Department of
Computer Science and Engineering
计算机科学与工程系

CSE5014 CRYPTOGRAPHY AND NETWORK SECURITY

Dr. QI WANG

Department of Computer Science and Engineering
Office: Room413, CoE South Tower
Email: wangqi@sustech.edu.cn

Cyclic groups

- Let G be a finite group of order m (written multiplicatively)

- Let g be some element of G

- Consider the set $\langle g \rangle = \{g^0, g^1, \ldots\}$
  - We know $g^m = 1 = g^0$, so the set has $\leq m$ elements
  - If the set has m elements, then it is all of G!
    - In this case, we say g is a *generator* of G
    - If G has a generator, we say G is *cyclic*

Uniform sampling

- Given cyclic group G of order q along with generator g, easy to sample a uniform $h \in G$
  - Choose uniform $x \in \{0, \ldots, q-1\}$: set $h := g^x$
- Fix *cyclic* group G of order q, and *generator* g
- We know that $\{g^0, g^1, \ldots, g^{q-1}\} = G$
  - For every $h \in G$, there is a unique $x \in \mathbb{Z}_q$, s.t. $g^x = h$
  - Define $\log_g h$ to be this x – the *discrete logarithm* of h with respect to g (in the group G)

Discrete-logarithm problem (informal)

- DLog problem in G:
  Given *generator* g and element h, compute $\log_g h$

- DLog assumption in G:
  Solving the discrete log problem in G is hard

Discrete-logarithm problem (informal)

- DLog problem in G:

  Given *generator* g and element h, compute $\log_g h$

- DLog assumption in G:

  Solving the discrete log problem in G is hard

- In $\mathbb{Z}^*_{3092091139}$
  - What is $\log_2 1656755742$?

Discrete-logarithm problem

- Let $\mathcal{G}$ be a group-generation algorithm
  - On input 1^n, outputs a cyclic group G, its order q (with $\|q\| = n$), and a generator g

- For algorithm A, define experiment $Dlog_{A,\mathcal{G}}(n)$:
  - Compute $(G, q, g) \leftarrow \mathcal{G}(1^n)$
  - Choose uniform $h \in G$
  - Run $A(G, q, g, h)$ to get x
  - Experiment evaluates to 1 if $g^x = h$

- **Definition 11.3** The *discrete-logarithm problem* is hard relative to $\mathcal{G}$ if for all PPT algorithms A,

$$\Pr[Dlog_{A,\mathcal{G}}(n) = 1] \leq negl(n)$$

Diffie-Hellman problems

- Fix cyclic group G and *generator* g
- Define $DH_g(h_1, h_2) = DH_g(g^x, g^y) = g^{xy}$

Diffie-Hellman problems

- Fix cyclic group G and *generator* g
- Define $DH_g(h_1, h_2) = DH_g(g^x, g^y) = g^{xy}$
- In $\mathbb{Z}_{11}^*$
  - $\langle 2 \rangle = \{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\}$
  - So $DH_2(7, 5) = ?$

Diffie-Hellman problems

- Fix cyclic group G and *generator* g
- Define $DH_g(h_1, h_2) = DH_g(g^x, g^y) = g^{xy}$
- In $\mathbb{Z}_{11}^*$
  - $\langle 2 \rangle = \{1, 2, 4, 8, 5, 10, 9, 7, 3, 6\}$
  - So $DH_2(7, 5) = ?$
- In $\mathbb{Z}_{3092091139}^*$
  - What is $DH_2(1656755742, 938640663) = ?$
  - Is 1994993011 the answer, or is it just a random element of $\mathbb{Z}_{3092091139}^*$?

Diffie-Hellman assumptions

- *Computational* Diffie-Hellman (CDH) problem:
  – Given g, h_1, h_2, compute $DH_g(h_1, h_2)$

Diffie-Hellman assumptions

- *Computational* Diffie-Hellman (CDH) problem:
  - Given g, h_1, h_2, compute $DH_g(h_1, h_2)$
- *Decisional* Diffie-Hellman (DDH) problem:
  - Given g, h_1, h_2, distinguish $DH_g(h_1, h_2)$ from a uniform element of G

DDH problem

- Let $\mathcal{G}$ be a group-generation algorithm
  - On input 1^n, outputs a cyclic group G, its order q (with $\|q\| = n$), and a generator g

DDH problem

- Let $\mathcal{G}$ be a group-generation algorithm
  - On input 1^n, outputs a cyclic group G, its order q (with $\|q\| = n$), and a generator g
- The DDH problem is hard relative to $\mathcal{G}$ if for all PPT algorithm A:

  $|\Pr[A(G, q, g, g^x, g^y, g^z) = 1] - \Pr[A(G, q, g, g^x, g^y, g^{xy}) = 1]| \leq \epsilon(n)$

Relating the Diffie-Hellman problems

- Relative to $\mathcal{G}$
  - If the discrete-logarithm problem is easy, so is the CDH problem
  - If the CDH problem is easy, so is the DDH problem

Relating the Diffie-Hellman problems

- Relative to $\mathcal{G}$
  - If the discrete-logarithm problem is easy, so is the CDH problem
  - If the CDH problem is easy, so is the DDH problem

  - I.e., the DDH assumption is *stronger* than the CDH assumption
  - I.e., the CDH assumption is *stronger* than the dlog assumption

Group selection

- The *discrete logarithm* is not hard in all groups!

