Cryptography

Public Key Cryptography

(“Asymmetric Encryption Systems”)

Professor Travis Peters
CSCI 476 - Computer Security
Spring 2020

Some slides and figures adapted from Wenliang (Kevin) Du’s
Computer & Internet Security: A Hands-on Approach (2nd Edition).
Thank you Kevin and all of the others that have contributed to the SEED resources!

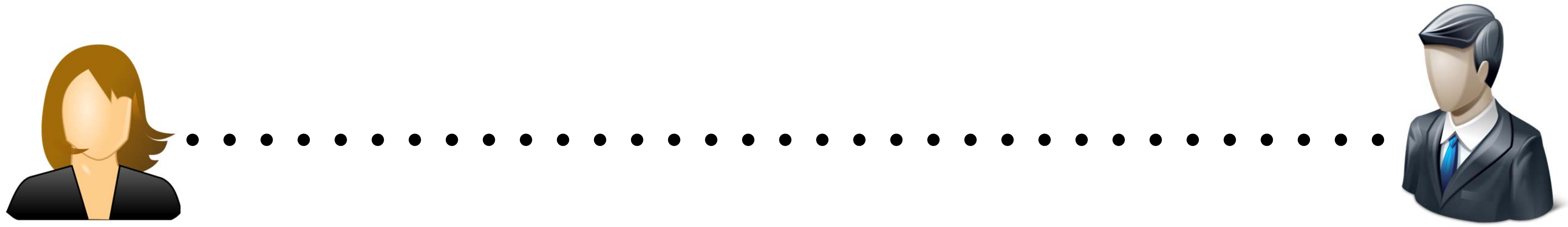
Introduction to Public Key Cryptography

This Video Covers:

- Overview of Public-Key Cryptography
- Overview of where we are going this week

Introduction & Overview

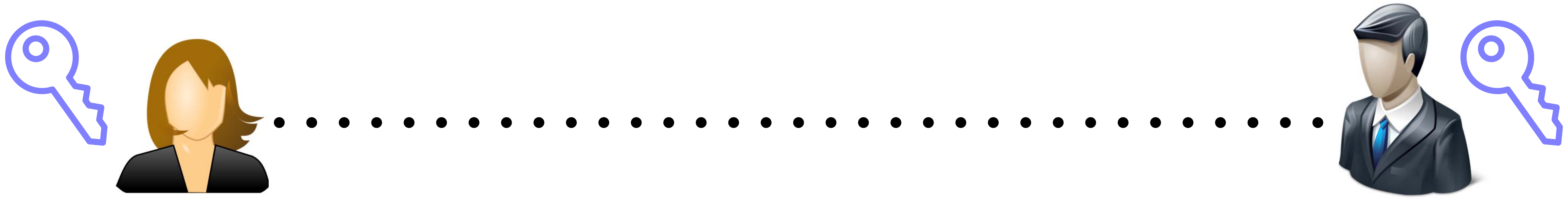
- **Public Key Cryptography** is at the foundation of today's secure communication
- Allows communicating parties to obtain a ***shared secret key***



Introduction & Overview

- **Public Key Cryptography** is at the foundation of today's secure communication
- Allows communicating parties to obtain a ***shared secret key***

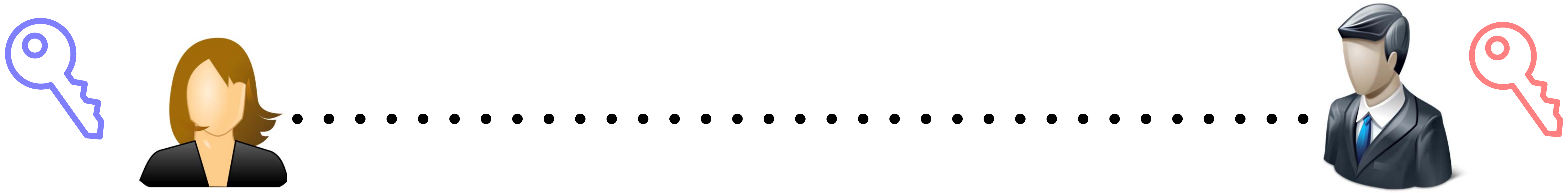
In **secret key crypto**, the **same key** was used for both encryption and decryption



Introduction & Overview

- **Public Key Cryptography** is at the foundation of today's secure communication
- Allows communicating parties to obtain a ***shared secret key***
- Public key (for ***encryption***) and Private key (for ***decryption***)
- Private key (to create ***digital signature***) and Public key (to ***verify signature***)

