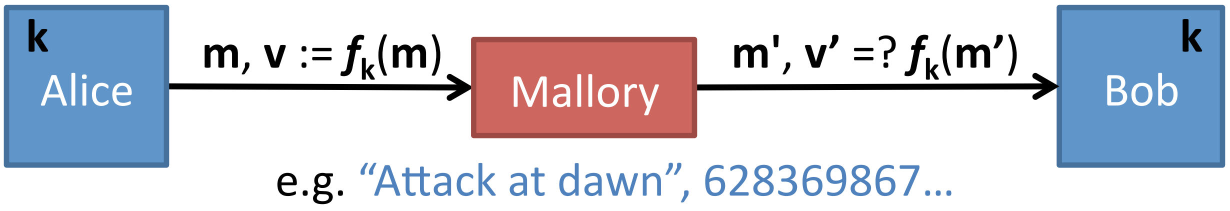
Public-Key Crypto

Review: Integrity

Problem: Sending a message over an **untrusted channel** without being changed

Provably-secure solution: **Random function**

Practical solution:



Pseudorandom function (**PRF**)

Input: arbitrary-length **k**

Output: fixed-length value

Secure if practically indistinguishable from a random function, unless know **k**

Real-world use:

Message authentication codes (**MACs**) built on cryptographic hash functions

Popular example: **HMAC-SHA256$_k$(m)**

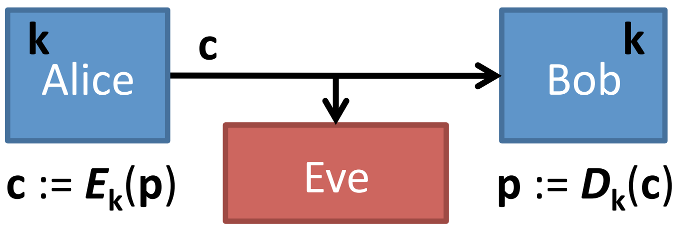
[Cautions?!]

Review: Confidentiality

Problem: Sending message in the presence of an **eavesdropper** without revealing it

Provably-secure solution: **One-time pad**

Practical solution:


k
Alice
c
k
Bob
Eve
$\mathbf{c} := E_k(\mathbf{p})$
$\mathbf{p} := D_k(\mathbf{c})$

Pseudorandom generator (**PRG**)

Input: fixed-length **k**

Output: arbitrary-length stream

Secure if practically indistinguishable from a random stream, unless know **k**

Real-world use:

Stream ciphers (can't reuse **k**)

Popular example: **AES-128** + **CTR mode**

Block ciphers (need **padding/IV**)

Popular example: **AES-128** + **CBC mode**

[Cautions?!]

Review: Diffie-Hellman Key Exchange

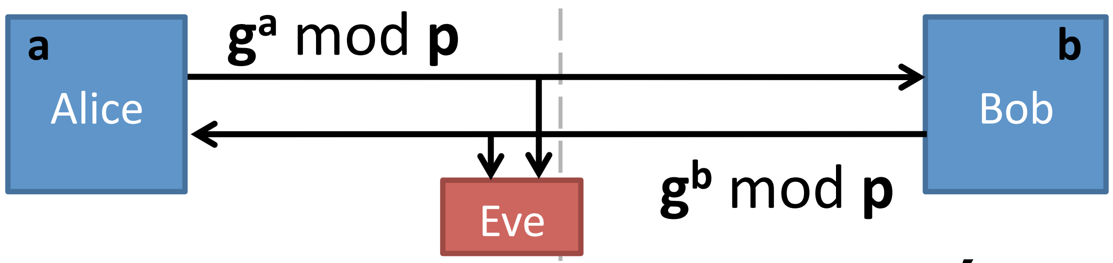
Lets Alice and Bob **agree on a shared secret** value by having a public conversation

Alice

Generates random secret **a** ($0 < \mathbf{a} < \mathbf{p}$)

Bob

Generates random secret **b** ($0 < \mathbf{b} < \mathbf{p}$)