Group selection

- The *discrete logarithm* is not hard in all groups!
- Nevertheless, there are certain groups where the problem is believed to be hard

Group selection

- The *discrete logarithm* is not hard in all groups!
- Nevertheless, there are certain groups where the problem is believed to be hard
- For cryptographic applications, best to use *prime-order* groups
  - The *dlog* problem becomes easier if the order of the group has small prime factors
  - Prime-order groups have several nice features: e.g., every element except identity is a generator

Group selection

- The *discrete logarithm* is not hard in all groups!
- Nevertheless, there are certain groups where the problem is believed to be hard
- For cryptographic applications, best to use *prime-order* groups
  - The *dlog* problem becomes easier if the order of the group has small prime factors
  - Prime-order groups have several nice features: e.g., every element except identity is a generator
- Two common choices of groups

Group selection: choice 1

- *Prime-order* subgroup of $\mathbb{Z}_p^*$, p prime
  - E.g., $p = tq + 1$ for q prime
  - Take the subgroup of t^{th} powers, i.e., $G = \{[x^t \bmod p] | x \in \mathbb{Z}_p^*\}$
    - This is a group
    - It has order $(p-1)/t = q$
    - Since q is prime, the group must be *cyclic*

Group selection: choice 1

- *Prime-order* subgroup of $\mathbb{Z}_p^*$, p prime
  - E.g., $p = tq + 1$ for q prime
  - Take the subgroup of t^{th} powers, i.e., $G = \{[x^t \bmod p] | x \in \mathbb{Z}_p^*\}$
    - This is a group
    - It has order $(p-1)/t = q$
    - Since q is prime, the group must be *cyclic*
  - Generalizations based on finite fields are also used

Group selection: choice 2

- *Prime-order* subgroup of an *elliptic curve* group
  - See book for details

Group selection: choice 2

- *Prime-order* subgroup of an *elliptic curve* group
  – See book for details

- We will describe algorithm in “abstract” groups
  – Can ignore details of the underlying group in the analysis
  – Can instantiate with any (appropriate) group for an implementation

Concrete parameters

- We have discussed two classes of cryptographic assumptions
  - *Factoring-based* (factoring, RSA assumptions)
  - *Dlog-based* (dlog, CDH, and DDH assumptions)

- All these problems are (believed to be) "hard", i.e., to have no polynomial-time algorithms
  - But how hard are they, concretely?

- The goal here is to give an idea as to how parameters are calculated, and what relevant parameters are

Security

- Recall: For symmetric-key algorithms
  - Block cipher with n-bit key $\approx$ security againt 2^n-time attacks
  - Hash functions with n-bit output $\approx$ security againt $2^{n/2}$-time attacks

Security

- Recall: For symmetric-key algorithms
  - Block cipher with n-bit key $\approx$ security againt 2^n-time attacks
  - Hash functions with n-bit output $\approx$ security againt $2^{n/2}$-time attacks

- Factoring of a modulus of size 2^n (i.e., length n) using exhaustive search takes $2^{n/2}$ time

Security

- Recall: For symmetric-key algorithms
  - Block cipher with n-bit key $\approx$ security againt 2^n-time attacks
  - Hash functions with n-bit output $\approx$ security againt $2^{n/2}$-time attacks

- Factoring of a modulus of size 2^n (i.e., length n) using exhaustive search takes $2^{n/2}$ time

- Computing discrete logarithms in a group of order 2^n takes 2^n time

Security

- Recall: For symmetric-key algorithms
  - Block cipher with n-bit key $\approx$ security againt 2^n-time attacks
  - Hash functions with n-bit output $\approx$ security againt $2^{n/2}$-time attacks
- Factoring of a modulus of size 2^n (i.e., length n) using exhaustive search takes $2^{n/2}$ time
- Computing discrete logarithms in a group of order 2^n takes 2^n time
  - Are these the best algorithms possible?

Algorithms for factoring

- There exist algorithms factoring an integer N that run in much less than $2^{\|N\|/2}$ time

- Best known algorithm (asymptotically): *general number field sieve*
  - Running time (heuristic): $2^{O(\|N\|^{1/3} \log^{2/3} \|N\|)}$
  - Makes a huge difference in practice
  - Exact constant term also important!

Algorithms for dlog

- Two classes of algorithms:
  - Ones that work for *arbitrary* ("generic") groups
  - Ones that target specific groups
    - Recall that in some groups the problem is not even hard

- Best "generic" algorithms:
  - Time $2^{n/2}$ in a group of order $\approx 2^n$
  - This is known to be optimal for generic algorithms

Algorithms for dlog

- Best known algorithm for (subgroups of) $\mathbb{Z}_p^*$: *number field sieve*
  - Running time (heuristic): $2^{O(\|p\|^{1/3} \log^{2/3} \|p\|)}$

- For (appropriately chosen) elliptic-curve groups, nothing better than generic algorithms is known!
  - This is why elliptic-curve groups can allow for more-efficient cryptography

Choosing parameters

- As recommended by NIST (112-bit security):
  - *Factoring*: 2048-bit modulus
  - *Dlog*: order-q subgroup of $\mathbb{Z}_p^*$: $\|q\| = 224$, $\|p\| = 2024$
  - *Dlog*, elliptic-curve group of order q: $\|q\| = 224$

Choosing parameters

- As recommended by NIST (112-bit security):
  - *Factoring*: 2048-bit modulus
  - *Dlog*: order-q subgroup of $\mathbb{Z}_p^*$: $\|q\| = 224$, $\|p\| = 2024$
  - *Dlog*, elliptic-curve group of order q: $\|q\| = 224$

- Much longer than for symmetric-key algorithms!
  - Explains in part why public-key crypto is less efficient than symmetric-key crypto

Private-key cryptography

- *Private-key cryptography* allows two users who *share a secret key* to establish a "secure channel"

Private-key cryptography

- *Private-key cryptography* allows two users who *share a secret key* to establish a “secure channel”
- The need to share a *secret key* has several drawbacks

Private-key cryptography

- *Private-key cryptography* allows two users who *share a secret key* to establish a "secure channel"
- The need to share a *secret key* has several drawbacks
- How do users share a key in the first place?
  - Need to share the key using a secure channel

Private-key cryptography

- *Private-key cryptography* allows two users who *share a secret key* to establish a “secure channel”
- The need to share a *secret key* has several drawbacks
- How do users share a key in the first place?
  - Need to share the key using a secure channel
- This problem can be solved in some settings
  - E.g., physical proximity, trusted courier, ...
  - **Note**: this does not make *private-key cryptography* useless!

Private-key cryptography

- *Private-key cryptography* allows two users who *share a secret key* to establish a "secure channel"
- The need to share a *secret key* has several drawbacks
- How do users share a key in the first place?
  - Need to share the key using a secure channel
- This problem can be solved in some settings
  - E.g., physical proximity, trusted courier, ...
  - **Note**: this does not make *private-key cryptography* useless!
- Can be difficult or expensive to solve in other settings

The key-management problem

- Imagine an organization with N employees, where each pair of employees might need to communicate securely

The key-management problem

- Imagine an organization with N employees, where each pair of employees might need to communicate securely

- Solution using *private-key cryptography*
  – Each user shares a key with all other users
  $\Rightarrow$ Each user must store/manage $N-1$ secret keys!
  $\Rightarrow$ $O(N^2)$ keys overall!

