رمزنگاری، امنیت اطلاعات و حریم خصوصی

ارائه: دکتر سیدعلی لاجوردی

بخش هشتم

The public-key setting

Review: private-key setting

- Two (or more) parties who wish to securely communicate share a uniform, secret key k
  in advance
- Same key k used for sending or receiving
  - Either party can send or receive
  - If multiple parties share a key, no way to distinguish them from based on the key
- Secrecy of k is critical
  - No security if attacker knows k

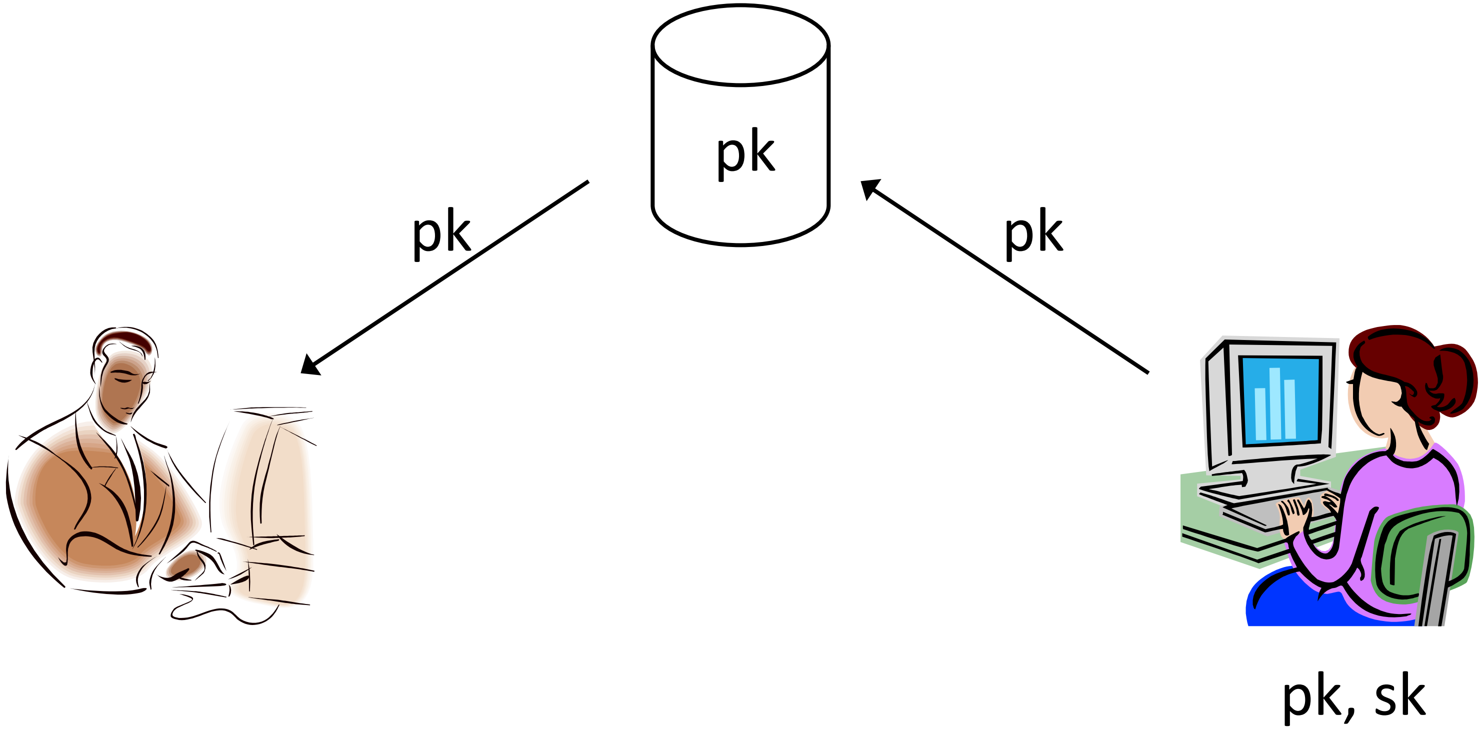
The public-key setting

- One party generates a pair of keys: public key pk and private key sk
  - Public key is widely disseminated
  - Private key is kept secret, and shared with no one

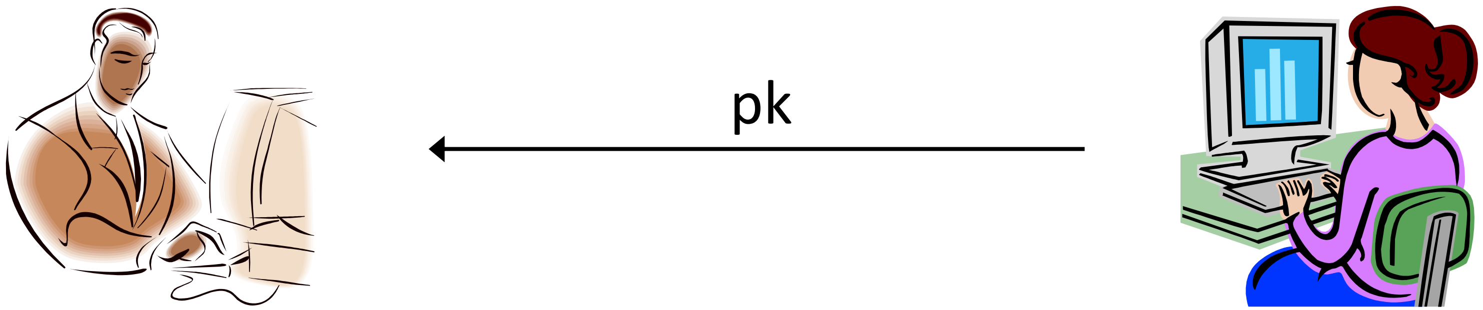
- Private key used by the party who generated it; public key used by anyone else
  - Also called asymmetric cryptography