Computes **x**

$= (\mathbf{g}^{b} \bmod \mathbf{p})^{a} \bmod \mathbf{p}$

$= \mathbf{g}^{ba} \bmod \mathbf{p}$

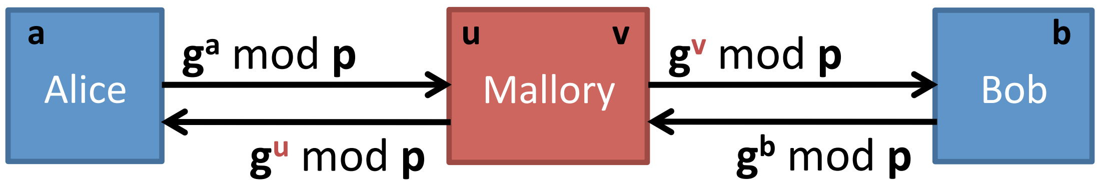
Computes **x′**

$= (\mathbf{g}^{a} \bmod \mathbf{p})^{b} \bmod \mathbf{p}$

$= \mathbf{g}^{ab} \bmod \mathbf{p}$

x == x′

Problem: **Man-in-the-middle attacks**



<u>Caution</u>: D-H gives you a shared secret, but don’t know who’s on the other end!

Suppose Alice publishes data to lots of people, and they all want to verify integrity…

Can't share an integrity key with *everybody*, or else *anybody* could forge messages

Suppose Bob wants to receive data from lots of people, confidentially…

Schemes we've discussed would require a separate key shared with each person

[What to do?]

Solution:

Public-key Crypto

So far, encryption key == decryption key
  “**symmetric key crypto**”

New idea: Keys are distinct, and
  you can’t find one from the other

Almost always used by splitting key-pair
  Alice keeps one key private (“**private key**”)
  Publishes the other key (“**public key**”)

Many applications

Invented in 1976 by Diffie and Hellman
  (earlier by Clifford Cocks of British intelligence, in secret)

Best known, most common public-key algorithm: **RSA**
  Rivest, Shamir, and Adleman 1978

Requirements for a public key crypto system to be secure

1. Computationally easy for B to generate a key pair: PU_b, PR_b
2. Computationally easy for sender A to generate the ciphertext for message M: $C=E(PU_b, M)$
3. Computationally easy for receiver B to decrypt the ciphertext: $M=D(PR_b, C)$
4. Computational infeasible to guess PR_b knowing PU_b.
5. Computational infeasible to recover M from PU_b and C.

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R.L. Rivest, A. Shamir, and L. Adleman*

How RSA works

Key generation:

1. Pick large (say, 1024 bits) random primes **p** and **q**
2. Compute **N** := **pq**
   (RSA uses multiplication mod **N**)
3. Pick **e** to be relatively prime to (**p**-1)(**q**-1)
4. Find **d** so that **ed** mod (**p**-1)(**q**-1) = 1
5. Finally: **Public key** is (e,N)
   Private key is (d,N)

To encrypt: $E(\mathbf{x}) = \mathbf{x}^{e} \bmod \mathbf{N}$

To decrypt: $D(\mathbf{x}) = \mathbf{x}^{d} \bmod \mathbf{N}$

Why RSA works

"It works" theorem:

For all $0 < \mathbf{x} < \mathbf{N}$,

can show that $\boldsymbol{D}(\boldsymbol{E}(\mathbf{x})) = \mathbf{x}$

Proof:

$$
\begin{aligned}
\boldsymbol{D}(\boldsymbol{E}(\mathbf{x})) &= (\mathbf{x}^{\mathbf{e}} \bmod \mathbf{pq})^{\mathbf{d}} \bmod \mathbf{pq} \\
&= \mathbf{x}^{\mathbf{ed}} \bmod \mathbf{pq} \\
&= \mathbf{x}^{\mathbf{a}(\mathbf{p}-1)(\mathbf{q}-1)+1} \bmod \mathbf{pq} \quad \text{for some } \mathbf{a} \\
&\qquad (\text{because } \mathbf{ed} \bmod (\mathbf{p}-1)(\mathbf{q}-1) = 1) \\
&= (\mathbf{x}^{(\mathbf{p}-1)(\mathbf{q}-1)})^{\mathbf{a}}\mathbf{x} \bmod \mathbf{pq} \\
&= (\mathbf{x}^{(\mathbf{p}-1)(\mathbf{q}-1)} \bmod \mathbf{pq})^{\mathbf{a}}\mathbf{x} \bmod \mathbf{pq} \\
&= 1^{\mathbf{a}}\mathbf{x} \bmod \mathbf{pq} \\
&\qquad (\text{because of the fact that if } \mathbf{p},\mathbf{q} \\
&\qquad \text{are prime, then for all } 0 < \mathbf{x} < \mathbf{N}, \\
&\qquad \mathbf{x}^{(\mathbf{p}-1)(\mathbf{q}-1)} \bmod \mathbf{pq} = 1) \\
&= \mathbf{x}
\end{aligned}
$$

