Security: An Introduction

Operating Systems

Baochun Li
Department of Electrical and Computer Engineering
University of Toronto

What is the overall objective when designing secure protocols?

Restrict access to information and resources to **just those principals** that are **authorized** to have access

Cryptographic algorithms

- Provides the basis and foundation for all security protocols
- For the purpose of this lecture, we need to understand **two things** about these algorithms

Cryptographic algorithms: our goals

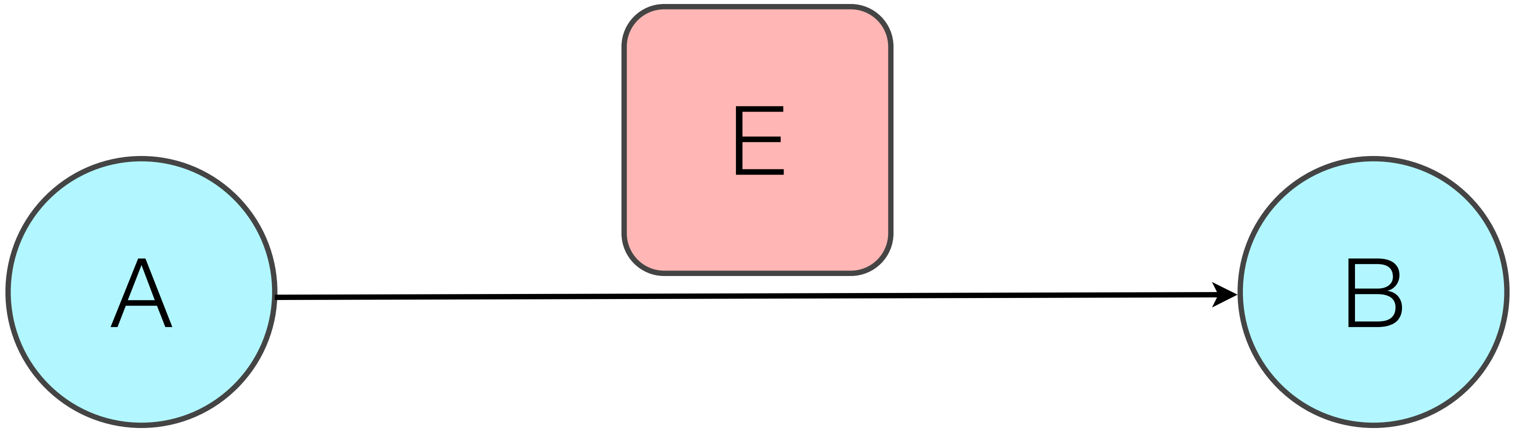
- Understand the basic ideas in the design of cryptographic algorithms
- Treat these cryptographic algorithms as "**blackboxes**," and understand how they are used to build secure protocols

What is a secure channel?

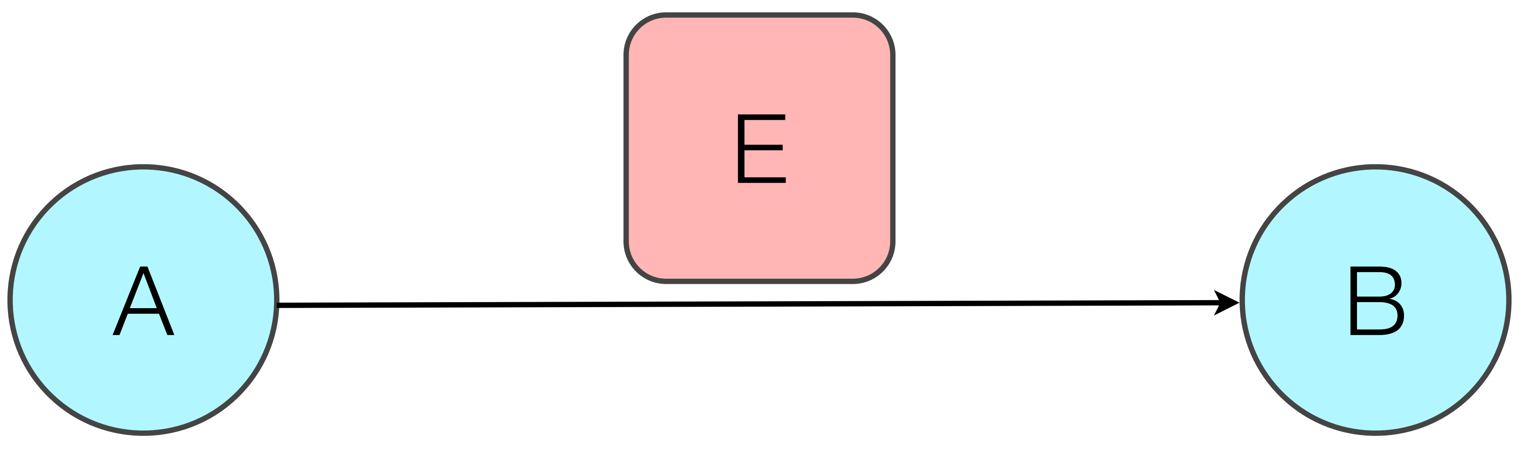
What can an **adversary** do?

