CS 5435: Asymmetric cryptography

Instructor: Tom Ristenpart

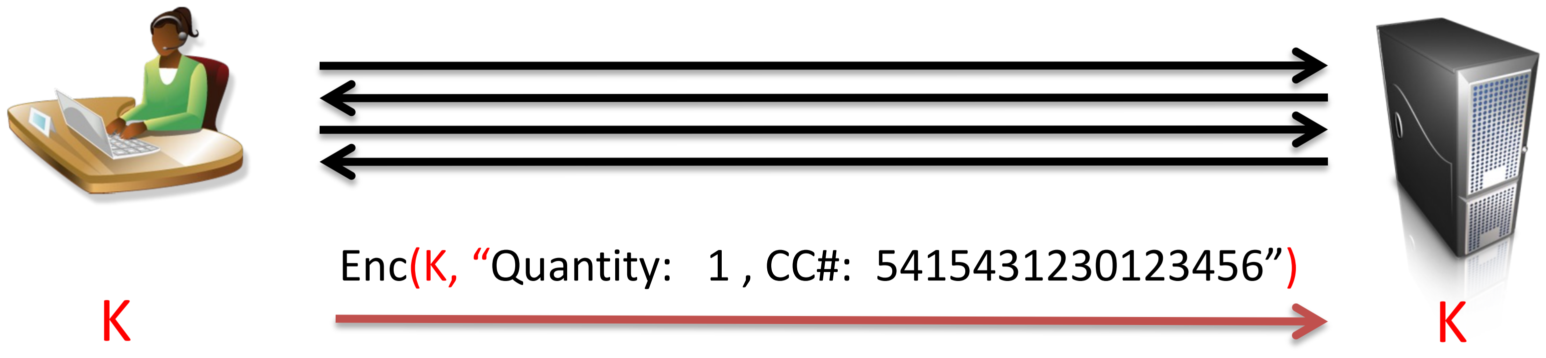
https://github.com/tomrist/cs5435-spring2024

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

Have (pk,sk) pair

pk ←

C <- Enc(pk,K,R) →

Choose fresh
symmetric key K

C <- Enc(pk,K,R)

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

TLS 1.2 handshake for RSA transport

Client

Server

Server has long-lived (pk,sk) pair

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

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

Pick random Ns

← CERT = (pk , signature over it)

Check CERT using CA public verification key

Pick random PMS
C <- Enc(pk,PMS)

C →

PMS <- Dec(sk,C)

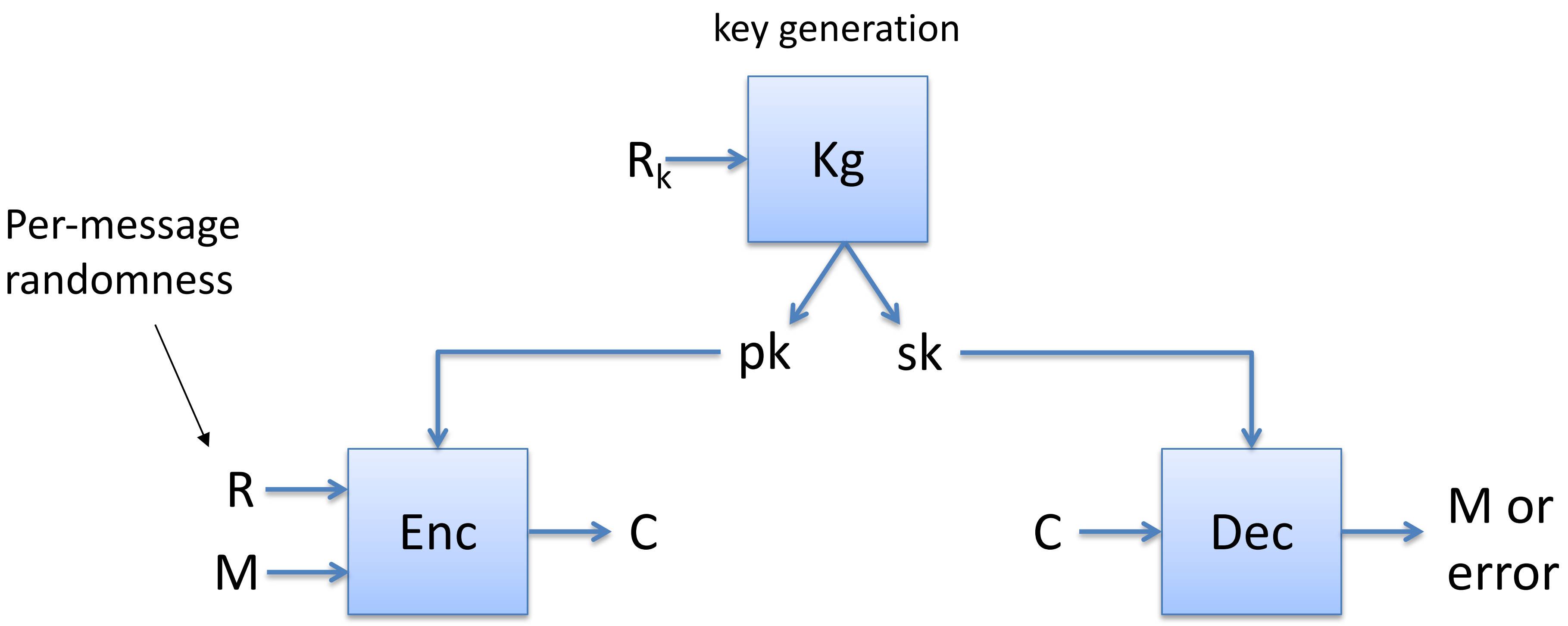
ChangeCipherSpec,
{ Finished, PRF(MS, “Client finished” || H(transcript)) } →

