Introduction to Information Security
14-741 Fall 2025

Unit 2: Lecture 2: Asymmetric Key Cryptography

Hanan Hibshi hhibshi@andrew

Acknowledgment: includes slides contributed by many people including N. Christian and L. Jia

Carnegie Mellon University
Information Networking Institute

This lecture's agenda

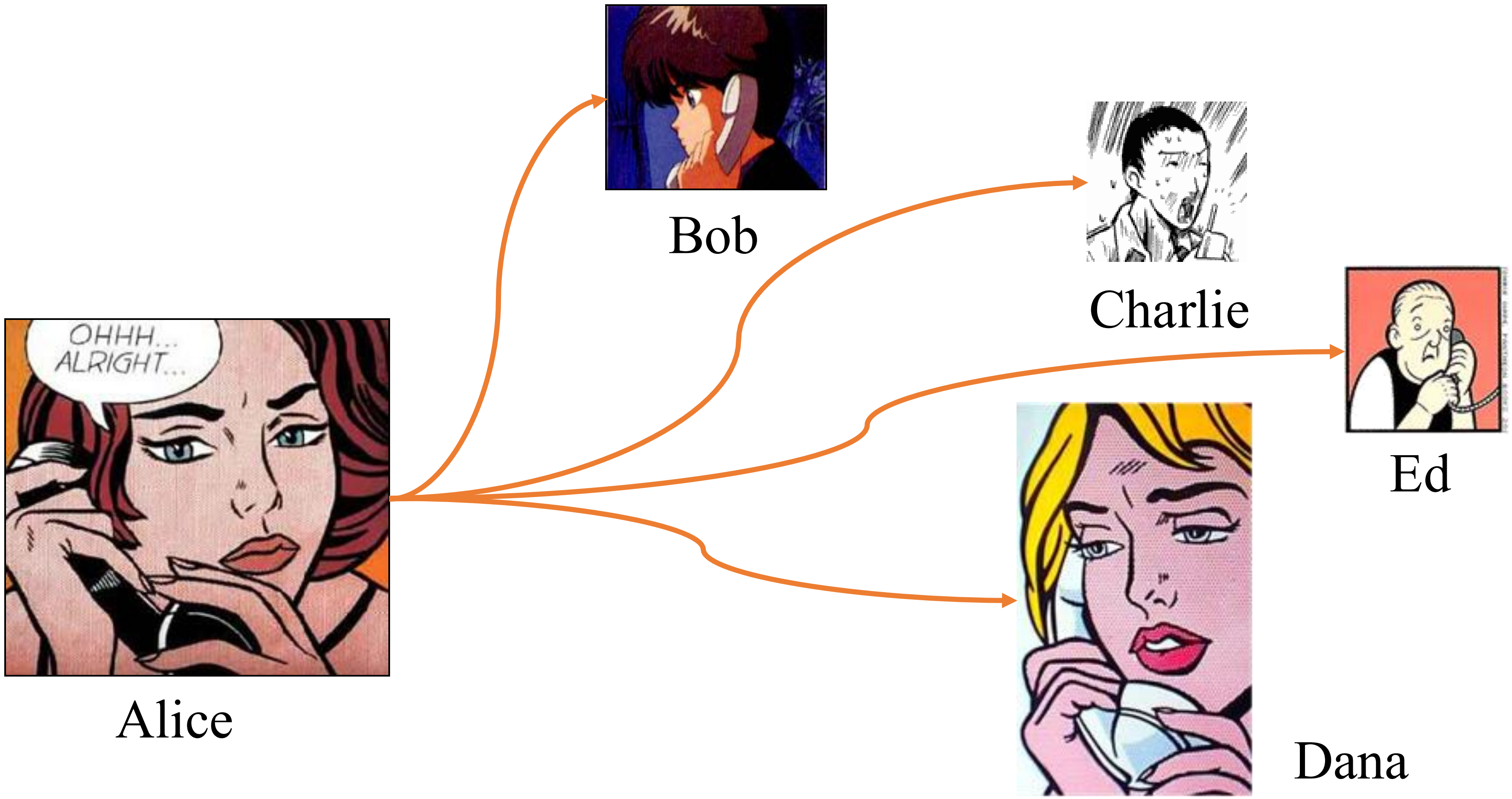
- Outline
  - Public key cryptography
    - Diffie-Hellman
    - RSA
    - Digital signatures
- Objectives
  - Complete our overview of basic cryptographic techniques

