Public Key Cryptography

EECS 388 F17

UNIVERSITY OF MICHIGAN

Review

Properties of a Secure Channel

Confidentiality

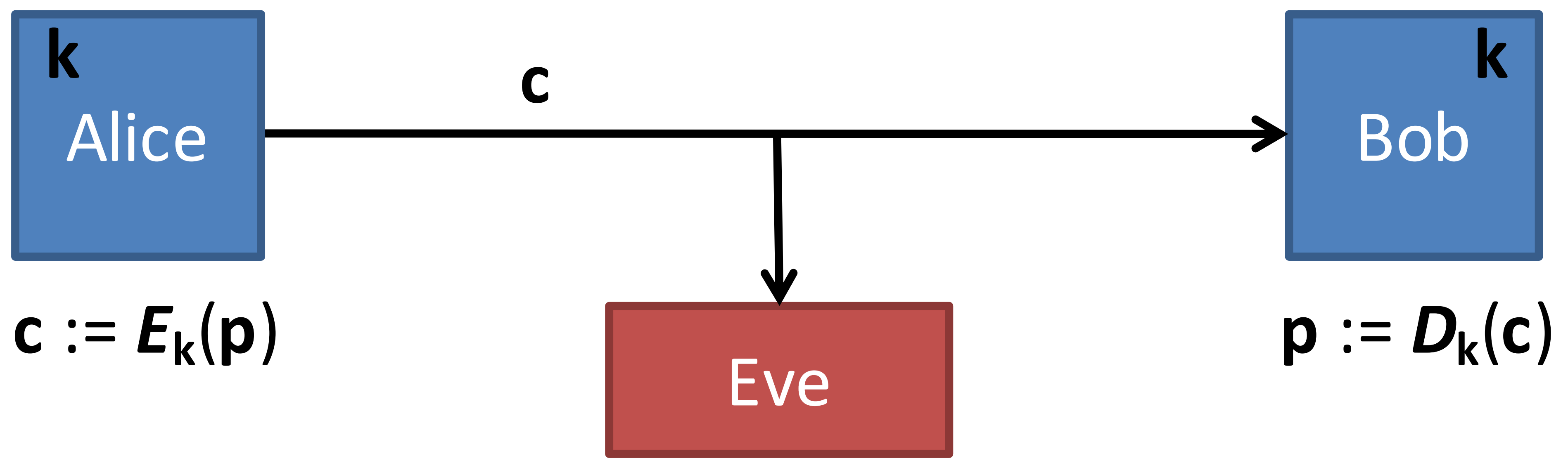
Integrity

Authentication (this is what we are doing today!)

Confidentiality Review

Goal: Keep contents of message ***p*** secret from an eavesdropper

This protects us from Eve (eavesdropper)



p	plaintext
c	ciphertext
m	message / plaintext
k	secret key
E	encryption function
D	decryption function

Integrity Review

1. Let f be a secure PRF.
2. In advance choose a random k known only to Alice and Bob.
3. Alice computes $\mathbf{v} := f_k(\mathbf{m})$.
4. 

5. Bob verifies $\mathbf{v'} = f_k(\mathbf{m'})$, accepts if and only if this is true.

- **This protects us from Mallory (MITM)**

Encryption / Integrity Ordering

Encrypt, then MAC

Encrypt, then MAC

Encrypt, then MAC

This protects from Cryptographic Doom (like the Padding Oracle)

AEAD

Authenticated Encryption and Associated Data

```
ciphertext, auth_tag := Seal(key, plaintext,
associated_data)

plaintext := Unseal(key, ciphertext, associated_data,
tag)
```

Combines integrity and encryption.

This protects us *<u>even better</u>* from Cryptographic Doom!