Lack of support for “open systems”

- Say two users *who have no prior relationship* want to communicate securely
  - When would they ever have shared a key?

Lack of support for "open systems"

- Say two users *who have no prior relationship* want to communicate securely
  - When would they ever have shared a key?

- This happens all the time!
  - Customer sending credit-card data to merchant
  - Contacting a friend-of-a-friend on social media
  - Emailing a colleague
  - ...

"Classical" cryptography offers no solution

644 IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. IT-22, NO. 6, NOVEMBER 1976

New Directions in Cryptography

Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER, IEEE

***Abstract*—Two kinds of contemporary developments in cryptography are examined. Widening applications of teleprocessing have given rise to a need for new types of cryptographic systems, which minimize the need for secure key distribution channels and supply the equivalent of a written signature. This paper suggests ways to solve these currently open problems. It also discusses how the theories of communication and computation are beginning to provide the tools to solve cryptographic problems of long standing.**

I. Introduction

We stand today on the brink of a revolution in cryptography. The development of cheap digital hardware has freed it from the design limitations of mechanical computing and brought the cost of high grade cryptographic devices down to where they can be used in such commercial applications as remote cash dispensers and computer terminals. In turn, such applications create a need for new types of cryptographic systems which minimize the necessity of secure key distribution channels and supply the equivalent of a written signature. At the same time, theoretical developments in information theory and computer science show promise of providing provably secure cryptosystems, changing this ancient art into a science.

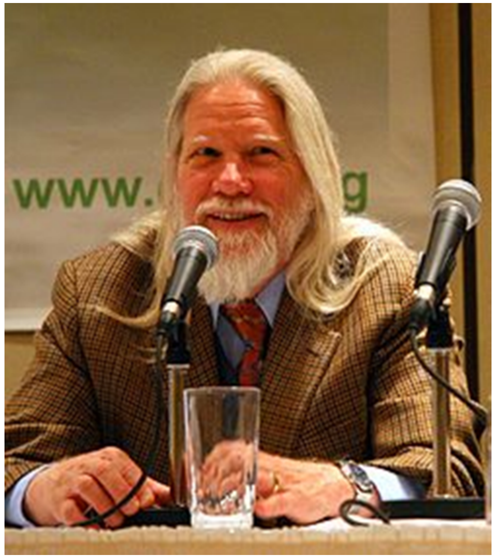
The best known cryptographic problem is that of privacy: preventing the unauthorized extraction of information from communications over an insecure channel. In order to use cryptography to insure privacy, however, it is currently necessary for the communicating parties to share a key which is known to no one else. This is done by sending the key in advance over some secure channel such as private courier or registered mail. A private conversation between two people with no prior acquaintance is a common occurrence in business, however, and it is unrealistic to expect initial business contacts to be postponed long enough for keys to be transmitted by some physical means. The cost and delay imposed by this key distribution problem is a major barrier to the transfer of business communications to large teleprocessing networks.

Section III proposes two approaches to transmitting keying information over public (i.e., insecure) channels without compromising the security of the system. In a *public key cryptosystem* enciphering and deciphering are governed by distinct keys, E and D, such that computing D from E is computationally infeasible (e.g., requiring 10^{100} instructions). The enciphering key E can thus be publicly disclosed without compromising the deciphering key D. Each user of the network can, therefore, place his enciphering key in a public directory. This enables any user of the system to send a message to any other user enci-

Cryptography History

- History (from 1976)
  - ◇ W. Diffie, M. Hellman, "New direction in cryptography", *IEEE Transactions on Information Theory, vol. 22, pp. 644-654, 1976.*

"We stand today on the brink of a revolution in cryptography."



Bailey W. Diffie

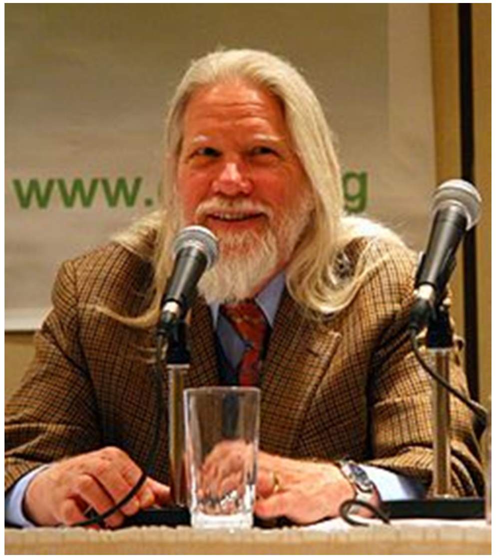
Martin E. Hellman

Cryptography History

- History (from 1976)
  - ◇ W. Diffie, M. Hellman, "New direction in cryptography", *IEEE Transactions on Information Theory, vol. 22, pp. 644-654, 1976.*

"We stand today on the brink of a revolution in cryptography."

