Today in Cryptography (5830)

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

References:
RSA discussed in many textbooks. See Katz & Lindell Sec. 8.1, 8.2
PKCS#1 encryption defined in PKCS#1 v1.5 standard
Diffie-Hellman discussed in many textbooks. See Katz & Lindell Sec. 8.3

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 mod $\phi(N) = 1$

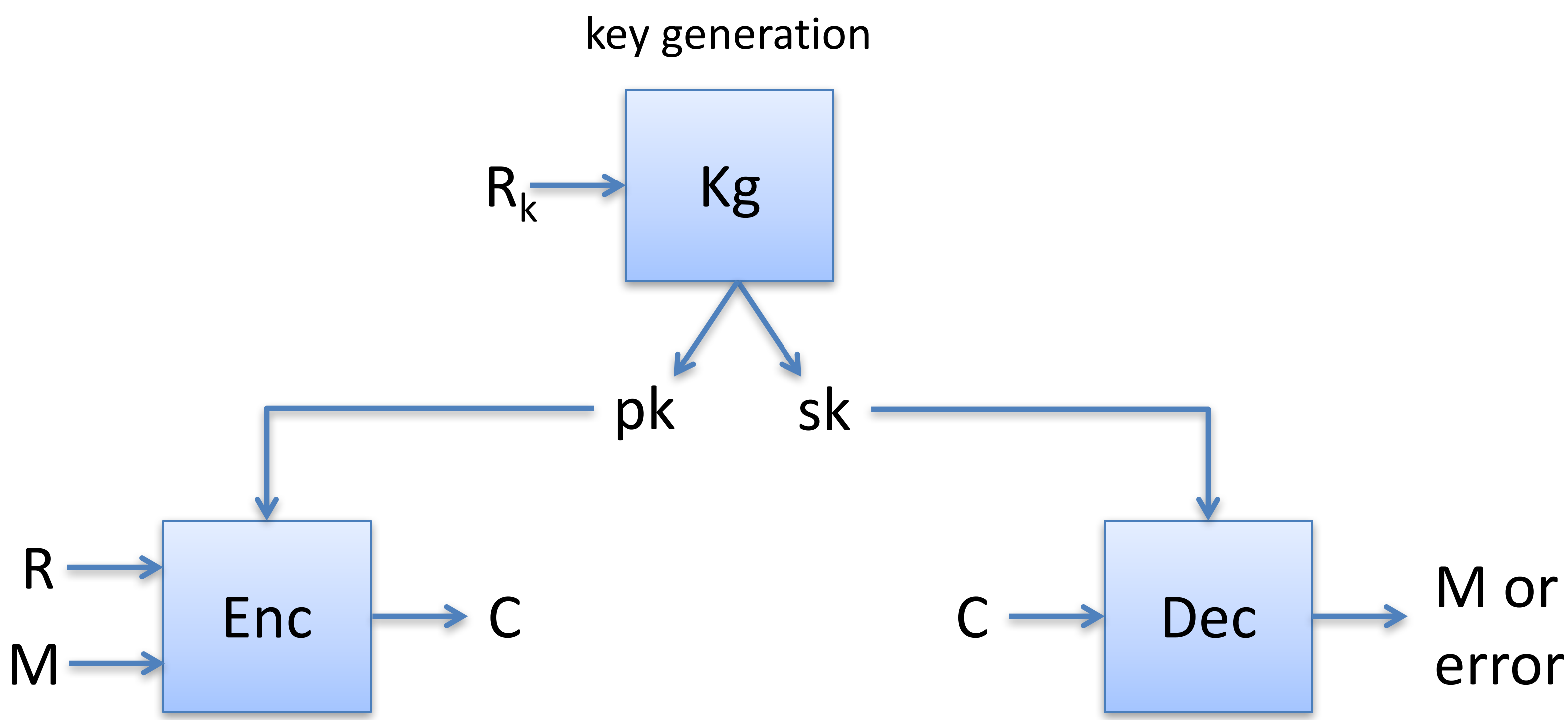
$f_{N,e}(x) = x^e \bmod N$ $g_{N,d}(y) = y^d \bmod N$

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

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)
  - Often store p,q with sk to use Chinese Remainder Theorem

Public-key encryption



C is a ciphertext

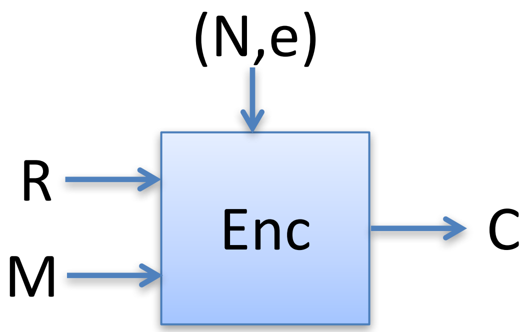
Correctness: D(sk , E(pk,M,R)) = M with probability 1 over randomness used

