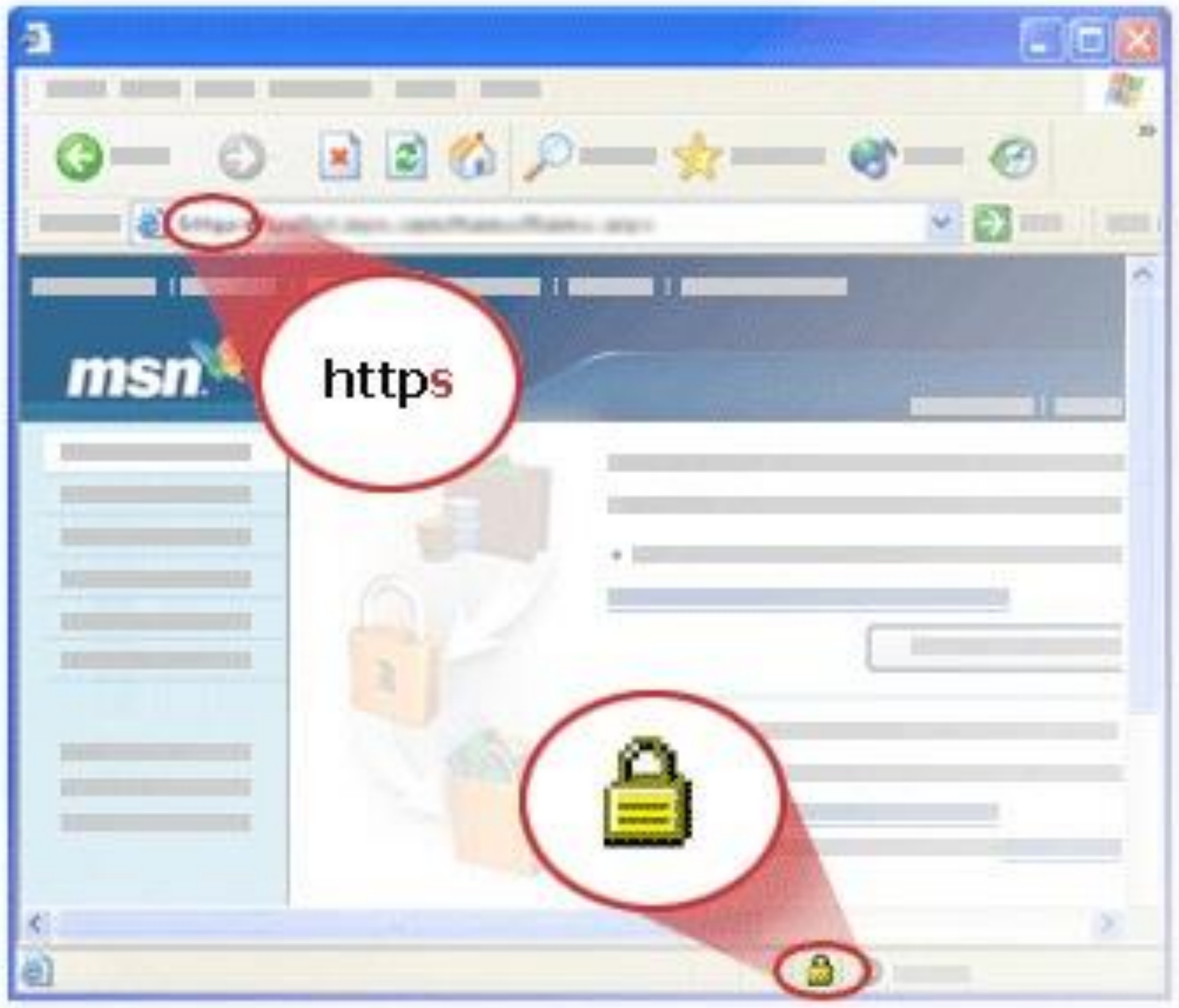
Public-Key Cryptography

Thanks to Ari Juels for parts of this deck!