The **adversary** (or the “enemy”) may gain access to the communication channel between authorized principals.

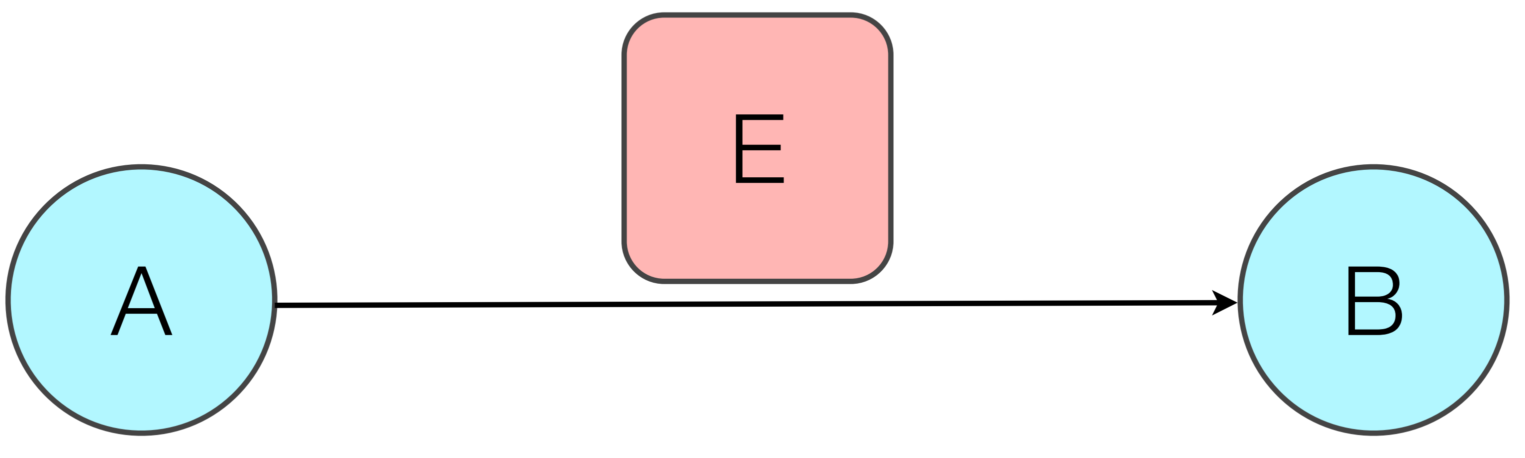
What can the enemy do?