Is RSA secure?

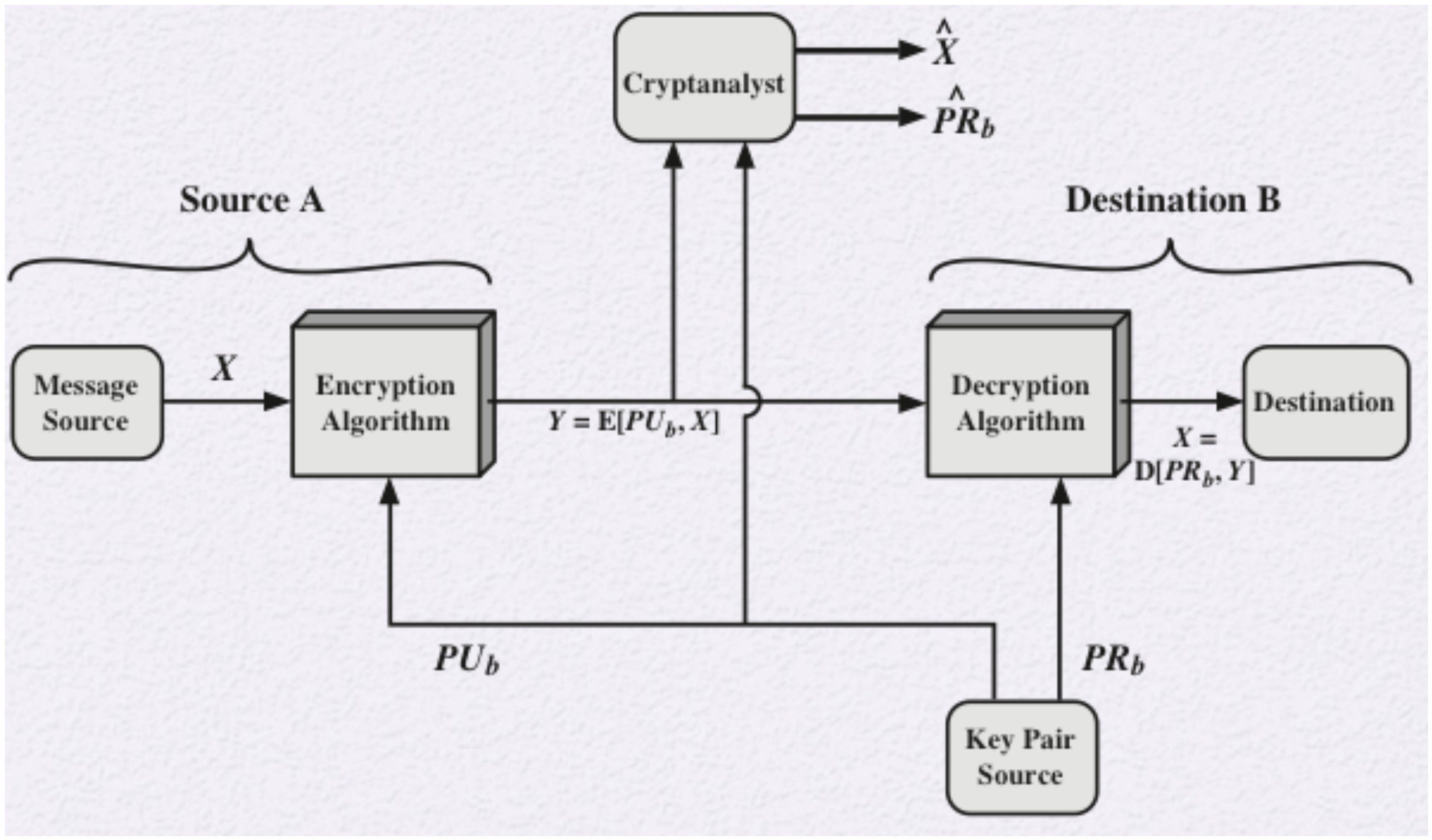
Best known way to compute **d** from **e**
is factoring **N** into **p** and **q**.