The magic trick you regularly perform

- When you log into a merchant's website (via HTTPS):
  - You've often got no shared secret key or password for encryption or authentication
  - Attackers can remotely eavesdrop and tamper with your communications
- Yet somehow, you create a secure (confidential, integrity-protected) channel over which you can safely send:
  - SSNs
  - Credit card numbers
  - Passwords, etc.



What's going on?

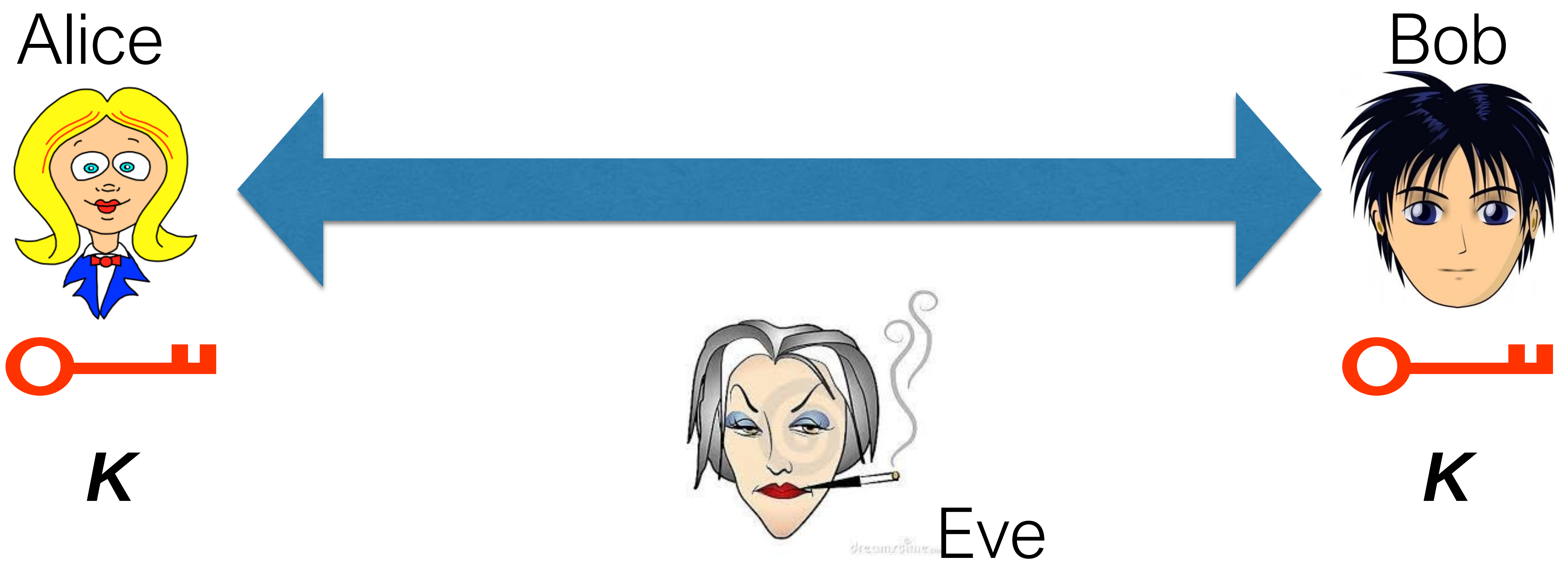
- **Key exchange**
- User and website somehow manage to choose and share a random, secret key *K*, despite:
  - No prior communication about *K*
  - Eavesdropper or malicious entity intercepting their communication

Diffie-Hellman key agreement

First practical public-key cryptosystem… and the simplest.

Goal



Alice and Bob want to share a secret key *K*, but:

- They've never met.
- They don't want Eve, who's eavesdropping, to learn *K*.

Discrete log problem

DL Problem: Given a group G of order q and the pair (g,y), where

- g is a generator of G and
- $y = g^x$ for random $x \in [0,q\text{-}1]$,

compute $x = \log_g y$.

DL assumption: The DL problem is hard (for certain groups).

