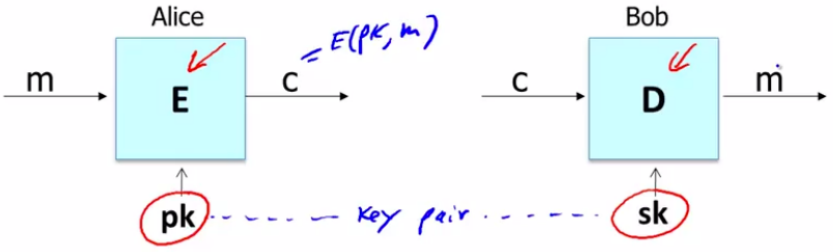
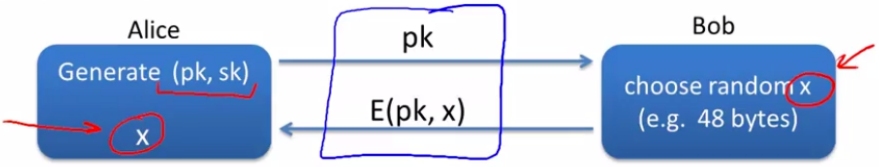
- **Public key encryption from trapdoor permutations**
- **definitions and security**
- public key encryption

Bob: generates (PK, SK) and gives PK to Alice



- Applications

Session setup (for now, only eavesdropping security)



Non-interactive applications: (e.g. Email)

- Bob sends email to Alice encrypted using pk_{alice}
- Note: Bob needs pk_{alice} (public key management)

- Public key encryption

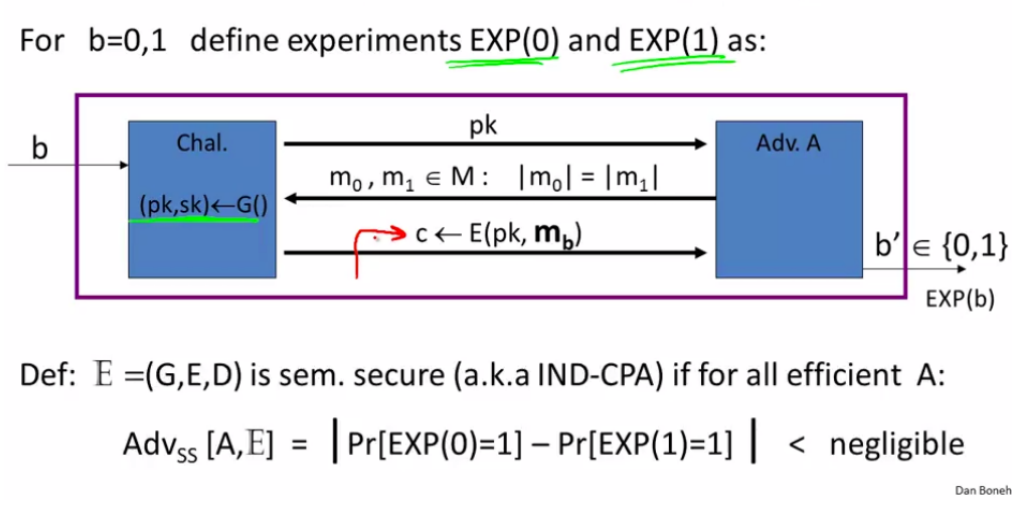
Def: a public-key encryption system is a triple of algs. (G, E, D)

- G(): randomized alg. outputs a key pair (pk, sk)
- E(pk, m): randomized alg. that takes $m \in M$ and outputs $c \in C$
- D(sk,c): det. alg. that takes $c \in C$ and outputs $m \in M$ or $\perp$

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

$\forall m \in M$: D(sk, E(pk, m)) = m

- Security: eavesdropping



- semantically secure if the attacker cannot tell if it is the first experiment of the second experiment

