CS459/698 Privacy, Cryptography, Network and Data Security

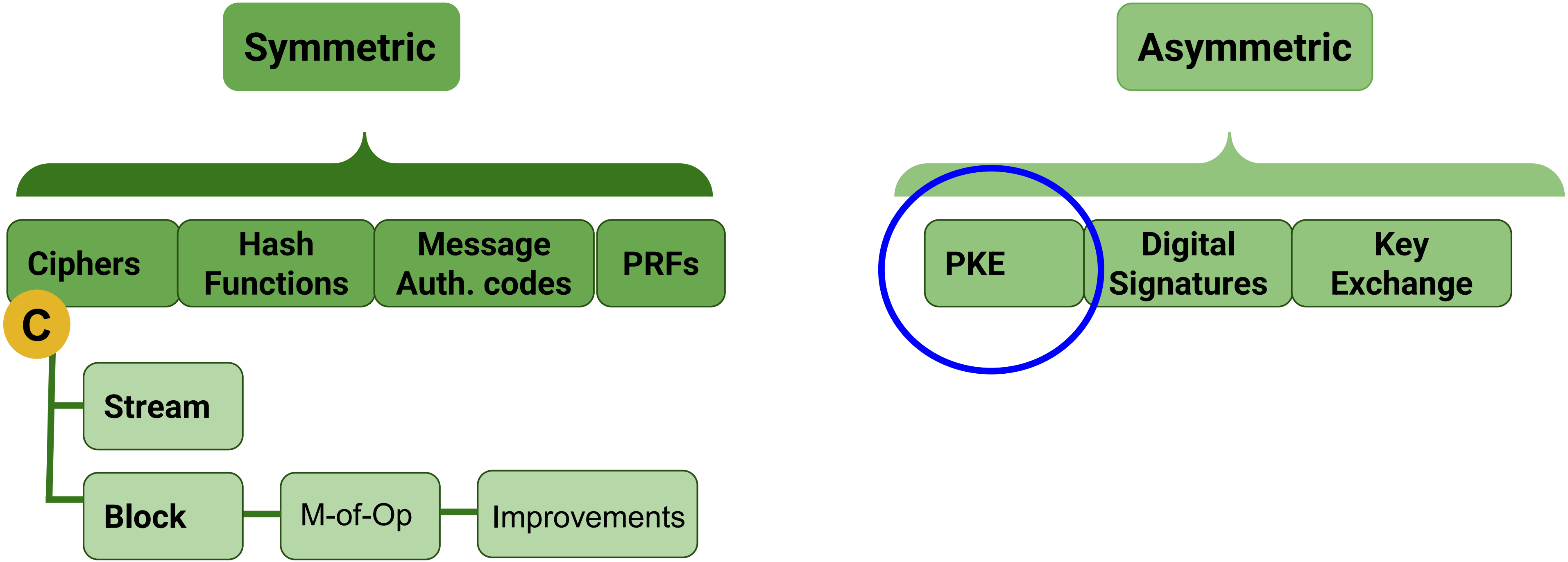
Public Key Cryptography (RSA)

Fall 2025, Tuesday/Thursday 8:30-9:50am

Assignment One

- Available on Learn today
- Due **Sep 30, 3pm**
- Written and programming

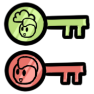
Cryptography Organization



Public-key Encryption

- Invented (in public) in the 1970’s
- Also called Asymmetric Encryption
  - Allows Alice to send a secret message to Bob without any prearranged shared secret!
  - In secret-key encryption, the same (or a very similar) key encrypts the message and also decrypts it

  - In public-key encryption, there’s one key for encryption, and a different key for decryption!



