The RSA Trapdoor Permutation

Recap

Public key encryption: (G, E, D)

G() ⟶ (pk, sk) , E(pk, m) ⟶ c , D(sk, c) ⟶ m

Constructions: (1) ElGamal encryption, (2) today: RSA

Security from last lecture:

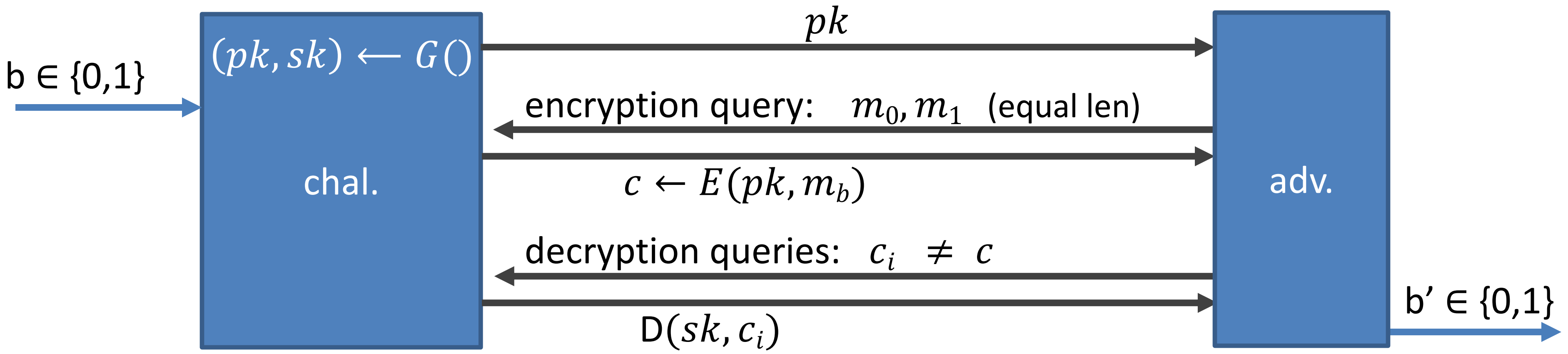
semantic security against an eavesdropper

In practice security against eavesdropping is insufficient:

adversary can make up ciphertexts
and see how recipient reacts

Security against chosen ciphertext attacks (CCA)

A PKE (G, E, D) is chosen-ciphertext secure if no "efficient" adversary can win the following game:



Thm: ElGamal encryption from last lecture is CCA secure assuming interactive-CDH in G holds, and H is a modeled as a random oracle

Recap

Public key encryption: (G, E, D)

$$\mathbf{G() \longrightarrow (pk, sk)\ , \qquad E(pk, m) \longrightarrow c\ , \qquad D(sk, c) \longrightarrow m}$$

Security: **semantic security against a chosen-ciphertext attack**

- Semantic security against adv. that can issue decryption queries

Constructions: (1) ElGamal encryption, (2) today: RSA

… but first: **trapdoor functions**

Trapdoor functions (TDF)

Def: a trapdoor func. $X \longrightarrow Y$ is a triple of efficient algs. (G, F, F^{-1})

- $G()$: randomized alg. outputs a key pair (pk, sk)
- $F(pk,\cdot)$: det. alg. that defines a function $X \longrightarrow Y$
- $F^{-1}(sk,\cdot)$: defines a function $Y \longrightarrow X$ that inverts $F(pk,\cdot)$

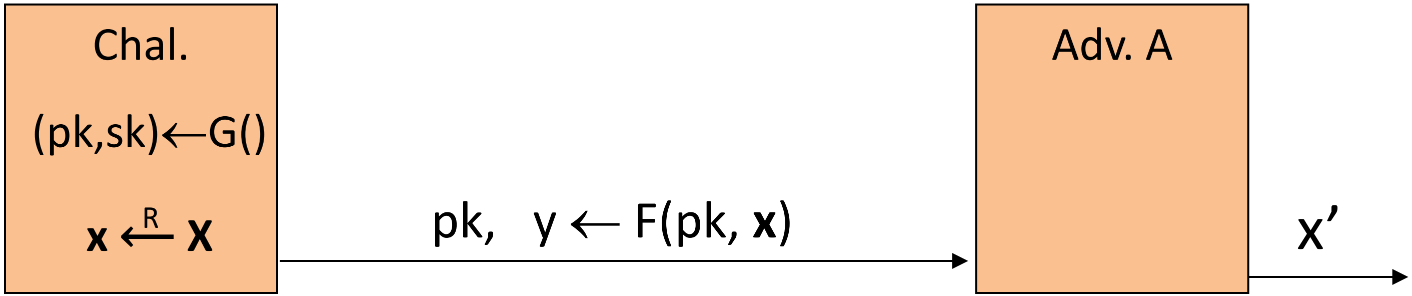
More precisely: $\forall (pk, sk)$ output by G