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

Bracket notation means contents encrypted

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

Public-key encryption



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

easy given N,e

X $\qquad\qquad$ $f_{pk}(X)$

hard given just N,e $\qquad$ easy given N,d

Long-standing open question

???

Factoring N into p,q reveals secret d $\Rightarrow$ Inverting RSA

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	$\mathcal{O}(e^{0.5 \ln(N)}) = \mathcal{O}(\mathrm{sqrt}(N))$
Quadratic sieve (QS)	$\mathcal{O}(e^{c})$ $c = d (\ln N)^{1/2} (\ln \ln N)^{1/2}$
Number Field Sieve (NFS)	$\mathcal{O}(e^{c})$ $c = 1.92 (\ln N)^{1/3} (\ln \ln N)^{2/3}$

Factoring records

Challenge	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
RSA-512	2015	NFS	$75 on EC2 / 4 hours
RSA-795	2019	NFS	4000 core-years (Xeon Gold 6130 CPU as reference)
RSA-829	2020	NFS	2700 core-years (same as above)

RSA-x is an RSA challenge modulus of size x bits
MIPS = million instructions per second
Recent academic paper: https://eprint.iacr.org/2020/697.pdf

Raw RSA example: QQ Browser circa 2018

- QQ browser popular in China, 100s millions of users
- Server chooses 1024-bit RSA key (N,e),(N,d).
- To send message M to server:

QQ servers

pk = (N,e) ←

C , C' →

OK ←

Client:

$K \leftarrow_{\$} \{0,1\}^{128}$

$C = K^e \bmod N$

$C' = \mathrm{Enc}(K,M)$

Server:

$X = C^d \bmod N$

$K' = \mathrm{Low128bits}(X)$

If Dec(K',C') fails then Ret FAIL

Ret OK

https://citizenlab.ca/2016/03/privacy-security-issues-qq-browser/ https://arxiv.org/pdf/1802.03367.pdf

An insecure example: QQ Browser circa 2018

$C = K^e \bmod N$ $\qquad$ $C' = \mathrm{Enc}(K, M^*)$

$C1 = 2^{127e}\, C \quad \bmod N$

$C1' = \mathrm{Enc}(10^{127}, M)$

$C1\ ,\ C1'$ →

OK ←

$C2\ ,\ C2'$ →

FAIL ←

⋮

$X = C^d \bmod N$

$K' = \mathrm{Low128bits}(X)$

If $\mathrm{Dec}(K',C')$ fails then Ret FAIL

Ret OK

First bit of K is 1 if return OK

First bit of K is 0 if return FAIL

$C2 = 2^{126e}\, C \quad \bmod N$

$C2' = \mathrm{Enc}(110^{126}, M)$

⋮

$K = \ldots 01$

Recover full key in 128 queries

https://citizenlab.ca/2016/03/privacy-security-issues-qq-browser/ https://arxiv.org/pdf/1802.03367.pdf

An insecure example: QQ Browser circa 2018

So many problems!