- Security must hold even if the attacker knows pk

Public-key distribution I



Public-key distribution II



pk, sk

Public-key distribution

- Previous figures (implicitly) assume parties are able to obtain correct copies of each others' public keys
  - I.e., the attacker is passive during key distribution

- We will revisit this assumption later

How does this address the drawbacks of private-key crypto…?

- Key distribution
  - Public keys can be distributed over public (but authenticated) channels
- Key management in system of N users
  - Each user stores 1 private key and N-1 public keys; only N keys overall
  - Public keys can be stored in a central, public directory
- Applicability to "open systems"
  - Even parties who have no prior relationship can find each others' public keys and use them

Public-key vs. private-key crypto

- Note that public-key cryptography is strictly stronger than private-key cryptography
  - Parties who wish to securely communicate could each generate public/private keys and then share them with each other
  - Use appropriate key depending on who is sending or receiving

Why study private-key crypto?

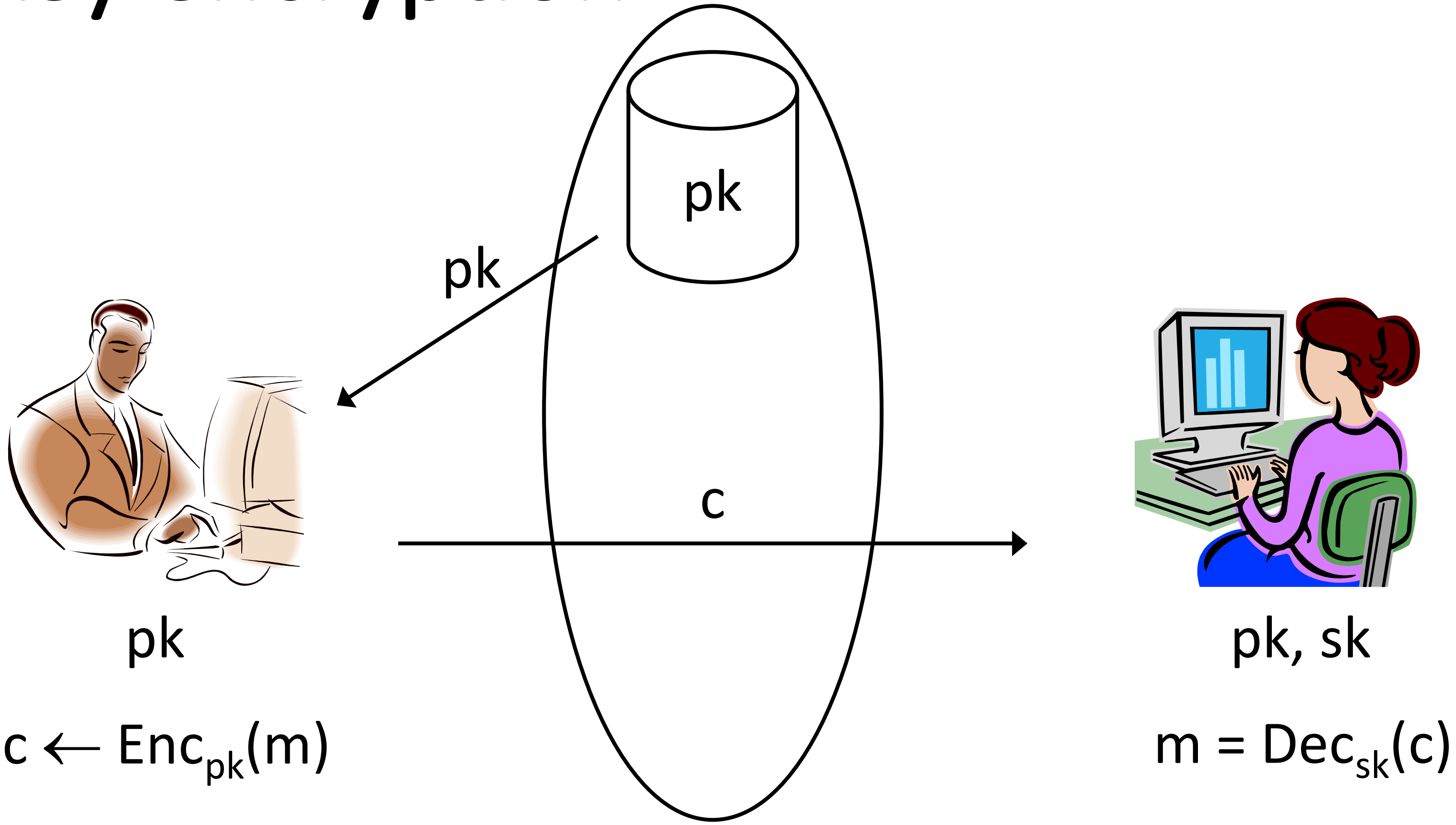
- Public-key crypto is roughly 2-3 orders of magnitude slower than private-key crypto
  - Also 2-10$\times$ higher communication
- Public-key cryptography requires stronger assumptions
- If private-key crypto is an option, better to use it!