$$\forall x \in X: \quad F^{-1}(sk, F(pk, x)) = x$$

Secure Trapdoor Functions (TDFs)

(G, F, F^{-1}) is secure if $F(pk, \cdot)$ is a "one-way" function:

can be evaluated, but cannot be inverted without sk



Def: (G, F, F^{-1}) is a secure TDF if for all efficient A:

$$Adv_{OW}[A,F] = \Pr[x = x'] < \text{negligible}$$

Dan Boneh

Public-key encryption from TDFs

- (G, F, F^{-1}): secure TDF $X \longrightarrow Y$
- (E_s, D_s) : symmetric auth. encryption defined over (K,M,C)
- $H: X \longrightarrow K$ a hash function

We construct a pub-key enc. system (G, E, D):

Key generation G: same as G for TDF

Public-key encryption from TDFs

- (G, F, F^{-1}): secure TDF $X \longrightarrow Y$
- (E_s, D_s) : symmetric auth. encryption defined over (K,M,C)
- $H: X \longrightarrow K$ a hash function

<u>E(pk, m)</u> :

$x \xleftarrow{R} X$, $y \longleftarrow F(pk, x)$

$k \longleftarrow H(x)$, $c \longleftarrow E_s(k, m)$

output (y, c)

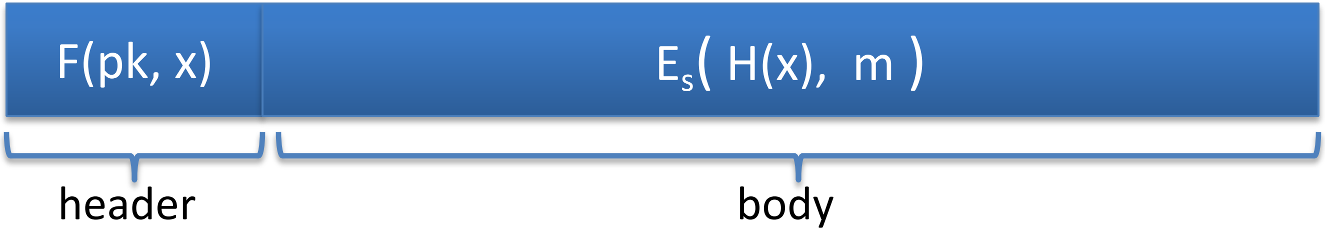
<u>D(sk, (y,c))</u> :

$x \longleftarrow F^{-1}(sk, y)$,

$k \longleftarrow H(x)$, $m \longleftarrow D_s(k, c)$

output m

In pictures:



<u>Security Theorem</u>:

If $\mathbf{(G, F, F^{-1})}$ is a secure TDF, $\mathbf{(E_s, D_s)}$ provides auth. enc.

and $\mathbf{H}$: $X \longrightarrow K$ is a "random oracle"

then $\mathbf{(G,E,D)}$ is CCA^{ro} secure.

Incorrect use of a Trapdoor Function (TDF)

Never encrypt by applying F directly to plaintext:

E(pk, m) :

output $c \longleftarrow F(pk, m)$

D(sk, c) :

output $F^{-1}(sk, c)$

Problems:

- Deterministic: cannot be semantically secure !!
- Many attacks exist (coming)

The RSA trapdoor permutation

Review: arithmetic mod composites

Let $N = p \cdot q$ where p,q are prime

$Z_N = \{0,1,2,\ldots,N-1\}$; $(Z_N)^*$ = {invertible elements in Z_N}

<u>Facts</u>: $x \in Z_N$ is invertible $\Leftrightarrow$ $\gcd(x,N) = 1$

– Number of elements in $(Z_N)^*$ is $\varphi(N) = (p-1)(q-1) = N-p-q+1$

<u>Euler's thm</u>: $\forall\, x \in (Z_N)^* \;:\; x^{\varphi(N)} = 1$

The RSA trapdoor permutation

First published: Scientific American, Aug. 1977.