(Formally, given random y, the value x cannot be computed with non-negligible probability by a probabilistic polynomial-time adversary)

Typical choices of *G*

- A Diffie-Hellman setup for SSH (RFC 4419)
  - $p = 2q+1$, for primes p and q (or $q \mid p - 1$)
  - Computation is performed mod p
  - g generates ***cyclic subgroup G of order q***
  - So Alice's public-key is $A = g^a \bmod p$, for $a \in_R [0, q-1]$
  - Typical parameter choices: p is a 2048-bit prime, q is a 224-bit value
    - Public-key key sizes ***much longer*** than symmetric-key

- Another good choice is G on an elliptic curve
  - G a cyclic subgroup for an elliptic curve on a finite field
  - Yields very compact private keys, e.g., 256-bit (ECDSA in Bitcoin), and efficient computation.

Discrete log problem

DL Problem intuition:

Random values in exponent space are "hidden," e.g. x is hidden in

$$y=g^x$$

hidden

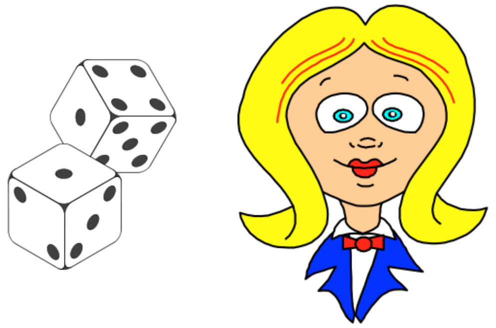
not hidden

So we can "compute secretly" in the exponent space.

Note: We'll now omit mod p for visual clarity.

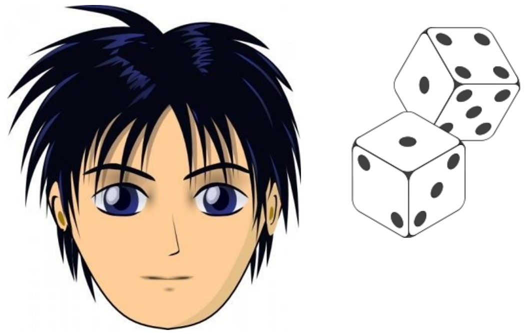
DH key agreement

Step 1: Key generation



Random private key: a

Public key: $A = g^{a}$



Random private key: b

Public key: $B = g^{b}$

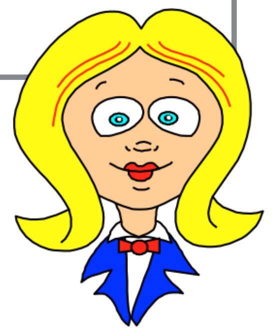
Eve

DH key agreement

(unauthenticated, simplified)

Step 2: Public-key exchange

Private / Public Keys:
(a, $A = g^a$)



$A = g^a$ →

← $B = g^b$

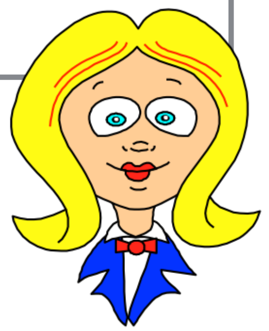
Private / Public Keys:
(b, $B = g^b$)

Eve

DH key agreement

Step 3: Computing secret, shared key

Private / Public Keys:
(a, $A = g^a$)



$A = g^a$

$B = g^b$

Private / Public Keys:
(b, $B = g^b$)

$\boldsymbol{K'} = B^a = (g^b)^a =$
$g^{ba} =$
$\boldsymbol{g^{ab}}$

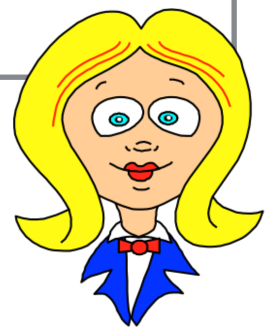
$\boldsymbol{K'} = A^b = (g^a)^b =$

$\boldsymbol{g^{ab}}$

Eve

Why can't Eve learn K?