- As we will see, private-key cryptography is used for efficiency even in the public-key setting

Primitives

	Private-key setting	Public-key setting
Secrecy	Private-key encryption	Public-key encryption
Integrity	Message authentication codes	Digital signature schemes

Public-key encryption



Public-key encryption

- A public-key encryption scheme consists of three PPT algorithms:
  - Gen: key-generation algorithm that on input 1n outputs pk, sk
  - Enc: encryption algorithm that on input pk and a message m outputs a ciphertext c
  - Dec: decryption algorithm that on input sk and a ciphertext c outputs a message m or an error $\perp$

For all m and pk, sk output by Gen,
$\mathrm{Dec}_{sk}(\mathrm{Enc}_{pk}(m)) = m$

CPA-security

- Fix a public-key encryption scheme Π and an adversary A
- Define experiment PubK-CPAA, Π(n):
  - Run Gen(1n) to get keys pk, sk
  - Give pk to A, who outputs (m0, m1) of same length
  - Choose uniform b $\in$ {0,1} and compute the ciphertext c $\leftarrow$ Encpk(mb); give c to A
  - A outputs a guess b', and the experiment evaluates to 1 if b'=b

- Public-key encryption scheme Π is CPA-secure if for all PPT adversaries A:

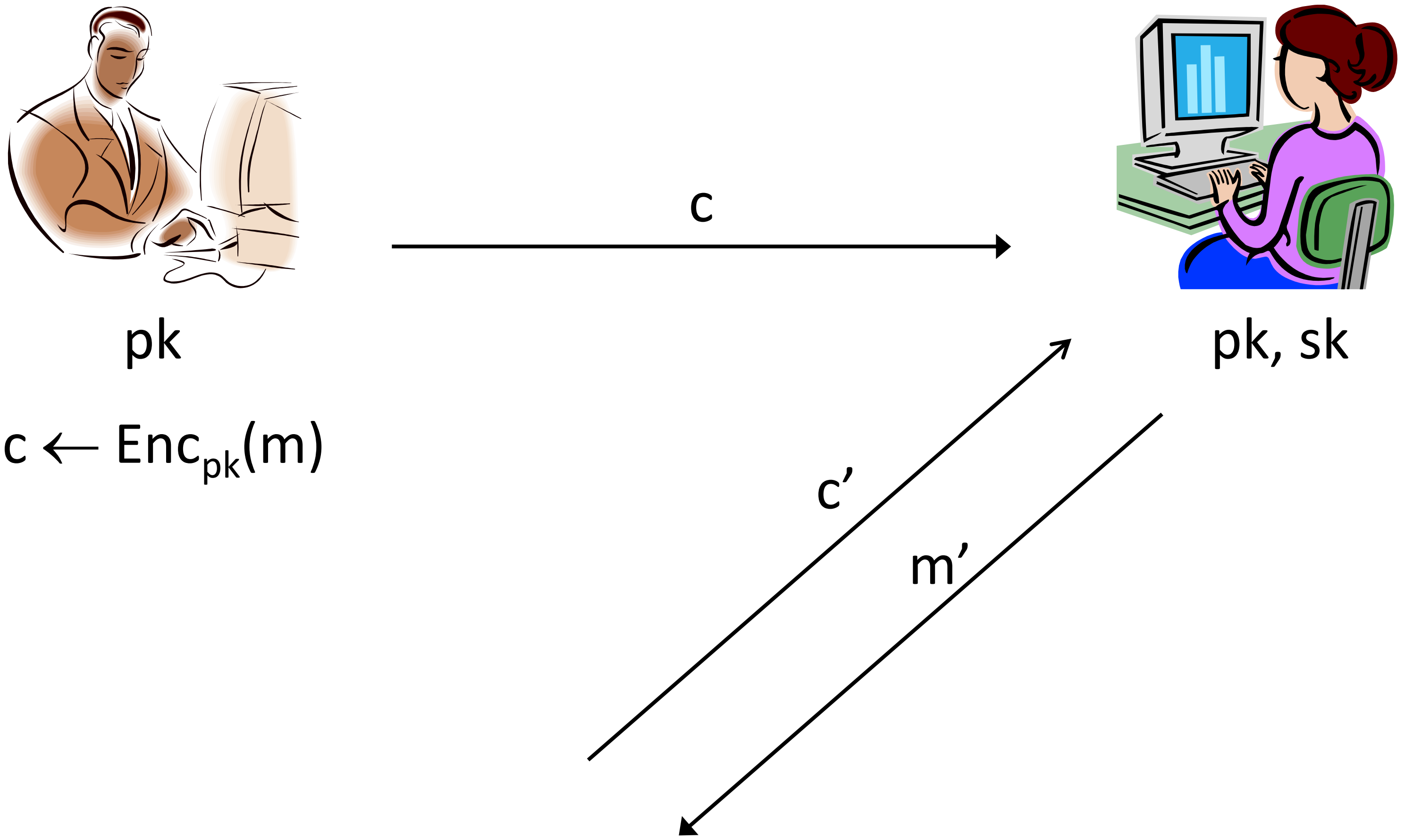
  Pr[PubK-CPAA, Π(n) = 1] ≤ ½ + negl(n)