Applications:

- HTTPS: web certificates
- deprecated for key exchange in TLS 1.3

The RSA trapdoor permutation

G(): choose random primes p,q ≈1024 bits. Set **N=pq**.

choose integers **e , d** s.t. **$e \cdot d = 1 \pmod{\varphi(N)}$**

output pk = (N, e) , sk = (N, d)

F(pk, x): $\mathbb{Z}_N^* \rightarrow \mathbb{Z}_N^*$; **RSA(x) = x^e** (in Z_N)

F^{-1}(sk, y) = y^d ; $y^d = \mathbf{RSA(x)^d} = x^{ed} = x^{k\varphi(N)+1} = \left(x^{\varphi(N)}\right)^k \cdot \mathbf{x} = x$

The RSA assumption

RSA_e assumption: RSA with exp. e is a one-way permutation

For all efficient algs. A:

$$\Pr\left[A(N,e,\mathbf{y}) = \mathbf{y}^{1/e} \right] < \text{negligible}$$

where $p,q \xleftarrow{R}$ n-bit primes, $N \leftarrow pq$, $y \xleftarrow{R} Z_N^*$

RSA pub-key encryption (ISO std)

(E_s, D_s): symmetric enc. scheme providing auth. encryption.

H: $\mathbb{Z}_N \to K$ where K is key space of (E_s, D_s)

- **G**(): generate RSA params: pk = (N,e), sk = (N,d)
- **E**(pk, m): (1) choose random x in $\mathbb{Z}_N^*$

  (2) $y \leftarrow RSA(x) = x^e$, $k \leftarrow H(x)$

  (3) output $(y, E_s(k,m))$
- **D**(sk, (y, c)): output $D_s(H(RSA^{-1}(y)), c)$

Textbook RSA is insecure

Textbook RSA encryption:

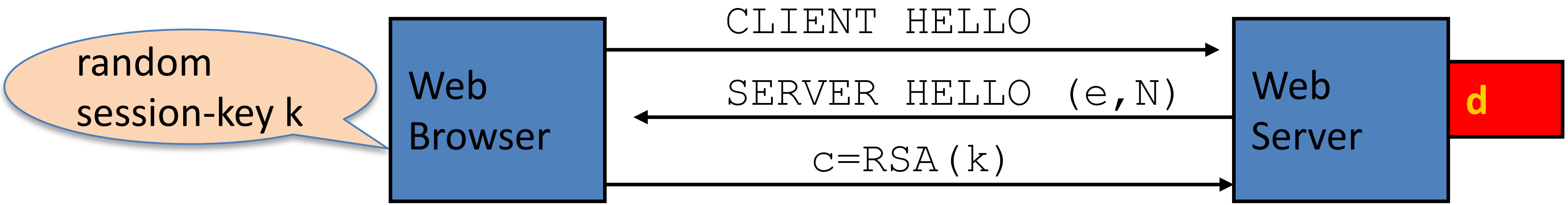
- public key: **(N,e)** Encrypt: $\mathbf{c} \longleftarrow \mathbf{m^e}$ (in Z_N)
- secret key: **(N,d)** Decrypt: $\mathbf{c^d} \longrightarrow \mathbf{m}$

Insecure cryptosystem !!

- Is not semantically secure and many attacks exist

⇒ The RSA trapdoor permutation is not an encryption scheme !

