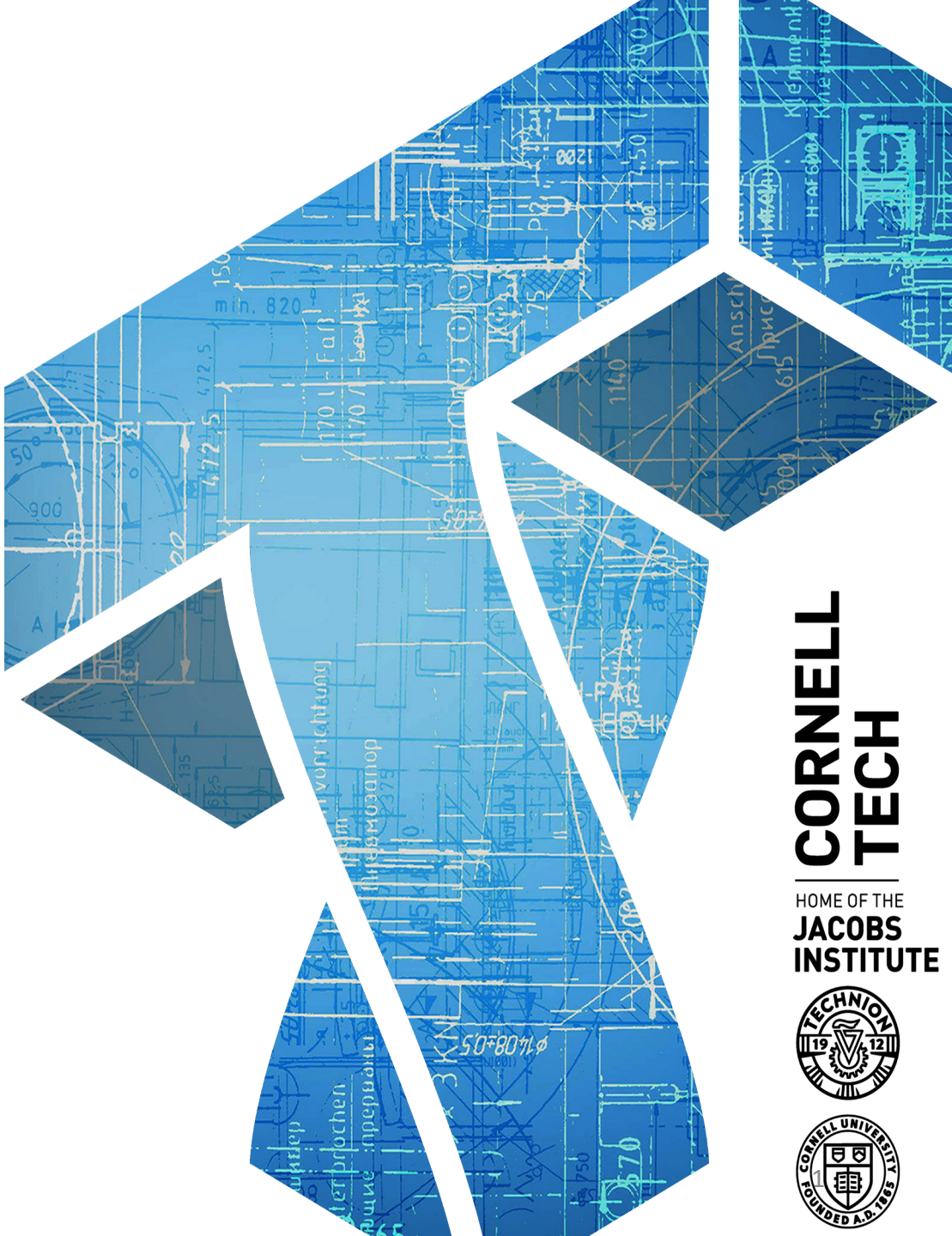
CS 5435: Asymmetric cryptography

Instructor: Tom Ristenpart

https://github.com/tomrist/cs5435-fall2019



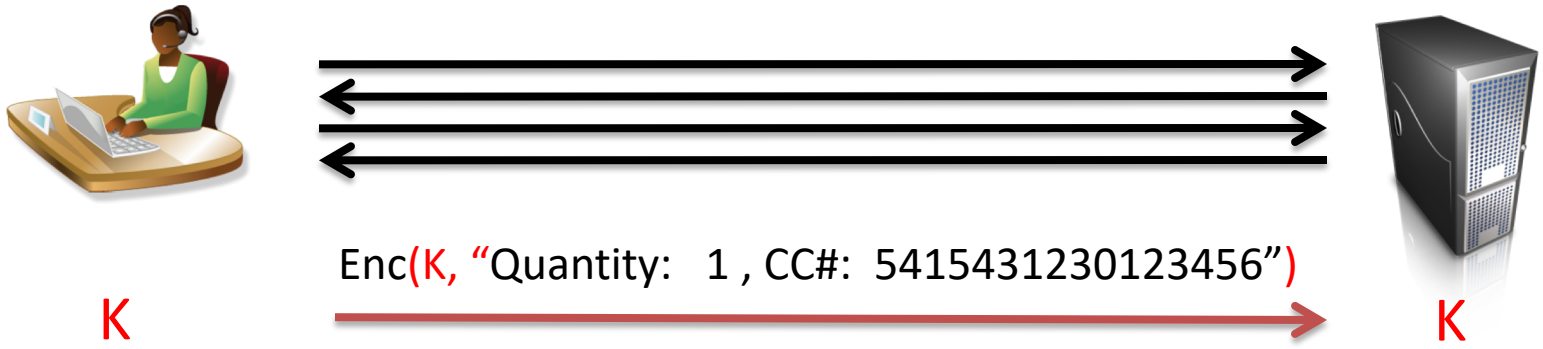
CORNELL TECH
HOME OF THE JACOBS INSTITUTE

Today's lecture

- Key exchange, high level (against passive adversaries)
  - Key transport
- Public-key encryption
- Forward secrecy for key exchange
- Diffie-Hellman groups and computational assumptions (discrete log problem)
- Active man-in-the-middle attacks

- Next time: digital signatures, PKI, & resisting MitM attacks

Recall two steps to secure channels:



Step 1: Key exchange protocol to share secret K

Step 2: Send data via secure channel

Authenticated encryption

Key exchange via public-key encryption

Choose fresh
symmetric key K

C <- Enc(pk,K,R)

pk

C <- Enc(pk,K,R)