Best known factoring algorithm:
General number field sieve
Takes more than polynomial time
but less than exponential time
to factor **n**-bit number.

(Still takes way too long if **p**,**q**
are large enough and random.)

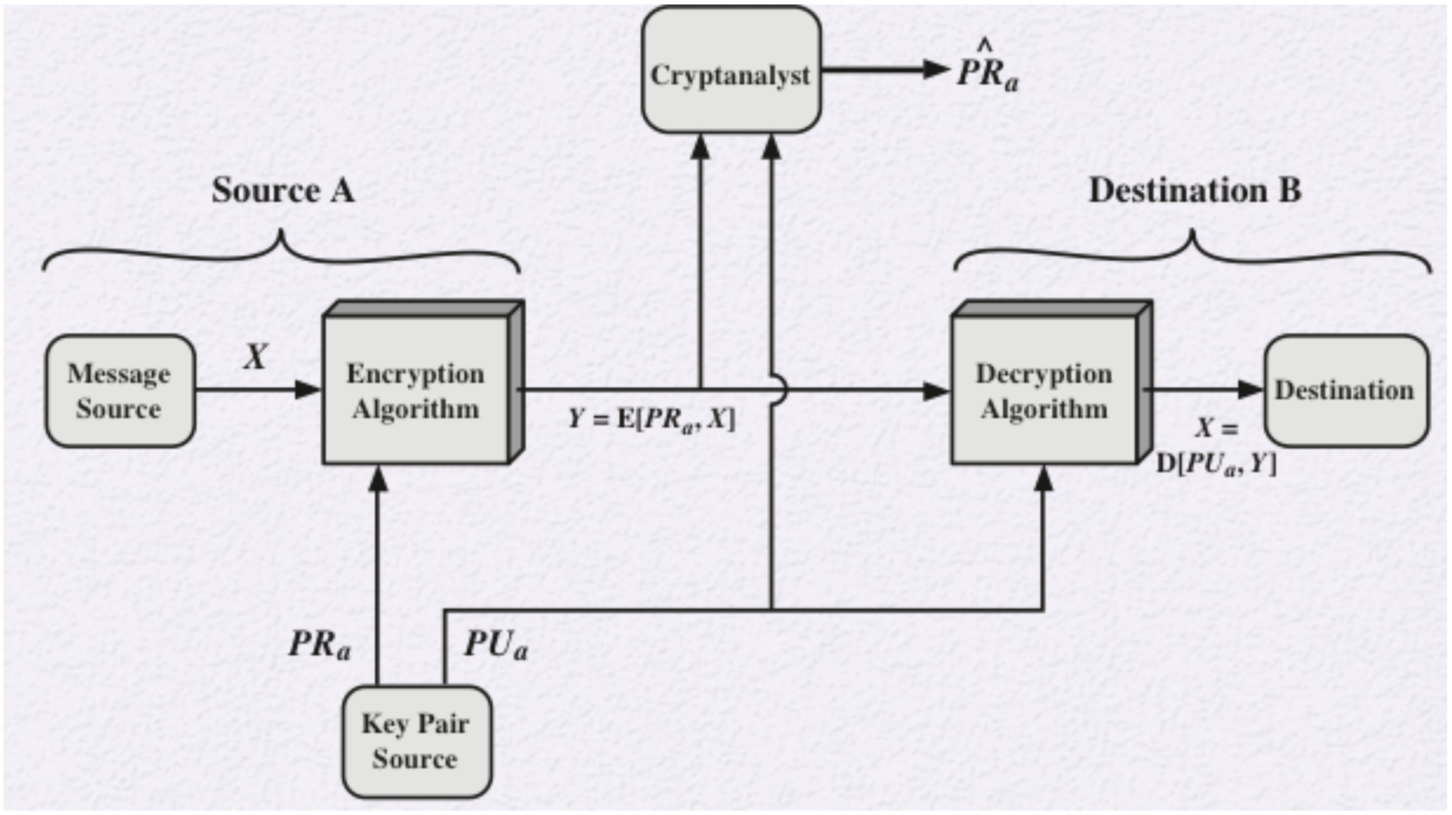
Fingers crossed…
but can't rule out a breakthrough!