Notes on the definition

- No encryption oracle?!
  - Encryption oracle redundant in public-key setting

- $\Rightarrow$ No perfectly secret public-key encryption
- No deterministic public-key encryption scheme can be CPA-secure
- CPA-security implies security for encrypting multiple messages (as in the private-key case)

Chosen-ciphertext attacks



Chosen-ciphertext attacks

- Chosen-ciphertext attacks are arguably even a greater concern in the public-key setting
  - Attacker might be a legitimate sender
  - Easier for attacker to obtain full decryptions of ciphertexts of its choice

- Related concern: malleability
  - I.e., given a ciphertext c that is the encryption of an unknown message m, might be possible to produce ciphertext c' that decrypts to a related message m'
  - This is also undesirable in the public-key setting
- Can define CCA-security for public-key encryption by analogy to the definition for private-key encryption

Hybrid encryption and KEMs

Encrypting long messages

- Public-key encryption schemes “natively” defined for “short” messages

- How can longer messages be encrypted?

Encrypting long messages

- Can always encrypt block-by-block
  - I.e., to encrypt $M = m_1, m_2, \ldots, m_\ell$, do:
    $\mathrm{Enc}_{pk}(m_1), \ldots, \mathrm{Enc}_{pk}(m_\ell)$

- If the underlying scheme is CPA-secure (for short messages), then this is CPA-secure (for arbitrary length messages)
  - Why?

Note

- (Public-key) encryption is NOT a block cipher
  - F_k is deterministic, one-to-one, and looks random
  - Enc_{pk} is randomized and not one-to-one (if it is CPA-secure), and may not look random

  $\Rightarrow$ CTR-mode/CBC-mode don't make sense for public-key encryption

  Also may not be secure...
  - "ECB mode" is secure for public-key encryption
    - Because underlying scheme is randomized

Encrypting long messages

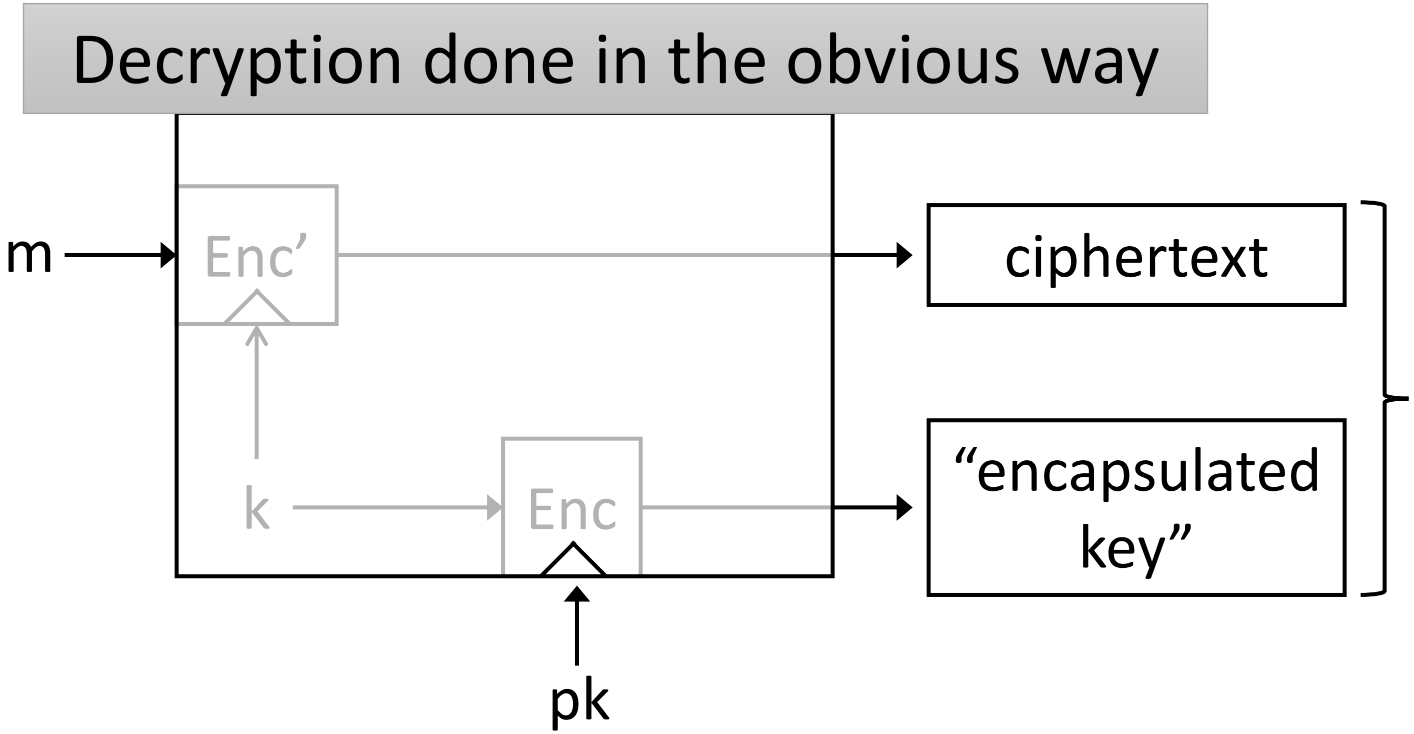
- Encrypting block-by-block is inefficient
  - Ciphertext expansion in each block
  - Public-key encryption is “expensive”

- Can we do better?