Private / Public Keys:
(a, $A = g^a$)



$A = g^a$

$B = g^b$

Private / Public Keys:
(b, $B = g^b$)

$\mathbf{K'} = B^a = (g^b)^a =$
$g^{ba} =$
$\mathbf{g^{ab}}$

$\mathbf{K'} = A^b = (g^a)^b =$
$\mathbf{g^{ab}}$

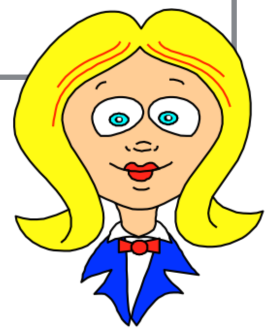
Intuition:

- Values in red are in *exponent space, so they remain hidden.*
- Alice can *multiply* hidden value b by *known* value a; vice versa for Bob.
- Eve doesn't know a or b, can't do secret multiplication (DH assumption).
- Eve can only compute, e.g., $AB = g^a g^b = g^{a+b}$.

Why can't Eve learn K?

$A = g^{a}$

Private / Public Keys:
(a, $A = g^{a}$)



Private / Public Keys:
(b, $B = g^{b}$)

$B = g^{b}$

$\boldsymbol{K'} = B^{a} = (g^{b})^{a} = g^{ba} = \boldsymbol{g^{ab}}$

$\boldsymbol{K'} = A^{b} = (g^{a})^{b} = \boldsymbol{g^{ab}}$

- g^{ab} is hashed to obtain symmetric key, e.g., AES key
- I.e., $K = H(K')$

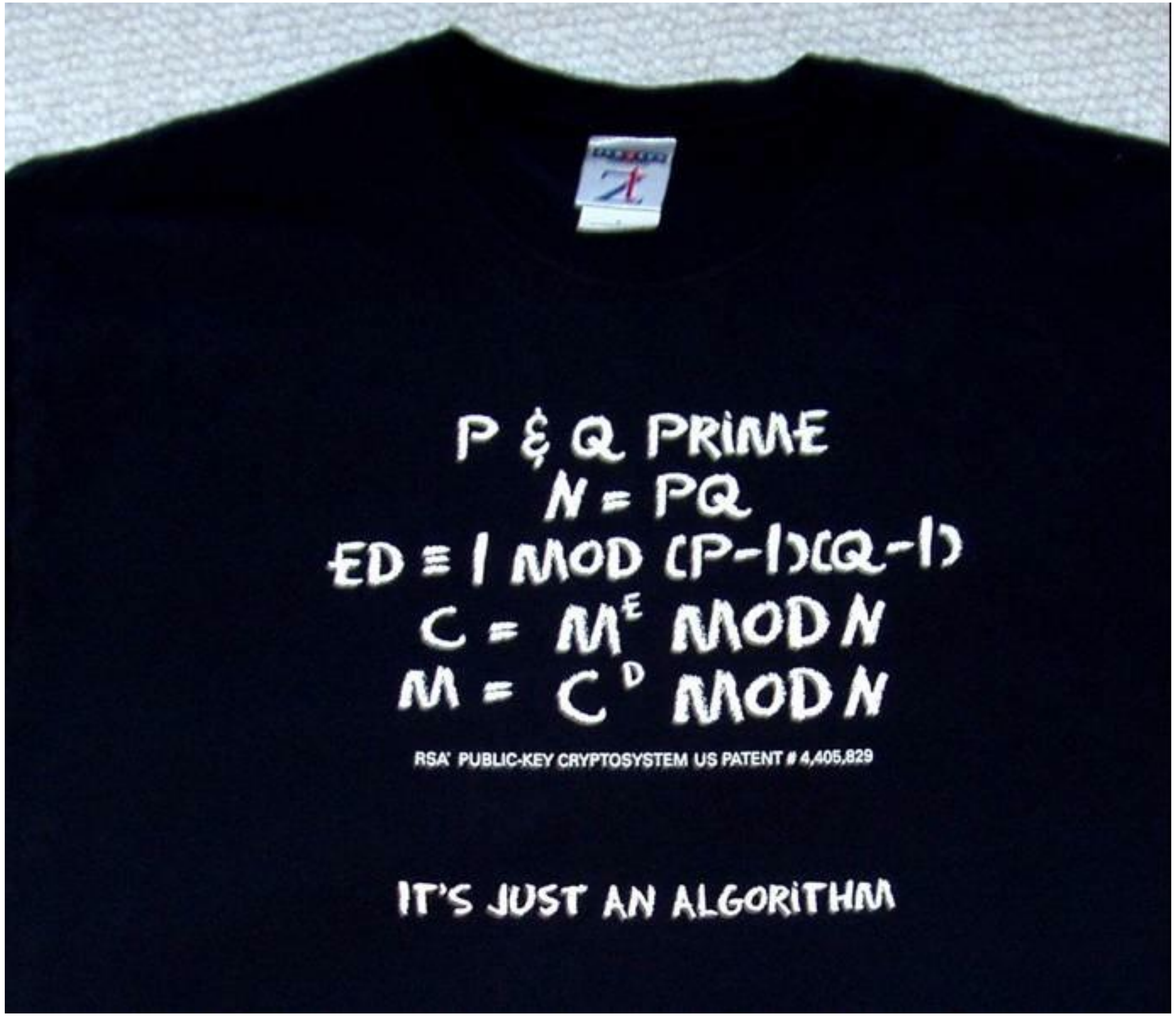