Diffie-Hellman Key Exchange

Standard g (generator), and p (prime, or modulus)

Alice picks secret a

$A = g^a \bmod p \;\longrightarrow$

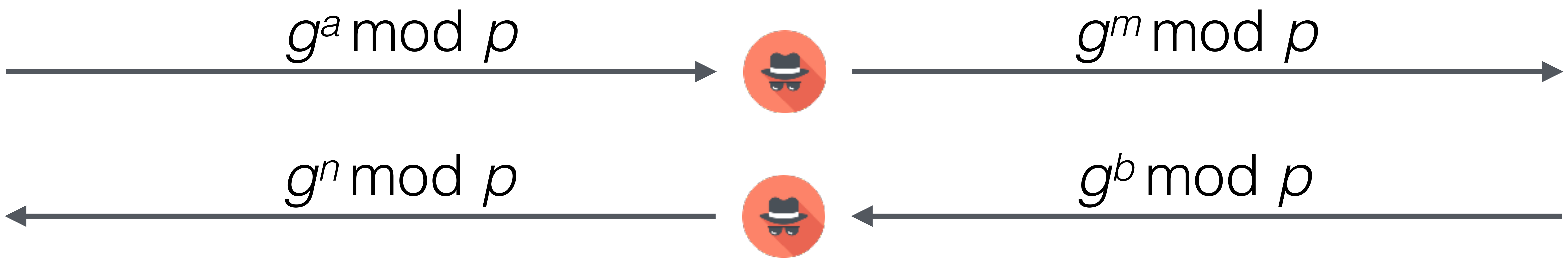
$\longleftarrow\; B = g^b \bmod p$

Bob picks secret b

$\mathbf{k} := A^b \bmod p = g^{ab} \bmod p = g^{ba} \bmod p = B^a \bmod p$

Diffie-Hellman Key Exchange (MITM)

Standard g (generator), and p (prime, or modulus)



key := $g^{an} \bmod p = g^{na} \bmod p$

key := $g^{mb} \bmod p = g^{bm} \bmod p$

DH really only protects them from Eve…not from Mallory… :-(

Public Key Cryptography (digital signatures) to the rescue… :-)

New Stuff

Problem: Scaling

- Suppose Alice publishes data to lots of people, and they all want to verify integrity…
  - ➡ Can't publish an integrity key [why?]
- Suppose Bob wants to receive data from lots of people, confidentially…
  - ➡ Impractical [why?]

Solution: Public-Key Cryptography

So far: encryption key == decryption key

New idea: keys are different, and *you can't find one from the other*

Typically: get a key pair and split it

- **Public Key**: make this public
- **Private Key**: keep this private

History

Invented in 1976 by Diffie and Hellman

(…but also secretly in 1973 by Clifford Cocks at GCHQ)

Most famous: **RSA** in 1978 by 1978 by (R)ivest, (S)hamir, and (A)dleman

How RSA Works

1. Pick two large (say, 2048 bits) random primes **p** and **q**.

2. **N := p * q** (RSA does multiplication mod **N**)

3. Pick **e** to be relatively prime to **(p – 1) * (q – 1)**.

4. Find **d** so that **(e * d)** mod **((p –1) * (q – 1)) = 1**

5. Public key **:=** (**e**, **N**)   <— the key is actually a pair of values
   Private key **:=** (**d**, **N**)

6. **Encryption: $\mathbf{Enc_e(x) = x^e}$** mod **N**
   Decryption: $\mathbf{Dec_d(x) = x^d}$ mod **N**

How RSA works

Again with the colors!

https://youtu.be/wXB-V_Keiu8?t=127

Why RSA Works

For all $0 < \mathbf{x} < \mathbf{N}$, we can show that $\mathbf{E(D(x))} = \mathbf{D(E(x))} = \mathbf{x}$