Difficulties w/ symmetric keys

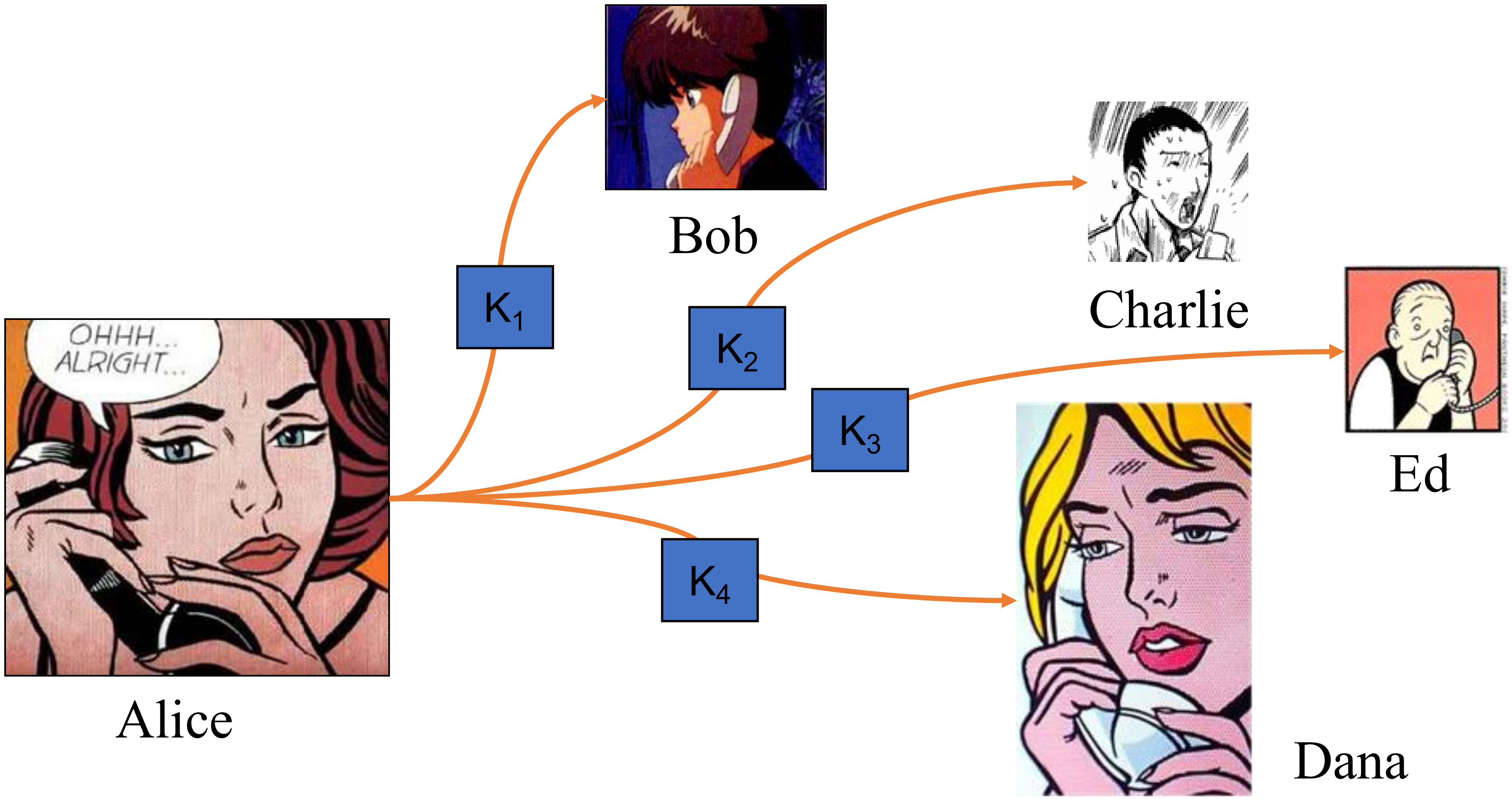
- Suppose Alice wants to talk to Bob but doesn't want Eve to be able to listen
- Symmetric crypto
  - E.g., DES, AES, …

How can Alice and Bob share the secret key?