A simple attack on textbook RSA



Suppose k is 64 bits: $k \in \{0,\ldots,2^{64}\}$. Eve sees: $c = k^e$ in Z_N

If $\mathbf{k = k_1 \cdot k_2}$ where $k_1, k_2 < 2^{34}$ (prob. ≈20%) then $\mathbf{c/k_1^e = k_2^e}$ in Z_N

Step 1: build table: $c/1^e, c/2^e, c/3^e, \ldots, c/2^{34e}$. time: 2^{34}

Step 2: for $k_2 = 0,\ldots, 2^{34}$ test if k_2^e is in table. time: 2^{34}

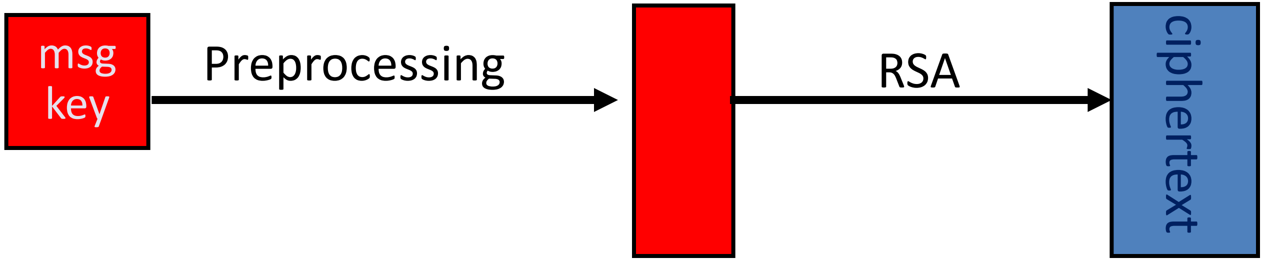
Output matching (k_1, k_2). Total attack time: $\approx 2^{34} << 2^{64}$

RSA in practice

RSA encryption in practice

Never use textbook RSA.

RSA in practice (since ISO standard is not often used) :

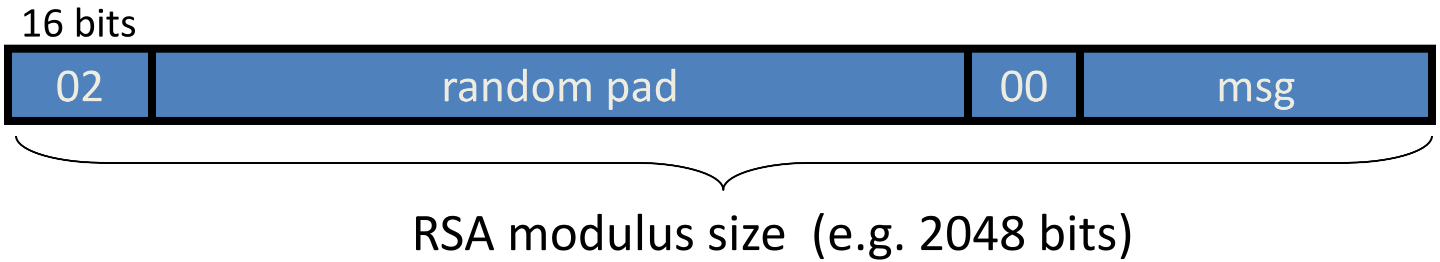


Main questions:

– How should the preprocessing be done?
– Can we argue about security of resulting system?

PKCS1 v1.5

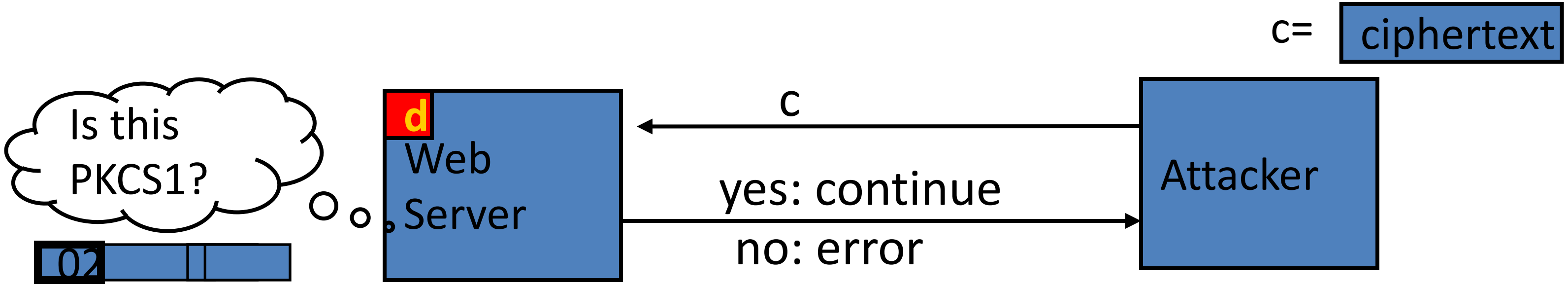
PKCS1 mode 2: (encryption)