In **public key crypto**, different keys are used for encryption and decryption

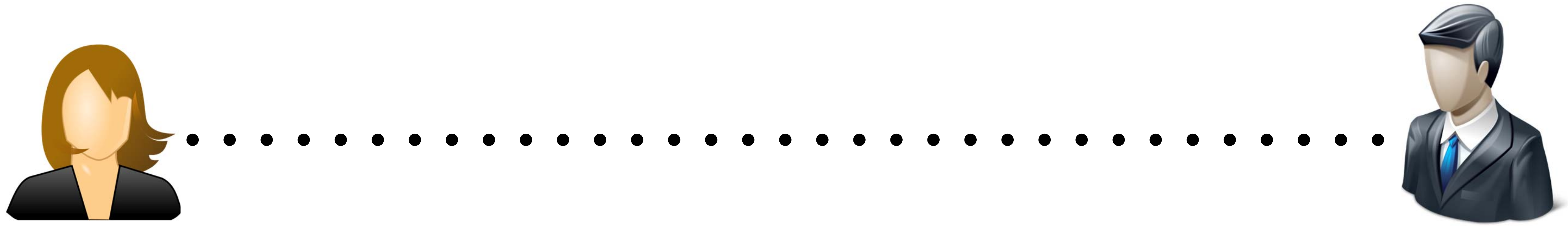


A Brief History Lesson

- Historically same key was used for encryption and decryption
- **Challenge:** exchanging the secret key (e.g. face-to-face meeting)

- 1976: Whitfield Diffie and Martin Hellman
  - DH key exchange protocol
  - proposed a new public-key cryptosystem

- 1978: Ron Rivest, Adi Shamir, and Leonard Adleman (all from MIT)
  - attempted to develop a public-key cryptosystem
  - created RSA algorithm



Outline

- Public-key algorithms
  - Diffie-Hellman key exchange
  - RSA algorithm
  - Digital signatures
- Public-key crypto & Python
- Applications
  - Authentication
  - HTTPS and TLS/SSL
  - Chip Technology Used in Credit Cards

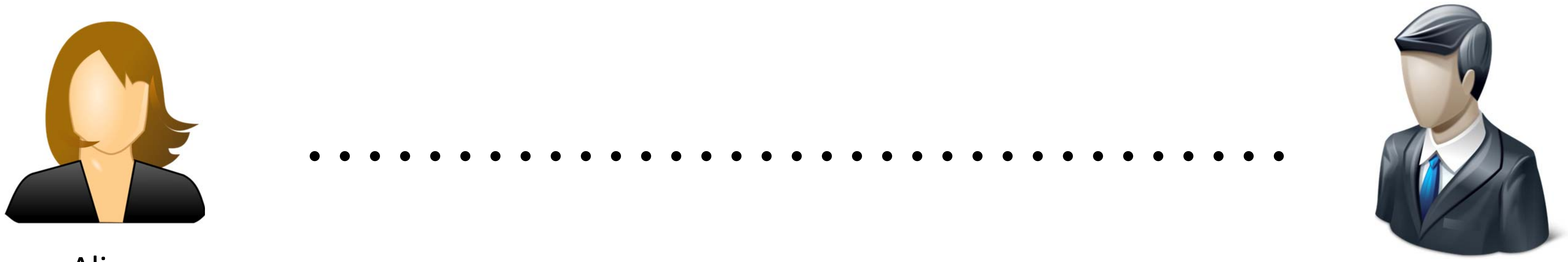
Diffie-Hellman Key Exchange

This Video Covers:

- The DH key exchange protocol
- How to exchange (symmetric) keys

Diffie-Hellman Key Exchange *(High-Level)*

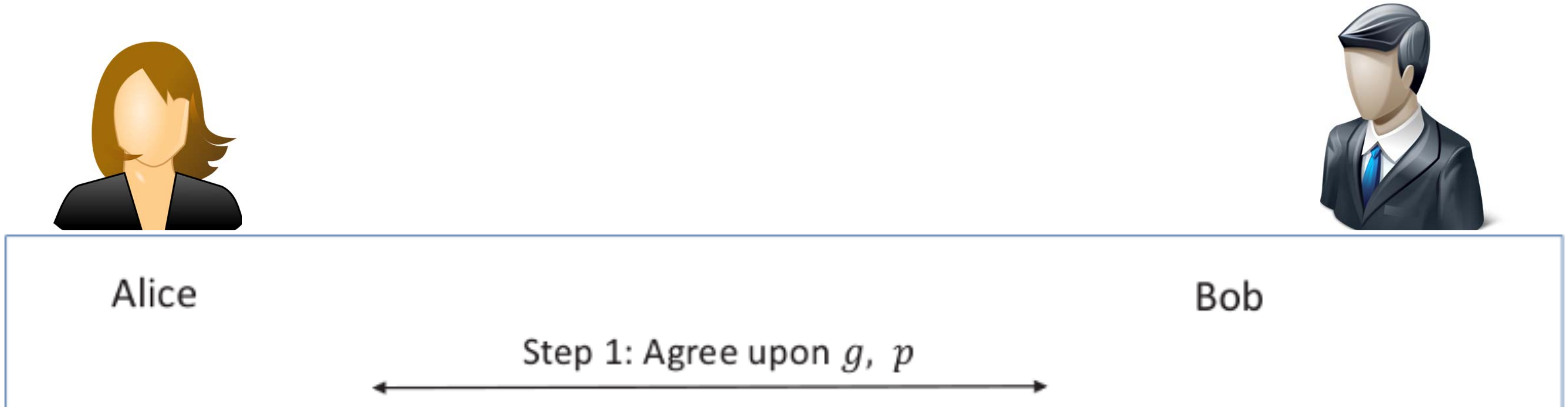
Allows communicating parties *with* ***no prior knowledge*** to ***exchange shared secret keys over an insecure channel***