More difficulties w/ sym. keys



More difficulties w/ sym. keys



More difficulties w/ sym. keys

For n participants in the system, you need $O(n^2)$ keys if everyone can talk to everyone else!

Common Misconception

- Yes, $O(n^2)$ keys are a lot of keys
  - 8 billion people (+ almost countless IoT devices)
- No, the existence of a large number of keys is <u>not</u> the problem
  - Even if O(n), 8 billion keys are already a large number of keys
  - Plus, we happily establish a new "session key" all the time (to be discussed later)
- **How to distribute the keys ("key distribution")** <u>is</u> the problem to solve
- But it is also true that some Asymmetric Key Crypto schemes generate O(n) keys for n entities, hence the misconception

Outline

- Diffie-Hellman key exchange
- Asymmetric (public) key crypto
  - RSA
  - Digital signature schemes

Diffie-Hellman-Merkle key exchange

- Attempts to solve the problem of secret key distribution by having people compute the secret key independently, using publicly available information and personal secrets
- Proposed by Diffie & Hellman in 1976
  - Different ways of doing crypto than had been proposed in the previous 4,000+ years
  - Foundation for public key crypto (RSA, ElGamal, etc.)
  - Diffie and Hellman won Turing Award for this in 2015
- Side notes:
  - Merkle credited by Hellman as a strong inspiration for the design
  - Similar method developed in the 1960s at GCHQ (UK) by James Ellis, but classified…

Merkle, Hellman and Diffie (1977)

Diffie-Hellman-Merkle key exchange

Alice

1. Agree g (base) and p (prime)
2. Make information public (doesn't matter who gets it)

Bob

Diffie-Hellman-Merkle key exchange

3A. Pick secret value *A*

Alice

1. Agree *g* (base) and *p* (prime)
2. Make information public (doesn't matter who gets it)

3B. Pick secret value *B*

Bob

Diffie-Hellman-Merkle key exchange

4A. Send g^A mod p

Insecure physical channel

3A. Pick secret value A

Alice

1. Agree g (base) and p (prime)
2. Make information public (doesn't matter who gets it)

3B. Pick secret value B

Bob

4B. Send g^B mod p

Diffie-Hellman-Merkle key exchange

4A. Send g^A mod p

Insecure physical channel

3A. Pick secret value A

Alice

5A. Compute
$(g^B \bmod p)^A \bmod p = g^{AB} \bmod p$

3B. Pick secret value B

Bob

5B. Compute
$(g^A \bmod p)^B \bmod p = g^{AB} \bmod p$

4B. Send g^B mod p

Diffie-Hellman-Merkle key exchange

4A. Send g^A mod p

Insecure physical channel

3A. Pick secret value A

Alice

Alice and Bob independently computed **the same secret number** by only sharing g and p

$= g^{AB}$ mod p

3B. Pick secret value B

Bob

5B. Compute
$(g^A \bmod p)^B \bmod p = g^{AB} \bmod p$

4B. Send g^B mod p

Why Diffie-Hellman works

Eve

- Based on a hard *discrete logarithm* problem
  - Given two large prime numbers *g* and *p*, *and* $x = g^A \bmod p$, computing *A* is very hard
  - The best-known algorithm for finding *A* is **exponential** in time, (i.e., roughly **equivalent** to a brute-force attack)
- Eve (eavesdropper)
  - Can easily get $g^A \bmod p$, $g^B \bmod p$
  - But can't compute (easily) $g^{AB} \bmod p$ without *A* and *B*
- Later work on asymmetric key encryption uses different hard mathematical problems

What's missing?