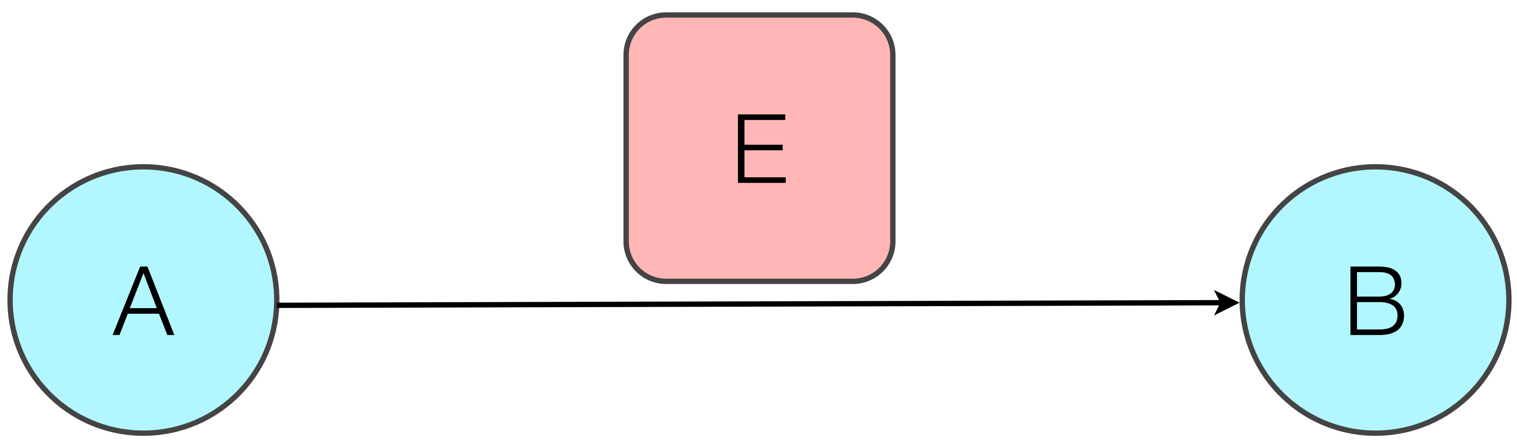
Read and copy messages
(**eavesdropping**)



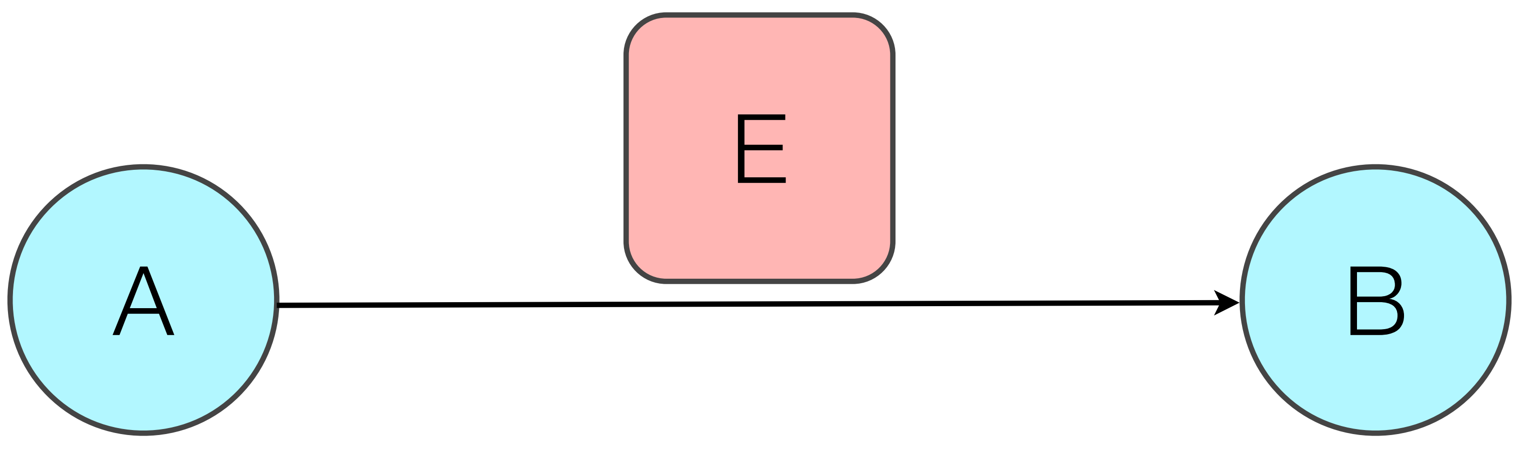
Inject arbitrary messages
(**tampering** and **replaying**)



send or receive messages using the identity of another principal (**masquerading**, **man-in-the-middle attacks**)



Can it prevent messages from getting through?