- Earlier version: used RSA with 128 bit modulus

  245406417573740884710047745869965023463

- Used ms-precision timestamp as randomness source to generate K
- Responses to requests actually didn't use K, used hard-coded key K*

https://citizenlab.ca/2016/03/privacy-security-issues-qq-browser/

Typical vulnerabilities using RSA

Direct use of “raw” RSA for anything except well-studied mode of operation

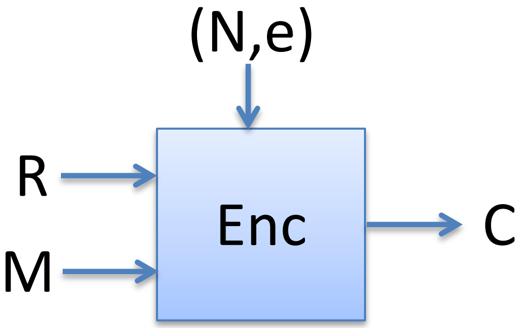
Using too-small modulus size (<2048 these days)

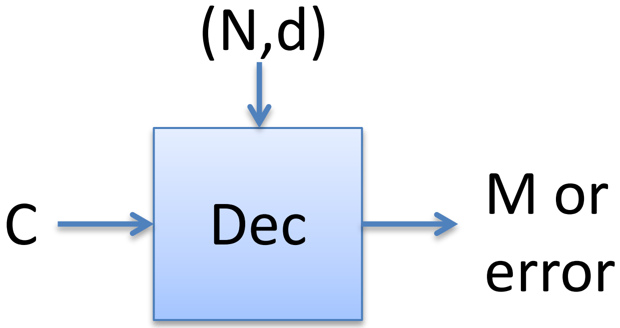
PKCS#1 v1.5 RSA encryption

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 $|M|_8 = m$





<u>Enc((N,e), M, R)</u>

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

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

Return X^e mod N

<u>Dec((N,d), C)</u>

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

Bleichenbacher attack

I've just learned some information about C_1^d mod N

C_1 →

← padding error?

C_2 →

← padding error?

…

$\underline{\text{Dec((N,d), C)}}$

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

If (aa ≠ 00) or (bb ≠ 02) or ($00 \notin$ w)

    Return error

pad || 00 || M = w

Return M

We can take a target C and decrypt it using a sequence of chosen ciphertexts C_1, … , C_q where q ≈ 1 million

[Bardou et al. 2012] q = 9400 ciphertexts on average

Bleichenbacher attack

Given ciphertext C,
learn $X = C^d \bmod N$

C_1

OK

<u>Dec((N,d), C)</u>
$X' = C^d \bmod N$; $aa||bb||w = X'$
If $(aa \neq 00)$ or $(bb \neq 02)$ or $(00 \notin w)$
    Return FAIL
$pad \,||\, 00 \,||\, M = w$
Return OK

$C_1 = C\, s1^e \bmod N$

<u>Response OK:</u>
$X' = (C\, s1^e)^d \bmod N = X\, s1 \bmod N$
So we know that:
$2*2^{8(n-2)} \leq X*s1 \bmod N < 3*2^{8(n-2)}$

Leaks some information about X!

Bleichenbacher attack

Given ciphertext C, learn $X = C^d \bmod N$

$C_1 = C\, s1^e \bmod N$

$C_2 = C\, s2^e \bmod N$

⋮

C_1 →

← padding error?

C_2 →

← padding error?