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 $|M|_8 = m$

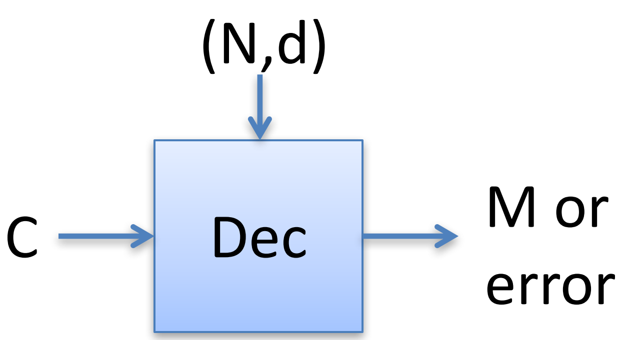


<u>Enc((N,e), M, R)</u>

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 mod 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

TLS handshake for RSA transport

Bank customer

Bank

Pick random Nc — ClientHello, MaxVer, Nc, Ciphers/CompMethods →

ServerHello, Ver, Ns, SessionID, Cipher/CompMethod ← — Pick random Ns

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(PS, "master secret" || Nc || Ns)

Security of RSA PKCS#1

- Passive adversary sees (N,e),C
- Attacker would like to invert C
- Possible attacks?

Inverting RSA : given N, e, y find x such that $x^e \equiv y \pmod{N}$

$\uparrow$ EASY because $f^{-1}(y) = y^d \bmod N$

Know d

$\uparrow$ EASY because $d = e^{-1} \bmod \varphi(N)$

Know $\varphi(N)$

$\uparrow$ EASY because $\varphi(N) = (p-1)(q-1)$

Know p, q

$\uparrow$? Learning p,q from N is the factoring problem

Know N

We don’t know if inverse is true, whether inverting RSA implies ability to factor

Factoring composites

- What is p,q for N = 901?

```
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)

$\mathcal{O}(\text{sqrt}(N)) = \mathcal{O}(e^{0.5 \ln(N)})$

Factoring composites

Algorithm	**Time to factor N**
Naïve	$\mathcal{O}(e^{0.5 \ln(N)})$
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-x is an RSA challenge modulus of size x bits

Security of RSA PKCS#1

- Passive adversary sees (N,e),C
- Attacker would like to invert C
- Possible attacks?
  - Pick |N| > 1024 and factoring will fail
  - Active attacks?

Bleichanbacher attack

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

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 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 chosen-ciphertext secure encryption
  - OAEP is common choice

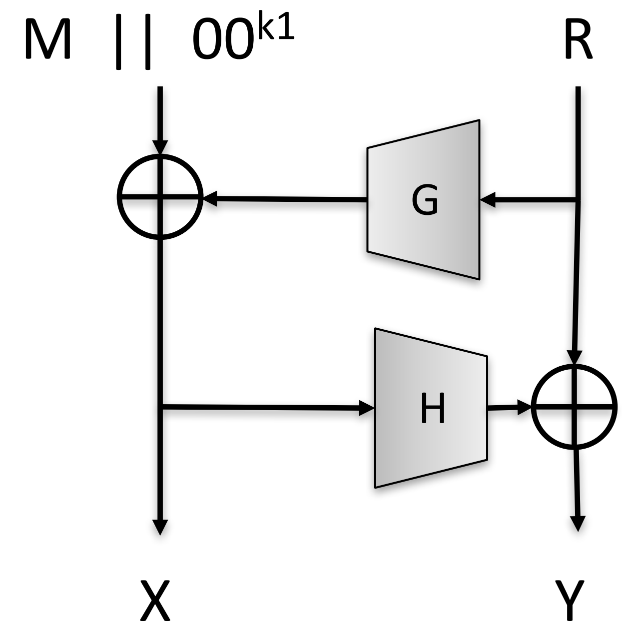
OAEP

[Bellare, Rogaway, 1994]

(optimal asymmetric encryption padding)

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

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



Basically a Feistel network using hash functions:

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

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
- RSA PKCS#1 v1.5 is insecure due to padding oracle attacks. Don't use it in new systems.
  - Use OAEP instead

Forward security

- If long-term secret keys of server are compromised, prior sessions remain secure
  - I.e., security against future compromises
- RSA key transport is ***not*** forward secure

TLS handshake for RSA transport

Client

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)

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

Textbook exponentiation

Let G be cyclic group.
How do we compute h^x for any $h \in G$?

```
ModExp(h,x)
X' = h
For i = 2 to x do
      X' = X'*h
Return X'
```

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

```
SqrAndMulExp(h,x)
b_k,...,b_0 = x
f = 1
For i = k down to 0 do
      f = f^2 mod N
      If b_i = 1 then
            f = f*h
Return f
```

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

```
SqrAndMulExp(h,x)
b_k,…,b_0 = x
f = 1
For i = k down to 0 do
    f = f^2 mod N
    If b_i = 1 then
        f = f*h
Return f
```

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

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

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

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

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

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

$b_1 = 1$ $\quad f_0 = (h^4 \bullet h)^2 \bullet h = h^8 \bullet h^2 \bullet h$

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

$$\begin{tikzcd}[column sep=huge]
x \arrow[r, bend left, "\text{easy}"] & g^x \arrow[l, bend left, "\text{hard}"]
\end{tikzcd}$$

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

$\mathcal{A}$(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.

Diffie-Hellman Key Exchange

X →

← Y

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 , 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)

Summary

- Diffie-Hellman provides forward security
  - Very efficient when using elliptic curve cryptography
  - 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