2015 **Turing Award**



Bailey W. Diffie

Martin E. Hellman

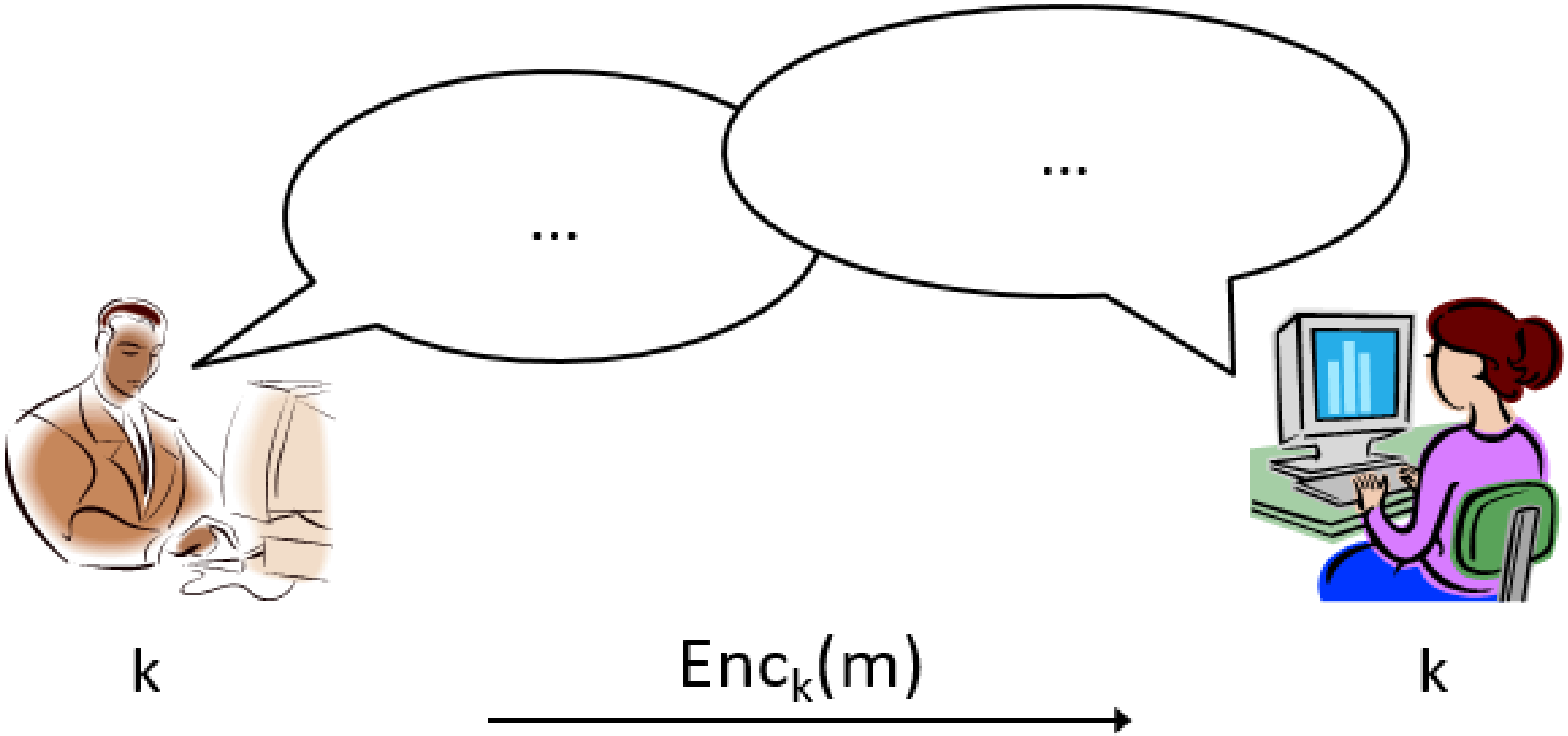
2015	Martin E. Hellman Whitfield Diffie	For fundamental contributions to **modern cryptography**. Diffie and Hellman's groundbreaking 1976 paper, "New Directions in Cryptography,"[39] introduced the ideas of public-key cryptography and digital signatures, which are the foundation for most regularly-used security protocols on the internet today.[40]

New directions

- Main ideas:
  - Some problems exhibit *asymmetry* - easy to compute, but hard to invert (*factoring*, *RSA*, *group exponentiation*, ...)
  - Use this *asymmetry* to enable two parties to agree on a shared secret key via public discussion
    - *Key exchange*

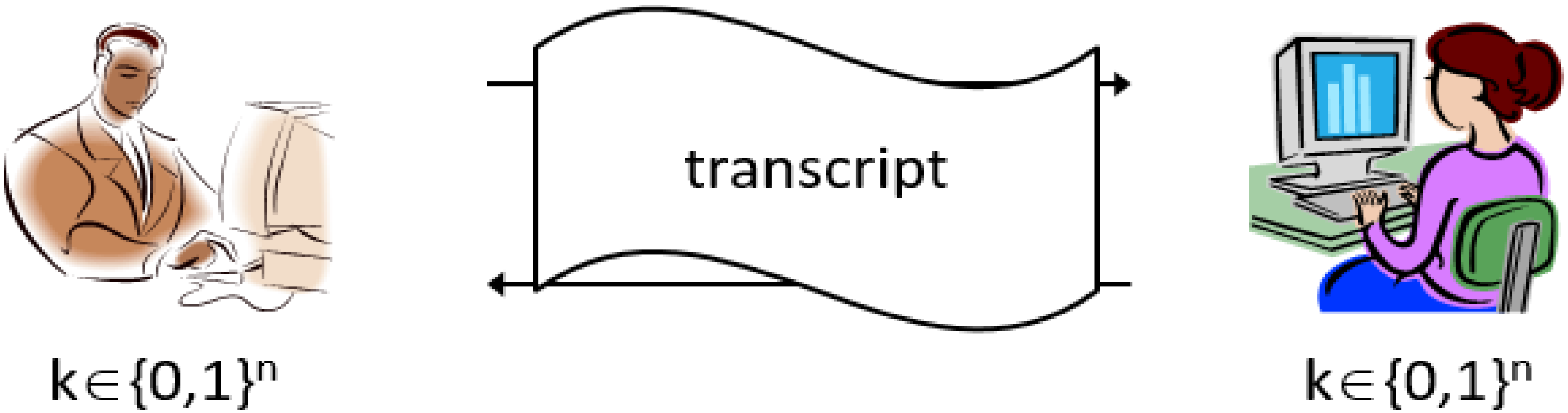
Key exchange

- *Secure* against an eavesdropper who sees everything!



More formally ...

