Today in Cryptography (5830)

RSA Recap
Active attacks against RSA PKCS#1 RSA encryption
Diffie-Hellman key exchange

Client

TLS handshake for RSA transport

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Pick random Ns

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

Check CERT
using CA public
verification key

← CERT = (pk of bank, signature over it)

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

C →

PMS <- D(sk,C)

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)

The RSA trapdoor permutation

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

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

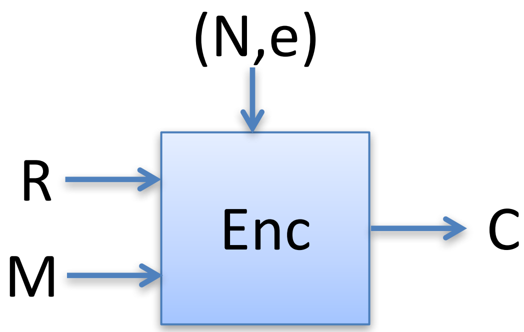
$$\begin{tikzcd}
X \arrow[rr, bend left, "\text{easy given } N{,}e"] & & f_{pk}(X) \arrow[ll, bend left, "\begin{array}{c}\text{hard given } N{,}e \\ \text{easy given } N{,}d\end{array}"]
\end{tikzcd}$$

PKCS #1 RSA encryption

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

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

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

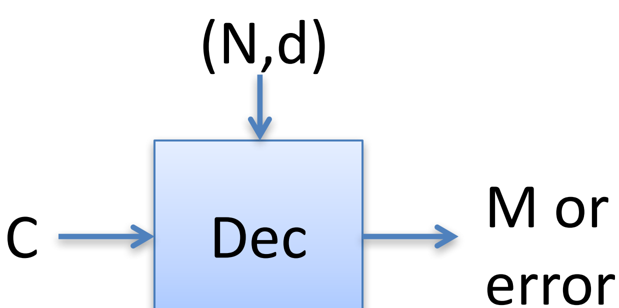


<u>Enc((N,e), M, R)</u>

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>

$X = C^d \bmod N$; aa||bb||w = X

If (aa ≠ 00) or (bb ≠ 02) or (00 ∉ w)

Return error

pad || 00 || M = w

Return M

Security of RSA PKCS#1

- Passive adversary sees (N,e),C
- Attacker would like to invert C
- Attacks?
  - Key generation failures
  - Active attacks

TLS handshake for RSA transport

Bank customer

Bank

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods

Pick random Ns

ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

Check CERT using CA public verification key

CERT = (pk of bank, signature over it)

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

C

PMS <- D(sk,C)