- Desired properties
  - Only Alice and Bob know $K=g^{AB} \bmod p$
  - After exchange, if Alice thinks she shares a key K with Bob, then Bob also thinks he shares the same key K with Alice
- Diffie-Hellman key exchange
  - Does not provide authentication of the protocol participants

4A. Send $g^A \bmod p$

Goal: exchange a shared secret key between Alice and Bob

4B. Send $g^B \bmod p$

Alice

Bob

Man-in-the-Middle

- Desired properties:
  - Only Alice and Bob know $K=g^{AB} \bmod p$
  - After the exchange, if Alice thinks she shares a key K with Bob, then Bob also thinks he shares the same key K with Alice

Alice

$g^A \bmod p$

$g^M \bmod p$

Mallory

$g^M \bmod p$

$g^B \bmod p$

Bob

Practical Knowledge about DH [not in exam]

- How do we pick *g* and *p* in practice?
  - Start reading IETF RFC 5114 "*Additional Diffie-Hellman Groups for Use with IETF Standards*", then follow the references therein
- Can we deal with the MITM attack against DH?
  - Read up on "*Station-to-Station Protocol*"
- DH is from 1976; are there new developments about this topic?
  - "*Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice*" by David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann @ ACM CCS 2015 (https://weakdh.org/)

Outline

- Diffie-Hellman key exchange
- Asymmetric (public) key crypto
  - RSA
  - Digital signature schemes

Public Key (asymmetric) crypto

- Everybody has a key pair: private and public key
  - The private key is not communicated to anyone
  - The public key is freely distributed
    - In fact, public keys are often meant to be posted on a *trusted* source (public key server/registry, or some widely-circulated publications)
- Allows encryption and authentication