Alice

Bob

Diffie-Hellman Key Exchange *(cont.)*



Diffie-Hellman Key Exchange *(cont.)*

Alice

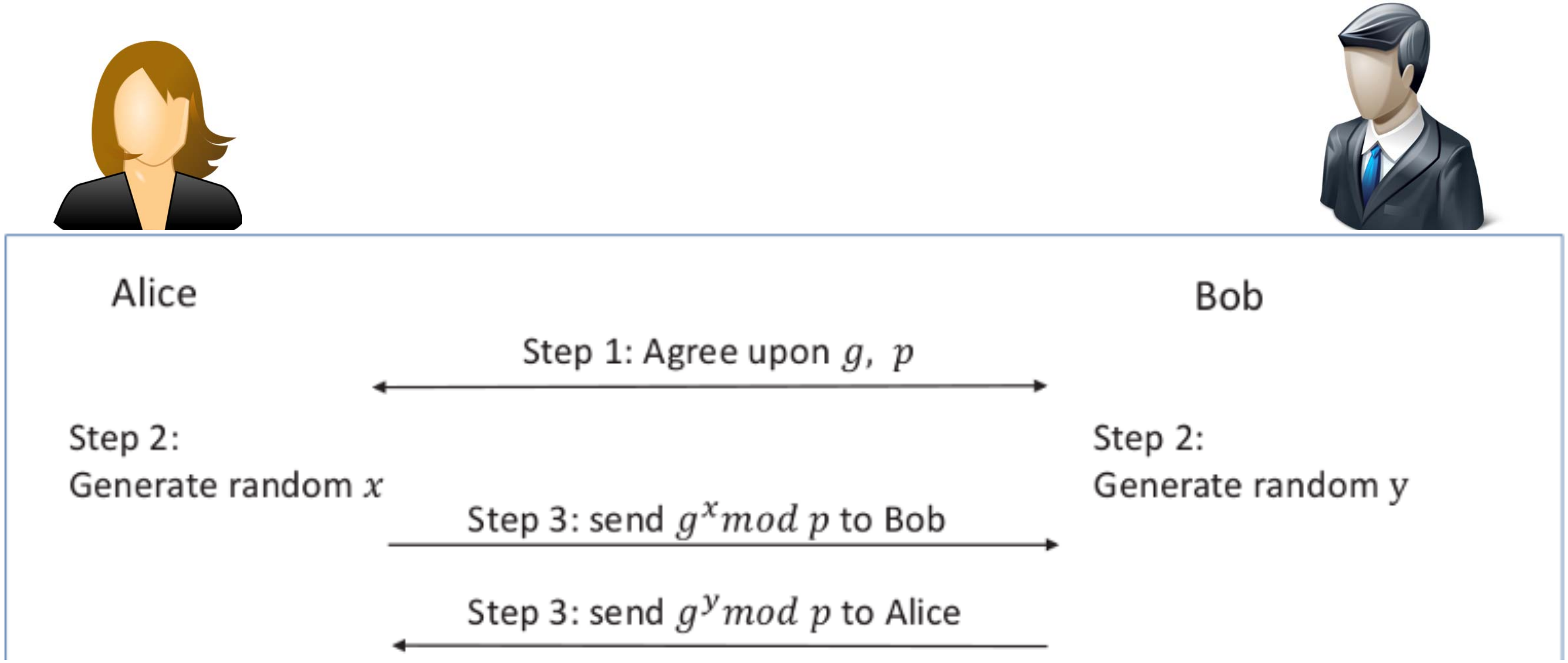
Bob

Step 1: Agree upon $g,\ p$

Step 2:
Generate random x

Step 2:
Generate random y

Diffie-Hellman Key Exchange *(cont.)*



Diffie-Hellman Key Exchange *(cont.)*

Alice

Bob

Step 1: Agree upon $g,\ p$

Step 2:
Generate random x

Step 2:
Generate random y

Step 3: send $g^x mod\ p$ to Bob

Step 3: send $g^y mod\ p$ to Alice

Step 4: Compute
$K = (g^y mod\ p)^x\ mod\ p$
$= \boldsymbol{g^{xy}\ mod\ p}$

Step 4: Compute
$K = (g^x mod\ p)^y\ mod\ p$
$= \boldsymbol{g^{xy}\ mod\ p}$

Turn DH Key Exchange into a Public-Key Encryption Algorithm!

- DH key exchange protocol allows two parties to exchange ***a secret***
- Protocol can be tweaked to turn into a public-key encryption scheme if…
  - **Public key:** known to the public and used for encryption
  - **Private key:** known only to the owner, and used for decryption
  - Establish algorithm(s) for encryption and decryption

Turn DH Key Exchange into a Public-Key Encryption Algorithm!

Alice

Key Generation

Public key: $\boldsymbol{g},\ \boldsymbol{p},\ \boldsymbol{g^x mod\ p}$

Private key: $\boldsymbol{g},\ \boldsymbol{p},\ \boldsymbol{x}$

Alice's Public key → Bob

Bob

Encryption

- Generate a random number y
- Compute $K = (g^x mod\ p)^y\ mod\ p$
  $= g^{xy}\ mod\ p$
- Ciphertext $C = ENCRYPT_K(M)$

$C,\ g^y\ mod\ p$ → Alice

Decryption

- Compute $K = (g^y mod\ p)^x\ mod\ p$
  $= g^{xy}\ mod\ p$
- Plaintext $M = DECRYPT_K(C)$

Turn DH Key Exchange into a Public-Key Encryption Algorithm! *(cont.)*

Alice determines $g, p, g^x \bmod p$ — publish!

Alice

Bob

Key Generation

Public key: $\boldsymbol{g}$, $\boldsymbol{p}$, $\boldsymbol{g^x mod\ p}$

Private key: $\boldsymbol{g}$, $\boldsymbol{p}$, $\boldsymbol{x}$

Keep x private

Alice's Public key →

Encryption

- Generate a random number y
- Compute $K = (g^x mod\ p)^y\ mod\ p$
  $= g^{xy}\ mod\ p$
- Ciphertext $C = ENCRYPT_K(M)$

← $C,\ g^y\ mod\ p$

Decryption

- Compute $K = (g^y mod\ p)^x\ mod\ p$
  $= g^{xy}\ mod\ p$
- Plaintext $M = DECRYPT_K(C)$

Turn DH Key Exchange into a Public-Key Encryption Algorithm! *(cont.)*

Alice

Bob

Key Generation

Public key: $\boldsymbol{g,\ p,\ g^x mod\ p}$

Private key: $\boldsymbol{g,\ p,\ x}$

Alice's Public key

Encryption

- Generate a random number y
- Compute $K = (g^x mod\ p)^y\ mod\ p$
  $= g^{xy}\ mod\ p$
- Ciphertext $C = ENCRYPT_K(M)$

can compute K based on y and params. from Alice's pub. key

use, e.g., AES-CBC to encrypt m

$C,\ g^y\ mod\ p$

send to Alice

Decryption

- Compute $K = (g^y mod\ p)^x\ mod\ p$
  $= g^{xy}\ mod\ p$
- Plaintext $M = DECRYPT_K(C)$

Turn DH Key Exchange into a Public-Key Encryption Algorithm! *(cont.)*

Alice

Bob

Key Generation

Public key: $g,\ p,\ g^x \bmod p$

Private key: $g,\ p,\ x$

Alice's Public key →

Encryption

- Generate a random number y
- Compute $K = (g^x \bmod p)^y \bmod p$
  $= g^{xy} \bmod p$
- Ciphertext $C = ENCRYPT_K(M)$