- *Security goal*: even after observing the transcript, the shared key k should be *indistinguishable* from a uniform key



Formally

- Fix a key-exchange protocol Π and an attacker (passive eavesdropper) A

Formally

- Fix a key-exchange protocol Π and an attacker (passive eavesdropper) A
- Define the following experiment $KE_{A,\Pi}(n)$:
  - Honest parties run Π using security parameter n, resulting in a transcript *trans* and (shared) key k
  - Choose uniform bit b. If $b = 0$, then set $k' = k$; if $b = 1$, then choose uniform $k' \in \{0, 1\}^n$
  - Give *trans* and k' to A, which outputs a bit b'
  - Exp't evaluates to 1 (A *succeeds*) if $b = b'$

Formally

- Fix a key-exchange protocol Π and an attacker (passive eavesdropper) A
- Define the following experiment $KE_{A,\Pi}(n)$:
  - Honest parties run Π using security parameter n, resulting in a transcript *trans* and (shared) key k
  - Choose uniform bit b. If $b = 0$, then set $k' = k$; if $b = 1$, then choose uniform $k' \in \{0,1\}^n$
  - Give *trans* and k' to A, which outputs a bit b'
  - Exp't evaluates to 1 (A *succeeds*) if $b = b'$
- **Definition 12.1** Key-exchange protocol Π is *secure* (against passive eavesdropping) if for *all* PPT A it holds that

$$\Pr[KE_{A,\Pi}(n) = 1] \leq 1/2 + negl(n)$$

Notes

- Being unable to compute the key given the transcript is not a strong enough guarantee

- Indistinguishability of the shared key from uniform is a much stronger guarantee
  - And is necessary if the shared key will subsequently be used for *private-key crypto*!

Diffie-Hellman key exchange

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

$\xleftarrow{h_2}$

$k_1 = (h_2)^x$

$k_2 = (h_1)^y$

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

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

In practice

$G,\ q,\ g$

$\xrightarrow{h_1}$

$\xleftarrow{h_2}$

$k_1 = (h_2)^x$

$k_2 = (h_1)^y$

$x \leftarrow \mathbb{Z}_q$
$h_1 = g^x$

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

Recall: Diffie-Hellman assumptions

- *Computational* Diffie-Hellman (CDH) problem:
  - Given g, h_1, h_2, compute $DH_g(h_1, h_2)$
- *Decisional* Diffie-Hellman (DDH) problem:
  - Given g, h_1, h_2, distinguish $DH_g(h_1, h_2)$ from a uniform element of G

Security

- Eavesdropper sees G, q, g, g^x, g^y
- Shared key k is g^{xy}

Security

- Eavesdropper sees G, q, g, g^x, g^y
- Shared key k is g^{xy}
- Computing k from the transcript is exactly the *computational* Diffie-Hellman problem

Security

- Eavesdropper sees G, q, g, g^x, g^y
- Shared key k is g^{xy}
- Computing k from the transcript is exactly the *computational* Diffie-Hellman problem
- Distinguishing k from uniform group element is exactly the *decisional* Diffie-Hellman problem
  $\Rightarrow$ If the DDH problem is hard relative to $\mathcal{G}$, this is a *secure* key-exchange protocol!

A subtlety

- We wanted our key-exchange protocol to give us a uniform(-looking) key $k \in \{0, 1\}^n$

A subtlety

- We wanted our key-exchange protocol to give us a uniform(-looking) key $k \in \{0, 1\}^n$

- Instead we have a uniform(-looking) group element $k \in G$
  - Not clear how to use this as, e.g., an AES key

A subtlety

- We wanted our key-exchange protocol to give us a uniform(-looking) key $k \in \{0,1\}^n$
- Instead we have a uniform(-looking) group element $k \in G$
  - Not clear how to use this as, e.g., an AES key
- Solution: *key derivation*
  - Set $k' = H(k)$ for suitable hash function H

Modern key-exchange protocols

- Security against passive eavesdroppers is insufficient

- Want *authenticated* key exchange
  - This requires some form of setup in advance