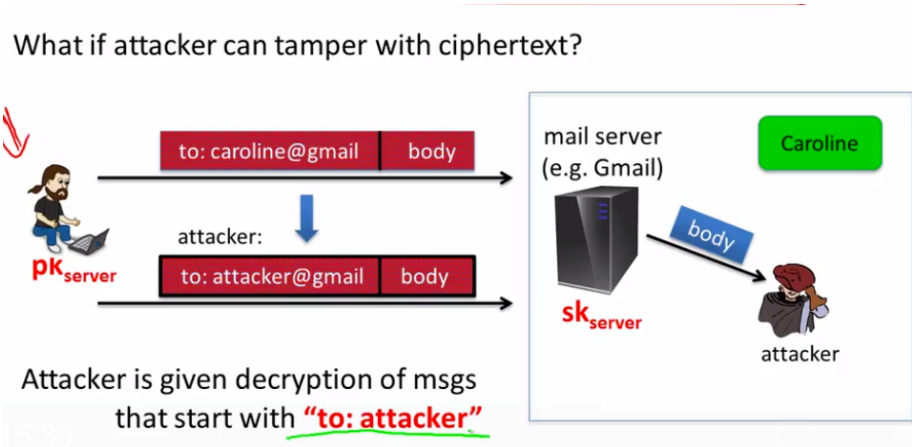
- Relation to symmetric cipher security

Recall: for symmetric ciphers we had two security notions:
- One-time security and many-time security (CPA)
- We showed that one-time security ⇏ many-time security

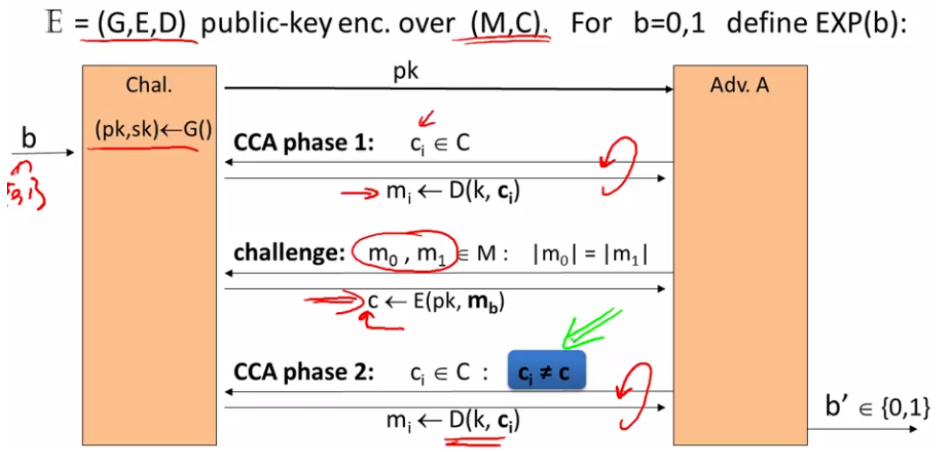
For public key encryption:
- One-time security ⇒ many-time security (CPA)

(follows from the fact that attacker can encrypt by himself)

- Security against active attacks



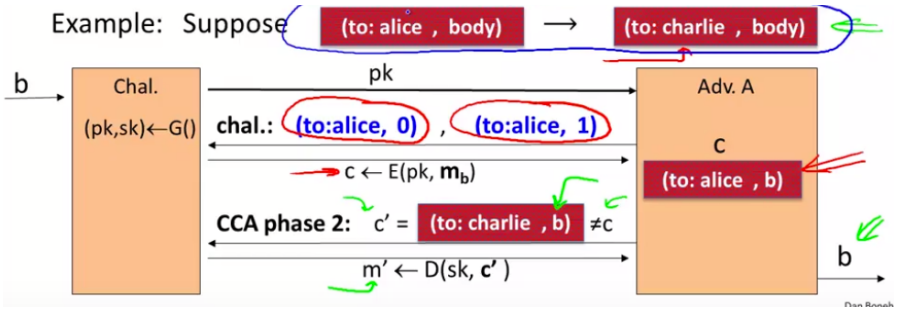
- (pub-key) chosen ciphertext security: definition



- Chosen ciphertext security: definition

Def: E is CCA secure (a.k.a IND-CCA) if for all efficient A:

$Adv_{CCA}[A,E] = |Pr[EXP(0)=1] - Pr[EXP(1)=1]|$ is negligible.



- constructing CCA secure pub-key systems

- **Constructions**
- Trapdoor functions (TDF)

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

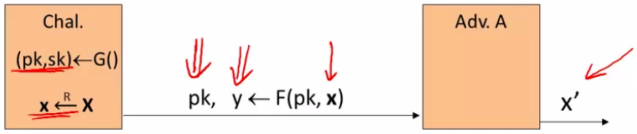
- G(): randomized alg. outputs a key pair (pk, sk)
- F(pk,·): det. alg. that defines a function $X \rightarrow Y$
- F^{-1}(sk,·): defines a function $Y \rightarrow X$ that inverts F(pk,·)

More precisely: ∀(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:

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

- 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):