- Side note:
  - Diffie and Hellman **conjectured** this existed

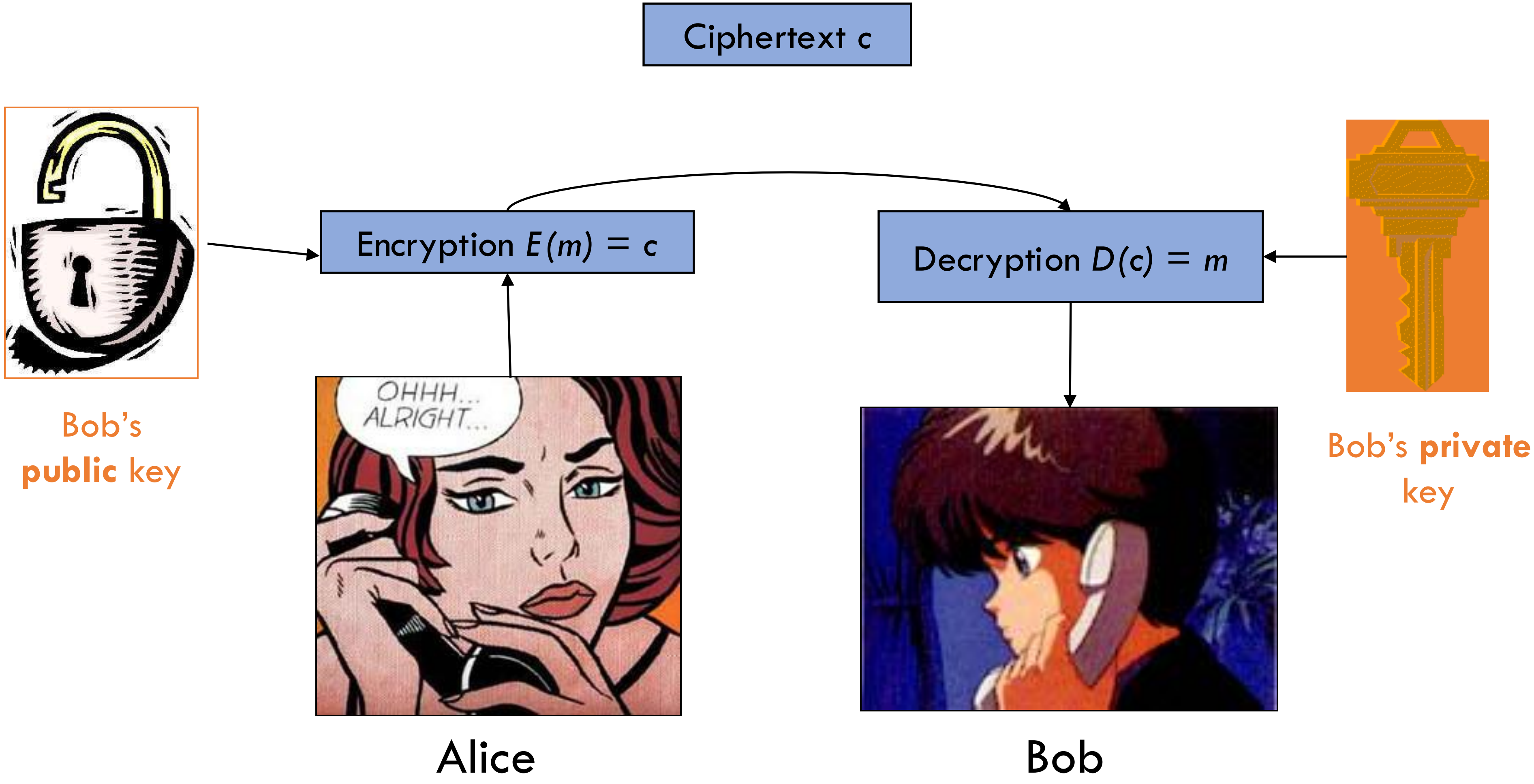
Requirements

- Public (encryption) and private (decryption) keys must be different
- Private key must be impossible (or, more formally, “extremely hard to”) to derive from the public key
- The ciphertext should not reveal anything about the private key
- Must be easy to encrypt/decrypt if knowing the right keys

Informal Definition of Public Key Encryption

- A public key encryption scheme is a triple
  $\langle G, E, D\rangle$ of efficiently computable functions
  - G outputs a "public key" K and a "private key" K^{-1}
    $$\langle K, K^{-1}\rangle \leftarrow G(\cdot)$$
  - E takes public key K and plaintext m as input, and outputs a ciphertext
    $$c \leftarrow E_K(m)$$
  - D takes a ciphertext c and private key K^{-1} as input, and outputs $\perp$ or a plaintext
    $$m \leftarrow D_{K^{-1}}(c)$$
  - If $c \leftarrow E_K(m)$, then $m \leftarrow D_{K^{-1}}(c)$
  - If $c \leftarrow E_K(m)$, then c and K should reveal "no information" about m

Public Key Encryption



RSA (1975-1978)

- Developed shortly after Diffie-Hellman paper
- Takes its name from the initials of its inventors
  - Ron **R**ivest
  - Adi **S**hamir
  - Leonard **A**delman
- Possibly best-known public key algorithm
- Allows encryption and authentication
- R, S, and A won Turing Award for this in 2002
- Clifford Cocks (w/ James Ellis and Malcolm Williamson), at GCHQ (UK), invented independently a particular case of this method 3 years before RSA, but it was classified by British intelligence
  - Declassified in 1997

Shamir, Rivest and Adelman ↑
From: http://www.usc.edu/dept/molecular-science/RSApics.htm

Clifford Cocks →
From:http://www.ulm.ccc.de/old/chaos-seminar/krypto2

RSA

- Key generation (G):
  - Choose two large prime numbers $\boldsymbol{p}$ and $\boldsymbol{q}$ such that $\boldsymbol{p} \neq \boldsymbol{q}$, randomly and independently of each other. Let $\mathbf{n} = \boldsymbol{p} * \boldsymbol{q}$.
  - Pick integer $\boldsymbol{e}$ co-prime with $(\boldsymbol{p}-\mathbf{1})(\boldsymbol{q}-\mathbf{1})$ (i.e., $\gcd(e,(p-1)(q-1)) = 1$)
  - Compute $\boldsymbol{d}$ such that