- Modern key-exchange protocols provide this
  - We will return to this later

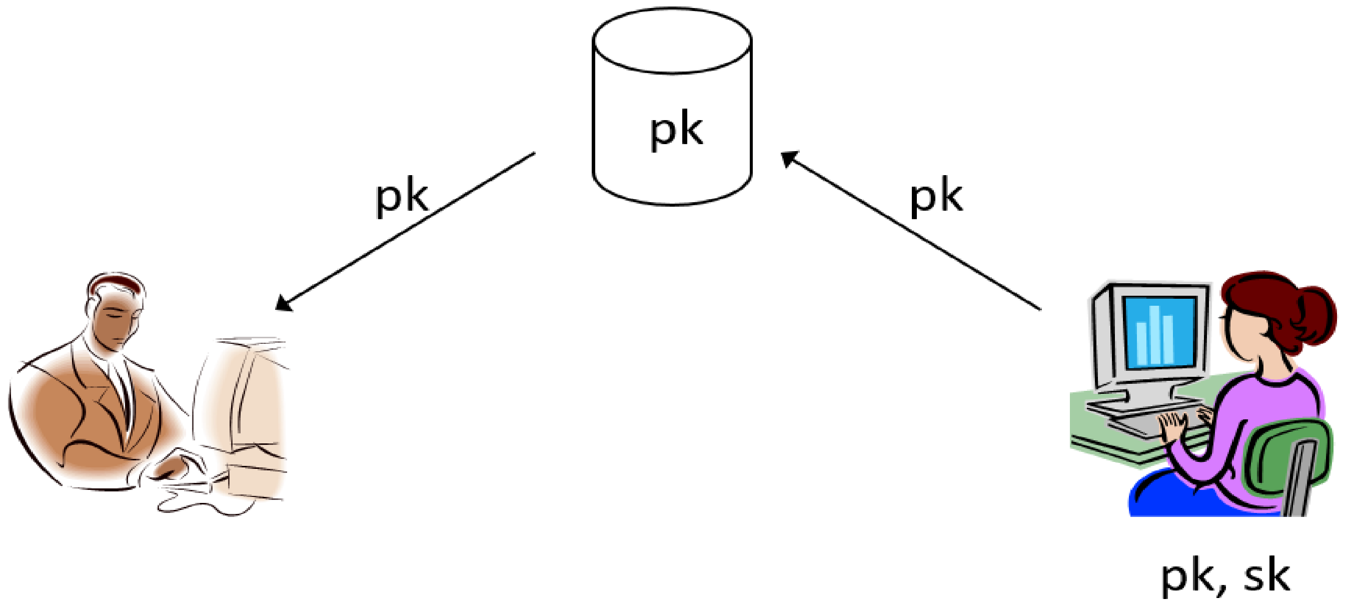
The public-key setting

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

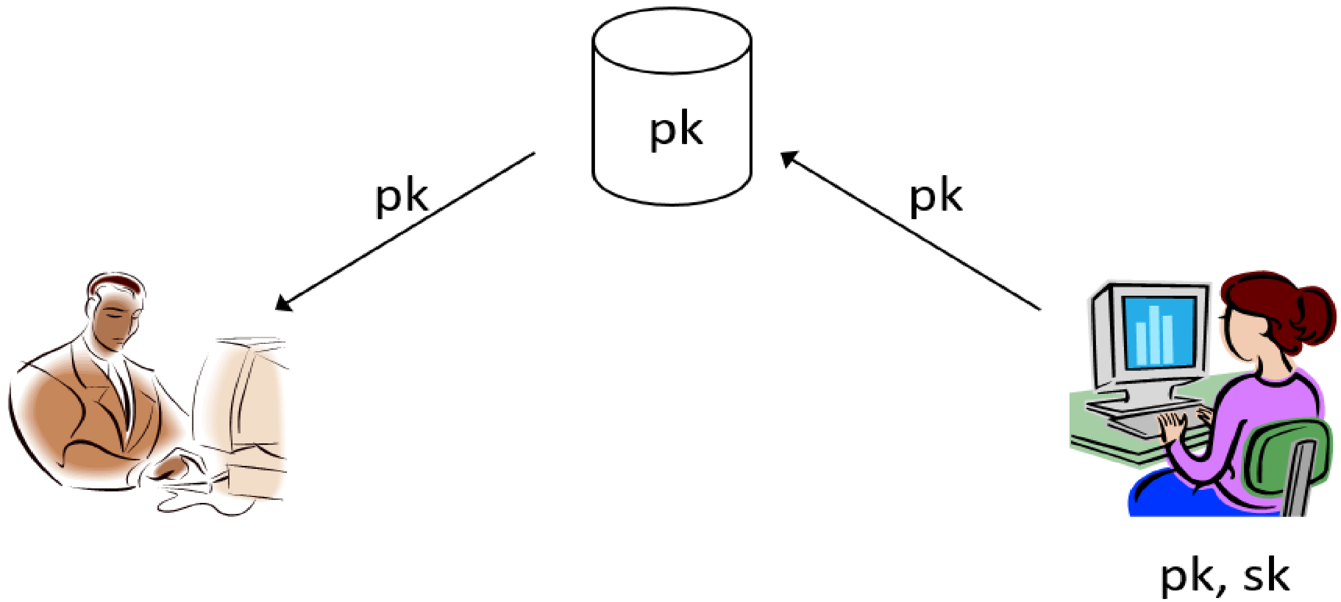
- Private key used by the party who generated it; public key used by everyone else
  - Also called *asymmetric cryptography*

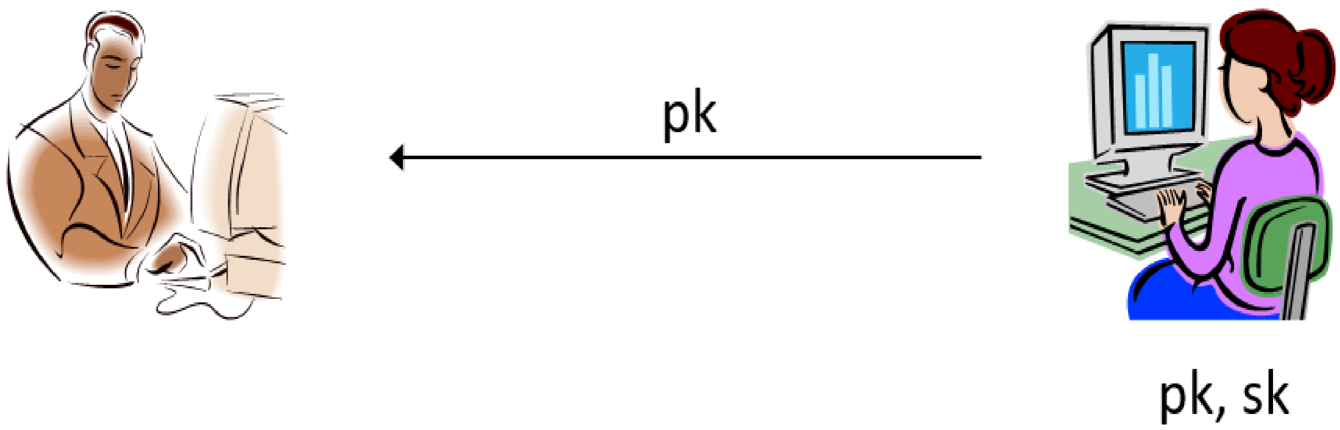
- Security must hold even if the attacker knows *pk*

Public-key distribution I, II



Public-key distribution I, II





Public-key distribution

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

Public-key distribution

- Previous figures (implicitly) assume parties are able to obtain correct cipies of each others' public keys
  - I.e., the attacker is *passive* during key distribution
- We will revisit this assumption later

Public-key distribution

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

- We will revisit this assumption later

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

Addressing drawbacks of private-key crypto

- Key distribution
  - Public keys can be distributed over *public* (but *authenticated*) channels!

Addressing drawbacks of private-key crypto

- Key distribution
  - Public keys can be distributed over *public* (but *authenticated*) channels!
- Key management in large systems 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 directory

Addressing drawbacks of private-key crypto

- Key distribution
  - Public keys can be distributed over *public* (but *authenticated*) channels!
- Key management in large systems 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 directory
- Applicability in "open systems"
  - Even parties who have no prior relationship can find each others' public keys and use them

Why study private-key crypto?