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

Bracket notation means contents encrypted

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

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

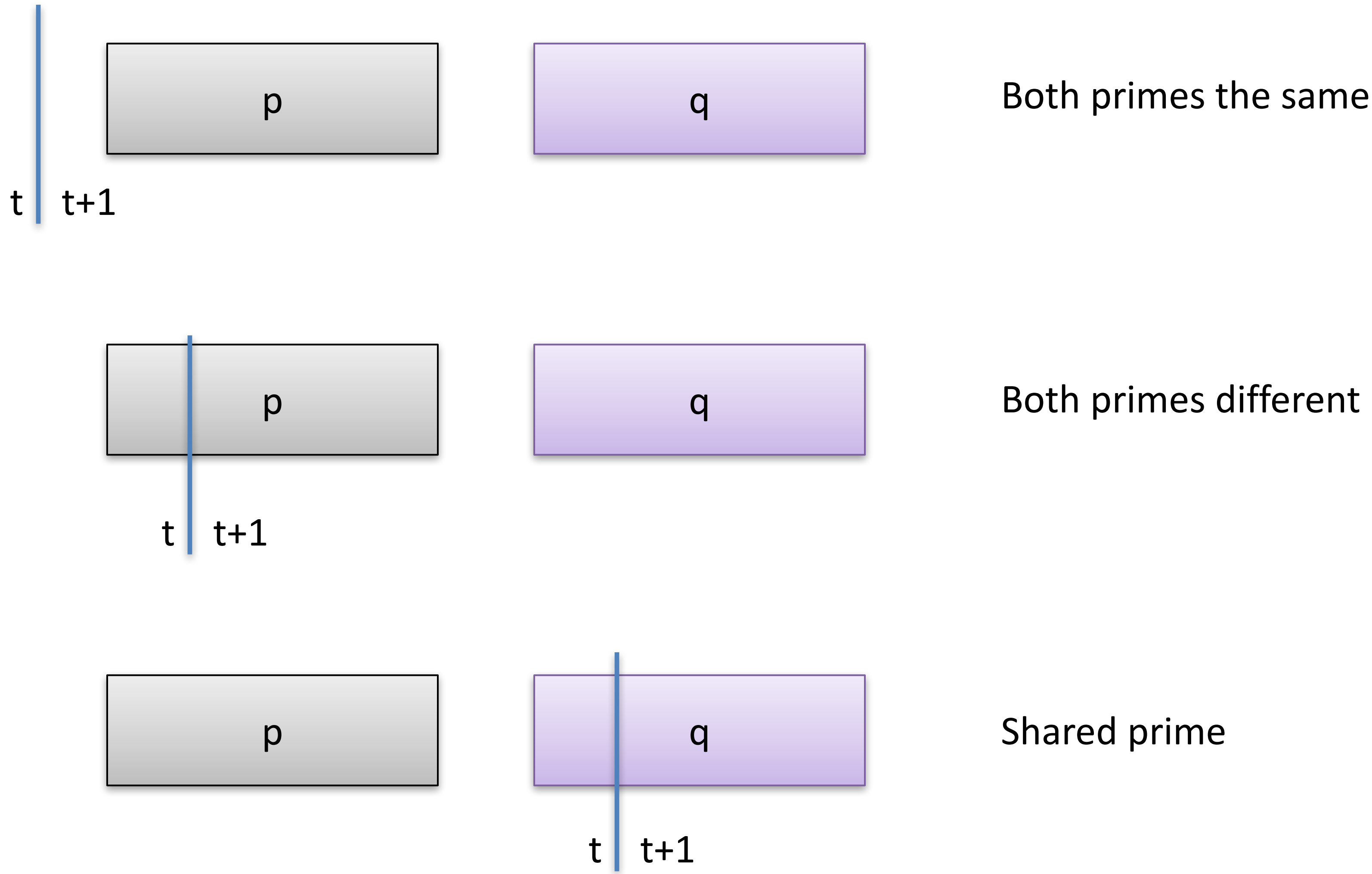
RSA key generation summary

- Find 2 large primes p, q . Let N = pq
  - random integers + primality testing
- Choose e (usually 65,537)
  - Compute d using $\phi(N) = (p-1)(q-1)$
- pk = (N,e) and sk = (N,d)

Weak RSA keys

- Factoring is hard for large key sizes (>=1024)
- But what could go wrong in key generation?
- Reuse p and q values accidentally
- Reuse p with different q:
    - Ex: $N1 = p * q$ $N2 = p * q'$
    - Compute GCD of large integers in milliseconds
    - Use Bernstein's all-pairs GCD to scale up

RNGs and RSA key generation



Weak keys

	Our TLS Scan	
Number of live hosts	12,828,613	(100.00%)
...using repeated keys	7,770,232	(60.50%)
...using vulnerable repeated keys	714,243	(5.57%)
...using default certificates or default keys	670,391	(5.23%)
...using low-entropy repeated keys	43,852	(0.34%)
...using RSA keys we could factor	64,081	(0.50%)
...using DSA keys we could compromise		
...using Debian weak keys	4,147	(0.03%)
...using 512-bit RSA keys	123,038	(0.96%)
...identified as a vulnerable device model	985,031	(7.68%)
...model using low-entropy repeated keys	314,640	(2.45%)

From [Heninger et al. 2012]

Client

TLS handshake for RSA transport

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Pick random Ns

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

Check CERT using CA public verification key

← CERT = (pk of bank, signature over it)

Pick random PMS

C <- E(pk,PMS)

C →

PMS <- D(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)

Security of RSA PKCS#1

- Passive adversary sees (N,e),C
- Attacker would like to invert C
- Attacks?
  - Key generation failures
  - Active attacks

Bleichanbacher attack

C_1

padding error?