← $C,\ g^y \bmod p$

Decryption

- Compute $K = (g^y \bmod p)^x \bmod p$
  $= g^{xy} \bmod p$
- Plaintext $M = DECRYPT_K(C)$

Given info from Bob, compute K & decrypt C

The RSA Algorithm

This Video Covers:

- **Modulo Operation (no video)**
- Euler's Theorem
- Extended Euclidean Algorithm
- RSA Algorithm
- Examples

Modulo Operation

- The RSA algorithm is based on **modulo operations**

modulus

remainder/residue

$$a \bmod n = r$$

Modulo Operation

- The RSA algorithm is based on **modulo operations**

modulus

remainder/residue

$$a \bmod n = r$$

- Examples:
  - $10 \ \mathtt{mod}\ 3 = ?$
  - $15 \ \mathtt{mod}\ 5 = ?$

Modulo Operation

- The RSA algorithm is based on **modulo operations**

modulus

remainder/residue

$$a \bmod n = r$$

- Examples:
  - $10 \bmod 3 = 1$
  - $15 \bmod 5 = 0$

Modulo Operation

- The RSA algorithm is based on **modulo operations**

modulus

remainder/residue

$$a \bmod n = r$$

- Examples:
  - $10 \bmod 3 = 1$
  - $15 \bmod 5 = 0$

- Modulo operations are ***distributive***:

$$(a + b) \bmod n = [(a \bmod n) + (b \bmod n)] \bmod n$$

$$a * b \bmod n = [(a \bmod n) * (b \bmod n)] \bmod n$$

$$a^x \bmod n = (a \bmod n)^x \bmod n$$

The RSA Algorithm

This Video Covers:

- Modulo Operation
- **Euler's Theorem (no video)**
- Extended Euclidean Algorithm
- RSA Algorithm
- Examples

Euler's Theorem → easily reduce large powers modulo n

- Euler's totient function φ(n) counts the positive integers up to a given integer n that are *relatively prime* to n
  - $\varphi(n) = n - 1$, if n is a prime number.

- Euler's totient function property:
  - if m and n are relatively prime, $\varphi(mn) = \varphi(m) * \varphi(n)$

- Euler's theorem states:
  - $a^{\varphi(n)} = 1 \pmod{n}$

Euler's Theorem *(cont.)* → easily reduce large powers modulo n

- **Example:** Calculate $4^{100003} \bmod 33$

Euler's Theorem *(cont.)* → easily reduce large powers modulo n

- **Example:** Calculate $4^{100003} \bmod 33$

use $a^{\phi(n)} = 1 \bmod n$
to simplify

$a = 4$
$n = 33$
$\phi(n) = \phi(33) = \ldots$

Euler's Theorem *(cont.)* → **easily reduce large powers modulo n**

- **Example:** Calculate $4^{100003} \bmod 33$
  - $\varphi(33) = \varphi(3) * \varphi(11) = (3 - 1) * (11 - 1) = 20$
  - $100003 = 5000\varphi(33) + 3$

Euler's Theorem *(cont.)* → **easily reduce large powers modulo n**

- **Example:** Calculate 4^{100003} mod 33
  - $\varphi(33) = \varphi(3) * \varphi(11) = (3 - 1) * (11 - 1) = 20$
  - $100003 = 5000\varphi(33) + 3$

$$4^{100003} \bmod 33 = 4^{20 \cdot 5000+3} \bmod 33$$

Euler's Theorem *(cont.)* → easily reduce large powers modulo n

- **Example:** Calculate 4^{100003} mod 33
  - $\varphi(33) = \varphi(3) * \varphi(11) = (3 - 1) * (11 - 1) = 20$
  - $100003 = 5000\varphi(33) + 3$

$$\begin{aligned} 4^{100003} \bmod 33 &= 4^{20 \cdot 5000 + 3} \bmod 33 \\ &= \left(4^{20}\right)^{5000} * 4^{3} \bmod 33 \end{aligned}$$

Euler's Theorem *(cont.)* → easily reduce large powers modulo n

- **Example:** Calculate 4^{100003} mod 33
  - $\varphi(33) = \varphi(3) * \varphi(11) = (3 - 1) * (11 - 1) = 20$
  - $100003 = 5000\varphi(33) + 3$

$$
\begin{aligned}
4^{100003} \bmod 33 &= 4^{20\cdot 5000+3} \bmod 33 \\
&= \left(4^{20}\right)^{5000} * 4^{3} \bmod 33 \\
&= \left[\left(4^{20}\right)^{5000} \bmod 33\right)\Big] * 4^{3} \bmod 33 \text{ \textbf{(applying distributive rule)}}
\end{aligned}
$$