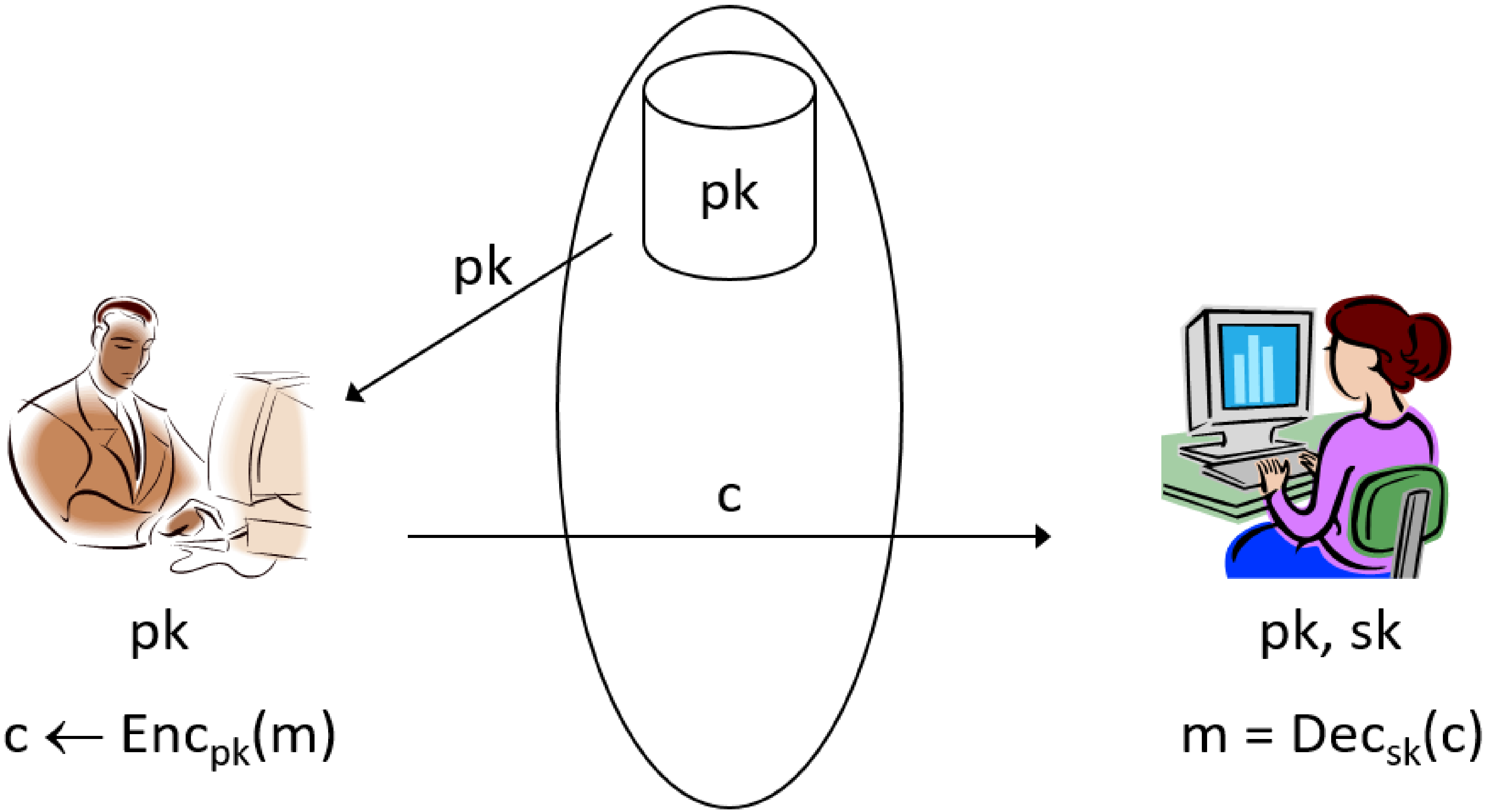
- Private-key cryptography is more suitable for certain applications
  - E.g., disk encryption

Why study private-key crypto?

- Private-key cryptography is more suitable for certain applications
  - E.g., disk encryption

- Public-key crypto is roughly $2 - 3$ orders of magnitude *slower* than private-key crypto
  - If private-key crypto is an option, use it!
  - Private-key crypto is used for *efficiency* even in the public-key setting

Public-key encryption



- **Theorem 12.2** A *public-key encryption* scheme is composed of three PPT algorithms:
  - *Gen*: *key-generation algorithm* that on input 1^n 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 $\perp$

Public-key encryption

- **Theorem 12.2** A *public-key encryption* scheme is composed of three PPT algorithms:
  - *Gen*: *key-generation algorithm* that on input 1^n 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,

$$Dec_{sk}(Enc_{pk}(m)) = m$$

CPA-security

- Fix a public-key encryption scheme Π and an adversary A
- Define experiment $PubK\text{-}CPA_{A,\Pi}(n)$:
  - Run $Gen(1^n)$ to get keys pk, sk
  - Give pk to A, who outputs m_0, m_1 of same length
  - Choose uniform $b \in \{0,1\}$ and compute the ciphertext $c \leftarrow Enc_{pk}(m_b)$; give c to A
  - A outputs a guess b', and the experiment evaluates to 1 if $b' = b$

CPA-security

- Fix a public-key encryption scheme Π and an adversary A
- Define experiment $PubK\text{-}CPA_{A,\Pi}(n)$:
  – Run $Gen(1^n)$ to get keys pk, sk
  – Give pk to A, who outputs m_0, m_1 of same length
  – Choose uniform $b \in \{0,1\}$ and compute the ciphertext $c \leftarrow Enc_{pk}(m_b)$; give c to A
  – A outputs a guess b', and the experiment evaluates to 1 if $b' = b$
- **Theorem 12.3** Public-key encryption scheme Π is *CPA-secure* if for all PPT adversaries A:

$$\Pr[PubK\text{-}CPA_{A,\Pi}(n) = 1] \le 1/2 + negl(n)$$

Notes on the definition

- No encryption oracle?!