- Resulting value is RSA encrypted
- Widely deployed, e.g. in HTTPS (TLS 1.2)

Attack on PKCS1 v1.5 (Bleichenbacher 1998)

PKCS1 used in HTTPS:



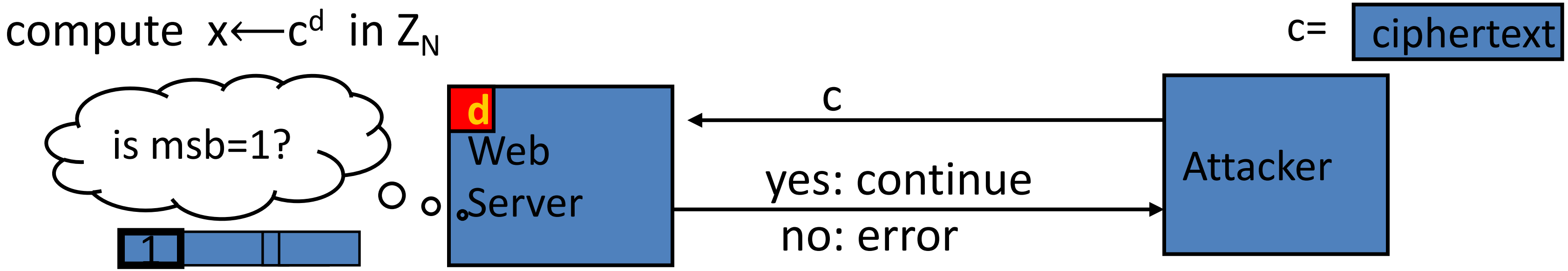
$\Rightarrow$ attacker can test if 16 MSBs of plaintext = '02'

Chosen-ciphertext attack: to decrypt a given ciphertext c do:

– Choose $r \in Z_N$. Compute $c' \leftarrow r^e \cdot c = (r \cdot \text{PKCS1}(m))^e$

– Send c' to web server and use response

Baby Bleichenbacher



Suppose N is $N = 2^n$ (an invalid RSA modulus). Then:

- Sending c reveals msb(x)
- Sending $\mathbf{2^e \cdot c = (2x)^e}$ in Z_N reveals msb(2x mod N) = $\mathbf{msb_2(x)}$
- Sending $\mathbf{4^e \cdot c = (4x)^e}$ in Z_N reveals msb(4x mod N) = $\mathbf{msb_3(x)}$

… and so on to reveal all of x

HTTPS Defense (RFC 5246)

Attacks discovered by Bleichenbacher and Klima et al. … can be avoided by treating incorrectly formatted message blocks … in a manner indistinguishable from correctly formatted RSA blocks.

In other words:

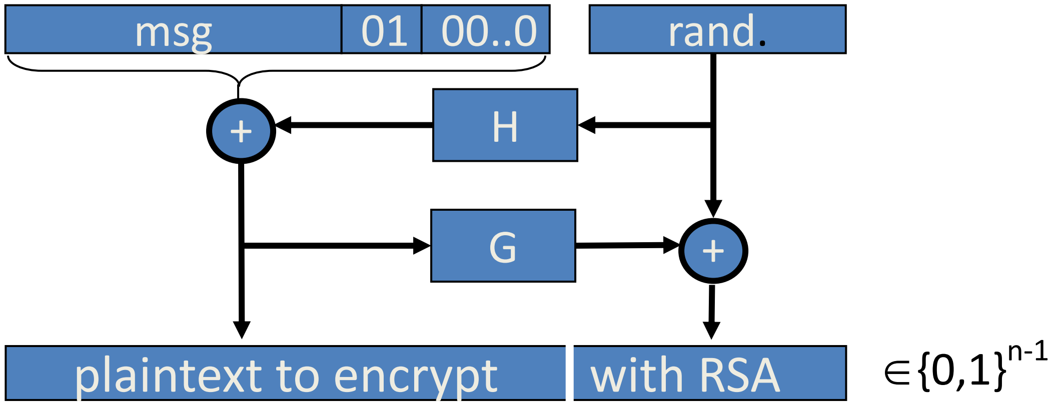
1. *Generate a string **R** of 46 random bytes*
2. *Decrypt the message to recover the plaintext M*
3. *If the PKCS#1 padding is not correct*

   *pre_master_secret = **R***

PKCS1 v2.0: OAEP

New preprocessing function: OAEP [BR94]

check pad
on decryption.
reject CT if invalid.