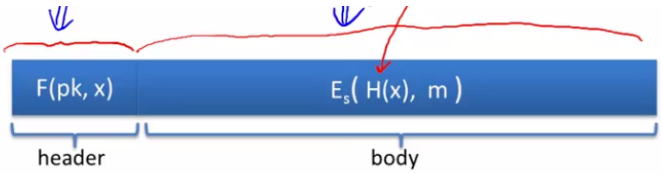
Key generation G: same as G for TDF

- Encryption and decryption

E(pk, m) :
$x \xleftarrow{R} X$, $y \leftarrow F(pk, x)$
$k \leftarrow H(x)$ $c \leftarrow E_s(k, m)$
output (y, c)

D(sk, (y,c)) :
$x \leftarrow F^{-1}(sk, y)$,
$k \leftarrow H(x)$, $m \leftarrow D_s(k, c)$
output m

In pictures:



Security Theorem:

If **(G, F, F^{-1})** is a secure TDF, **(E_s, D_s)** provides auth. enc.
and **H:** $X \longrightarrow K$ is a "random oracle"
then **(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) :	**D(sk, c) :**
output $c \leftarrow F(pk, m)$	output $F^{-1}(sk, c)$

Problems:

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

  - never apply the trapdoor function to the message m
- **Public Key Encryption from Trapdoor Permutations: RSA**
- **The RSA Trapdoor Permutation**
- Review: trapdoor permutations

Three algorithms: (G, F, F^{-1})

- G: outputs pk, sk. pk defines a function $F(pk, \cdot): X \rightarrow X$
- $F(pk, x)$: evaluates the function at x
- $F^{-1}(sk, y)$: inverts the function at y using sk

Secure trapdoor permutation:

The function $F(pk, \cdot)$ is one-way without the trapdoor sk

- 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}

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

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

Euler's thm: $\forall x \in (Z_N)^* : x^{\varphi(N)} = 1$

  - 1 in Zn or 1 mod N

- The RSA trapdoor permutation

First published: Scientific American, Aug. 1977.

Very widely used:

- SSL/TLS: certificates and key-exchange
- Secure e-mail and file systems

... many others

- The RSA Trapdoor permutation
  - encryption exponent e
  - decryption exponent d

G(): choose random primes p,q ≈1024 bits. Set **N=pq**.
choose integers **e , d** s.t. **e·d = 1 (mod φ(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 = **RSA(x)^d** = x^{ed} = $x^{k\varphi(N)+1}$ = $(x^{\varphi(N)})^k \cdot x$ = x

$y = RSA(x) = x^e$

$ed = k \cdot \varphi(N) + 1$

$= 1$

- The RSA assumption

RSA assumption: RSA is one-way permutation

For all efficient algs. A:

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

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

- Review: RSA pub-key encryption (ISO std)
  - symmetric encryption system

(E_s, D_s): symmetric enc. scheme providing auth. encryption.
$H: Z_N \to K$ 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 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 $D_s(H(RSA^{-1}(y)), c)$ → m

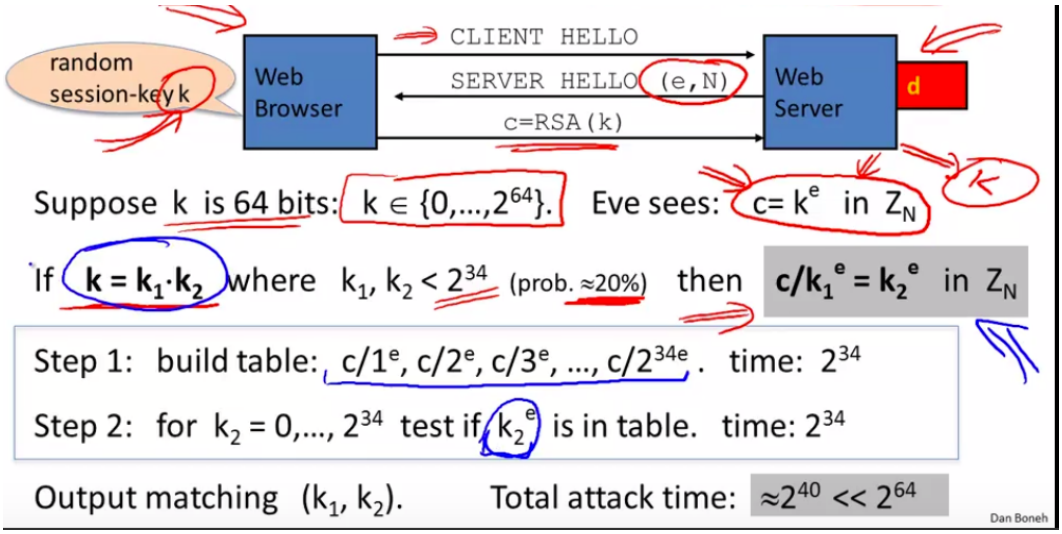
- Textbook RSA is insecure

Textbook RSA encryption:
- public key: **(N,e)** Encrypt: $c \leftarrow m^e$ (in Z_N)
- secret key: **(N,d)** Decrypt: $c^d \rightarrow m$

Insecure cryptosystem !!
- Is not semantically secure and many attacks exist

⇒ The RSA trapdoor permutation is not an encryption scheme !