Notes on the definition

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

Notes on the definition

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

$\Rightarrow$ No *perfectly secret* public-key encryption

$\Rightarrow$ No *deterministic* public-key encryption can be CPA-secure

$\Rightarrow$ CPA-security implies security for encryption multiple messages as in the private-key case

Perfectly secret public-key encryption

- **Definition 12.4** A public-key encryption scheme is *perfectly secret* if for all public keys pk, all messages m_0, m_1, all ciphertexts c, and all algorithms A, we have:
  $$\Pr[A(pk, c) = 0 | c \leftarrow Enc_{pk}(m_0)] = \Pr[A(pk, c) = 0 | c \leftarrow Enc_{pk}(m_1)]$$

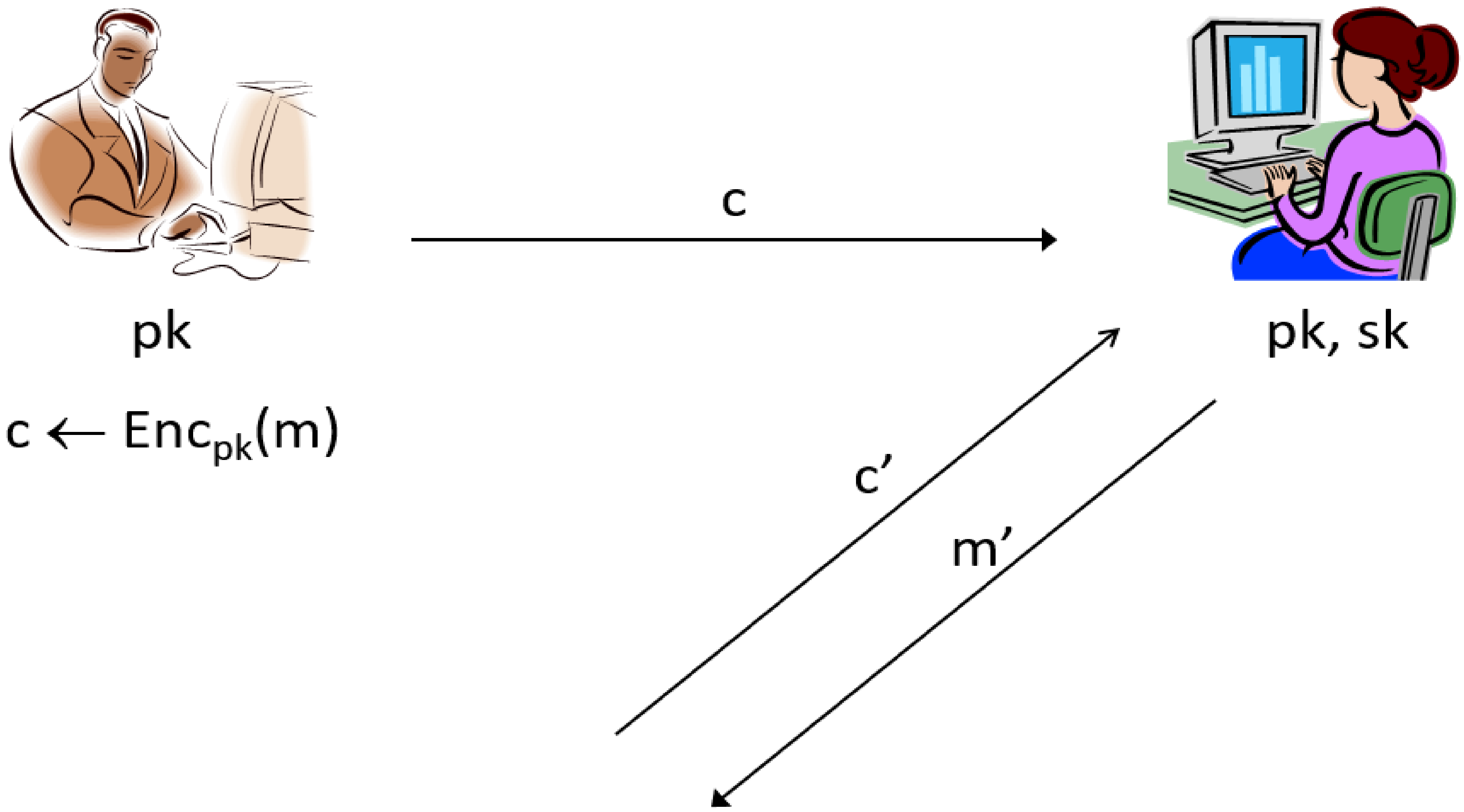
Perfectly secret public-key encryption

- **Definition 12.4** A public-key encryption scheme is *perfectly secret* if for all public keys pk, all messages m_0, m_1, all ciphertexts c, and all algorithms A, we have:
  $\Pr[A(pk,c)=0|c \leftarrow Enc_{pk}(m_0)] = \Pr[A(pk,c)=0|c \leftarrow Enc_{pk}(m_1)]$

  Theorem 12.5 No public-key encryption scheme is *prefectly secret*.

  Proof.

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

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

Recall: plain RSA

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

Recall: plain RSA

- Choose random, equal-length primes p, q
- 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 $x^d \bmod N$

$$(x^d)^e = x^{de} = x \bmod N$$

Recall: plain RSA

- Choose random, equal-length primes p, q
- 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 $x^d \bmod N$
$$(x^d)^e = x^{de} = x \bmod N$$

- *RSA assumption*: given N, e only, it is *hard* to compute the e^{th} root of a uniform $c \in \mathbb{Z}_N^*$

“Plain” RSA encryption

N, e →

← c

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

$pk = (N, e)$

$sk = d$

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

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

Security

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

Security

- The scheme is *deterministic*
  – *Cannot* be *CPA-secure*!

- *RSA assumption* only refers to hardness of computing the e^{th} roots of *uniform* c
  – c is not uniform unless m is
  – Easy to recover “small” m from c

Security

- The scheme is *deterministic*
  – *Cannot* be *CPA-secure*!

- *RSA assumption* only refers to hardness of computing the e^{th} roots of *uniform* c
  – c is not uniform unless m is
  – Easy to recover "small" m from c

- *RSA assumption* only refers to hardness of computing the e^{th} roots *in its entirety*
  – Partial information about the e^{th} root may be leaked

Security

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

- *RSA assumption* only refers to hardness of computing the e^{th} roots of *uniform* c
  - c is not uniform unless m is
  - Easy to recover "small" m from c

- *RSA assumption* only refers to hardness of computing the e^{th} roots *in its entirety*
  - Partial information about the e^{th} root may be leaked

- Plain RSA should *never* be used!

Next Lecture

- El Gamal encryption, digital signature ...