Have (pk,sk) pair

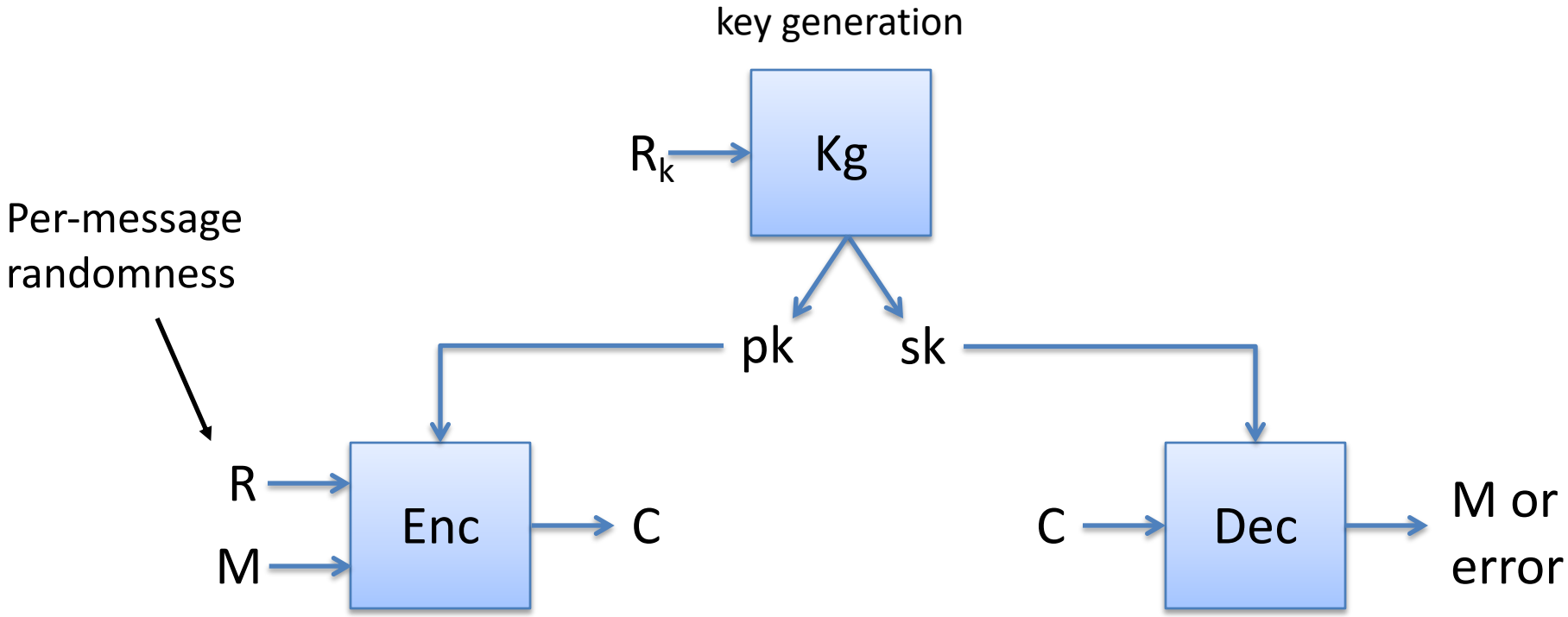
K <- Dec(sk,C)

Server picks long-lived (pk,sk) pair; pk sent to client

Client encrypts a key K using pk and some fresh randomness R

Ciphertext C sent to server; server decrypts using sk

Public-key encryption



Correctness: Dec(sk , Enc(pk,M,R)) = M with probability 1 over random choise of R

RSA trapdoor permutation

pk = (N,e) N = pq for large primes p, q

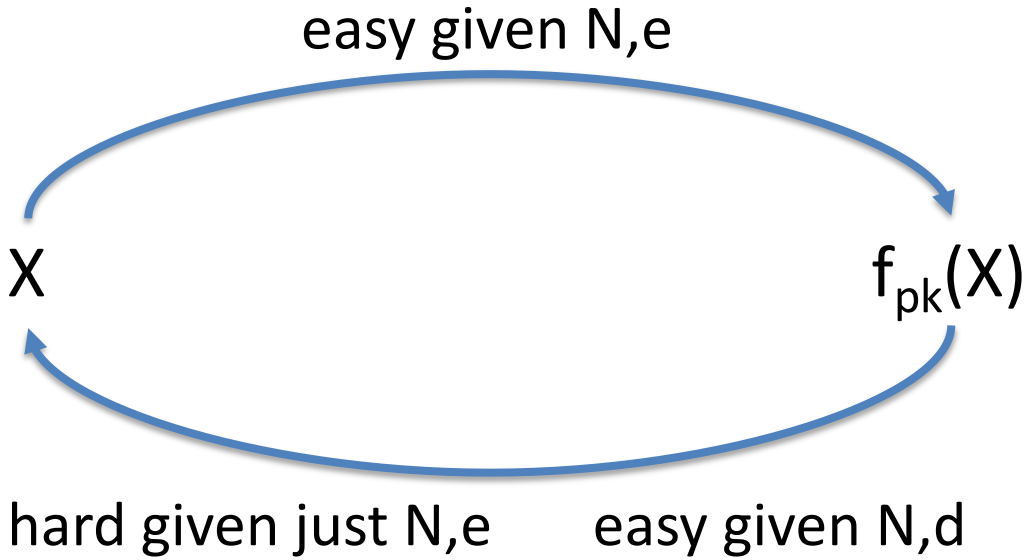
sk = (N,d) e,d chosen so that $x^{ed} \bmod N = x \bmod N$ for all x