Proof of $E(D(\mathbf{x}))$ side:

$$
\begin{aligned}
E(D(\mathbf{x})) &= (\mathbf{x}^{\mathbf{d}} \bmod \mathbf{pq})^{\mathbf{e}} \bmod \mathbf{pq} \\
&= \mathbf{x}^{\mathbf{de}} \bmod \mathbf{pq} \\
&= \mathbf{x}^{\mathbf{a}(\mathbf{p}-1)(\mathbf{q}-1)+1} \bmod \mathbf{pq} \quad \text{for some } \mathbf{a} \text{ (because } \mathbf{ed} \bmod (\mathbf{p}-1)(\mathbf{q}-1) = 1) \\
&= (\mathbf{x}^{(\mathbf{p}-1)(\mathbf{q}-1)})^{\mathbf{a}}\mathbf{x} \bmod \mathbf{pq} \\
&= (\mathbf{x}^{(\mathbf{p}-1)(\mathbf{q}-1)} \bmod \mathbf{pq})^{\mathbf{a}}\mathbf{x} \bmod \mathbf{pq} \\
&= 1^{\mathbf{a}}\mathbf{x} \bmod \mathbf{pq}
\end{aligned}
$$

(because of the fact that if $\mathbf{p},\mathbf{q}$ are prime, then for all $0 < \mathbf{x} < \mathbf{N}$, $\mathbf{x}^{(\mathbf{p}-1)(\mathbf{q}-1)} \bmod \mathbf{pq} = 1$)

$$= \mathbf{x}$$

Is RSA Secure?

Best known way to compute **d** given **e** is by factoring **N** into **p** and **q**…

This is very hard to do…if p and q are 2048 bits, N is 4096 bits…

Best known factoring algorithm (general number field sieve)

This takes more than polynomial time, but less than exponential time to factor an **n**-bit number. Takes "a while" with 4096 bits…

So…Fingers crossed!

“Fun” RSA Facts

RSA can be used for confidentiality, integrity/authentication, or both!

Confidentiality:

1. Alice encrypts with Bob’s public key: $\mathbf{E_{pubkeyBob}(m) = m^{e} \bmod N = c}$
2. Bob decrypts with his private key: $\mathbf{D_{privkeyBob}(c) = c^{d} \bmod N = m}$

Integrity/Authentication: <— called a digital signature

1. Alice signs with her private key: $\mathbf{Sign_{privkeyAlice}(m):\ s = m^{d} \bmod N}$
2. Bob verifies using Alice’s public key: $\mathbf{Verify_{pubkeyAlice}(m,s):\ m \mathrel{?=} s^{e} \bmod N}$

Note that both m ***and*** s are sent to Bob.

Both (w/two key pairs):

1. Alice encrypts with **pubkeyBob**, then signs with **privkeyAlice**
2. Bob verifies with **pubkeyAlice**, then decrypts with **privkeyBob**