Hybrid encryption

- Main idea
  - Use public-key encryption to establish a (shared, secret) key k
  - Use k to encrypt the message *with a symmetric-key encryption scheme*
- Benefits
  - Lower ciphertext expansion
  - Amortized efficiency of *symmetric-key* encryption

Hybrid encryption



The *functionality* of public-key encryption
with the (asymptotic) *efficiency* of private-key encryption!

Formally

- Let Π be a public-key scheme, and let Π' be a symmetric-key scheme
- Define Π_{hy} as follows:
  - Gen_{hy} = Gen (i.e., same as Π)
  - $Enc_{hy}(pk, m)$:
    - Choose $k \leftarrow \{0,1\}^n$
    - $c \leftarrow Enc_{pk}(k)$
    - $c' \leftarrow Enc'_k(m)$
    - Output c, c'
  - Decryption done in the natural way…

Security of hybrid encryption

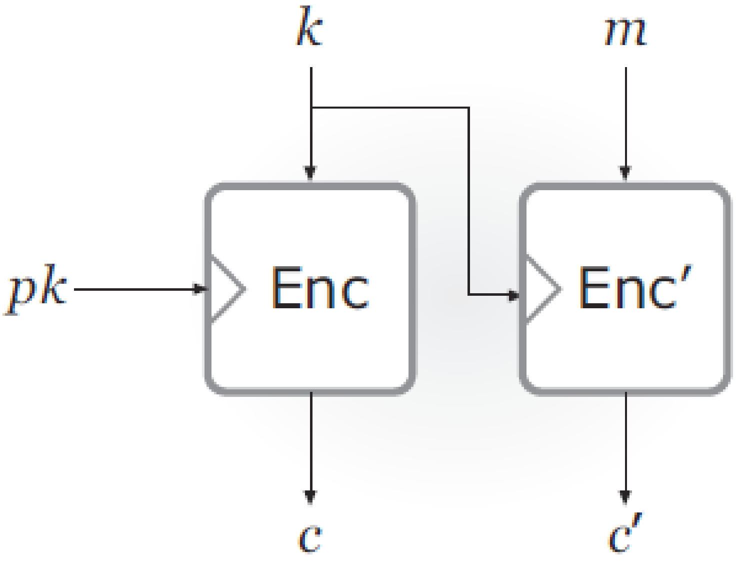
- If Π is a CPA-secure public-key scheme, and Π' is a CPA-secure private-key scheme, then Π_{hy} is a CPA-secure public-key scheme
  - In fact, suffices for Π' to be EAV-secure

- If Π is a CCA-secure public-key scheme, and Π' is a CCA-secure private-key scheme, then Π_{hy} is a CCA-secure public-key scheme

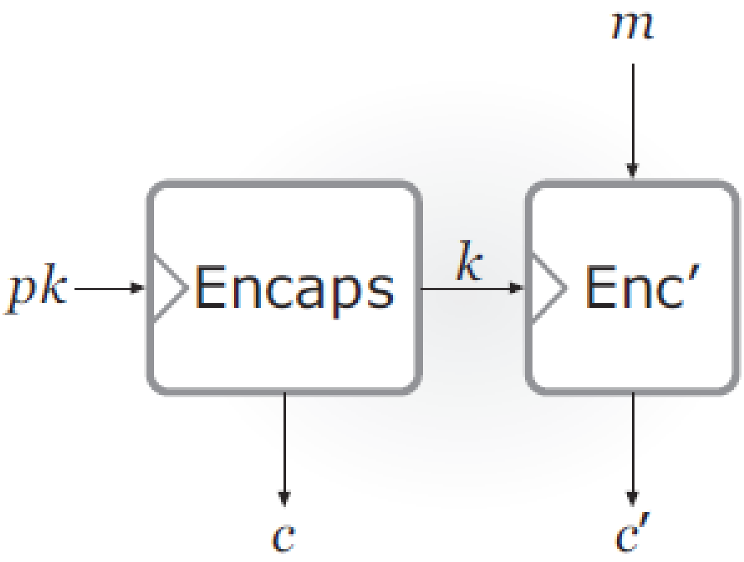
KEM/DEM paradigm

- For hybrid encryption, something *weaker* than public-key encryption suffices
- Sufficient to have a “key encapsulation mechanism” (KEM) that takes a public key and outputs a ciphertext c and a key k
  - Correctness: k can be recovered from c given sk
  - Security: k is indistinguishable from uniform given pk and c; can formally define CPA-/CCA-security
- Can still combine with symmetric-key encryption (DEM) as before!