Seminal papers

- [DH] W. Diffie and M. Hellman, New directions in cryptography, IEEE TIT 22(6):644-654 (1976)
  - First practical public-key cryptographic algorithm—for key exchange (not encryption)
  - Many other conceptual contributions, e.g.,
    - Notion of digital signatures
    - Relationship between crypto and complexity theory
      - Idea that crypto should be predicated on seemingly hard problems
      - Requirement for average-case hardness given random selection of instance
- [RSA] R. Rivest, A. Shamir, and L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, CACM 21(2):120-126 (1978)
  - First public-key encryption *and* digital signature scheme
  - Introduced Alice and Bob

RSA encryption

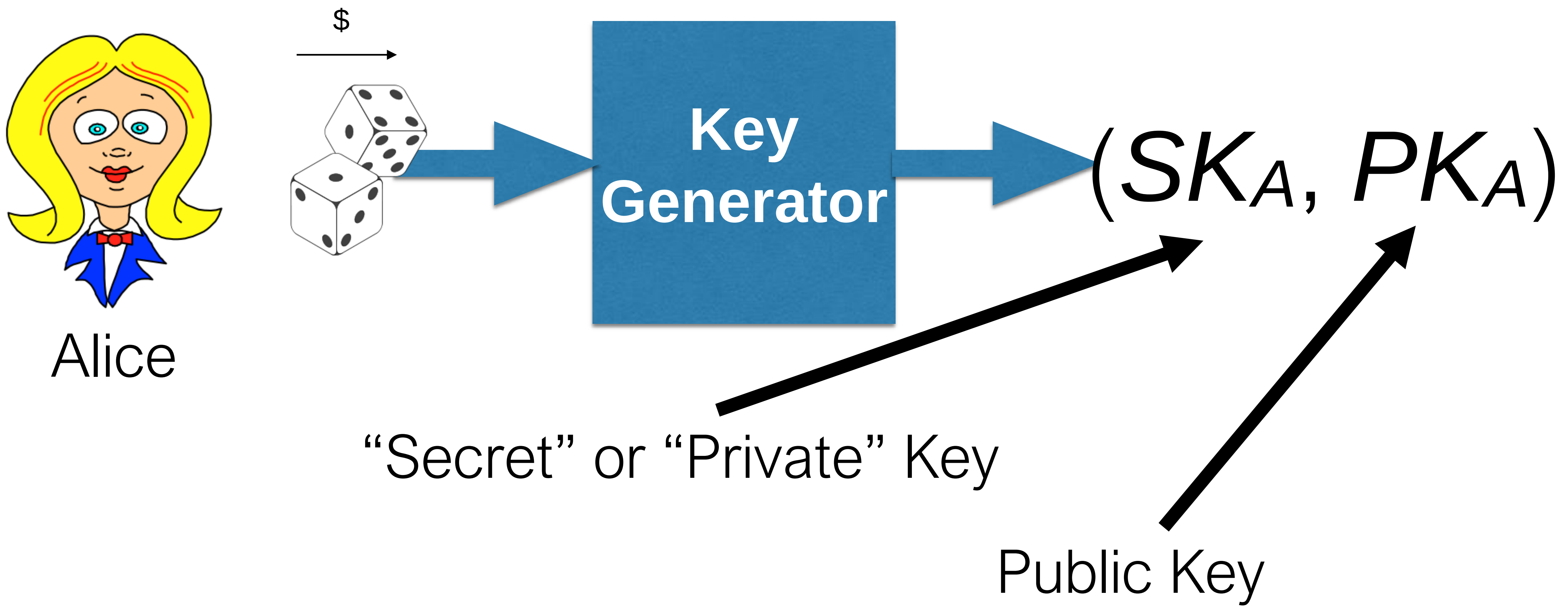
- Uses modular exponentiation (like D-H)
- Security related to hardness of *factoring*
  - Given pq for large primes p and q, compute p and q