Drawbacks

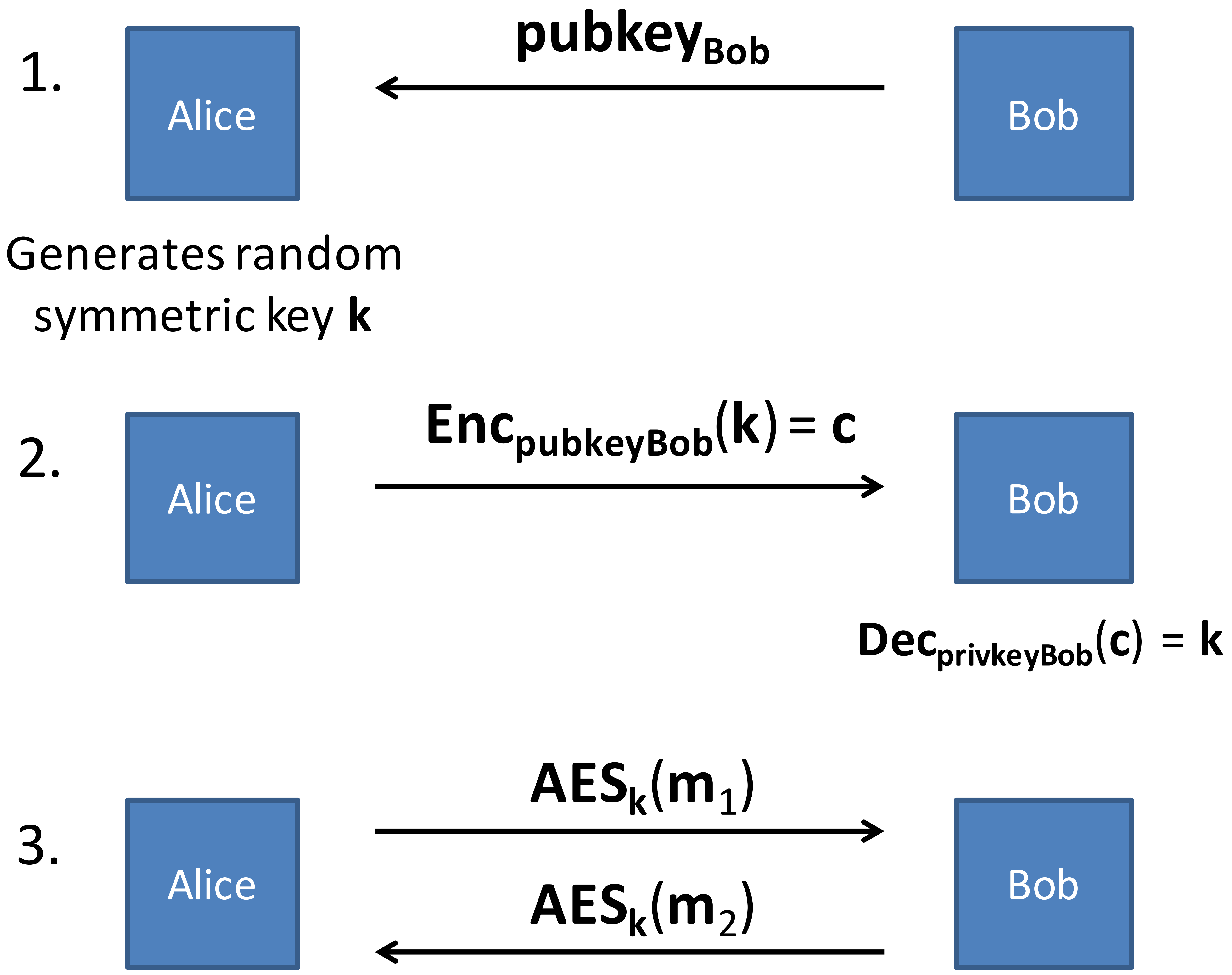
1000x (or more!) slower than AES

Runtime is dominated by exponentiation (goes up as cube of key size)

Message must be shorter than **N**

Keys must be large

RSA In Practice: Encryption (Confidentiality)



This is called "bootstrapping symmetric crypto with public key crypto":

- We no longer need to worry about the inefficiency of RSA
- We no longer have to worry about message length

RSA In Practice: Signing (Integrity/Authentication)

1. **v :=** PRF(**m**)

   In other words, digest/“shorten” the message using a PRF or a hash so that we can send messages of arbitrary length.

2. Use RSA to sign a *carefully padded* version of **v**

RSA Gotchas!

1. **Problems during key generation** (PRGs without enough entropy)

Alice

$\text{pubkey}_{\text{Alice}} = (\mathbf{e}_1, \mathbf{N})$

Bob

$\text{pubkey}_{\text{Bob}} = (\mathbf{e}_2, \mathbf{N})$

The public keys for both Alice and Bob have the same value for N. How can one of them attack the other?

Alice

$\text{pubkey}_{\text{Alice}} = (\mathbf{e}, \mathbf{N}_1 = \mathbf{p}\mathbf{q}_1)$