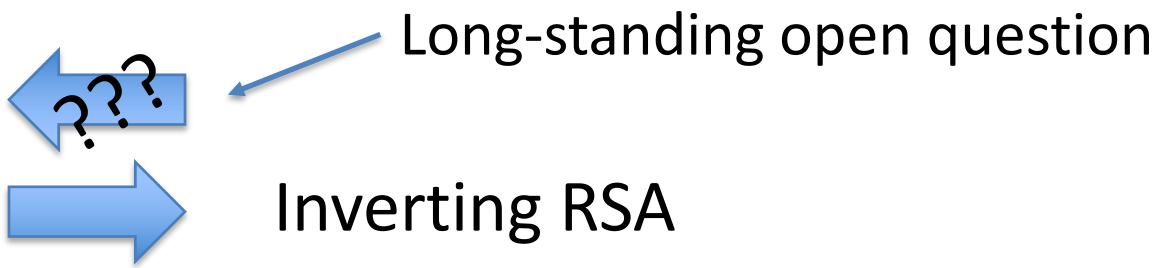
$f_{N,e}(x) = x^e \bmod N$ $g_{N,d}(y) = y^d \bmod N$

Multiply y by itself d times, reducing modulo N
Algorithms can do this in time $O(\log N)$

RSA trapdoor permutation

Conjectured computational difficulty of inverting RSA given only N,e for random value X





Must choose p,q such that N is hard to factor

Factoring composites

- What is p,q for N = 901?
- What is an algorithm for factoring N?

```
Factor(N):
for i = 2 , … ,  sqrt(N) do
    if N mod i = 0 then
            p = i
            q = N / p
            Return (p,q)
```

Woops… we can always factor

But not always efficiently:
Run time is sqrt(N)

Factoring composites

Algorithm	**Time to factor N**
Naïve	$O(e^{0.5 \ln(N)}) = O(\text{sqrt}(N))$
Quadratic sieve (QS)	$O(e^{c})$ $\quad c = d\,(\ln N)^{1/2}\,(\ln \ln N)^{1/2}$
Number Field Sieve (NFS)	$O(e^{c})$ $\quad c = 1.92\,(\ln N)^{1/3}\,(\ln \ln N)^{2/3}$

Factoring records

RSA-x is an RSA challenge modulus of size x bits

Algorithm	Year	Algorithm	Time
RSA-400	1993	QS	830 MIPS years
RSA-478	1994	QS	5000 MIPS years
RSA-515	1999	NFS	8000 MIPS years
RSA-768	2009	NFS	~2.5 years

Nowdays: minimal recommended size is 2048-bit modulus

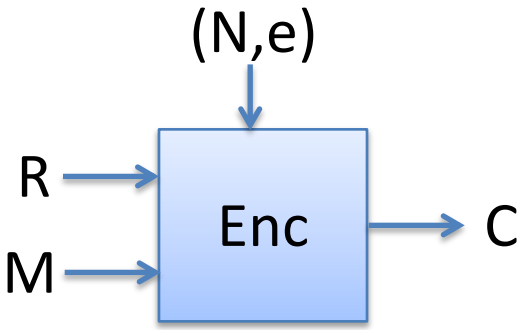
Recall: exponentiation in $O(\log N)$, and so size impacts performance