    $ed \equiv 1 \; mod \; (p-1)(q-1)$, i.e., $ed \; mod \; (p-1)(q-1) = 1$
  - Private key = (n, d)
  - Public key = (n, e)
- Encryption (we assume $m < n$):
  - $E_{(n,e)}(m) = m^e \; mod \; n$
- Decryption (we know $c < n$; we can stop if otherwise):
  - $D_{(n,d)}(c) = c^d \; mod \; n$

Why RSA works

- $ed \bmod (p-1)(q-1) = 1$
- $n = pq$
- $E_{(n,e)}(m) = m^e \bmod n$
- $D_{(n,d)}(c) = c^d \bmod n$

- Need $D_{K^{-1}}\big(E_K(m)\big) = m$

- $(m^e \bmod n)^d \bmod n$

$$
\begin{aligned}
&= m^{ed} \bmod n \\
&= m^{h(p-1)(q-1)+1} \bmod n \\
&= (m^{h(p-1)(q-1)} * m) \bmod n \\
&= m \bmod n
\end{aligned}
$$

First term is 1 from Fermat's Little Theorem

Why RSA works

- Hard problems:
  - Integer factorization
    - Given a number n, find its prime factorization, i.e.,
    $$n = p_1^{e_1} p_2^{e_2} p_3^{e_3} p_4^{e_4} \ldots$$
    - Widely believed to be computationally infeasible to find factorization of $N = p*q$ on classical computers if p and q are large prime numbers
  - RSA problem:
    - Given (i) $c = m^e \bmod n$ and (ii) (n,e), compute m
    - The best algorithm so far is to factorize n

A note on RSA

- Only presented the mathematical intuition
- **Deploying RSA in practice is nowhere near that simple**
  - You need specific "add-ons" to avoid vulnerabilities (OAEP for encryption)
- Choosing parameters properly is paramount
  - Safely choosing and validating primes is mandatory
  - E.g., commonly chosen e=3 turns out to be less secure than previously thought
    - Instantiated as Bleichenbacher attack (2006) against Firefox
    - Now 65537 is recommended
- **Properly using RSA in practice requires more study/effort**

More attacks on RSA (don't be naive!)

- Don't pick too small e, say, $e = 3$ (Hastad's Broadcast attack)
  - Say you intercepted the ciphertext of the same plaintext M to 3 different people
    - $C1 = M^3 \, mod \, N1$
    - $C2 = M^3 \, mod \, N2$
    - $C3 = M^3 \, mod \, N3$
  - ① This is RSA, so we know $M < N1, \; M < N2, \; M < N3$, i.e., $M^3 < N1 * N2 * N3$
  - ② Chinese Remainder Theorem allows you to compute $C' = M^3 \, mod \, (N1 * N2 * N3)$
    - Observe C' is a number between 0 and $N1 * N2 * N3 - 1$
  - Using ①, we simplify ② to $C' = M^3$ (← no more modulo arithmetic here)
  - Take the cube root to recover plaintext: $\sqrt[3]{C'} = M$
- **In short, this is an example showing an attacker with some math knowledge!**

More attacks on RSA (don’t be naive!)

- **Padding Attacks**
- **Timing Attacks**
  - Powermod algorithm uses repeated squaring and multiplication
  - Measure time to figure out if multiplications occur
- Power Attacks
  - Measure smartcard power consumption during signature generation

Interesting read: Lenstra, Arjen K., James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung and Christophe Wachter. “*Ron was wrong, Whit is right*.” IACR Cryptol. ePrint Arch. 2012 (2012): 64.