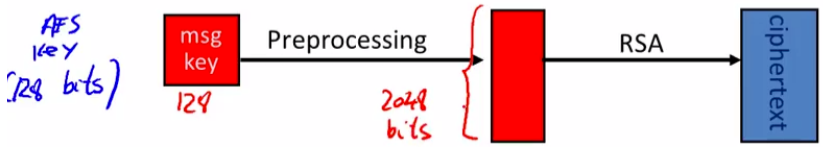
- A simple attack on textbook RSA



- **PKCS 1**
- RSA encryption in practice

Never use textbook RSA.

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

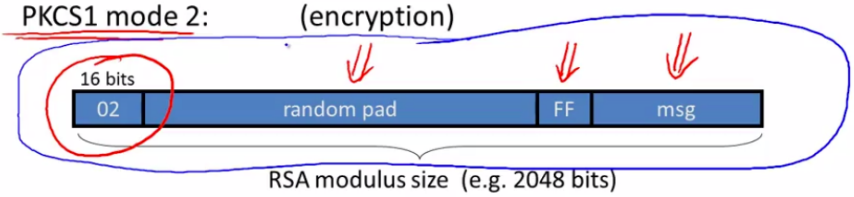


Main questions:

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

- PKCS1 v1.5

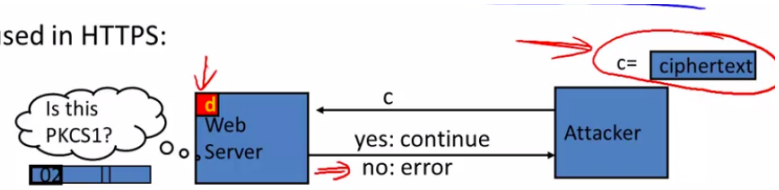
PKCS1 mode 2: (encryption)



- Resulting value is RSA encrypted
- Widely deployed, e.g. in HTTPS

- Attack on PKCS v1.5

PKCS1 used in HTTPS:

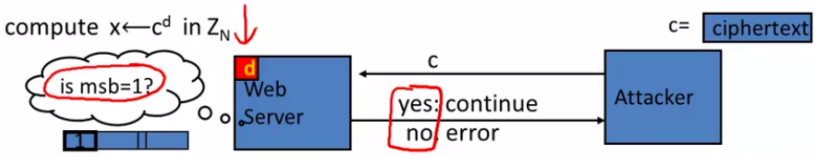


⇒ attacker can test if 16 MSBs of plaintext = '02'

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 $2^e \cdot c = (2x)^e$ in Z_N reveals msb(2x mod N) = $msb_2(x)$
- Sending $4^e \cdot c = (4x)^e$ in Z_N reveals msb(4x mod N) = $msb_3(x)$
- … and so on to reveal all of x

- HTTPS Defense

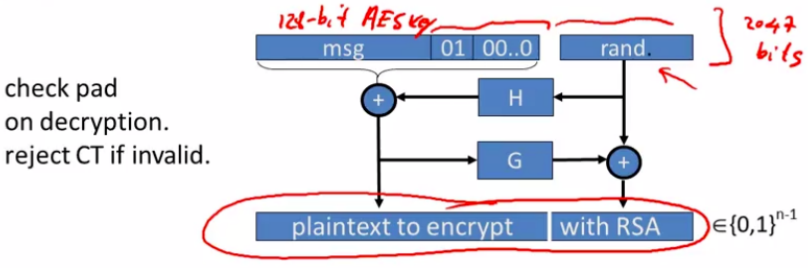
Attacks discovered by Bleichenbacher and Klima et al. … can be avoided by treating incorrectly formatted message blocks … in a manner indistinguishable from correctly formatted RSA blocks. In other words:

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 (Optimal asymmetric encryption padding)

New preprocessing function: OAEP [BR94]