…

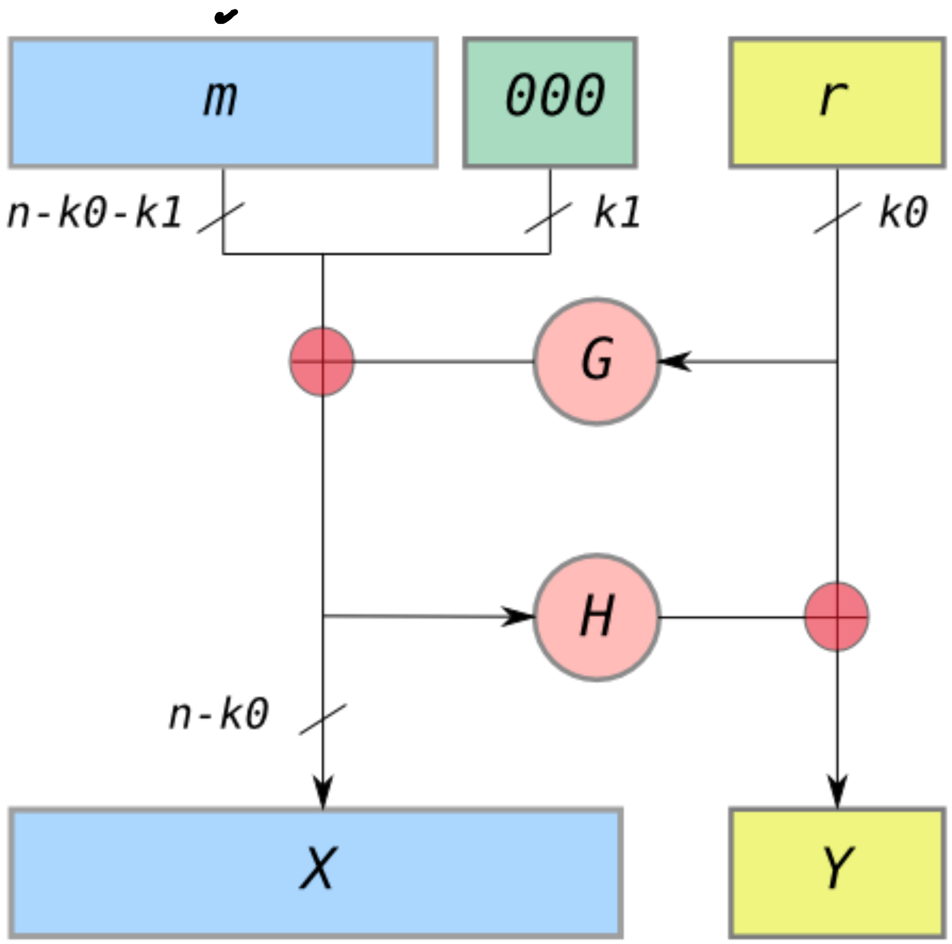
<u>Dec((N,d), C)</u>
$X' = C^d \bmod N$; $aa||bb||w = X'$
If $(aa \neq 00)$ or $(bb \neq 02)$ or $(00 \notin w)$
    Return FAIL
pad || 00 || M = w
Return OK

We can take a target C and decrypt it using a sequence of carefully chosen ciphertexts $C_1, \ldots, C_q$ where $q \approx 1$ million

[Bardou et al. 2012] q = 9400 ciphertexts on average

RSA-OAEP (optimal asymmetric encryption padding)

- Provide better padding scheme than PKCS#1v1.5
- OAEP is such a padding scheme
  - r chosen randomly
  - G,H hash functions
  - $C = (X||Y)^e \bmod N$
- RSA one-wayness implies CCA security



Forward secrecy?

Choose fresh
symmetric key K

C <- Enc(pk,K,R)

pk

C <- Enc(pk,K,R)

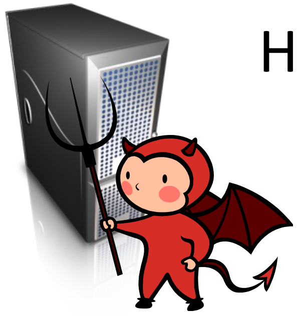
Record encrypted
transcript

Have (pk,sk) pair

K <- Dec(sk,C)

Sometime later… break in and steal sk

Can adversary recover K? Yes!



Have (pk,sk) pair

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

Can we build RSA-based key exchange with forward-secrecy?

Generate ***fresh*** RSA (pk,sk) pair

pk ←

C <- Enc(pk,K,R) →

Choose fresh symmetric key K

C <- Enc(pk,K,R)

Record encrypted transcript

K <- Dec(sk,C)

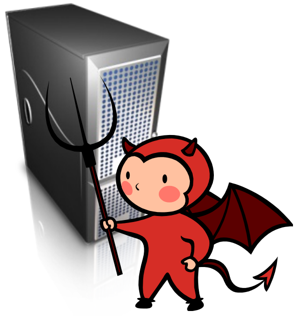
Delete sk

Sometime later… break in ~~and steal sk~~

Can adversary recover K? Nope!