- Some common examples:
  - RSA, ElGamal, ECC, NTRU, McEliece

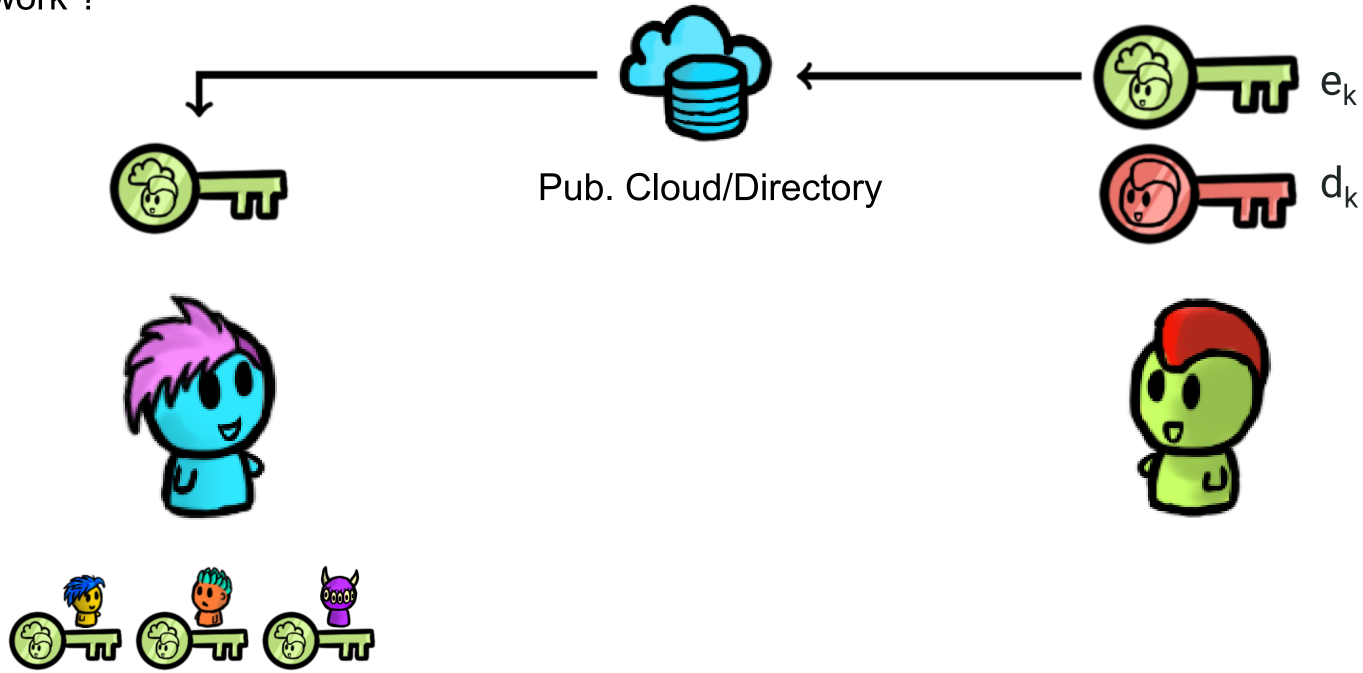
Public-key Encryption

How does it work ?

key pair (e_k , d_k)

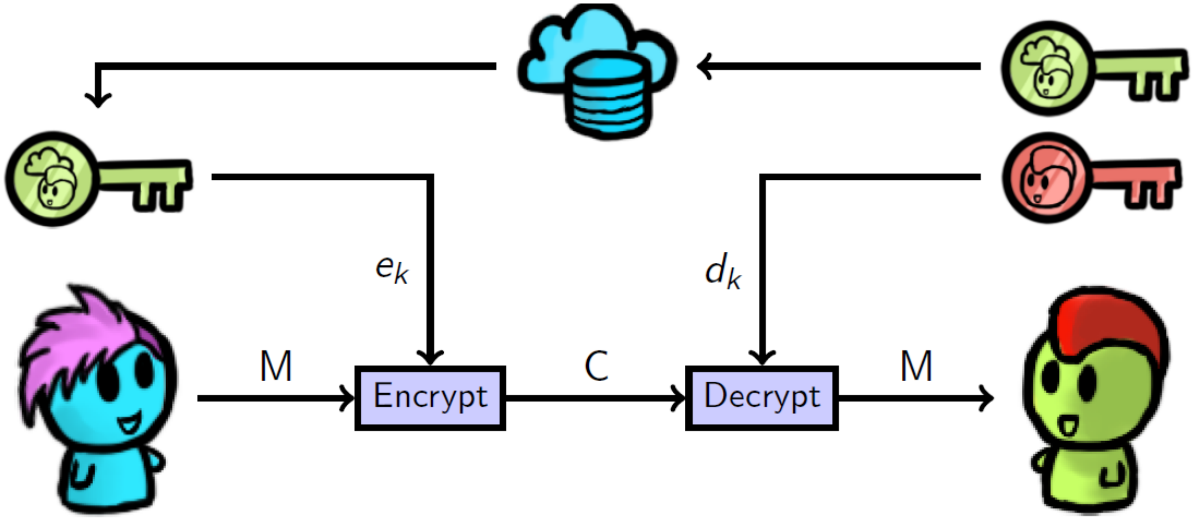
Public-key Encryption

How does it work ?