RSA

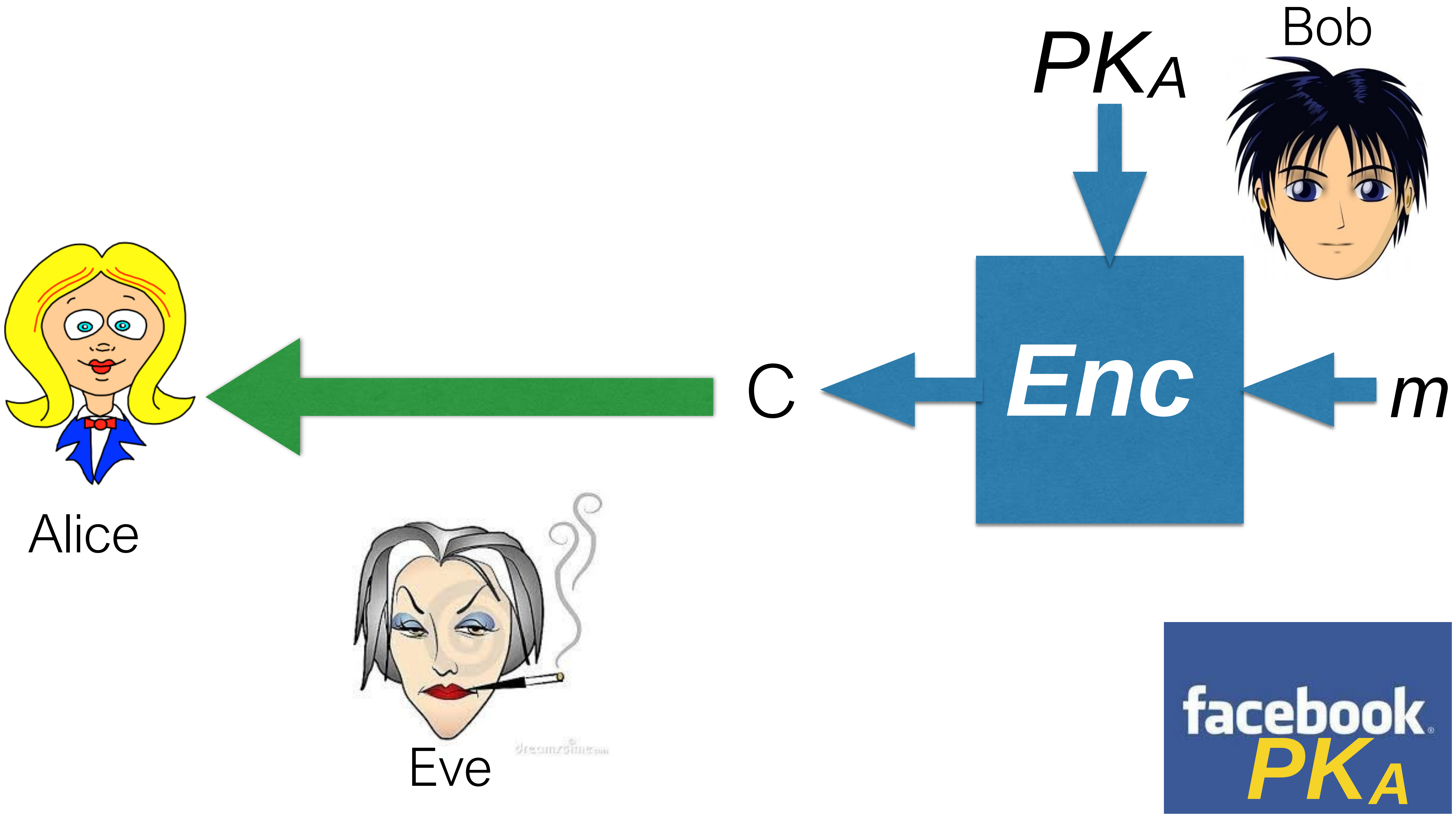

P & Q PRIME
N = PQ
ED ≡ 1 MOD (P-1)(Q-1)
C = M^E MOD N
M = C^D MOD N
RSA PUBLIC-KEY CRYPTOSYSTEM US PATENT # 4,405,829
IT'S JUST AN ALGORITHM

Public-key encryption (a la RSA)