We don’t use this approach in practice, why?

Performance: RSA key generation is pretty slow



Diffie-Hellman math

Let p be a large prime number
Consider set $Z_p^* = \{1,2,\ldots,p\text{-}1\}$ 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 mod 7	1	2	4	1	2	4	1
3^x mod 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

Usually g,p fixed, public parameters

$$\xleftarrow{g,p,X}$$

$$\xrightarrow{Y}$$

Pick random y

$Y = g^y \mod p$

Pick random x

$X = g^x \mod p$

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

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

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

TLS handshake for Diffie-Hellman Key Exchange

Client

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Pick random Ns

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

Pick random x

$X = g^x \bmod p$

← CERT = (pk_s , signature over it)

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

Check CERT using CA public verification key

Check σ

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

Easy given g,p,x

x $g^x \bmod p$

Hard given just g,p

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 mod p then
       Return i
```

Very slow for large groups!

$\mathcal{O}(p)$

Baby-step giant-step is better:

$\mathcal{O}(p^{0.5})$

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 mod p for random x, compute x

```
Think of x as x = az + b  with z = ceil(p^0.5)
                g^x g^-az   =   g^b  mod p
For b = 1, ..., z
     Store (b,g^b mod p)
For a = 1, ..., z
     If g^x g^-az mod p equals one of precomputed g^b mod 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 $\Rightarrow$ 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 $O(p^{0.5})$

Client

TLS handshake for Diffie-Hellman Key Exchange

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Pick random Ns

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

Pick random x

$X = g^x$

← CERT = (pk_s , signature over it)

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

~~Check CERT using CA public verification key~~

~~Check σ~~

Pick random y

$Y = g^y$

Y →

PMS = g^{xy}

PMS = g^{xy}

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

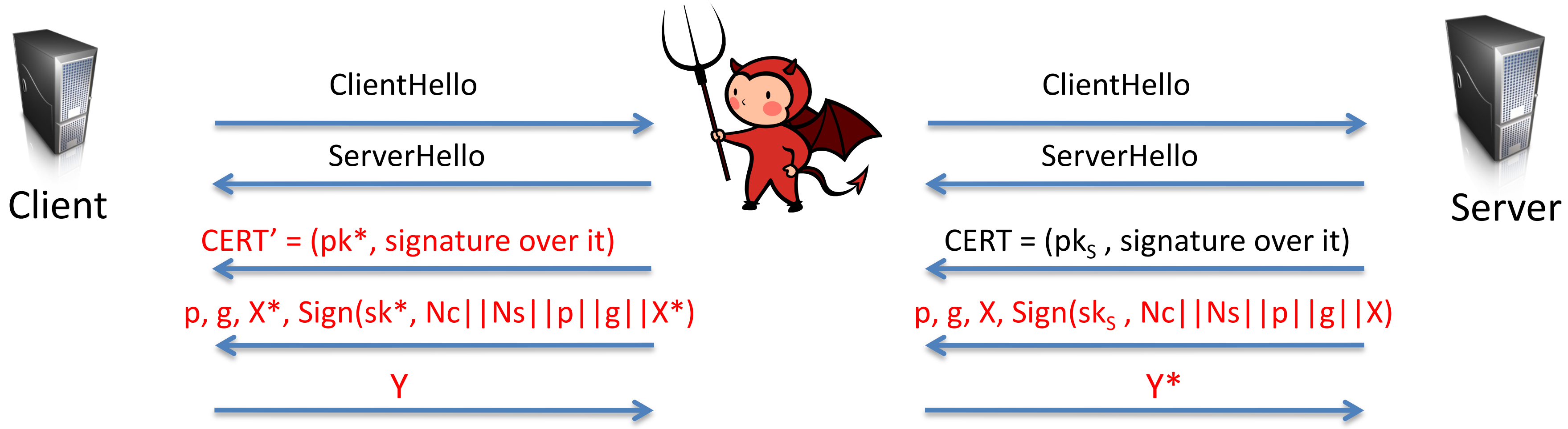
Bracket notation means contents encrypted