Euler's Theorem *(cont.)* → easily reduce large powers modulo n

- **Example:** Calculate 4^{100003} mod 33
  - φ(33) = φ(3) * φ(11) = (3 − 1) * (11 − 1) = 20
  - 100003 = 5000φ(33) + 3

$$
\begin{aligned}
4^{100003} \bmod 33 &= 4^{20\cdot 5000+3} \bmod 33 \\
&= \left(4^{20}\right)^{5000} * 4^{3} \bmod 33 \\
&= \left[\left(4^{20}\right)^{5000} \bmod 33\right)\Big] * 4^{3} \bmod 33 \text{ \textbf{(applying distributive rule)}} \\
&= \left[\left(4^{20} \bmod 33\right)\right]^{5000} * 4^{3} \bmod 33 \text{ \textbf{(applying distributive rule)}}
\end{aligned}
$$

Euler's Theorem *(cont.)* → easily reduce large powers modulo n

- **Example:** Calculate 4^{100003} mod 33
  - φ(33) = φ(3) * φ(11) = (3 – 1) * (11 – 1) = 20
  - 100003 = 5000φ(33) + 3

$$
\begin{aligned}
4^{100003} \bmod 33 &= 4^{20\cdot 5000+3} \bmod 33 \\
&= \left(4^{20}\right)^{5000} * 4^3 \bmod 33 \\
&= \left[\left(\left(4^{20}\right)^{5000} \bmod 33\right)\right] * 4^3 \bmod 33 \text{ (applying distributive rule)} \\
&= \left[\left(4^{20} \bmod 33\right)\right]^{5000} * 4^3 \bmod 33 \text{ (applying distributive rule)} \\
&= 1^{5000} * 64 \bmod 33 \text{ (applying Euler's theorem)}
\end{aligned}
$$

Euler's Theorem *(cont.)* → easily reduce large powers modulo n

- **Example:** Calculate 4^{100003} mod 33
  - $\varphi(33) = \varphi(3) * \varphi(11) = (3 - 1) * (11 - 1) = 20$
  - $100003 = 5000\varphi(33) + 3$

$$
\begin{aligned}
4^{100003} \bmod 33 &= 4^{20\cdot 5000+3} \bmod 33 \\
&= \left(4^{20}\right)^{5000} * 4^3 \bmod 33 \\
&= \left[\left(\left(4^{20}\right)^{5000} \bmod 33\right)\right] * 4^3 \bmod 33 \text{ (applying distributive rule)} \\
&= \left[\left(4^{20} \bmod 33\right)\right]^{5000} * 4^3 \bmod 33 \text{ (applying distributive rule)} \\
&= 1^{5000} * 64 \bmod 33 \text{ (applying Euler's theorem)} \\
&= 31
\end{aligned}
$$

The RSA Algorithm

This Video Covers:

- Modulo Operation
- Euler's Theorem
- **Extended Euclidean Algorithm (no video)**
- RSA Algorithm
- Examples

RSA and the Extended Euclidean Algorithm

- **Euclid's algorithm:** an efficient method for computing GCD of two #'s
- **Extended Euclidean algorithm:**
  - computes GCD of integers a and b
  - finds integers x and y, such that: $ax + by = g = \gcd(a, b)$

RSA and the Extended Euclidean Algorithm

- **Euclid's algorithm:** an efficient method for computing GCD of two #'s
- **Extended Euclidean algorithm:**
  - computes GCD of integers a and b
  - finds integers x and y, such that: ax + by = g = gcd(a, b)

- RSA uses Extended Euclidean algorithm:
  - e and n are components of public key
  - Find solution to equation:

    $e * x + \varphi(n) * y = \text{gcd}(e, \varphi(n)) = 1$

```
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, x, y = egcd(b % a, a)
        return (g, y - (b // a) * x, x)
```

RSA and the Extended Euclidean Algorithm

- **Euclid's algorithm:** an efficient method for computing GCD of two #'s
- **Extended Euclidean algorithm:**
  - computes GCD of integers a and b
  - finds integers x and y, such that: ax + by = g = gcd(a, b)

- RSA uses Extended Euclidean algorithm:
  - e and n are components of public key
  - Find solution to equation:

    $e * x + \varphi(n) * y = \gcd(e, \varphi(n)) = 1$
  - x is private key *(also referred as d)*
  - Equation results: $e * d \bmod \varphi(n) = 1$

```
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, x, y = egcd(b % a, a)
        return (g, y - (b // a) * x, x)
```

The RSA Algorithm

This Video Covers:

- Modulo Operation
- Euler's Theorem
- Extended Euclidean Algorithm
- **RSA Algorithm**
- Examples

RSA: Key Generation

Key Generation → *Encryption* → *Decryption*

- **Need to generate:** modulus n, public key exponent e, private key exponent d

- **Approach:**
  - Choose p, q → *large random prime numbers (secret!)*
  - $n = pq$ → *should be LARGE; computationally hard to factor n → Euler's Theorem*
  - Choose e, $1 < e < \varphi(n)$ and e is *relatively prime* to $\varphi(n)$
    → *e is the "public-key exponent" (e.g., e = 65537)*
  - Find d, $ed \bmod \varphi(n) = 1$
    → *solve using the Extended Euclidean Algorithm; d is the "private-key exponent" (secret!)*
    Can be solved in polynomial time if you know p, q, and e!

- **Result:**
  - (e, n) is public key → *without knowledge of p or q, computationally hard to find d*
  - d is private key

RSA: Encryption & Decryption

*Key Generation → **Encryption** → **Decryption***

Encryption

- Treat the plaintext as a number
- Assuming $M < n$
- $C = M^e \texttt{ mod } n$