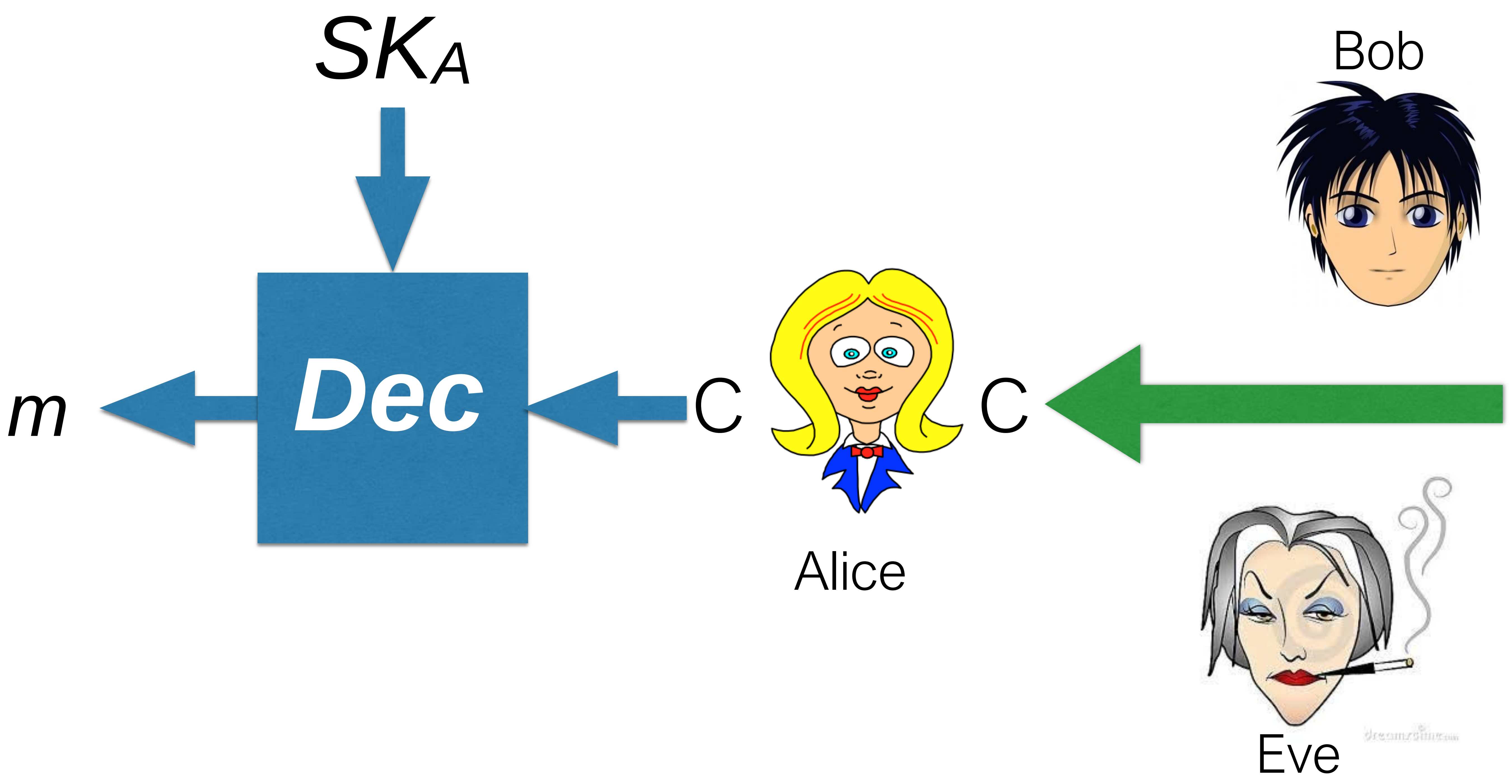
- Key generation



Encryption



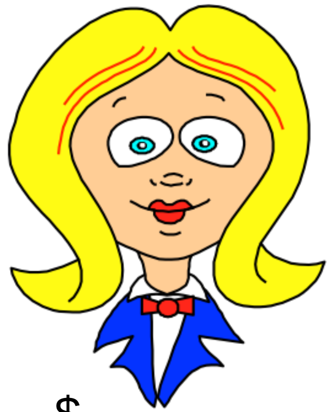
Decryption



SSL / TLS using RSA

(grossly simplified)

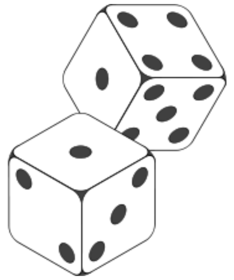
Alice



(SK,PK)

PK

$\xrightarrow{\$}$



K

(pre-master secret)

$c = \mathrm{Enc}_{PK}[K]$

Symmetric encryption under key K^*

Decrypt K

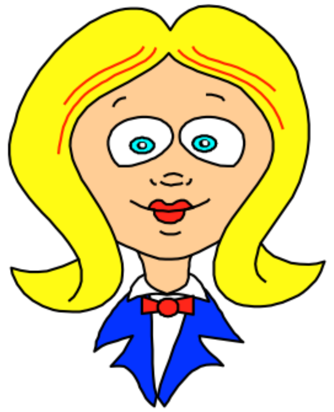
Notes:

- Session key K^* derived from K and other inputs
- Symmetric-key encryption in authenticated mode
- Whole protocol is *extremely* complicated, with cipher-suite negotiation, etc.

Can also use D-H

(*SK*,*PK*)

Alice



PK

K derived using D-H

Symmetric encryption
under key *K**

Notes:
- Session key K^* derived from *K* and other inputs
- Symmetric-key encryption in authenticated mode
- Whole protocol is ***extremely*** complicated, with cipher-suite negotiation, etc.

What can we learn from this design?

- Why do Alice and the server switch to symmetric-key encryption?
  - Public-key encryption allows negotiation of secrets over public channels.
  - But symmetric-key encryption is far faster than public-key encryption.
    - E.g., 32,000 RSA decryptions / second in coprocessor (Freescale C293)