KEM/DEM paradigm



Hybrid encryption



KEM/DEM

Security of KEM/DEM

- If Π is a CPA-secure KEM, and Π' is a CPA-secure private-key scheme, then combination is a CPA-secure public-key scheme
  - Suffices for Π' to be EAV-secure

- If Π is a CCA-secure KEM, and Π' is a CCA-secure private-key scheme, then combination is a CCA-secure public-key scheme

KEMs vs. PKE schemes

- For short messages, direct encryption using a PKE scheme (with no hybrid encryption) can sometimes be the best choice

- For anything longer, KEM/DEM or hybrid encryption will be more efficient
  - This is how things are done in practice (unless very short messages are being encrypted)

Dlog-based PKE

Diffie-Hellman key exchange

$(G, q, g) \leftarrow \mathcal{G}(1^n)$

$x \leftarrow \mathbb{Z}_q$

$h_1 = g^x$

$$\xrightarrow{G,\ q,\ g,\ h_1}$$

$y \leftarrow \mathbb{Z}_q$

$h_2 = g^y$

$$\xleftarrow{h_2}$$

$k = (h_1)^y$

$c_2 = k \cdot m$

$$\xleftarrow{c_2}$$

$k = (h_2)^x$

$m = c_2/k$

El Gamal encryption

Public key

G, q, g, h_1 →

← $h_2, h_1^y \cdot m$

← c_2

$(G, q, g) \leftarrow \mathcal{G}(1^n)$
$x \leftarrow \mathbb{Z}_q$
$h_1 = g^x$

$k = (h_2)^x$
$m = c_2/k$

$y \leftarrow \mathbb{Z}_q$
$h_2 = g^y$

$k = (h_1)^y$
$c_2 = k \cdot m$

El Gamal encryption

- $\mathrm{Gen}(1^n)$
  - Run $\mathcal{G}(1^n)$ to obtain G, q, g. Choose uniform $x \in \mathbb{Z}_q$. The public key is (G, q, g, g^x) and the private key is x

- $\mathrm{Enc}_{pk}(m)$, where pk = (G, q, g, h) and $m \in G$
  - Choose uniform $y \in \mathbb{Z}_q$. The ciphertext is $g^y, h^y \cdot m$

- $\mathrm{Dec}_{sk}(c_1, c_2)$, where sk = x
  - Output $c_2/c_1^x = c_2 \cdot c_1^{-x}$

Security?

- If the DDH assumption is hard for $\mathcal{G}$, then the El Gamal encryption scheme is CPA-secure
  - Follows from security of Diffie-Hellman key exchange, or can be proved directly
  - Note that the discrete-logarithm assumption alone is not enough here

$\Rightarrow$ Secure for encryption of multiple messages (using the same public key)!

  - Note that sender(s) must use fresh randomness for each encryption

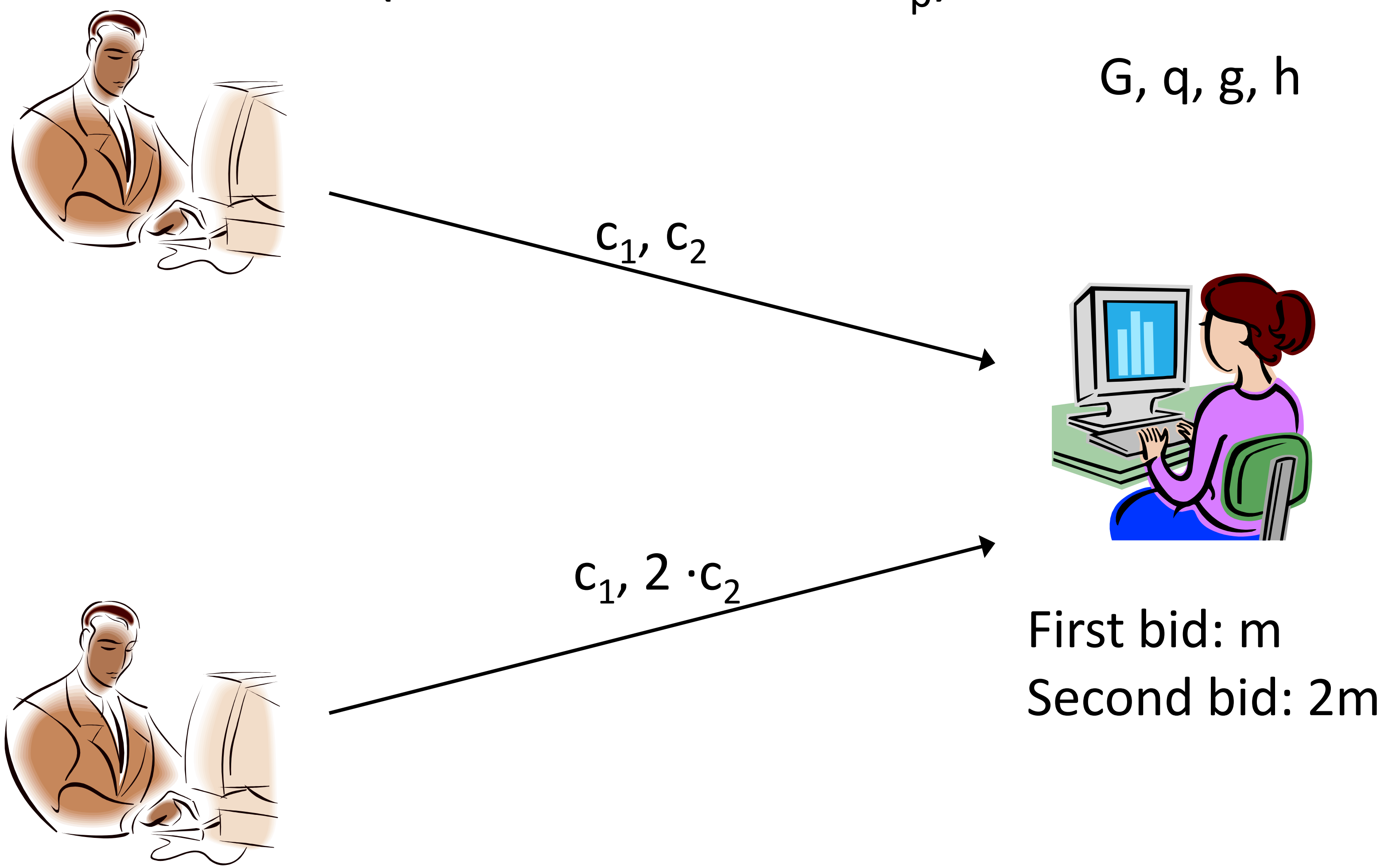
Chosen-ciphertext attacks?