Denial of Service attacks

(flooding a channel in order to
deny access of resources to others,
challenges **availability** of the resources)

Even though they may be able to, it is not very interesting to assume this, as there are **no countermeasures** (with basic cryptographic tools)!

Objective: to implement a **secure channel** between two principals, **A** and **B**

Requirements of a secure channel

- We need to be able to pass data from A to B subject to a selection of the following constraints:
  - **Secrecy**: the data can not be read by unintended recipients
  - **Integrity**: the data can not be altered without detection
  - **Authentication**: the data is attributed to the correct originator

To achieve **secrecy**, we wish to design a tool to —

encode a message to hide its contents from access without a secret key

Secret key algorithms

- **Secret key** algorithms are also called **symmetric** cryptographic algorithms
  - the sender and recipient share the knowledge of a **secret key, K**

Using secret key algorithms

Required property #1: secret key algorithms — **Correctness**

- If E(K, M) and D(K, M) are the encryption and decryption algorithms, respectively, then:

Required property #2: secret key algorithms — **Security**

- In the absence of knowledge of K, it must be **very awkward** (computationally infeasible) to recover M from E(K, M).

Modern symmetric cryptographic algorithms (block **ciphers**)

- Outdated: Data Encryption Standard (DES) — 64-bit blocks, 56-bit keys
- Since 1997: Advanced Encryption Standard (AES) — 128-bit blocks, 128, 192, or 256 bit keys

How do we use symmetric cryptographic algorithms to establish a secure channel?

Review of our objective: to implement a secure channel between two principals, **A** and **B**

Establishing a secure channel using symmetric cryptographic algorithms

One idea —

If we assume **A** and **B** share a secret key **K**, we can use symmetric cryptographic algorithms to encrypt and decrypt the message. **A** encrypts the message **M** with **E(K, M)**, send it to **B**, and **B** computes: **D(K, E(K, M)) = M**.

Does it satisfy the requirements of a secure channel?

Does it satisfy the requirements of a secure channel?

- **Secrecy**: Yes, due to the properties of the symmetric cryptographic algorithm (our “blackbox”).
- **Authentication**: Yes, **if** we may assume that only **A** and **B** shares knowledge of the secret key **K**.

How about **integrity**?

It seems that the shared secret key is sufficient to establish a secure channel, except that the assumption that **only** the two principals A and B share the secret key is **strong**, as it may need a separate protocol to achieve this.

How do we securely share keys between A and B?

Establishing a secure channel using asymmetric crypto algorithms

Establishing a secure channel using asymmetric crypto algorithms

If we let B keep $K_{B,\mathbf{priv}}$ strictly to itself, and publish $K_{B,\mathbf{pub}}$ to the entire system, then A can send $E(K_{B,\mathbf{pub}}, M)$ to B, and B can perform

$$D(K_{B,\mathbf{priv}}, E(K_{B,\mathbf{pub}}, M))$$

and then obtain **M**.

Asymmetric cryptographic algorithms

- **Public key** algorithms (asymmetric cryptographic algorithms)
  - The sender, Alice, uses a **public key** of the recipient (published by the recipient to everyone in the system) to encrypt the message
  - The recipient, Bob, then uses his own **private key** to decrypt the message
- If $E(\cdot)$ and $D(\cdot)$ are used to denote the encryption and decryption algorithms using public key algorithms, then we have