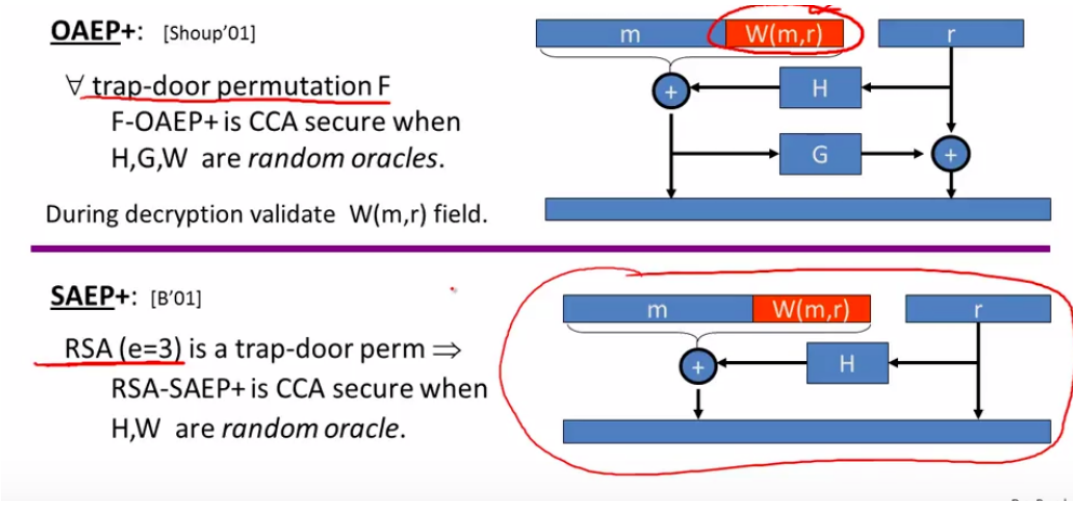
check pad
on decryption.
reject CT if invalid.

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

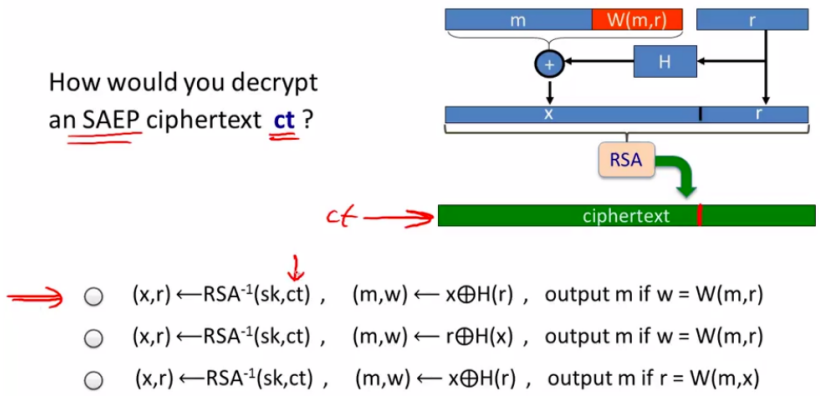
in practice: use SHA-256 for H and G

- optimal because the ciphertext is short as possible
- Thm is false if use general trap door permutation

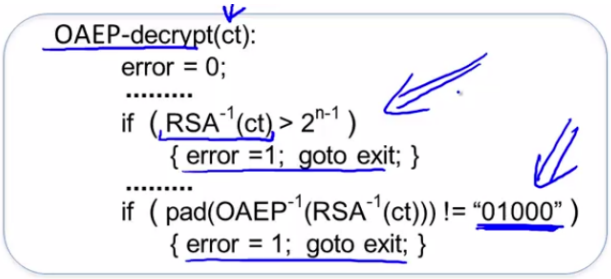
- OAEP Improvements



- Example



- Subtleties in implementing OAEP



Problem: timing information leaks type of error

⇒ Attacker can decrypt any ciphertext

- **Public Key Encryption from Trapdoor Permutations: attacks**
- **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 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 ($d \approx 2^{128}$)