I've just learned some information about $C_1^d \bmod N$

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 error

$pad \,||\, 00 \,||\, M = w$

Return M

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

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

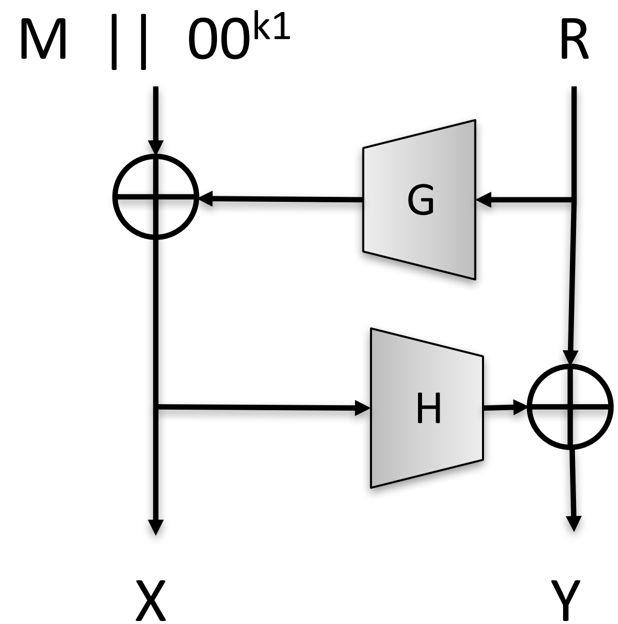
Response to this attack

- Ad-hoc fix: Don't leak whether padding was wrong or not
  - This is harder than it looks (timing attacks, control-flow side channel attacks, etc.)
  - What was used in TLS 1.0, 1.1, 1.2, XML encryption, elsewhere
- Better:
  - use scheme secure against chosen-ciphertext attacks
  - OAEP is common choice

OAEP
[Bellare, Rogaway, 1994]

(optimal asymmetric encryption padding)

<u>Enc((N,e), M, R)</u>
$X = G(R) \oplus M||00^{k1}$
$Y = H(X) \oplus R$
Return $(X||Y)^e \bmod N$



R is k2 random padding bytes
k1 = n – k2 - |M| (in bytes)
G,H are hash functions

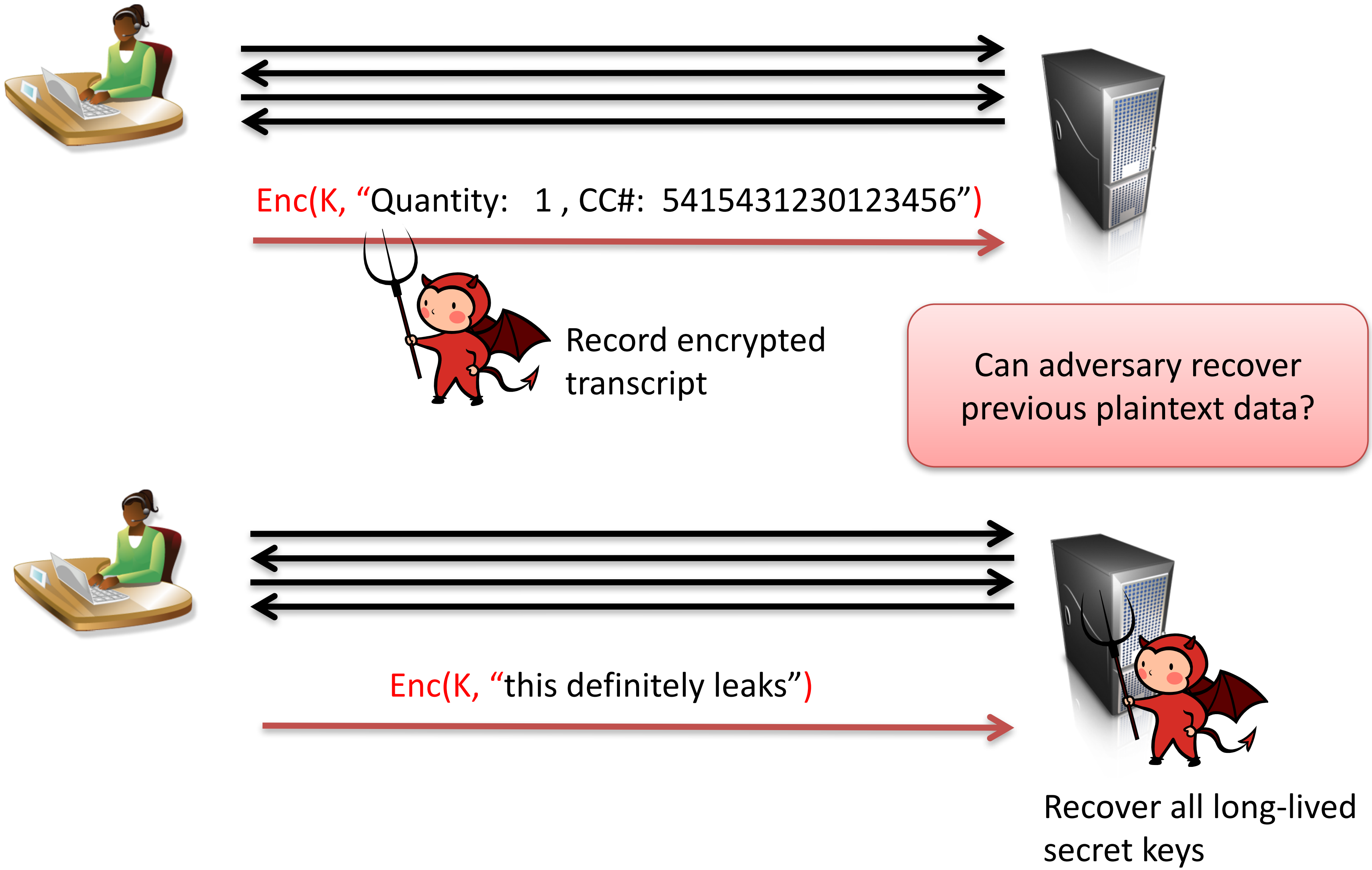
Basically a Feistel network using (unkeyed) hash functions:

