HTTPS Connection (SSL / TLS)

- Browser (client) connects via TCP to Amazon's HTTPS server
- Client picks 224-bit random number $R_B$, sends over list of crypto protocols it supports
- Server picks 224-bit random number $R_S$, selects protocols to use for this session
- Server sends over its certificate
- (all of this is in the clear)
- **Client now validates cert**

```
Hello. My rnd # = R_B. I support (TLS+RSA+AES128+SHA1) or (SSL+RSA+3DES+MD5) or ...
```

```
My rnd # = R_S. Let’s use TLS+RSA+AES128+SHA1
```

```
Here’s my cert
```

~2-3 KB of data
HTTPS Connection (SSL / TLS), con’t

• For RSA, browser constructs long (368 bits) “Premaster Secret” PS

• Browser sends PS encrypted using Amazon’s public RSA key $K_{Amazon}$

• Using PS, $R_B$, and $R_S$, browser & server derive symm. cipher keys ($C_B$, $C_S$) & MAC integrity keys ($I_B$, $I_S$)
  – One pair to use in each direction

These seed a cryptographically strong pseudo-random number generator (PRNG). Then browser & server produce $C_B$, $C_S$, etc., by making repeated calls to the PRNG.
HTTPS Connection (SSL / TLS), con’t

- For RSA, browser constructs long (368 bits) “Premaster Secret” $PS$
- Browser sends PS encrypted using Amazon’s public RSA key $K_{Amazon}$
- Using PS, $R_B$, and $R_S$, browser & server derive symm. cipher keys ($C_B$, $C_S$) & MAC integrity keys ($I_B$, $I_S$)
  - One pair to use in each direction
- Browser & server exchange MACs computed over entire dialog so far
- If good MAC, Browser displays
- All subsequent communication encrypted w/ symmetric cipher (e.g., AES128) cipher keys, MACs
  - Messages also numbered to thwart replay attacks