Elliptic Curve Cryptography (ECC)

- Discrete Logarithm Problem; algebraic structure of elliptic curves over finite fields
- "*finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point is infeasible*"
  - Keys are harder to crack; shorter keys compared to RSA
  - Computational and storage efficiency (less memory overhead than RSA)
- Example uses: TOR and iMessage
  - Elliptic Curve Digital Signature Algorithm (ECDSA) is used with cryptocurrencies to sign transactions and some web applications
  - ECDH: can be used with SSL (later)
- Requires reliable pseudorandom number generators (PRNG)
- Attacks exist; e.g., side-channel attacks
- Easy read on Cloudflare blog
- Good read
  - Hankerson, D., Menezes, A.J. and Vanstone, S., 2006. *Guide to elliptic curve cryptography*. Springer Science & Business Media.

Outline

- Diffie-Hellman key exchange
- Asymmetric (public) key crypto
  - RSA
  - Digital signature schemes

Digital Signatures (Informal Definition)

- A digital signature scheme is a triple $<G, S, V>$ of efficiently computable algorithms
  - G outputs a "public key" K and a "private key" K^{-1}
    $$< K, K^{-1}> \leftarrow G(\cdot)$$
  - S takes a "message" m and K^{-1} as input and outputs a "signature" σ
    $$\sigma \leftarrow S_{K^{-1}}(m)$$
  - V takes a message m, signature σ and public key K as input, and outputs a bit b
    $$b \leftarrow V_K(m, \sigma)$$
  - If $\sigma \leftarrow S_{K^{-1}}(m)$, then $V_K(m, \sigma)$ outputs 1 ("valid")
- Security requirement
  - Given only K and message/signature pairs $\{<m_i, S_{K^{-1}}(m_i)>\}_i$, it is computationally infeasible to compute $<m, \sigma>$ such that
    $$V_K(m, \sigma) = 1$$
    for any new $m \neq m_i$

Digital Signatures (Public Key authentication)

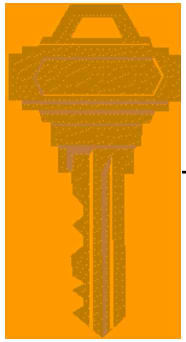
- Scenario:
  - Alice signs a message M with her **private** key
    - Adds a tag (bits) that is binded with the message
  - Bob can verify that M comes from Alice using Alice's **public** key
  - No one but Alice could sign the message that way

    (duplicating a private key is impossible unless the key is leaked)

- Very effective defense against man-in-the-middle attacks
  - But you need a trusted way to verify keys
    - (e.g., certificate authority that signs them)

Public Key authentication (e.g., RSA)

m, c



Alice's
private
key

$D(m) = c$

$E(c) = m'$
$m = m'$?

Alice's
public
key

Alice

Bob

Digital signatures properties

- Authentication
  - Entity; data origin: who originated the message that was signed
- Integrity
  - The signed content is the same as the original
- Non-repudiation
  - When Alice signs a message, cannot deny it later
  - Assumption: only Alice has access to her private key
    - In practice: can users easily protect the leakage of their private keys?

Digital signatures compromises

- Existential forgery
  - The attacker manages to forge a signature of (at least) one message, but not necessarily of their choice
- Selective forgery
  - The attacker manages to forge a signature of (at least) one message of their choice
- Universal forgery
  - The attacker manages to forge a signature of any message
- Total break
  - The attacker can compute the signer’s private key

Encrypt? Sign? In which order?

- Sign-then-Encrypt ?
  - No
  - Add the recipient’s name and/or
  - Sign-Encrypt-Sign
- Encrypt-then-Sign
  - No
  - Encrypt the sender's name and/or
  - Encrypt-sign-Encrypt
- Better solution: Authenticated Encryption (AE)

good read: https://theworld.com/~dtd/sign_encrypt/sign_encrypt7.html

Outline

- Diffie-Hellman key exchange
- Asymmetric (public) key crypto
  - RSA
  - Digital signature schemes

Comparison sym. vs. asym. crypto

Symmetric crypto (AES)

- Need shared secret
- 256-bit key for high security
- 1,000,000 ops/s on a 1 GHz processor
- >100x speedup in hardware