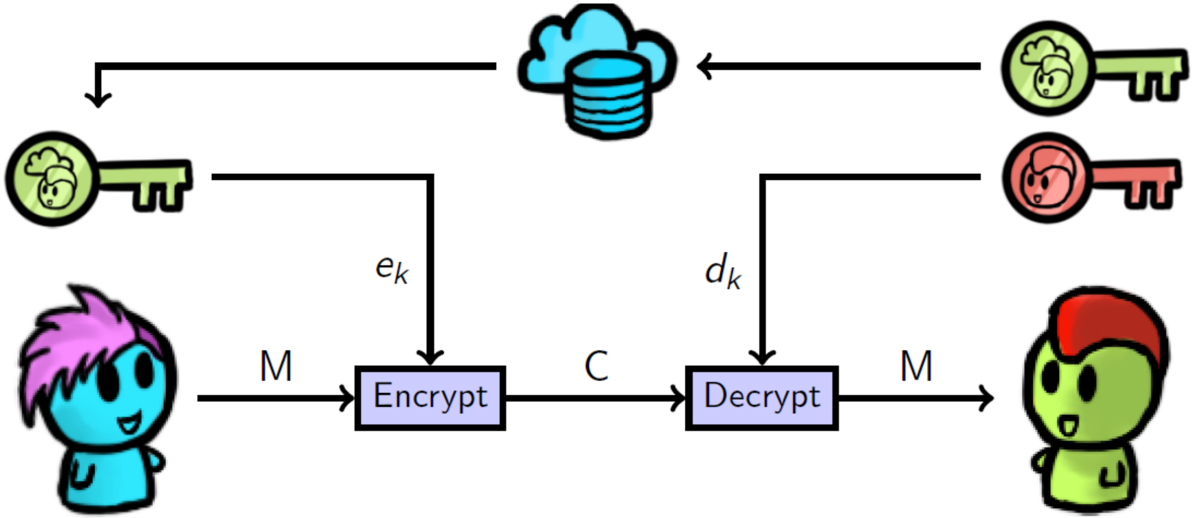
Public-key Encryption

How does it work ?



Public-key Encryption

How does it work ?



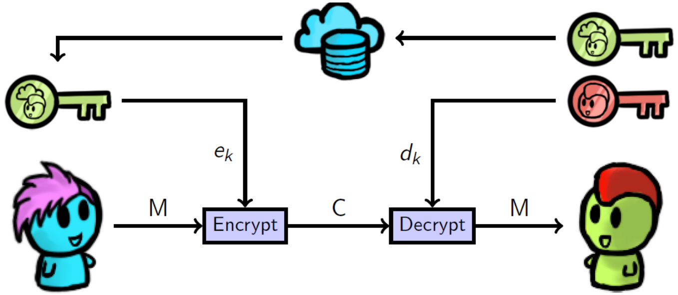
- ✓ Eve can't decrypt; she only has the encryption key e_k
- ✓ Neither can Alice!
- ✓ It must be HARD to derive d_k from e_k

Steps for PKE

1. Bob creates a key pair

2. Bob gives everyone the public key

3. Alice encrypts m and sends it
4. Bob decrypts using private key



5. Eve and Alice can't decrypt, only have encryption key

Requirements for PKE

- The encryption function?
  - Must be easy to compute 
- The inverse, decryption?
  - Must be hard for anyone without the key  vs. 

Thus, we require so called “one-way” functions for this.

Requirements for PKE

- The encryption function?
  - Must be easy to compute 
- The inverse, decryption?
  - Must be hard for anyone without the key  vs. 

Thus, we require so called “one-way” functions for this.