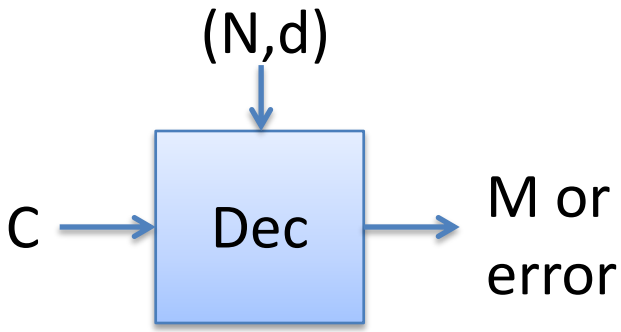
PKCS#1 v1.5 RSA encryption

Kg outputs (N,e),(N,d) where $|N|_8 = n$ (n bytes long)

Let $B = \{0,1\}^8 / \{00\}$ be set of all bytes except 00

Want to encrypt messages of length $|M|_8 = m$





Enc((N,e), M, R)

pad = first n - m - 3 bytes from R that are in B

X = 00 || 02 || pad || 00 || M

Return X^e mod N

Dec((N,d), C)

X = C^d mod N ; aa||bb||w = X

If (aa ≠ 00) or (bb ≠ 02) or (00 ∉ w)

  Return error

pad || 00 || M = w

Return M

Vulnerable to padding oracle attacks!

RSA-OAEP better: secure against chosen-ciphertext attacks

Forward secrecy?

Choose fresh
symmetric key K

C <- Enc(pk,K,R)

pk

C <- Enc(pk,K,R)

Record encrypted
transcript

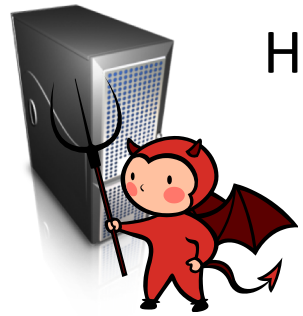
Have (pk,sk) pair

K <- Dec(sk,C)

Sometime later… break in and steal sk

Can adversary recover K? Yes!

We want key exchange protocol that provides ***forward secrecy***: later compromises don't reveal previous sessions



Have (pk,sk) pair

Diffie-Hellman math

Let p be a large prime number
Consider set $Z_p^* = \{1,2,\ldots,p-1\}$ and multiplication modulo p

Fact. There exists $g \in Z_p^*$, called the *generator*, such that

$$Z_p^* = \{ g^0 \bmod p , g^1 \bmod p , g^2 \bmod p , \ldots , g^{p-2} \bmod p\}$$

Example: p = 7. Is 2 or 3 a generator for Z_7^* ?

x	0	1	2	3	4	5	6
$2^x \bmod 7$	1	2	4	1	2	4	1
$3^x \bmod 7$	1	3	2	6	4	5	1

Z_p^* with modular multiplication is just one choice. More generally: cyclic finite group

Diffie-Hellman Key Exchange

g,p,X

Y

Usually g,p fixed, public parameters

Pick random y

$Y = g^y \bmod p$

Pick random x

$X = g^x \bmod p$

$K = H(X^y \bmod p)$

$K = H(Y^x \bmod p)$

Get the same key. Why? $Y^x = g^{yx} = g^{xy} = X^y \bmod p$

TLS handshake for Diffie-Hellman Key Exchange

Client

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Check CERT using CA public verification key
Check σ

← ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

Pick random Ns

Pick random x
$X = g^x \bmod p$

← CERT = (pk_s , signature over it)