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

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

RSA math

Let N be a positive number

Looking ahead: $N = pq$ for large primes p,q

N will be called the modulus

$p = 7$, $q = 13$, gives $N = 91$

$p = 17$, $q = 53$, gives $N = 901$

RSA math

Let N be a positive number
Looking ahead: N = pq for large primes p,q
N will be called the modulus

$\mathbf{Z}_N = \{0,1,2,3,\ldots, N-1\}$

$\mathbf{Z}_N^* = \{ i \mid \gcd(i,N) = 1 \text{ and } i < N\}$

$\gcd(X,Y) = 1$ if greatest common divisor of X,Y is 1

RSA math

$\mathbf{Z}_N^* = \{ i \mid \gcd(i,N) = 1 \}$

N = 13 $\mathbf{Z}_{13}^* = \{ 1,2,3,4,5,6,7,8,9,10,11,12 \}$

N = 15 $\mathbf{Z}_{15}^* = \{ 1,2,4,7,8,11,13,14 \}$

The size of a set S is denoted by |S|

Def. $\phi(N) = |\mathbf{Z}_N^*|$ (This is Euler's totient function)

$\phi(13) = 12$

$\phi(15) = 8$

$\mathbf{Z}_{\phi(15)}^* = \mathbf{Z}_8^* = \{ 1,3,5,7 \}$

RSA math

$\mathbf{Z}_N^* = \{ i \mid \gcd(i,N) = 1 \}$

Fact. For any a,N with N > 0, there exists unique q,r such that

$$a = Nq + r \qquad \text{and} \qquad 0 \leq r < N$$

$$17 \bmod 15 = 2 \qquad\qquad 105 \bmod 15 = 0$$

Def. $a \bmod N = r \in \mathbf{Z}_N$

Def. $a \equiv b \pmod N$ iff $(a \bmod N) = (b \bmod N)$

Operations work in natural way:

$$a \bullet b \bmod N \qquad\qquad a+b \bmod N$$

RSA math

$\mathbf{Z}_N^*$ = { i | gcd(i,N) = 1 }

($\mathbf{Z}_N^*$, •) is a **group** where • denotes multiplication mod N

Group (G,•) is a set G and operator • that satisfy:

1. *Closure*: for all a,b ∈ G it holds that a•b ∈ G
2. *Associativity*: for all a,b,c ∈ G it holds that a•(b•c) = (a•b)•c
3. *Identity*: Exists I ∈ G s.t. for all a ∈ G a•I = a
4. *Inverses*: for a ∈ G there exists a^{-1} ∈ G s.t. $a \bullet a^{-1} = \mathrm{I}$

Abelian group is additionally commutative:

for all a,b ∈ G it holds that a•b = b•a

RSA math

$\mathbf{Z}_N^* = \{ i \mid \gcd(i,N) = 1 \}$

$(\mathbf{Z}_N^*, \bullet)$ is a **group**

$$\mathbf{Z}_{15}^* = \{ 1,2,4,7,8,11,13,14 \}$$

$2 \bullet 7 \equiv 14 \pmod{15}$

$4 \bullet 8 \equiv 2 \pmod{15}$

Closure: for any $a,b \in \mathbf{Z}_N^*$ $\quad a \bullet b \bmod N \in \mathbf{Z}_N^*$

Def. $\quad a^i \bmod N = \underbrace{a \bullet a \bullet a \bullet \ldots \bullet a}_{i \text{ times}} \bmod N$

Some needed algorithms

Algorithm	**Running time (n = log N)**
Modular multiplication $a \bullet b \bmod N$	$\mathcal{O}(n^2)$
Modular exponentation $a^i \bmod N$	$\mathcal{O}(n^3)$
Modular inverse $a^{-1} \bmod N$	$\mathcal{O}(n^2)$

Textbook exponentiation

How do we compute $h^x \bmod N$?

```
Exp(h,x,N)
X' = h
For i = 2 to x  do
      X' = X'•h mod N
Return X'
```

Requires time $O(|G|)$ in worst case.

```
SqrAndMulExp(h,x,N)
b_k,…,b_0 = x
f = 1
For i = k down to 0 do
        f = f•f mod N
        If b_i = 1 then
              f = f•h mod N
Return f
```

Requires time $O(k)$ multiplies and squares in worst case.