But because of decryption, we need a “Trapdoor”

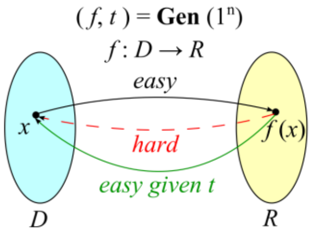
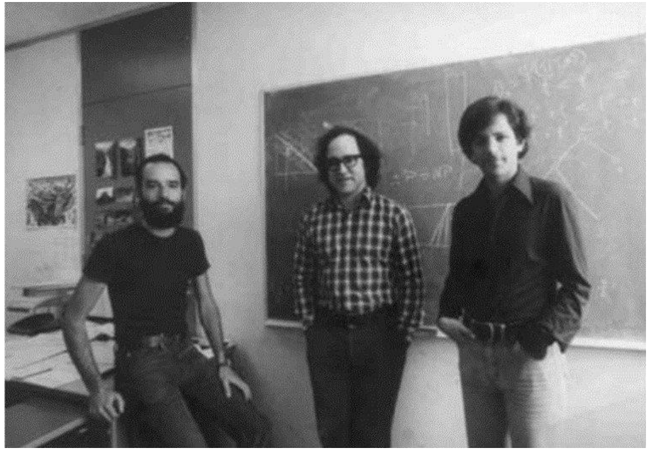


Image Credit: https://en.wikipedia.org/wiki/Trapdoor_function

Textbook RSA

- Relies on the practical difficulty of the "Factoring problem"
- Modular arithmetic: integer numbers that "wrap around"



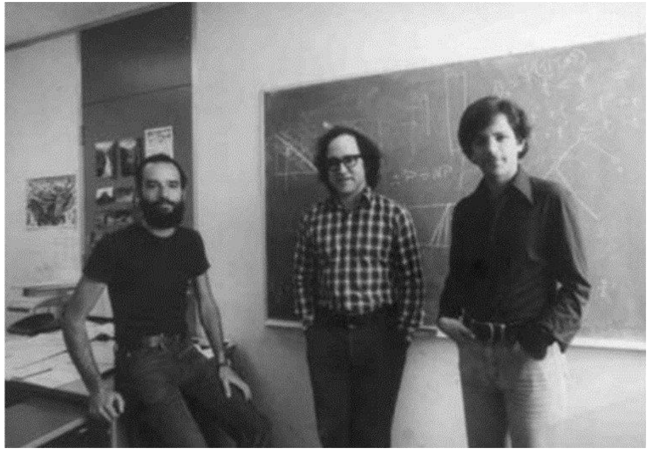
Left to right: Ron Rivest, Adi Shamir, and Leonard Adleman.

Fun (?) Facts:

- RSA was the first popular public-key encryption method, published in 1977

Textbook RSA

- Relies on the practical difficulty of the "Factoring problem"
- Modular arithmetic: integer numbers that "wrap around"



Left to right: Ron Rivest, Adi Shamir, and Leonard Adleman.

Example of modular arithmetic:

7 mod 5 = 2
12 mod 5 = 2

$7 \equiv 12 \bmod 5$
(*congruent* modulo 5)
(same remainder when divided by 5)

Fun (?) Facts:

- RSA was the first popular public-key encryption method, published in 1977

Prime Numbers

- **Prime:** a natural number greater than 1 that can only be divided by 1 and itself
- **Primes and factorization:** An integer number can be written as a unique product of prime numbers
  - E.g., 1234567 = 127 * 9721

How to know if a number is prime?

Run a primality test algorithm (Solovay-Strassen, Miller-Rabin, etc.)

How to discover a number's factors?

Run a factorization algorithm (Pollard p-1, etc.)

Textbook RSA

- High-level idea
  - It is easy to find large integers e, d, and n (=p*q) that satisfy

$$(m^e)^d \equiv m \pmod{n}$$

- Computational difficulty of the **factoring problem**
  - Given two large primes p*q = n, it is very hard to factor n.

Easy for me to pick **e**, **d**, and **n** that satisfy that equation

Ugh. I know **e** and **n** and (even **m**) extremely hard to find **d**!!!

Textbook RSA

- Encryption:

$$C = m^e \pmod{n}$$

The ciphertext is equal to **m** multiplied by itself **e** times modulo **n**.