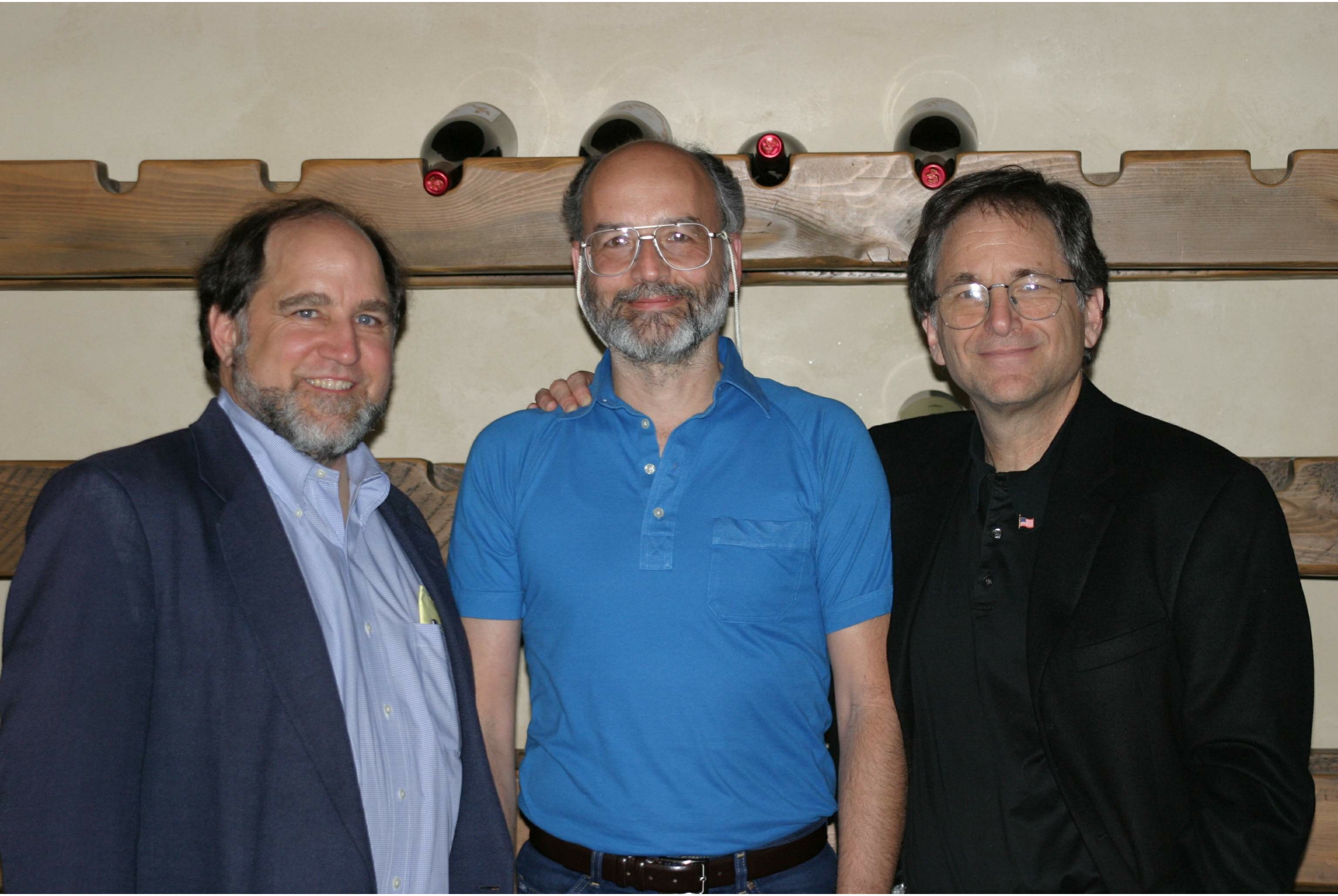
$$D(K_{\text{priv}}, E(K_{\text{pub}}, M)) = M \text{ and}$$

$$D(K_{\text{pub}}, E(K_{\text{priv}}, M)) = M \text{ as well.}$$

Required properties: public key algorithms

- Knowing the public key, it is **very awkward** (computationally infeasible) to compute the private key
- In the absence of knowledge of $K_{\mathbf{priv}}$, it must be very awkward (computationally infeasible) to recover M from $E(K_{\mathbf{pub}}, M)$.
- Given M and $E(K_{\mathbf{pub}}, M)$, it should be very awkward to recover the private key $K_{\mathbf{priv}}$.

The world’s first and most widely used asymmetric cryptographic algorithm:
RSA (1977)
(Rivest, Shamir, Adleman)



Does it satisfy the requirements of a secure channel?

Secrecy?

Does it satisfy the requirements of a secure channel?

