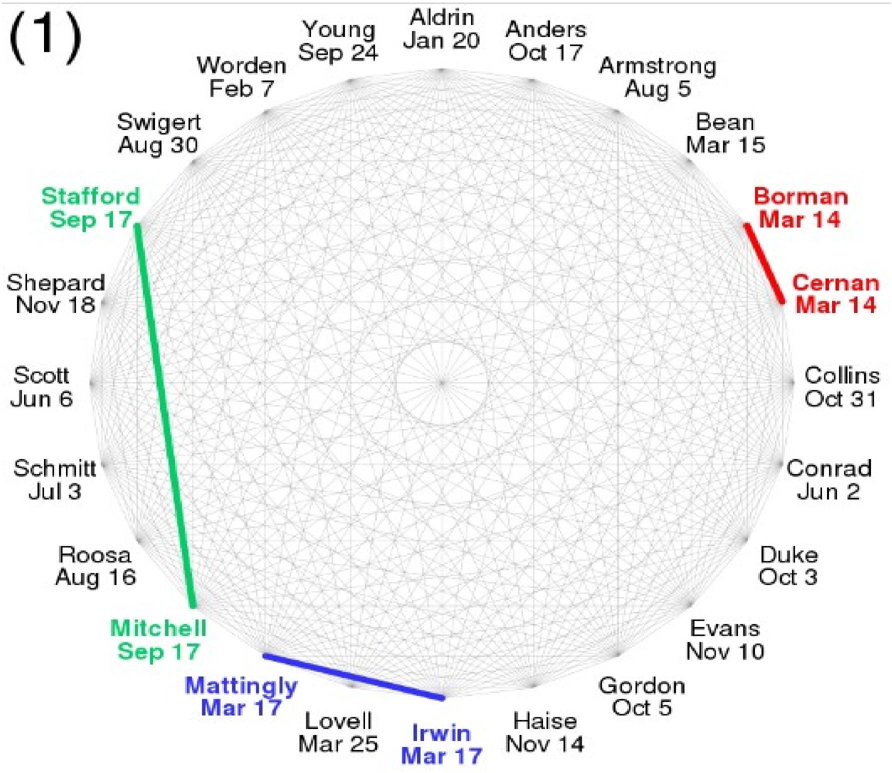
Chosen-prefix collision attack

- Given two prefixes p_1 and p_2, find m_1 and m_2 such that $hash(p_1||m_1)=hash(p_2||m_2)$
- p1 and p2 could be domain names in a certificate, images, PDFs, *etc.* … any digital image.
- This is one of the two ways MD5 is broken (other is plain old collision resistance), and is how we generated the two images with the same MD5 sum for the example from the Citizen Lab report

Birthday attack

- Probability of collision is *1* in 2^n, but the expected number of hashes until two of them collide is $sqrt(2^n)=2^{n/2}$
  - Why? Third try has two opportunities to collide, fourth has three opportunities, fifth has six, and so on...

24 people, same birthday?



https://commons.wikimedia.org/wiki/File:Birthday_attack_vs_paradox.svg

Length extension attack

```
jedi@mariposa:~$ echo "password='lDEnr45#d3'&donut=choc&quantity=1" | md5sum
91a9fc74a98997dba291a26a91c9648e  -
jedi@mariposa:~$ echo "password='lDEnr45#d3'&donut=choc&quantity=100" | md5sum
8fdd2d4515bcba887b1b80a653f21e0c  -
```

```
jedi@mariposa:~$ echo "password=               '&donut=choc&quantity=1" | md5sum
91a9fc74a98997dba291a26a91c9648e  -
jedi@mariposa:~$ echo "password=               '&donut=choc&quantity=100" | md5sum
8fdd2d4515bcba887b1b80a653f21e0c  -
```

MD5 and SHA-1 vulnerable, SHA-3 is not

References

- [Cryptography Engineering] *Cryptography Engineering: Design Principles and Applications,* by Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. Wiley Publishing, 2010.
- [Cryptovirology] *Malicious Cryptography: Exposing Cryptovirology*, by Adam Young and Moti Yung. Wiley Publishing, 2004.
- Lots of images and info plagiarized from Wikipedia