Public key: $Pub_{Key} = (e, n)$

Textbook RSA

- Decryption:

$$m = C^{d} \bmod n = (m^{e})^{d} \bmod n = m^{ed} \bmod n$$

Decryption relies on number **d** satisfying **e*d** = 1 mod $\boldsymbol{\varphi}$**(n)** ,

s.t. $m^{ed} \bmod n = m^{1} \bmod n = m$

  - In other words, **d** is the multiplicative inverse of **e** mod $\boldsymbol{\varphi}$**(n)**

Private key: $Priv_{Key}$ = d (other numbers can be discarded)

Key Generation (how to choose **e** and find **d**)

- Pick two random primes **p** and **q** such that **p*q** = **n**
- Generate $\varphi(n) = (p-1)(q-1)$
  - We know all relative primes to (p−1)(q−1) form a group with respect to multiplication and are invertible
  - $\varphi(n)$ is the order of the multiplicative group of units modulo n
- Pick **e** as a random prime smaller than $\varphi(n)$
  - **e** chosen as relative prime to (p−1)(q−1) to ensure it has a multiplicative inverse mod (p−1)(q−1)
- Generate **d** (the inverse of e mod $\varphi(n)$)
  - **e*d** = 1 mod $\varphi(n)$
  - Can be obtained via the extended Euclidean algorithm

*If gcd(a,b) = 1, then we say that a and b are **relatively prime** (or coprime).

Textbook RSA (summary)

1. Choose two “**large primes**” *p* and *q* (secretly)
2. Compute n = p*q
3. “Choose” value *e* and find *d* such that
   - $(m^e)^d \equiv m \bmod n$
4. **Public key**: (e, n)
5. **Private key**: d
6. Encryption: $C = m^e \bmod n$
7. Decryption: $m = C^d \bmod n$

- ✓ Note that the decryption works.
- ✓ This is textbook RSA, never do this!! (we’ll see one of the reasons next)

Example (Tiny RSA)

Parameters:

- $p=53$, $q=101$, $n=5353$
- $\varphi(n) = (53-1)(101-1) = 5200$
- $e=139$ (random pick)
- $d=1459$ (extended Euclidean)

- Message:

  m=<u>20</u>

Encryption: $c = m^e \bmod n$

$$C = 20^{139} \bmod 5353 = 5274$$

Decryption: $m = c^d \bmod N$

$$m = 5274^{1459} \bmod 5353 = \underline{20}$$

Example (Tiny RSA)

Parameters:

- p=53, q=101, n=5353
- $\varphi(n) = (53-1)(101-1) = 5200$
- e=139 (random pick)
- d=1459 (extended Euclidean)

- Message:

  m=20

Encryption: $c = m^e \bmod n$

$$C = 20^{139} \bmod 5353 = 5274$$

Decryption: $m = c^d \bmod N$

$$m = 5274^{1459} \bmod 5353 = \underline{20}$$

Applying **e** or **d** to encrypt does not really matter from a functionality perspective

Size of message on textbook RSA

- Overview:

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

m has to be strictly smaller than **n**, otherwise decryption will produce erroneous values.

Size of message on textbook RSA

- Overview:

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

m has to be strictly smaller than **n**, otherwise decryption will produce erroneous values.

Ok! So we can break the message in **chunks**! But perhaps we're better served with **hybrid** schemes… Let's look more into this later…

Attacking RSA (Bad primes)

I know **e** and **n**…
What can I do to find **d**?

Parameters:

- p=53, q=101, n=5353
- φ(n) = (53-1)(101-1) = 5200
- e=139
- d=1459

- c = 5274

Attack idea:

- Factor **n** to obtain **p** and **q**
- Obtain $\boldsymbol{\varphi}$**(n)**
- From $\boldsymbol{\varphi}$**(n)** and **e**, generate **d** just like Alice would

Attacking RSA (Bad primes)

... e ... N...
Wh... find **d**?

WARNING: Factoring is hard, but...

Attack idea:

- Factor **n** to obta... **p** and ...
- Obtain φ(**n**)
- From φ(**n**) and **e**, generate **d** just like Alice would

Parameters:

- p=53, q=101, n=5353
- φ(n) = (53-1).(101-1) = 5200
- e=139
- ...1459
- c ...5274

Factoring and RSA

You want to factor the public modulus?

- Good news, abundant literature on factoring algorithms
- Bad news, “appropriate” primes will not be defeated

Factoring and RSA

You want to factor the public modulus?

- Good news, abundant literature on factoring algorithms
- Bad news, "appropriate" primes will not be defeated

Bad primes: easily factored

Approach at Factoring

Strawman approach:

- Try to divide a number by all numbers smaller than it until you find a number **a** that divides n
- Then, carry on to divide n with **a+1** and so on…
- We end up with a list of factors of n

Way too computationally expensive.