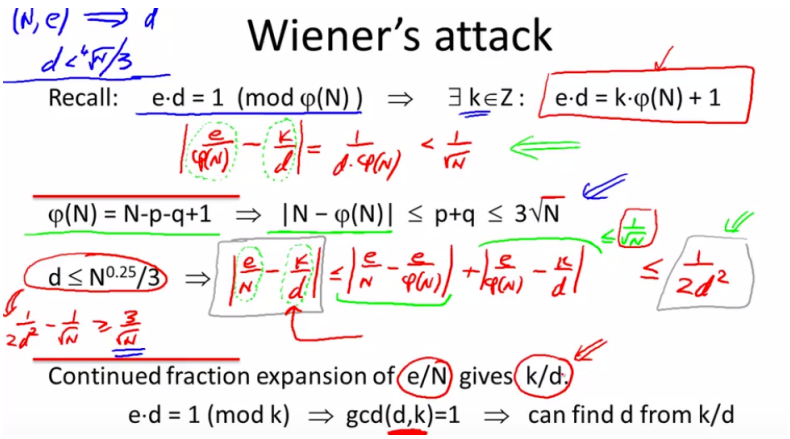
$\Rightarrow$ $c^d = m \pmod N$

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

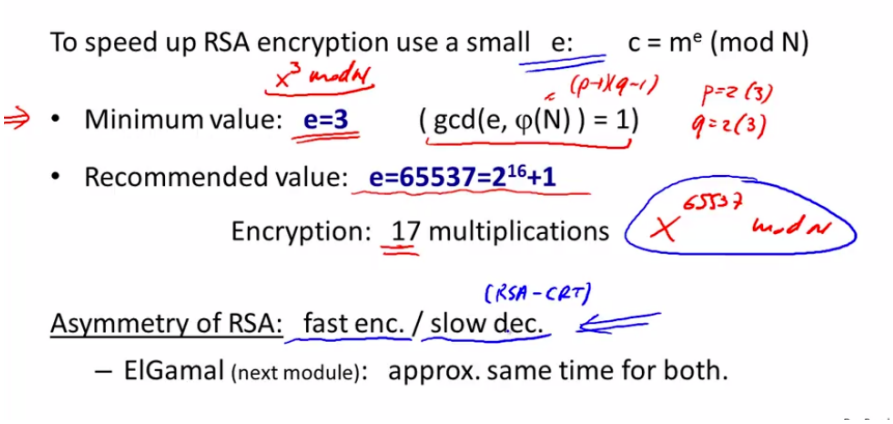
BD'98: if $d < N^{0.292}$ then RSA is insecure (open: $d < N^{0.5}$)

Insecure: priv. key d can be found from (N,e)

- Wiener's attack
  - given (N,e) recover d
  - d less than the 4th root n divided by 3



- **RSA in Practice**
- RSA With Low public exponent



- 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 - decryption time should be independent of the arguments
  - Power attack - defend against power analysis attacks
  - Faults attack - one error reveals secret key

Timing attack: (Kocher 97)
The time it takes to compute $c^d \pmod N$ can expose d

Power attack: (Kocher 99)
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

x4 speedup

Suppose error occurs when computing $\hat{x}_q$, 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 $\Rightarrow$ $\gcd((x')^e - c, N) = p$

$= 0 \pmod p$, $\neq 0 \pmod q$

Dan Boneh

- RSA Key Generation Trouble

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 $\Rightarrow$ $\gcd(N_1, N_2) = p$

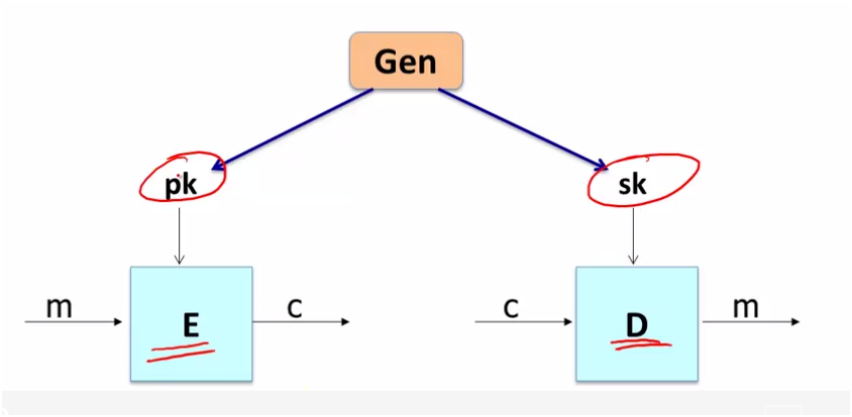
- RSA Key Generation Trouble

Experiment: factors 0.4% of public HTTPS keys !!

