Public Key Encryption from trapdoor permutations

Public key encryption: definitions and security

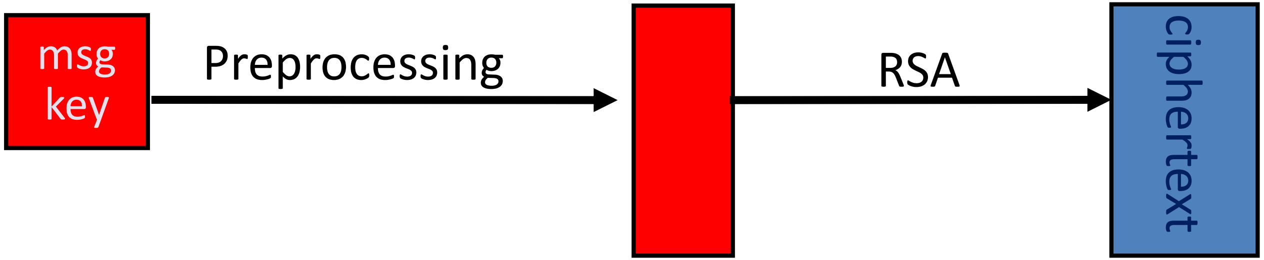
Public Key Encryption from trapdoor permutations

PKCS 1

RSA encryption in practice

Never use textbook RSA.

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

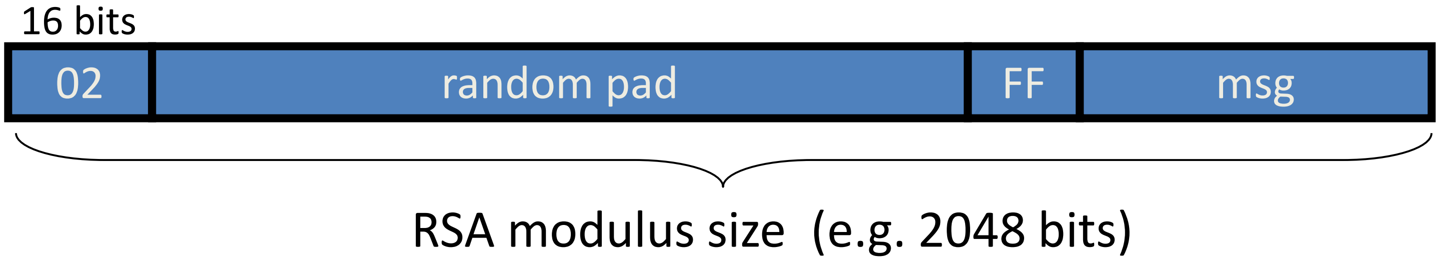


Main questions:

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

PKCS1 v1.5

PKCS1 mode 2: (encryption)

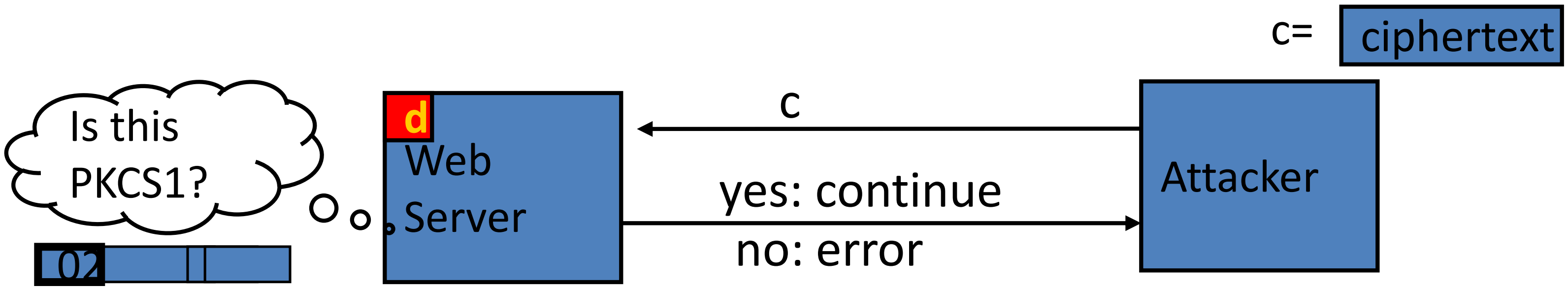


- Resulting value is RSA encrypted

- Widely deployed, e.g. in HTTPS

Attack on PKCS1 v1.5 (Bleichenbacher 1998)