- Recovering any bit of message requires recovering all of associated X,Y
- Formal reduction to one-wayness of RSA even for chosen ciphertext attacks

RSA summary

- RSA is example of trapdoor one-way function
  - Security conjectured. Relies on factoring being hard
- RSA security scales somewhat poorly with size of primes due to factoring algorithms
  - Key generation must be carefully implemented
- RSA PKCS#1 v1.5 is insecure due to padding oracle attacks. Don’t use it in new systems.
  - Use OAEP instead

Forward-secrecy



Bank customer

TLS handshake for RSA transport

Bank

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Pick random Ns

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

Check CERT using CA public verification key

← CERT = (pk of bank, signature over it)

Pick random PMS

C <- E(pk,PMS)

C →

PMS <- D(sk,C)

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

Bracket notation means contents encrypted

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

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

Forward-secrecy

Have to use ephemeral secret for each key exchange

Key exchange method	Forward security?
RSA transport	No
Static Diffie-Hellman	No
Ephemeral Diffie-Hellman	Yes

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)) } →

ChangeCipherSpec,
← { Finished, PRF(MS, "Server finished" || H(transcript')) }

Bracket notation means contents encrypted

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

Diffie-Hellman math

Let p be a large prime number

Fix the group G = $\mathbf{Z}_p^*$ = {1,2,3,…, p-1}

Then G is *cyclic*. This means one can give a member $g \in G$, called the generator, such that

$$G = \{ g^0, g^1, g^2, \ldots, g^{p-1} \}$$

Example: p = 7. Is 2 or 3 a generator for $\mathbf{Z}_7^*$?

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

The discrete log problem

Fix a cyclic group G with generator g

Traditionally: prime-order subgroup of $\mathbf{Z}_q^*$ for q prime

Pick x at random from $\mathbf{Z}_{|G|}$

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

$$\begin{tikzcd} x \arrow[rr, bend left, "\text{easy}"] & & g^x \arrow[ll, bend left, "\text{hard}"] \end{tikzcd}$$

The discrete log problem

Fix a cyclic group G with generator g

Pick x at random from $\mathbf{Z}_{|G|}$

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

```
𝒜(X):
for i = 2 , … ,  |G|-1 do
     if X  = g^i then
          Return i
```

Very slow for large groups!

$\mathcal{O}(|G|)$

Baby-step giant-step is better:

$\mathcal{O}(|G|^{0.5})$

Nothing faster is known for some groups.

Unauthenticated Diffie-Hellman Key Exchange

$\xrightarrow{\quad X \quad}$

$\xleftarrow{\quad Y \quad}$

Pick random x from $\mathbf{Z}_{|G|}$
$X = g^x$

Pick random y from $\mathbf{Z}_{|G|}$
$Y = g^y$

$K = H(Y^x)$

$K = H(X^y)$

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

What type of security does this protocol provide?

Computational Diffie-Hellman Problem

Fix a cyclic group G with generator g

Pick x,y both at random $\mathbf{Z}_{|G|}$

Give adversary g, $X = g^x$, $Y = g^y$.
Adversary must compute g^{xy}

For most groups, best known algorithm finds discrete log of X or Y.
But we have no proof that this is best approach.

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 , $\sigma = \mathrm{Sign}(sk_s, p \,||\, g \,||\, X)$ ←

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)) } →

ChangeCipherSpec,
{ Finished, PRF(MS, "Server finished" || H(transcript')) } ←

Bracket notation means contents encrypted

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

Summary

- Diffie-Hellman provides forward secrecy
  - Traditionally using $\mathbf{Z}_p^*$ for large prime p
  - DH very efficient when using elliptic curve groups
  - Key exchange protocol of choice these days
    - TLS 1.3 only supports DH-based key exchange
- Asymmetric crypto so far:
  - RSA
  - DH over finite cyclic group