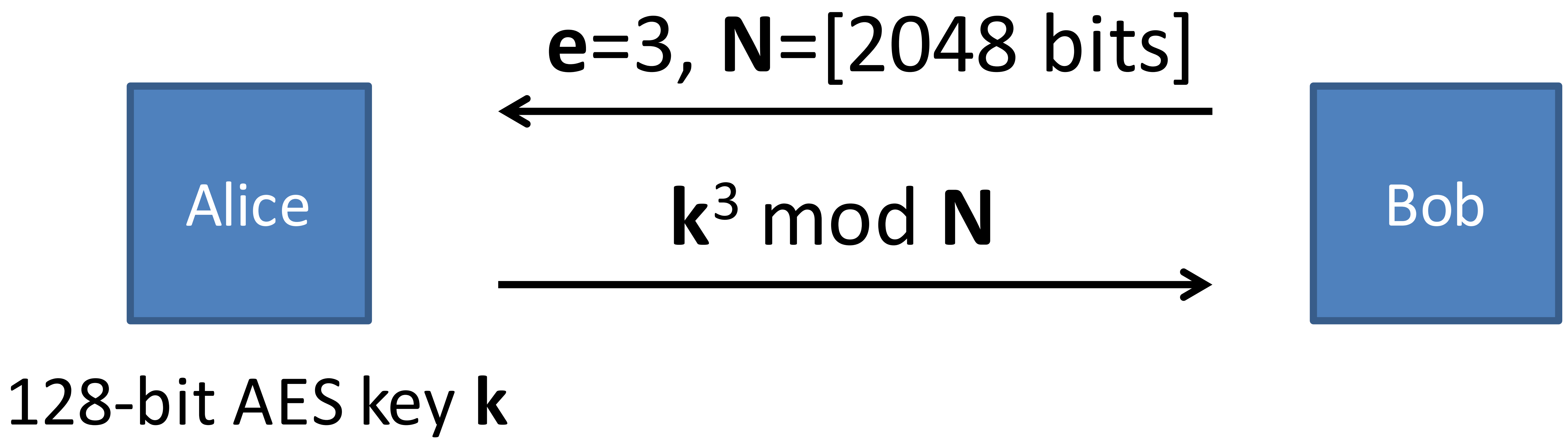
Bob

$\text{pubkey}_{\text{Bob}} = (\mathbf{e}, \mathbf{N}_2 = \mathbf{p}\mathbf{q}_2)$

The public keys for both Alice and Bob have the same value for p. How can ***Eve*** attack Alice and Bob?

RSA Gotchas!