PKCS1 used in HTTPS:



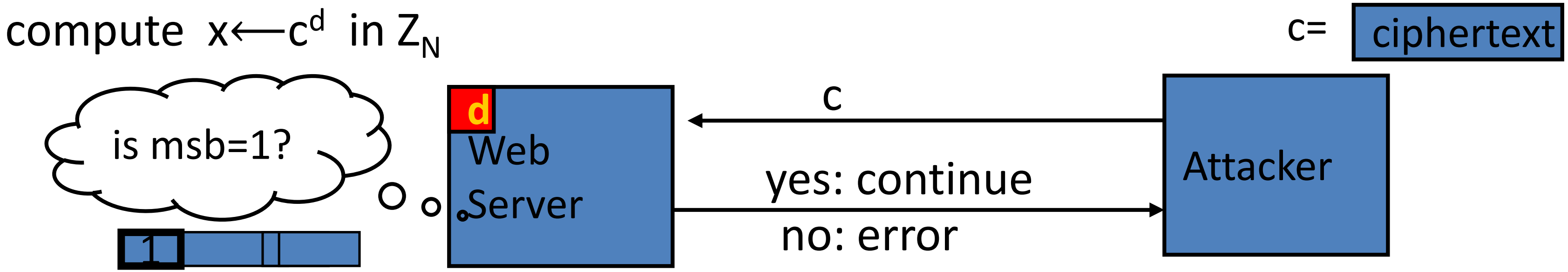
$\Rightarrow$ attacker can test if 16 MSBs of plaintext = '02'

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

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

– 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:

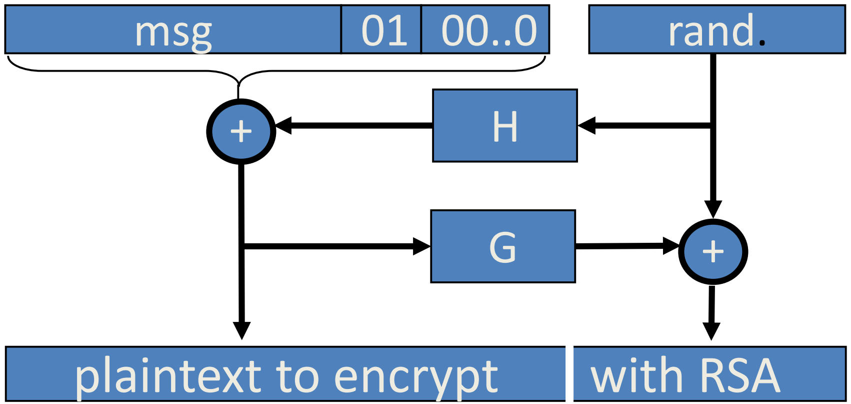
1. *Generate a string **R** of 46 random bytes*
2. *Decrypt the message to recover the plaintext M*
3. *If the PKCS#1 padding is not correct*

   *pre_master_secret = **R***

PKCS1 v2.0: OAEP

New preprocessing function: OAEP [BR94]

check pad
on decryption.
reject CT if invalid.



$\in \{0,1\}^{n-1}$

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

OAEP Improvements

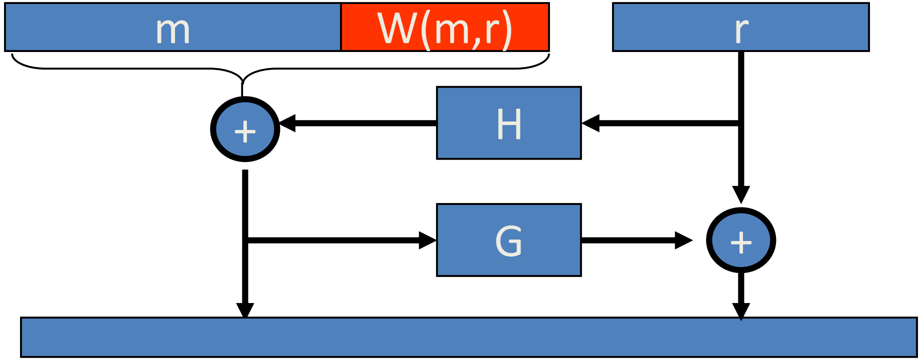
OAEP+: [Shoup'01]

$\forall$ trap-door permutation F

F-OAEP+ is CCA secure when

H,G,W are *random oracles*.

During decryption validate W(m,r) field.