← $p , g , X , \quad \sigma = \text{Sign}(sk_s, Nc||Ns||p||g||X)$

Pick random y
$Y = g^y \bmod p$

Y →

$PMS = g^{xy} \bmod p$

$PMS = g^{xy} \bmod p$

ChangeCipherSpec,
{ Finished, PRF(MS, “Client finished” || H(transcript)) } →

ChangeCipherSpec,
{ Finished, PRF(MS, “Server finished” || H(transcript’)) } ←

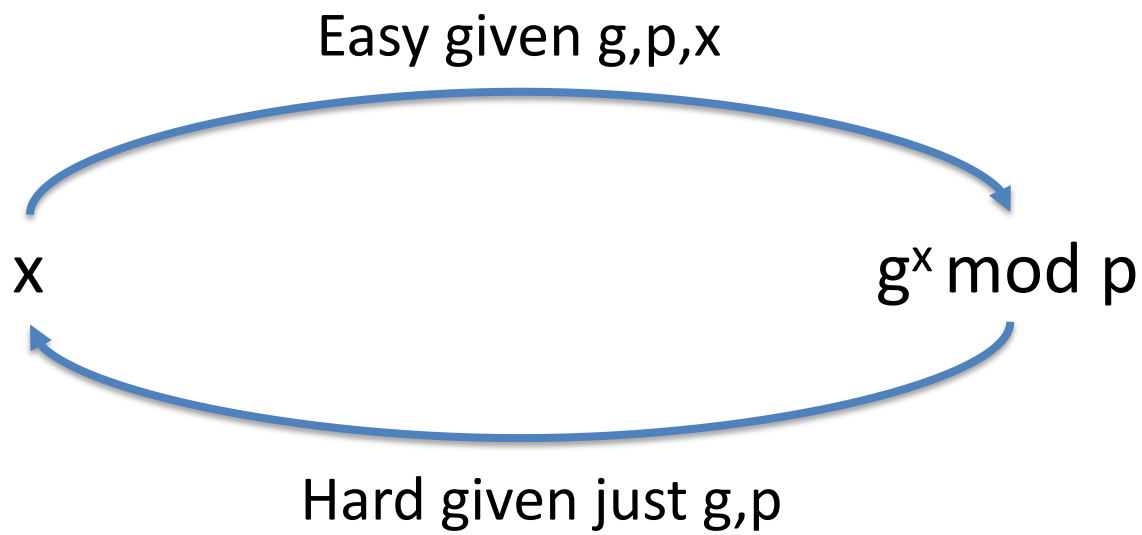
Bracket notation means contents encrypted

MS <- PRF(PMS, “master secret” || Nc || Ns)

The discrete log problem

Pick x at random

Give adversary g, $X = g^x \bmod p$. Adversary's goal is to compute x



The discrete log problem

Pick x at random

Give adversary g, $X = g^x \bmod p$. Adversary's goal is to compute x

$\mathcal{A}(X)$:

for i = 0 , … , p-2 do

  if $X = g^i \bmod p$ then

    Return i

Very slow for large groups!

$O(p)$

Baby-step giant-step is better:

$O(p^{0.5})$

Nothing faster is known for some groups

For Z_p^*, discrete log NFS algorithm with runtime same as factoring NFS

Baby-Step Giant-Step algorithm

- DLP: Given $g^x \bmod p$ for random x, compute x

Think of x as $x = az + b$ with $z = \text{ceil}(p^{0.5})$

$$g^x g^{-az} = g^b \bmod p$$

For b = 1, ..., z

    Store $(b, g^b \bmod p)$

For a = 1, ..., z

    If $g^x g^{-az} \bmod p$ equals one of precomputed $g^b \bmod p$ values

        Return az + b

- Works in time $O(p^{0.5})$ and space $O(p^{0.5})$
- Pollard rho method: reduce space to constant

Diffie-Hellman Key Exchange

Usually g,p fixed, public parameters

g,p,X

Y

Pick random y

$Y = g^y \bmod p$

Pick random x

$X = g^x \bmod p$

$K = H(X^y \bmod p)$

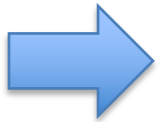
$K = H(Y^x \bmod p)$

Solving discrete log breaks DH key exchange

Could there be other ways of breaking?