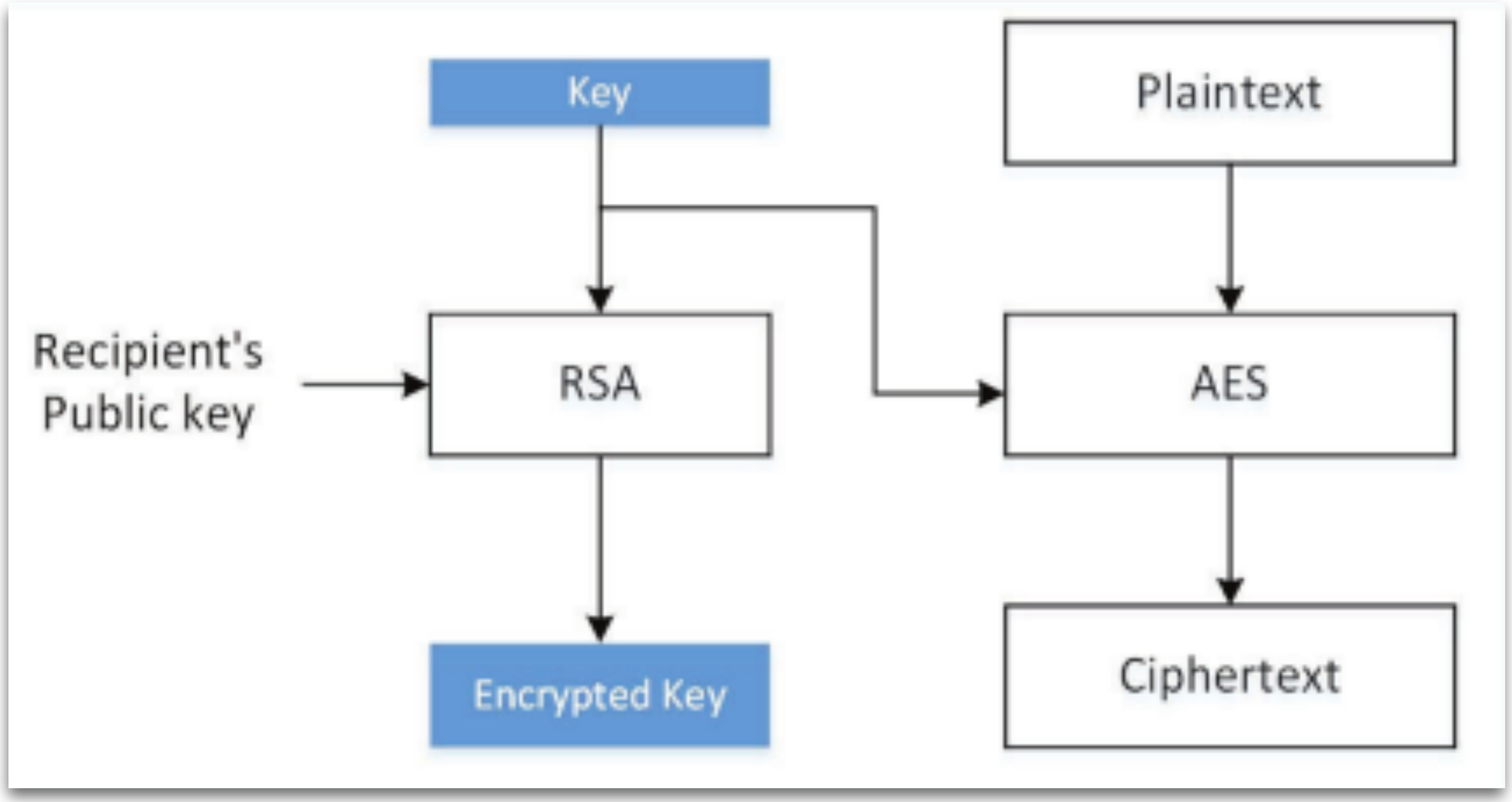
Decryption

- $M = C^d \texttt{ mod } n$

You can convince yourself (see below) that decryption does indeed yield back the message, M…

$$
\begin{aligned}
M^{ed} \bmod n &= M^{k\phi(n)+1} \bmod n \quad \textbf{(note: } ed = k\phi(n) + 1\textbf{)} \\
&= M^{k\phi(n)} * M \quad \bmod n \\
&= \left(M^{\phi(n)} \bmod n\right)^k * M \bmod n \quad \textbf{(applying distributive rule)} \\
&= 1^k * M \bmod n \quad \textbf{(applying Euler's theorem)} \\
&= M
\end{aligned}
$$

Hybrid Encryption



- Public-key encryption is computationally expensive (e.g., large-number multiplications)
- Use public key algorithms to ***exchange a secret session key***
- The key (data-encryption key) used to encrypt data using a symmetric-key algorithm (e.g., AES-128-CBC)

The RSA Algorithm

This Video Covers:

- Modulo Operation
- Euler's Theorem
- Extended Euclidean Algorithm
- RSA Algorithm
- **Examples (no video)**

RSA: Exercise w/ Small Numbers

- Choose two prime numbers $p = 13$ and $q = 17$
- Find e:
  - $n = pq = 221$
  - $\varphi(n) = (p-1)(q-1) = 192$
  - choose $e = 7 \rightarrow$ *7 is relatively prime to* $\varphi(n)$
- Find $\varphi(n)$:
  - $ed = 1 \mathtt{~mod~} \varphi(n)$
- Solving the above equation is equivalent to: $7d + 192y = 1$
- Using Extended Euclidean algorithm, we get $d = 55$ and $y = -2$

RSA: Exercise w/ Small Numbers *(cont.)*

- Encrypt $M = 36$

$$
\begin{aligned}
M^e \bmod n &= 36^7 \bmod 221 \\
&= \left(36^2 \bmod 221\right)^3 * 36 \bmod 221 \\
&= 191^3 * 36 \mathbf{\ mod\ } 221 \\
&= 179 \mathbf{\ mod\ } 221.
\end{aligned}
$$

- Ciphertext $C = 179$

RSA: Exercise w/ Small Numbers *(cont.)*

$$
\begin{aligned}
C^d \textbf{ mod } n &= 179^{55} \textbf{ mod } 221 \\
&= \left(179^2 \textbf{ mod } 221\right)^{27} * 179 \textbf{ mod } 221 \\
&= 217^{27} * 179 \textbf{ mod } 221 \\
&= \left(217^2 \textbf{ mod } 221\right)^{13} * 217 * 179 \textbf{ mod } 221 \\
&= 16^{13} * 217 * 179 \textbf{ mod } 221 \\
&= \left(16^2 \textbf{ mod } 221\right)^{6} * 16 * 217 * 179 \textbf{ mod } 221 \\
&= 35^{6} * 16 * 217 * 179 \textbf{ mod } 221 \\
&= \left(35^2 \textbf{ mod } 221\right)^{3} * 16 * 217 * 179 \textbf{ mod } 221 \\
&= 120^{3} * 16 * 217 * 179 \textbf{ mod } 221 \\
&= \left(120^2 \textbf{ mod } 221\right) * 120 * 16 * 217 * 179 \textbf{ mod } 221 \\
&= 35 * 120 * 16 * 217 * 179 \textbf{ mod } 221 \\
&= 36 \textbf{ mod } 221
\end{aligned}
$$

RSA: Exercise w/ Large Numbers

Example w/ larger numbers discussed in the text

+

`rsa.c`