Signing with the public key for confidentiality or secrecy:



Does this provide integrity?

Signing with private key for integrity/authentication.



Does this provide confidentiality?

Subtle fact: RSA can be used for either confidentiality or integrity

RSA for confidentiality:

Encrypt with public key
Decrypt with private key
"your eyes only" message

RSA for integrity:

Encrypt ("sign") with private key
Decrypt ("verify") with public key
called a **digital signature**

[What if we want both confidentiality and integrity on the same message?]

How to have both confidentiality and integrity (using RSA)?

Alice (A) wants to send a secret message to Bob (B) so that Bob can verify that it comes from Alice.

Which one(s) is/are secure?

1. $E(E(M, PR_A), PU_B)$
2. $E(E(M, PU_B), PR_A)$
3. $C=E(M, PR_A)$ $E(MAC(C), PU_B)$
    - Send C||MAC
4. $C=E(M, PU_B)$ $E(MAC(C), PR_A)$

RSA drawback: Performance

Factor of 1000 or more slower than AES.

Dominated by exponentiation – cost goes up (roughly) as cube of key size.

Message must be shorter than **N**.

[How big should the RSA keys be?]

Use in practice:

Encryption:

Use RSA to encrypt a random **x** < **N**, compute **k** := PRF(**x**), encrypt message using a symmetric cipher and key **k**

Signing:

Compute **v** := PRF(**m**), use RSA to sign a carefully padded version of **v**
(many gotchas!)

Almost always should use crypto libraries to get the details right

True or false:

Public-key encryption is more secure from cryptanalysis than symmetric encryption.

True or false:

Public-key encryption is a general-purpose technique that has made symmetric encryption obsolete

True or false:

Key distribution is trivial when using public-key encryption, compared to the cumbersome handshaking involved with key distribution centers for symmetric encryption.

Attacks against RSA

1. Brute force: trying all possible private keys

2. Mathematical attacks: factoring

3. Timing attacks: using the running time of decryption

4. Hardware-based fault attack: induce faults in hardware to generate digital signatures

5. Chosen ciphertext attack

Exercise

Suppose Bob uses RSA crypto with a very large modulus **n** for which the factorization cannot be found in a reasonable amount of time.

Suppose Alice sends a message to Bob by representing each alphabet letter as an integer between 0 and 25 (A->0, …, Z->25) and then encrypting each number separately using RSA with large **e** and large **n**.

Is this method secure?

If yes, why?

If not, how to efficiently attack this encryption method?

Solution:

For a set of message block values SM = {0, 1, 2, ..., 25}. The set of corresponding ciphertext block values SC = {0^e mod N, 1^e mod N, ..., 25^e mod N}, and can be computed by everybody with the knowledge of the public key of Bob.

The most efficient attack is to compute M^e mod N for all possible values of M, then create a look-up table with a ciphertext as an index, and the corresponding plaintext as a value of the appropriate location in the table.

So Far:

The Security Mindset

Message Integrity

Confidentiality

Key Exchange

Building a Secure Channel

Public Key Crypto

Next Week:

Begin Web Security Unit

HTTPS: Secure channels for the web

Review: Public-key Crypto

So far, encryption key == decryption key
"**symmetric key crypto**"

New idea: Keys are distinct.

RSA: **N** := **pq**

Public key is **(e,N)**

Private key is **(d,N)**

To encrypt: $E(\mathbf{x}) = \mathbf{x}^e \bmod \mathbf{N}$
To decrypt: $D(\mathbf{x}) = \mathbf{x}^d \bmod \mathbf{N}$

RSA for confidentiality:

Encrypt with public key
Decrypt with private key

*RSA for integrity (**digital signatures**):*

Encrypt ("sign") with private key
Decrypt ("verify") with public key

[Cautions?!]