Computational Diffie-Hellman (DH) Problem

Pick x,y at random
Give adversary g , $X = g^x \bmod p$, $Y = g^y \bmod p$
Adversary's goal is to compute $g^{xy} \bmod p$

Solving discrete log  Solving DH

For cryptographically strong groups that we use:
best known DH solver is discrete log solver

Asymmetric primitives

Security level	RSA size (log N)	DLP in finite field (log p)	ECC group size (log p)
80	1024	1024	160
112	2048	2048	224
128	3072	3072	256
256	15360	15360	512

Elliptic curve cryptography (ECC) uses cyclic subgroups of set of solutions to elliptic curves of size prime p

Best known attack is $O(p^{0.5})$

TLS handshake for Diffie-Hellman Key Exchange

Client

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

← ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

Pick random Ns

Check CERT using CA public verification key

Check σ

← CERT = (pk_s , signature over it)

← p , g , X , σ = Sign(sk_s, p || g || X)

Pick random x

$X = g^x$

Pick random y

$Y = g^y$

Y →

$PMS = g^{xy}$

$PMS = g^{xy}$

ChangeCipherSpec,

{ Finished, PRF(MS, “Client finished” || H(transcript)) } →

ChangeCipherSpec,

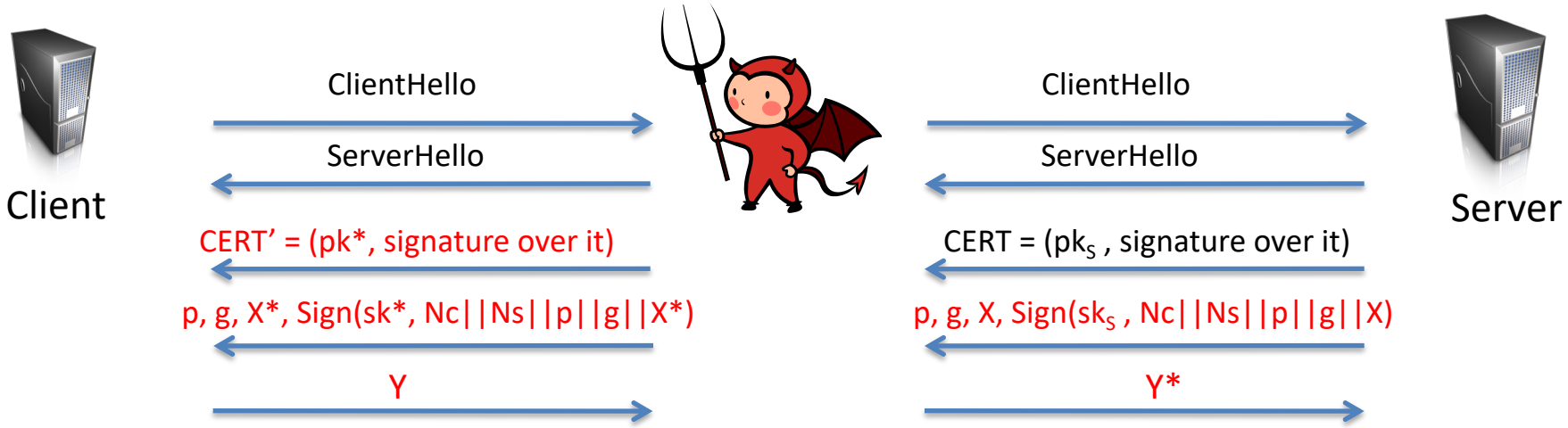
{ Finished, PRF(MS, “Server finished” || H(transcript’)) } ←

Bracket notation means contents encrypted

MS <- PRF(PMS, “master secret” || Nc || Ns)

Man-in-the-middle attacks

Suppose authentication vulnerability:
CERT can be forged, Client doesn't check CERT, etc.



Attacker can choose X*, Y*, so it knows discrete logs
Completes handshake on both sides
Client thinks its talking to Server
All communications decrypted by adversary, re-encrypted and forwarded to server

Next lecture

- Digital signatures
- Public key infrastructure