    - Intel AES-NI: about 1.3 cycles / byte for AES-128 (CBC-decrypt) on single-core Intel Core i7 Extreme Edition, i7-980
  - A hybrid approach achieves the best of both worlds…
    - Can also be used for message encryption via “key wrapping”
    - C = (Enc$_{PK}$[K], enc$_K$[m])
      - Enc is public-key, enc is symmetric-key

How hard is it to break RSA?

- Best known general attack involves factoring *N = pq*
- Difficulty of best classical factoring algorithm (general number field sieve) grows super-polynomially (but sub-exponentially)

$$\exp\left(\left(\sqrt[3]{\frac{64}{9}}+o(1)\right)(\ln n)^{\frac{1}{3}}(\ln \ln n)^{\frac{2}{3}}\right) = L_n\left[\frac{1}{3}, \sqrt[3]{\frac{64}{9}}\right]$$

- Here, *n* is bit length
- Note that faster computers *favor defenders.*
- A better method could arise, e.g.,
  - Algorithmic breakthrough
  - Factoring quantum computer (for which poly-time algorithms are known)…
- Similar story for D-H

In-class exercise

- Apart from TLS, what are some good applications of public-key encryption?
- Why is symmetric-key encryption alone not sufficient to achieve them?

Some applications of public-key encryption

- Secure e-mail
  - S/MIME
- Hard-drive encryption
  - E.g., FileVault, PGP full-disk encryption use RSA
  - Don't need secret key to add files!