Secrecy: Yes, due to the properties of the asymmetric cryptographic algorithm (our “blackbox”).

Does it satisfy the requirements of a secure channel?

Integrity?

Does it satisfy the requirements of a secure channel?

Integrity: To ensure integrity, we may add a checksum C(M) to the message M being sent by A, we can let A send $E(K_{B,pub}, \{M, C(M)\})$, and after B decrypts the message, it obtains {M, C(M)}, then computes C(M) using M (assuming the checksum function is known to both principals), and finally compare the computed result with the received C(M). If the two are identical, the message M is received intact.

How about authentication?

No. Anyone can send a message M to B, knowing B’s public key.

How do we change the protocol to satisfy the requirement of authentication?

Digital signatures

Digital signatures

A encrypts the message M by using its own private key $K_{A,\textbf{priv}}$ — $E(K_{A,\textbf{priv}}, M)$ — called a digital signature, or A *signs* M. Given a signed message (and presumably a hint that it may have been from A), an attempt to decrypt it using $K_{A,\textbf{pub}}$ will yield M, and thus the recipient, B, can infer that A carried out the encryption (the **signature**) originally.

The problem:

digital signatures do not satisfy the **secrecy** requirement, in that any principal can decrypt the message and obtain **M**.

How do we solve this problem?

Combined with the use of checksum C(M), this protocol satisfies all the requirements for a secure channel.

But there are **two** remaining problems

Remaining problem #1: The protocol requires applying asymmetric cryptographic algorithms on the entire message, which is not computationally efficient.

To solve this problem: use the idea of **message digests**

Message digests

- It turns out that it is possible to devise **functions** from the original messages to quantities of a fixed size, known as **message digests** or **secure hashes**

- Good examples include **SHA** (160 bits) and **MD5** (128 bits)

Properties of message digests

- The properties of a message digest function, H(M), are as follows:
- Given M, it should be easy to compute H(M);
- Given H(M), it should be hard to obtain M;
- **Collision resistance**: Given M and H(M), it should be exceedingly awkward (computationally infeasible) to find another message M', such that H(M') = H(M).

With **H(M)**, the sender can then sign the digest, not the entire message

Solution with secure hash functions

With the use of **H(M)** as the checksum, this protocol satisfies all requirements for a secure channel.

Sharing the Secret Key with Public Key Algorithms

Sharing the Secret Key with Public Key Algorithms

- An natural idea at this point is to send the secret key K, generated by A, to B, using the pubic key of B, $K_{B,pub}$
- Once B obtains K, it should be able to use K to establish a secure channel for later use
- As K is usually short, it is computationally efficient
- In general, SSL (TLS) and SHTTP use this idea

Remaining problem #2:
How does A know that $K_{B,pub}$ is really the public key of B?

A possible solution

A possible solution

- We can try to solve this problem by asking A to access a *trusted key* distribution entity S to obtain the public key of B, $K_{B,pub}$
- How does A know that $K_{B,pub}$ originates from S? We can solve this problem by asking S to sign using the private key of itself: $K_{S,priv}$
  - S will send $E(K_{S,\mathbf{priv}}, \{K_{B,\mathbf{pub}}\})$ to A, which is called a **certificate**.
  - When A decrypts the received message using the public key of S — $K_{S,pub}$, it authenticates it

But how does A know that $K_{S,pub}$ is really the public key of S?

How does A know it’s from S?

Apparently, A can access another trusted entity for a certificate to certify **S**, which creates a **chain of certificates**. This chain, however, has to **stop** somewhere.

The **chain of certificates stops** in your operating system.

Root certificate authorities

- We trust our operating system, which stores a list of public keys of well-known authorities, such as **verisign.com,** in the form of certificates
  - These authorities are called **root certificate authorities**

Hierarchical certificate system

- **Root**, **regional** and **local** certificate authorities (CA)
- Called the **X.509** Digital Certificate Framework: a standard

Security: an introduction

- The adversary
- Cryptographic algorithms
- The secure channel
- Establishing the **secure channel**
  - Using secret key algorithms
  - Using public key algorithms
  - Digital signatures
  - Message digests (secure hash functions)
  - Sharing the secret key with public key algorithms