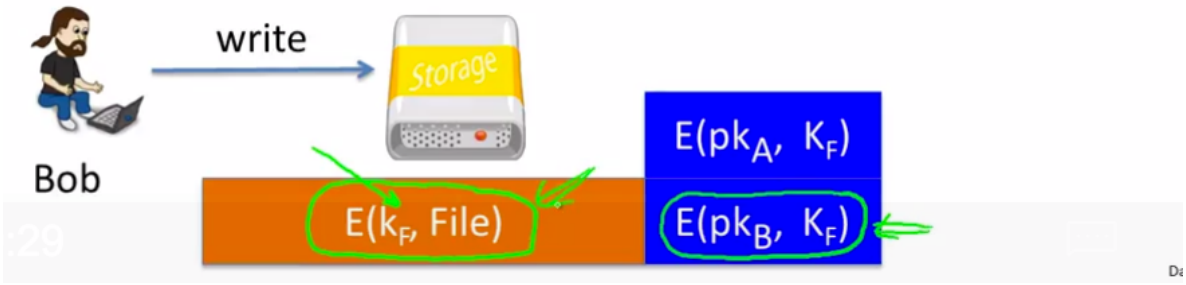
Lesson:

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

- **Public Key Encryption From Diffie-Hellman: ElGamal**
- **The ElGamal Public-key System**
- Recap: public key encryption: (Gen, E, D)



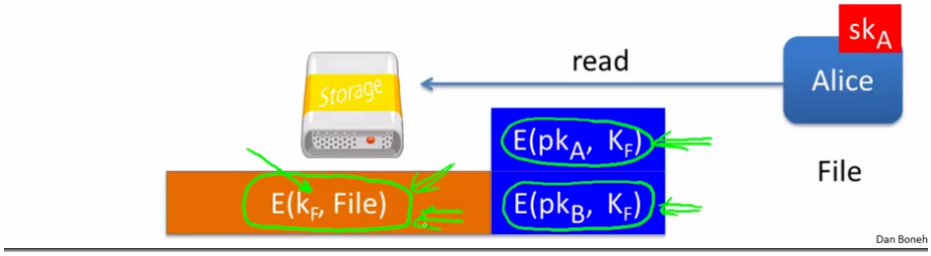
- Recap: public-key encryption applications
  - Key exchange (e.g. in HTTPS)
  - Encryption in non-interactive settings:
    - Secure Email: Bob has Alice's pub-key and sends her an email
    - Encrypted file systems
    - Bob encrypts the key Kf for alice using the public key set for alice the pk_a
    - When alice wants to decrypt the file alice uses the sk_a file that she has and decrypts the key
    - no interaction with bob but alice can read the file for her self



Key exchange (e.g. in HTTPS)

Encryption in non-interactive settings:

- Secure Email: Bob has Alice's pub-key and sends her an email
- Encrypted File Systems

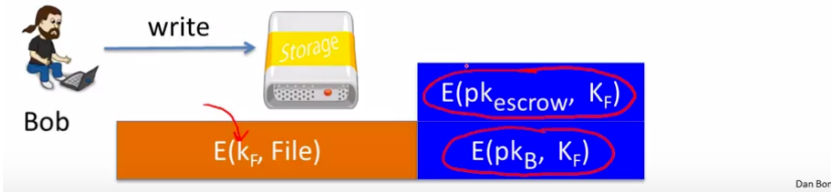


- Recap: public key encryption applications

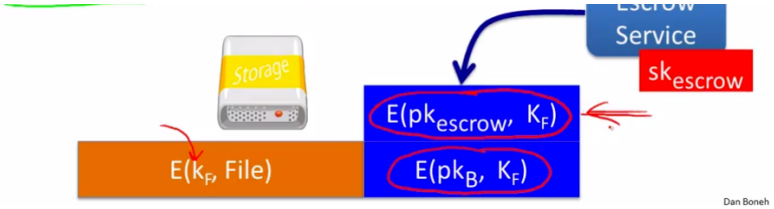
Key exchange (e.g. in HTTPS)

Encryption in non-interactive settings:

- Secure Email: Bob has Alice's pub-key and sends her an email
- Encrypted File Systems
- Key escrow: data recovery without Bob's key



- escrow service uses secret key to decrypt the header and obtain the key
- then using the key obtained through the use of the escrow service obtain the file through decryption