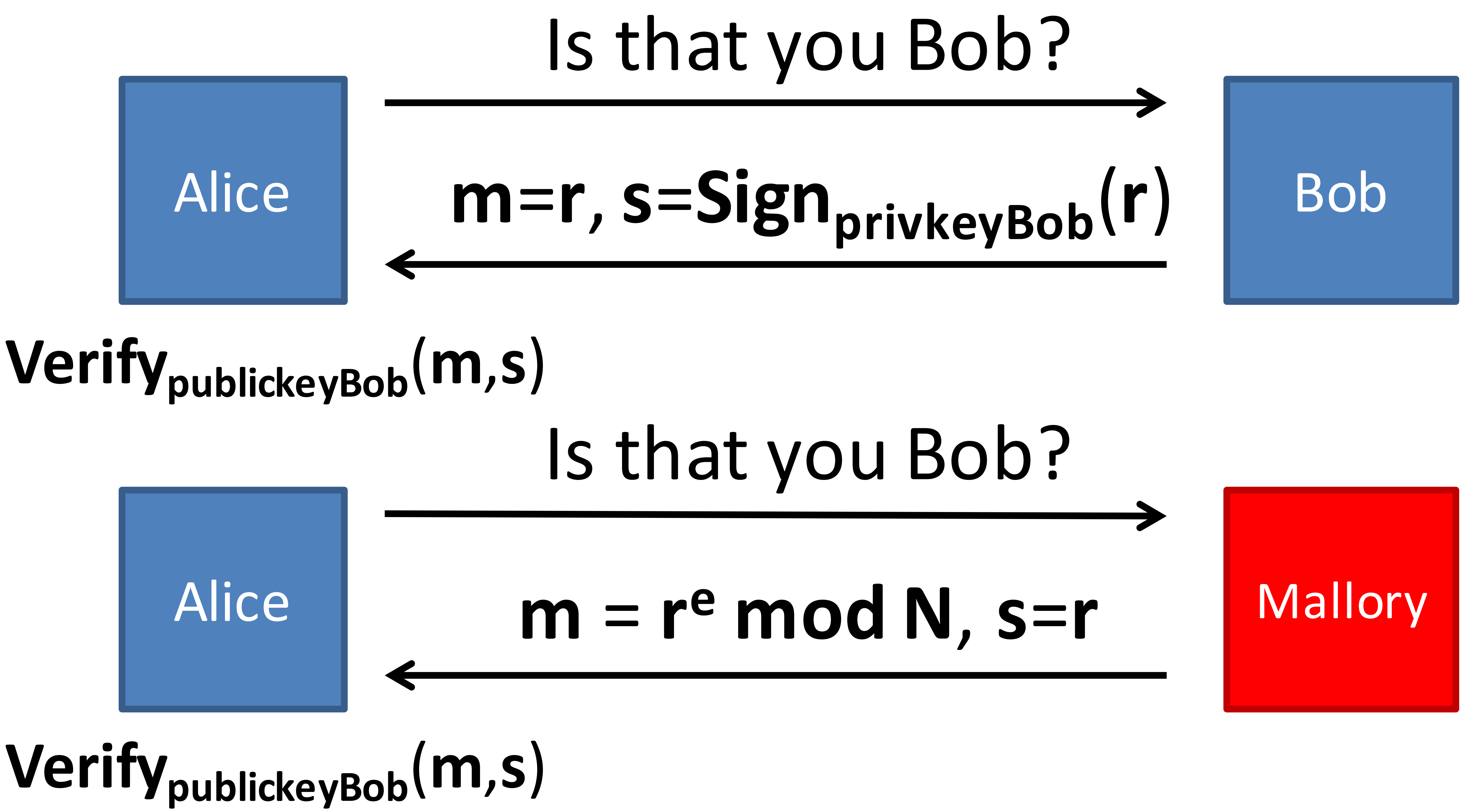
2. Problems during encryption



How can ***Eve*** learn **k**?

RSA Gotchas!

3. Problems with digital signatures:
Forgery when messages are arbitrary.



Alice wants to know if Bob is on the wire. She expects Bob to sign a random message, **r**, that she can verify with his public key.

How could Alice be tricked by ***Mallory***?

RSA Gotchas!

3. **Problems with digital signatures:**
   Forgery of specific messages.

Alice $\xrightarrow{\text{Please sign } \mathbf{m}}$ Bob — Checks **m** is not sensitive.

Bob $\xrightarrow{s=\text{Sign}_{\text{privkeyBob}}(m)}$ Alice

Mallory $\xrightarrow{\text{Please sign } \mathbf{m} = \mathbf{r^e m'}}$ Bob — Checks **m** is not sensitive.

Bob $\xrightarrow{s = \text{Sign}_{\text{privkeyBob}}(m)}$ Mallory

$$
\begin{aligned}
s' &= s/r \bmod N \\
&= (r^e m')^d / r \bmod N \\
&= (m')^d \bmod N \\
&= \text{Sign}(m')
\end{aligned}
$$

RSA ciphertexts are ***malleable***.

Fixes for some RSA Gotchas

1. e should be >= 65537

2. Always use padding.
   (Common standards: PKCS #1, OAEP)

3. Hash-and-sign paradigm:
   s = Signprivkey(H(m))

4. Use different keys for encryption and digital signatures.

5. In practice, never implement RSA yourself. Use respected crypto libraries!

Putting Together a Secure Channel

1. Establish a shared secret **k** using Diffie-Hellman (even if Mallory is there!)

2. Make sure they're really talking to each other by exchanging and verifying RSA signatures on **k** (if Mallory was there, this would fail)

3. Use a PRF, essentially a hash, to split **k** into four distinct keys (integrity and confidentiality in both directions)

4. To encrypt: Encrypt with symmetric cipher, then add MACs for integrity (encrypt, then MAC…)

So Far

Putting together a secure channel (all the parts)

Upcoming…

Begin Web/Network Security Unit

SSL/TLS (HTTPS) and Public-Key Infrastructure