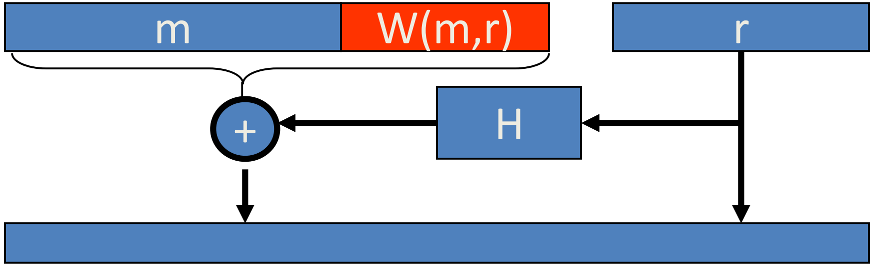


SAEP+: [B'01]

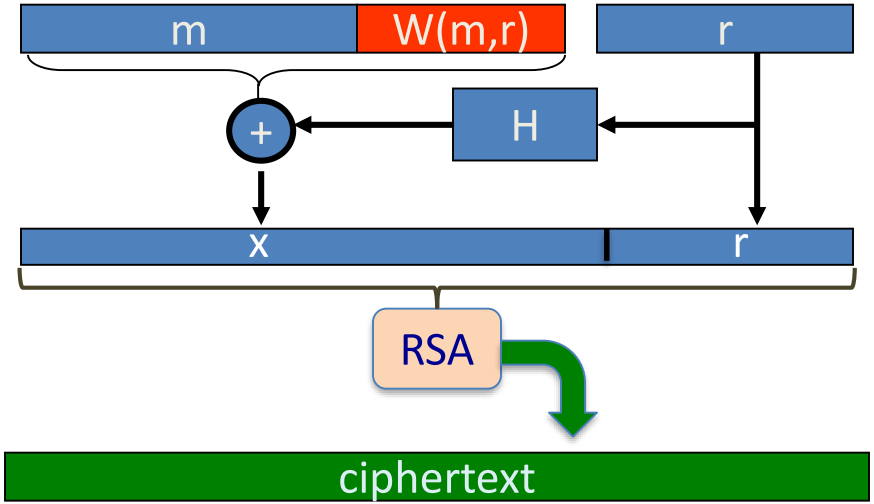
RSA (e=3) is a trap-door perm $\Rightarrow$

RSA-SAEP+ is CCA secure when

H,W are *random oracle*.



How would you decrypt an SAEP ciphertext **ct** ?



- $(x,r) \longleftarrow RSA^{-1}(sk,ct)$, $(m,w) \longleftarrow x \oplus H(r)$, output m if $w = W(m,r)$
- $(x,r) \longleftarrow RSA^{-1}(sk,ct)$, $(m,w) \longleftarrow r \oplus H(x)$, output m if $w = W(m,r)$
- $(x,r) \longleftarrow RSA^{-1}(sk,ct)$, $(m,w) \longleftarrow x \oplus H(r)$, output m if $r = W(m,x)$

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 !

End of Segment

Public Key Encryption from trapdoor permutations

Is RSA a one-way function?

Is RSA a one-way permutation?

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

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

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 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 ($d \approx 2^{128}$)

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

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

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

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

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

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

$$\leq \frac{3\sqrt{N}}{N} \cdot \frac{e}{\varphi(N)} \leq \frac{3}{\sqrt{N}} \leq \frac{1}{2d^2} - \frac{1}{\sqrt{N}}$$

$\Rightarrow$ 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

End of Segment

Public Key Encryption from trapdoor permutations

RSA in practice

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 (next module): 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
80 bits	1024 bits
128 bits	3072 bits
256 bits (AES)	**15360** bits

Implementation attacks

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

The time it takes to compute c^d (mod N) can expose d

Power attack: [Kocher et al. 1999)

The power consumption of a smartcard while it is computing c^d (mod N) can expose d.

Faults attack: [BDL'97]

A computer error during c^d (mod 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

$$\left.\begin{array}{lll} \text{decrypt mod p:} & x_p = c^d & \text{in } Z_p \\ \text{decrypt mod q:} & x_q = c^d & \text{in } Z_q \end{array}\right\} \text{combine to get } x = c^d \text{ in } Z_N$$

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

Further reading

- Why chosen ciphertext security matters, V. Shoup, 1998
- Twenty years of attacks on the RSA cryptosystem,
  D. Boneh, Notices of the AMS, 1999
- OAEP reconsidered, V. Shoup, Crypto 2001
- Key lengths, A. Lenstra, 2004

End of Segment