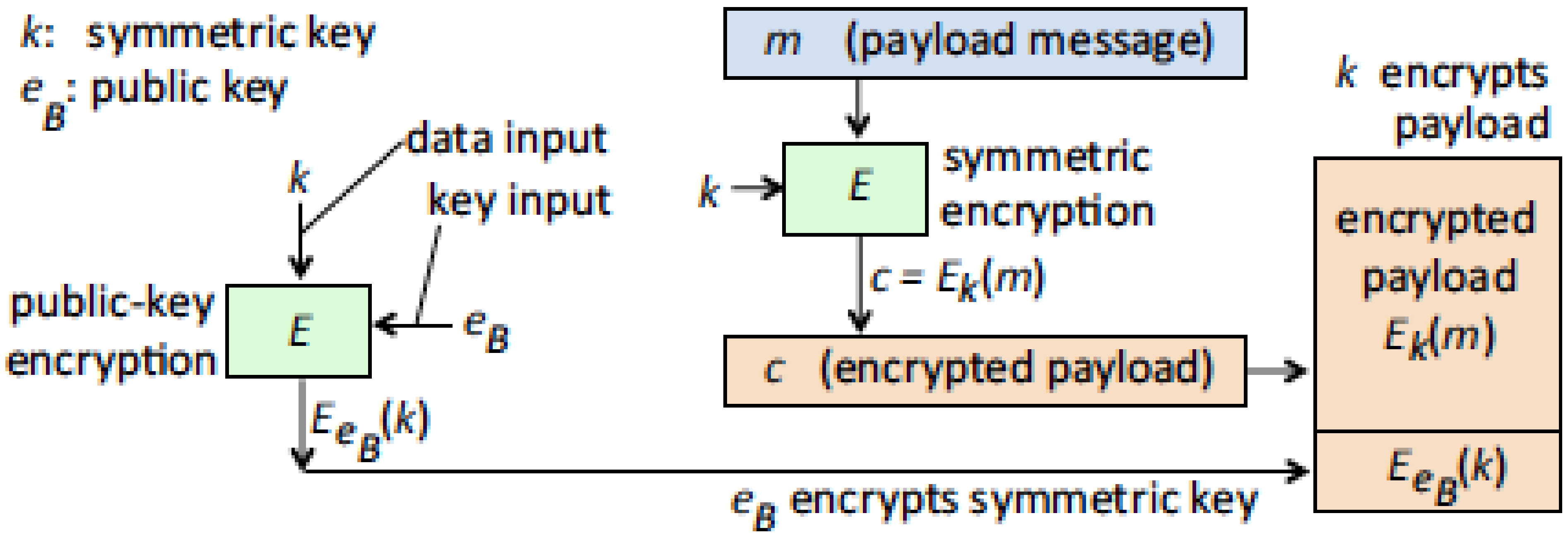
Asymmetric crypto*

- Need authentic public key
- 2048-bit key (RSA)
- 100 signatures/s and 1,000 verifications/s (RSA) on 1 GHz processor
- ~ 10x speedup in hardware

* Excludes Elliptic Curve Crypto

(With thanks to Adrian Perrig for this slide)

We can put things together



Source: Computer Security and the Internet: Tools and Jewels from Malware to Bitcoin, Second Edition by Paul C. van Oorschot. Springer, 2021.

Takeaway (1)

- Exchanging secret keys is difficult, and doesn't scale well
- Diffie-Hellman-Merkle key exchange protocol makes each party independently compute the secret key based on
    - publicly available information (g, p),
    - their own secret (A and B)
    - partial information about the other party's secret
  - The scheme does not support authentication

Takeaway (2)

- Public key crypto
  - Builds on Diffie-Hellman-Merkle's ideas
  - Provides encryption **and** authentication
    - Encryption: use the recipient's public key
    - Authentication: use your private key
  - Much slower than symmetric cryptography
  - Must be careful with implementation!
- Digital signatures
  - Rely on public key crypto
  - Useful for authentication (data origin), and to thwart man-in-the-middle attacks
  - Provides non-repudiation (theoretical)
  - Provide Integrity
- Always check the latest recommendations
  - NIST is a good source