- El Gamal encryption is *not* secure against chosen-ciphertext attacks
  - Follows from the fact that it is *malleable*

- Given ciphertext (c_1, c_2), transform it to obtain the ciphertext $(c_1, c'_2) = (c_1, \alpha \cdot c_2)$ for arbitrary α
  - Since $(c_1, c_2) = (g^y, h^y \cdot m)$,
    we have $(c_1, c'_2) = (g^y, h^y \cdot (\alpha m))$
  - I.e., encryption of m becomes an encryption of αm!

Attack!

(Assume $2 \in G \subset \mathbb{Z}^*_p$)



El Gamal in practice

- Parameters G, q, g are standardized and shared

- Need to encode message as a group element
  - In some groups, there are natural ways to do this
  - In other cases, not as easy
  - Can avoid this if using El Gamal as a KEM!

Hybrid encryption with El Gamal?

- To use hybrid encryption with El Gamal, would need to encode key k as a group element
  - Can we avoid this?

- The sender doesn't care about encrypting a *specific* key, it just needs to send a random key
  - Idea: encrypt a random group element K; define the key as k = H(K)

KEM based on El Gamal

- Gen(1^n)
  - Run $\mathcal{G}(1^n)$ to obtain G, q, g. Choose uniform $x \in \mathbb{Z}_q$. The public key is (G, q, g, g^x) and the private key is x
- Ecaps$_{pk}$, where pk = (G, q, g, h)
  - Choose uniform $y \in \mathbb{Z}_q$. The ciphertext is g^y, and the key is $k = H(h^y)$
- Decaps$_{sk}$(c), where sk = x
  - Output $k = H(c^x)$

Security?

- If the DDH assumption holds, and H is modeled as a random oracle, then this KEM is CPA-secure

Complete scheme

- Combine the KEM with private-key encryption

- I.e., encryption of message m is
  $$g^y, \mathrm{Enc}'_k(m),$$
  where $k = H(h^y)$ and Enc' is a symmetric-key encryption scheme (e.g., CTR-mode)
  - If Enc' is CPA-secure, DDH assumptions holds, and H is modeled as a random oracle, this is a CPA-secure public-key encryption scheme

Chosen-ciphertext security

- Under stronger assumptions, this approach can be proven to give CCA security
  - If Enc’ is a CCA-secure symmetric-key scheme
- Can at least see why the previous attack no longer works
- Standardized as DHIES/ECIES

RSA-based PKE

Recall…

- Let p, q be random, equal-length primes
- Compute modulus N=pq
- Choose e, d such that $e \cdot d = 1 \bmod \phi(N)$

- The e^{th} root of x modulo N is $[x^d \bmod N]$
  - I.e., easy to compute given p, q (or d)
- *RSA assumption*: given N, e <u>only</u>, it is hard to compute the e^{th} root of a uniform $c \in \mathbb{Z}_N^*$

“Plain” RSA encryption

$\xrightarrow{N,\ e}$

$\xleftarrow{c}$

$(N, e, d) \leftarrow \mathrm{RSAGen}(1^n)$

$pk = (N, e)$

$sk = d$

$m = [c^d \bmod N]$

$c = [m^e \bmod N]$

Security?

- This scheme is *deterministic*
  - Cannot be CPA-secure!

- RSA assumption only refers to hardness of computing the e^{th} root of a *uniform* c
  - c is not uniform unless m is
  - Why would m be uniform?
  - Easy to compute e^{th} root of $c = [m^e \bmod N]$ when m is small

- RSA assumption only refers to hardness of computing the e^{th} root of c *in its entirety*
  - *Partial* information about the e^{th} root may be leaked
  - (In fact, this is the case)

Chosen-ciphertext attacks

- Of course, plain RSA cannot be CCA-secure since it is not even CPA-secure…
  - …but chosen-ciphertext attacks are devastating

- Given ciphertext c for unknown message m, can compute $c' = [\alpha^e \cdot c \bmod N]$
  - What does this decrypt to?

How to fix plain RSA?