Notice these algorithms actually work for ***any group***

```
SqrAndMulExp(h,x,N)
b_k,…,b_0 = x
f = 1
For i = k down to 0 do
        f = f•f mod N
        If b_i = 1 then
                f = f•h mod N
Return f
```

$$x = \sum_{b_i \neq 0} 2^i$$

$$h^x = h^{\sum_{b_i \neq 0} 2^i} = \prod_{b_i \neq 0} h^{2^i}$$

$$h^{11} = h^{8+2+1} = h^8 \bullet h^2 \bullet h$$

$b_3 = 1$ $\quad f_3 = 1 \bullet h$

$b_2 = 0$ $\quad f_2 = h^2$

$b_1 = 1$ $\quad f_1 = (h^2)^2 \bullet h$

$b_0 = 1$ $\quad f_0 = (h^4 \bullet h)^2 \bullet h = h^8 \bullet h^2 \bullet h$

Don't implement this algorithm: side-channel attacks

RSA math

$\mathbf{Z}_N^* = \{ i \mid \gcd(i,N) = 1 \}$

Claim: Suppose $e,d \in \mathbf{Z}_{\phi(N)}^*$ satisfying $ed \bmod \phi(N) = 1$
then for any $x \in \mathbf{Z}_N^*$ we have that

$$(x^e)^d \bmod N = x$$

$$\begin{aligned}(x^e)^d \bmod N &= x^{1 + k\,\phi(N)} \bmod N \\ &= x^1\, x^{k\,\phi(N)} \bmod N \\ &= x \bmod N\end{aligned}$$

k is some positive integer

Last equality is
by Euler's Theorem:
$x^{\phi(N)} \bmod N = 1 \bmod N$

RSA math

$\mathbf{Z}_N^* = \{ i \mid \gcd(i,N) = 1 \}$

Claim: Suppose $e,d \in \mathbf{Z}_{\phi(N)}^*$ satisfying $ed \bmod \phi(N) = 1$ then for any $x \in \mathbf{Z}_N^*$ we have that

$$(x^e)^d \bmod N = x$$

$\mathbf{Z}_{15}^* = \{ 1,2,4,7,8,11,13,14 \}$ $\quad\quad$ $\mathbf{Z}_{\phi(15)}^* = \{ 1,3,5,7 \}$

$e = 3$, $d = 3$ gives $ed \bmod 8 = 1$

x	1	2	4	7	8	11	13	14
x^3 mod 15	1	8	4	13	2	11	7	14
y^3 mod 15	1	2	4	7	8	11	13	14

The RSA trapdoor permutation

pk = (N,e)   sk = (N,d)   with ed mod $\phi(N) = 1$

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

$$\begin{tikzcd} X \arrow[rr, bend left, "\text{easy given N,e}"] && f_{pk}(X) \arrow[ll, bend left, "\begin{array}{c}\text{hard given N,e}\\ \text{easy given N,d}\end{array}"] \end{tikzcd}$$

The RSA trapdoor permutation

pk = (N,e)   sk = (N,d)   with ed mod $\phi(N) = 1$

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

But how do we find suitable N,e,d ?

If p,q distinct primes and N = pq then $\phi(N) = (p-1)(q-1)$

Why?

$$
\begin{aligned}
\phi(N) &= |\{1,\ldots,N-1\}| - |\{ip : 1 \leq i \leq q-1\}| - |\{iq : 1 \leq i \leq p-1\}| \\
&= N-1 - (q-1) - (p-1) \\
&= pq - p - q + 1 \\
&= (p-1)(q-1)
\end{aligned}
$$

The RSA trapdoor permutation

pk = (N,e) sk = (N,d) with ed mod $\phi(N) = 1$

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

But how do we find suitable N,e,d ?

If p,q distinct primes and N = pq then $\phi(N) = (p-1)(q-1)$

Given $\phi(N)$, choose $e \in \mathbf{Z}^*_{\phi(N)}$ and calculate

$d = e^{-1} \bmod \phi(N)$

How to find suitable p,q prime?

Choose random numbers and test primality (Miller-Rabin testing)

https://eprint.iacr.org/2018/749.pdf