Thm [FOPS'01] : RSA is a trap-door permutation $\Rightarrow$
RSA-OAEP is CCA secure when H,G are *random oracles*

in practice: use SHA-256 for H and G

Subtleties in implementing OAEP [M '00]

```
OAEP-decrypt(ct):
        error = 0;
        ……..
        if ( RSA^-1(ct) > 2^(n-1) )
              { error =1;  goto exit; }
        ……..
        if ( pad(OAEP^-1(RSA^-1(ct))) != "01000" )
              { error = 1;  goto exit; }
```

Problem: timing information leaks type of error

$\Rightarrow$ Attacker can decrypt any ciphertext

Lesson: Don't implement RSA-OAEP yourself !

Is RSA a one-way function?

Is RSA a one-way permutation?

To invert the RSA one-way func. (without d) attacker must compute:

x from $c = x^e \pmod{N}$.

How hard is computing e'th roots modulo N ??

Best known algorithm:

- Step 1: factor N (hard)
- Step 2: compute e'th roots modulo p and q (easy)

Shortcuts?

Must one factor N in order to compute e'th roots?

To prove no shortcut exists show a reduction:

- Efficient algorithm for e'th roots mod N

  $\Rightarrow$ efficient algorithm for factoring N.

- Oldest open problem in public key cryptography.

Some evidence no reduction exists: (BV'98)

- "Algebraic" reduction $\Rightarrow$ factoring is easy.

How **not** to improve RSA's performance

To speed up RSA decryption use small private key d ($d \approx 2^{128}$)

$$c^d = m \pmod{N}$$

Wiener'87: if $d < N^{0.25}$ then RSA is insecure.

BD'98: if $d < N^{0.292}$ then RSA is insecure (open: $d < N^{0.5}$)

<u>Insecure:</u> priv. key d can be found from (N,e)

Wiener's attack

Recall: $e \cdot d = 1 \pmod{\varphi(N)} \Rightarrow \exists\, k \in Z: \quad e \cdot d = k \cdot \varphi(N) + 1$

$$\left| \frac{e}{\varphi(N)} - \frac{k}{d} \right| = \frac{1}{d \cdot \varphi(N)} \leq \frac{1}{\sqrt{N}}$$

Wiener's attack

Recall: $e \cdot d = 1 \pmod{\varphi(N)} \Rightarrow \exists\, k \in Z: \quad e \cdot d = k \cdot \varphi(N) + 1$

$$\left| \frac{e}{\varphi(N)} - \frac{k}{d} \right| = \frac{1}{d \cdot \varphi(N)} \leq \frac{1}{\sqrt{N}}$$

$\varphi(N) = N\text{-}p\text{-}q+1 \quad \Rightarrow \quad |N - \varphi(N)| \leq p+q \leq 3\sqrt{N}$

Wiener's attack

Recall: $e \cdot d = 1 \pmod{\varphi(N)} \Rightarrow \exists\, k \in Z: \quad e \cdot d = k \cdot \varphi(N) + 1$

$$\left| \frac{e}{\varphi(N)} - \frac{k}{d} \right| = \frac{1}{d \cdot \varphi(N)} \leq \frac{1}{\sqrt{N}}$$

$\varphi(N) = N - p - q + 1 \Rightarrow |N - \varphi(N)| \leq p + q \leq 3\sqrt{N}$

$d \leq N^{0.25}/3 \Rightarrow \left| \frac{e}{N} - \frac{k}{d} \right| \leq \left| \frac{e}{N} - \frac{e}{\varphi(N)} \right| + \underbrace{\left| \frac{e}{\varphi(N)} - \frac{k}{d} \right|}_{\leq 1/\sqrt{N}} \leq \frac{1}{2d^2}$