- One approach: use a *randomized* encoding

- I.e., to encrypt m
  - First compute some reversible, randomized mapping $M \leftarrow \text{Encode}(m)$
  - Then set $c := [M^e \bmod N]$

- To decrypt c
  - Compute $M := [c^d \bmod N]$
  - Recover m from M

PKCS #1 v1.5

- Standard issued by RSA labs in 1993
- Idea: introduce random padding
  - Encode(m) = r|m

- I.e., to encrypt m
  - Choose random r
  - Compute the ciphertext c := [(r|m)e mod N]

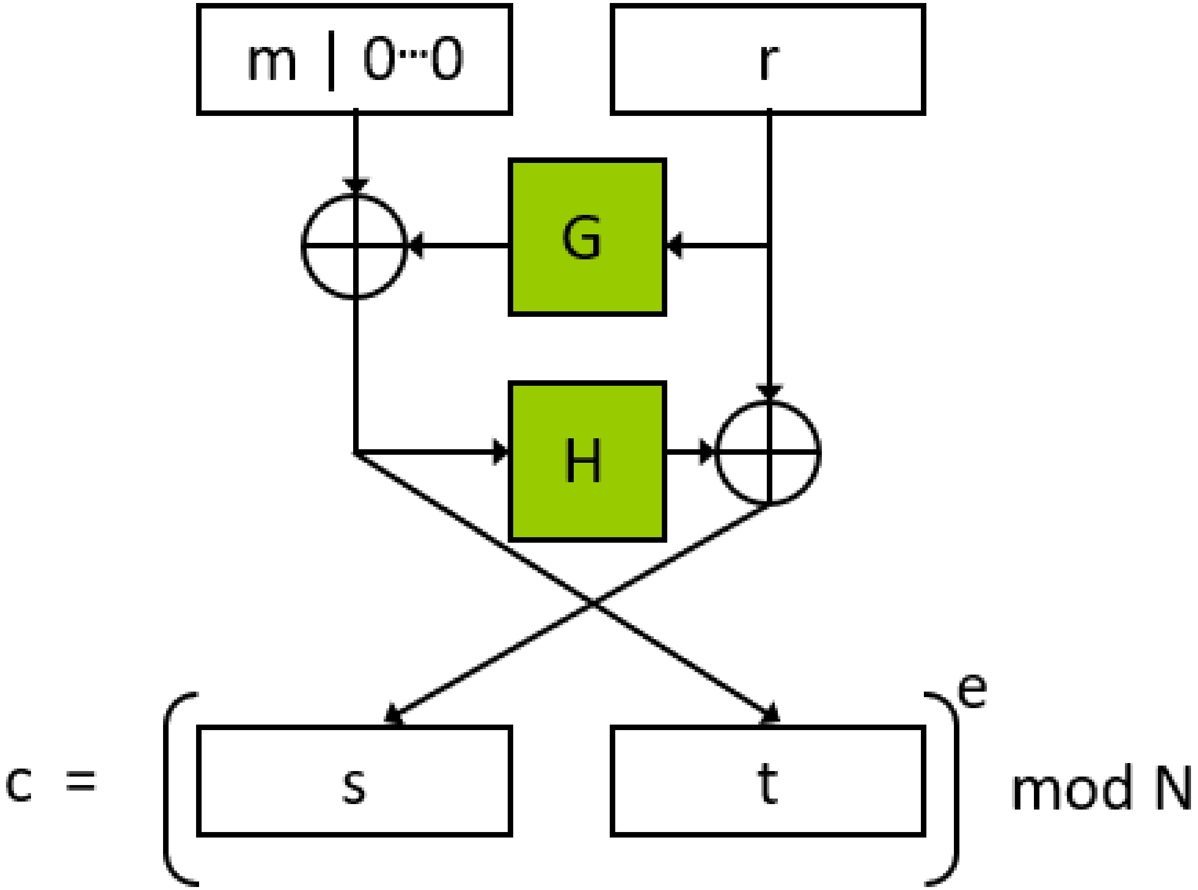
- Issues:
  - No proof of CPA-security (unless m is very short)
  - Chosen-plaintext attacks are known if r is too short
  - Chosen-ciphertext attacks are still possible

PKCS #1 v2.0

- *Optimal asymmetric encryption padding* (OAEP) applied to message first

- This padding introduces *redundancy*, so that not every c $\in \mathbb{Z}^*_N$ is a valid ciphertext
  - Need to check for proper format upon decryption
  - Return error if not properly formatted

OAEP



Security?

- RSA-OAEP can be proven CCA-secure under the RSA assumption, if G and H are modeled as random oracles

- Widely used in practice…

RSA-based KEM

- Idea: use plain RSA as before…

  …but on a random value!

- Then use that random value to derive a key

RSA-based KEM

- Encaps:
  - Choose uniform $r \in \mathbb{Z}^*_N$
  - Ciphertext is $c = [r^e \bmod N]$
  - Key is $k = H(r)$

- Decaps(c)
  - Compute $r = [c^d \bmod N]$
  - Compute the shared key $k = H(r)$

Security?

- This is CCA-secure under the RSA assumption, if H is modeled as a random oracle

Comparison to RSA-OAEP?

- The RSA-KEM must be used with a symmetric-key encryption scheme
- For very short messages (< 1500 bits), RSA-OAEP will have shorter ciphertexts
- For anything longer, ciphertexts will be the same length; RSA-KEM is simpler

PKE in practice

- What is the best way to encrypt a 1MB file?
  - Use 1MB parameters?
  - Use 1000-bit parameters; encrypt file in chunks
  - Use hybrid encryption/KEM-DEM approach

PKE in practice

- Current recommended parameters:
  - RSA-based schemes: ≈2000-bit modulus N
  - Dlog, order-q subgroup of $\mathbb{Z}^*_p$: $\|q\| \approx 256$, $\|p\| \approx 2000$
  - Dlog, order-q elliptic-curve group: $\|q\| \approx 256$; group elements require ≈256 bits