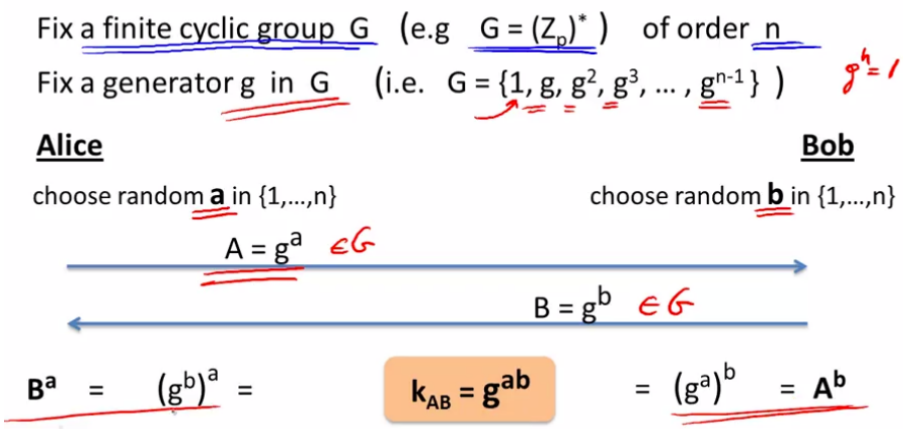
- Constructions

This week: two families of public-key encryption schemes

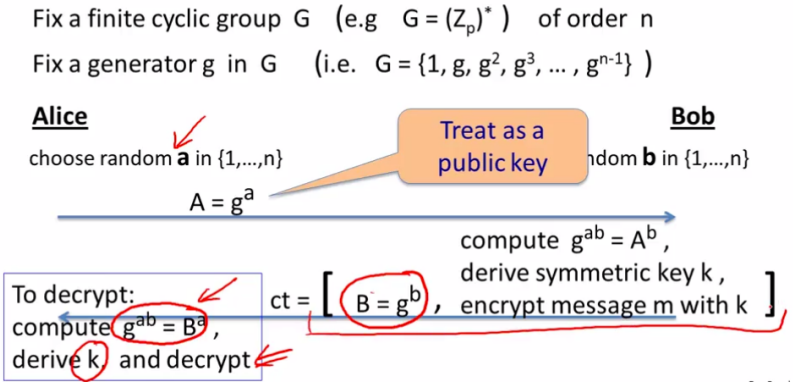
- Previous lecture: based on trapdoor functions (such as RSA)
  - Schemes: ISO standard, OAEP+, …
- This lecture: based on the Diffie-Hellman protocol
  - Schemes: ElGamal encryption and variants (e.g. used in GPG)

Security goals: chosen ciphertext security

- Review: the Diffie-Hellman protocol



  - the attacker is allowed to see both g to the a and g to the b. The shared secret is g to at the a b. The attacker is also allowed to see the g in the generator. However this is believed to be a hard or difficult problem
- ElGamal: converting to pub-key encryption



- The ElGamal System (a modern view)
  - symmetric system encryption decryption
  - better to choose a random generator every time

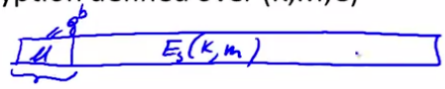
- G: finite cyclic group of order n
- (E_s, D_s): symmetric auth. encryption defined over (K,M,C)
- H: $G^2 \longrightarrow K$ a hash function

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

- Key generation Gen:
  - choose random generator g in G and random a in Z_n
  - output sk = a , pk = (g, h=g^a)

- The ElGamal System (a modern view)

  - G: finite cyclic group of order n
  - (E_s, D_s) : symmetric auth. encryption defined over (K,M,C)
  - H: $G^2 \longrightarrow K$ a hash function



E(pk=(g,h), m) :

$b \xleftarrow{R} Z_n$, $u \leftarrow g^b$, $v \leftarrow h^b$

$k \leftarrow H(u,v)$, $c \leftarrow E_s(k, m)$

output (u, c)

D(sk=a, (u,c)) :

$v \leftarrow u^a = (g^b)^a = g^{ab}$

$k \leftarrow H(u,v)$, $m \leftarrow D_s(k, c)$

output m

- ElGamal Performance
  - windowed exponentiation - is when you precompute the tables

E(pk=(g,h), m) :

$b \leftarrow Z_n$, $u \leftarrow g^b$, $v \leftarrow h^b$

D(sk=a, (u,c)) :

$v \leftarrow u^a$

Encryption: 2 exp. (fixed basis)

- Can pre-compute $[g^{(2^i)}, h^{(2^i)}$ for $i=1,\ldots,\log_2 n]$
- 3x speed-up (or more)

Decryption: 1 exp. (variable basis)

- **ElGamal Security**
- Computational Diffie-Hellman Assumption

G: finite cyclic group of order n

Comp. DH (CDH) assumption holds in G if: $g, g^a, g^b \not\Rightarrow g^{ab}$

for all efficient algs. A:

$$\Pr[\, A(g, g^a, g^b) = g^{ab} \,] < \text{negligible}$$

where $g \leftarrow \{\text{generators of } G\}$, $a, b \leftarrow Z_n$

- Hash Diffie-Hellman Assumption