Wiener's attack

Recall: $e \cdot d = 1 \pmod{\varphi(N)} \Rightarrow \exists\, k \in \mathbb{Z}: \; e \cdot d = k \cdot \varphi(N) + 1$

$$\left| \frac{e}{\varphi(N)} - \frac{k}{d} \right| = \frac{1}{d \cdot \varphi(N)} \le \frac{1}{\sqrt{N}}$$

$\varphi(N) = N - p - q + 1 \Rightarrow |N - \varphi(N)| \le p+q \le 3\sqrt{N}$

$d \le N^{0.25}/3 \Rightarrow$

$$\left| \frac{e}{N} - \frac{k}{d} \right| \le \left| \frac{e}{N} - \frac{e}{\varphi(N)} \right| + \left| \frac{e}{\varphi(N)} - \frac{k}{d} \right| \le \frac{1}{2d^2}$$

$$\left| \frac{e}{\varphi(N)} - \frac{k}{d} \right| \le 1/\sqrt{N}$$

$$\left| \frac{e}{N} - \frac{e}{\varphi(N)} \right| \le \frac{3\sqrt{N}}{N} \cdot \frac{e}{\varphi(N)} \le \frac{3}{\sqrt{N}} \le \frac{1}{2d^2} - \frac{1}{\sqrt{N}}$$

$$\frac{1}{2d^2} - \frac{1}{\sqrt{N}} \ge \frac{3}{\sqrt{N}}$$

$\Rightarrow$ Continued fraction expansion of e/N gives k/d.

$e \cdot d = 1 \pmod{k} \Rightarrow \gcd(d,k) = 1 \Rightarrow$ can find d from k/d

RSA With Low public exponent

To speed up RSA encryption use a small e: $c = m^e \pmod{N}$

- Minimum value: **e=3** ($\gcd(e, \varphi(N)) = 1$)
- Recommended value: **$e=65537=2^{16}+1$**

Encryption: 17 multiplications

Asymmetry of RSA: fast enc. / slow dec.

- ElGamal: approx. same time for both.

Key lengths

Security of public key system should be comparable to security of symmetric cipher:

Cipher key-size	RSA Modulus size	Elliptic Curve Modulus size
80 bits	1024 bits	160 bits
128 bits	3072 bits	256 bits
256 bits (AES)	**15360** bits	512 bits

Best factoring algorithm (GNF): n-bits integer, time ≈ $\exp(n^{1/3})$

Implementation attacks

Timing attack: [Kocher et al. 1997] , [BB'04]

The time it takes to compute $c^d \pmod{N}$ can expose d

Power attack: [Kocher et al. 1999)

The power consumption of a smartcard while it is computing $c^d \pmod{N}$ can expose d.

Faults attack: [BDL'97]

A computer error during $c^d \pmod{N}$ can expose d.

A common defense:: check output. 10% slowdown.

An Example Fault Attack on RSA (CRT)

A common implementation of RSA decryption: $x = c^d$ in Z_N

decrypt mod p: $x_p = c^d$ in Z_p

decrypt mod q: $x_q = c^d$ in Z_q

} combine to get $x = c^d$ in Z_N

Suppose error occurs when computing x_q, but no error in x_p

Then: output is x' where $x' = c^d$ in Z_p but $x' \neq c^d$ in Z_q

$\Rightarrow$ $(x')^e = c$ in Z_p but $(x')^e \neq c$ in Z_q $\Rightarrow$ $\gcd\big((x')^e - c, N\big) = p$

RSA Key Generation Trouble [Heninger et al./Lenstra et al.]

OpenSSL RSA key generation (abstract):

```
prng.seed(seed)
p = prng.generate_random_prime()
prng.add_randomness(bits)
q = prng.generate_random_prime()
N = p*q
```

Suppose poor entropy at startup:

- Same p will be generated by multiple devices, but different q
- N_1 , N_2 : RSA keys from different devices $\Rightarrow$ $\gcd(N_1,N_2) = p$

RSA Key Generation Trouble [Heninger et al./Lenstra et al.]

Experiment: factors 0.4% of public HTTPS keys !!

Lesson:

- Make sure random number generator is properly seeded when generating keys

THE END