A Smarter Approach at Factoring

- We only need to test prime numbers (not every a < n)

- We only need to test those smaller than $\sqrt{n}$
  - If both p and q are larger than $\sqrt{n}$, then $p*q > n$, which is impossible

A Smarter Approach at Factoring

- We only need to test prime numbers (not every a < n)

- We only need to test those smaller than $\sqrt{n}$
  - If both p and q are larger than n, then p*q > n, which is impossible

Still too computationally expensive for large n.

n = 4096 bits requires about 2^{128} operations

AMD's EPYC or Intel's Xeon series, 3 teraflops/sec
≈ 13.8 billion years

Attacking "bad primes"

- Some primes are not suited to be used for RSA, as they make n easier to factor

- Examples:
  - Either **p** or **q** are small numbers
  - **p** and **q** are too close together
  - **p** and **q** are both close to 2^b, where b is a given bound
  - $n = \mathbf{p}^r\mathbf{q}^s$ and $r > \log p$
  - …

Don't build your own RSA implementation!

So far so good, but…

WARNING: this was textbook RSA, do **not** use!!!

Why not "Textbook RSA"?

Encryption: $c \equiv m^e \pmod{n}$, **Decryption:** $m = c^d \pmod{n}$

- Compute $C_1 = Enc_e(m_1)$.
- Compute $C_2 = Enc_e(m_2)$.
- Compute $m = Dec_d(C_1 * C_2)$. What is happening? Why?

A: The decryption would yield the product of the original plaintexts.

$$(m_1)^e * (m_2)^e \equiv (m_1 * m_2)^e$$

Malleability: it is possible to transform a ciphertext into another ciphertext that decrypts to a transformation of the original plaintext.

This is typically (but not always!) undesirable.

Attacking RSA (CCA)

Chosen Ciphertext Attack (CCA)

- We are Eve. Alice is using RSA and her public key is (e, n). 
- Bob sends secret message m, encrypted as $c = Enc_e(m)$. 
- We intercept c. 
- Alice is convinced her textbook RSA is very secure, so she is willing to decrypt any ciphertext we send her (except for c).

Attacking RSA (CCA)

Chosen Ciphertext Attack (CCA)

- We are Eve. Alice is using RSA and her public key is (e, n). 
- Bob sends secret message m, encrypted as c = $Enc_e(m)$. 
- We intercept c. 
- Alice is convinced her textbook RSA is very secure, so she is willing to decrypt any ciphertext we send her (except for c).

Goal: Ask Alice to decrypt something (other than c) that helps us guess m

Attacking RSA (CCA)

Chosen Ciphertext Attack (CCA): Solution

- Alice's public key is (e, n). 
- Bob sends $c_1 = Enc_e(m)$. We intercept c_1. 

Q: Ask Alice to decrypt, e.g., $c_2 = 2^e \cdot c_1$.

Attacking RSA (CCA)

Chosen Ciphertext Attack (CCA): Solution

- Alice's public key is (e, n). 
- Bob sends $c_1 = Enc_e(m)$. We intercept c_1. 

Q: Ask Alice to decrypt, e.g., $c_2 = 2^e \cdot c_1$.

A: This decryption yields $(2^e \cdot c_1)^d \equiv 2m$.
We divide the result by 2, and we get m.

Example: given m=5, e=3, and n=33 → $c_1 = 26$, $c_2 = 208$ → $m_2 = 10$

Attacking RSA (CCA)

Chosen Ciphertext Attack (CCA): Solution

- Alice's public key is (e, n). 
- Bob sends $c_1 = Enc_e(m)$. We intercept c_1. 

Q: Ask Alice to decrypt, e.g., $c_2 = 2^e \cdot c_1$.

A: This decryption yields $(2^e \cdot c_1)^d \equiv 2m$.
We divide the result by 2, and we get m.

- ✓ Textbook RSA is vulnerable against chosen ciphertext attacks (among other things)
- ✓ We can fix this with padding techniques (RSA-OAEP).

Textbook RSA is not IND-CPA Secure

IND-CPA: **Ind**istinguishability under **C**hosen-**P**laintext **A**ttack

1. Attacker chooses two plaintexts and gives them to challenger
2. Challenger encrypts one of the plaintexts and gives ciphertext to attacker
3. Attacker needs to guess which plaintext was encrypted
4. Attacker guesses successfully with probability > 0.5

Textbook RSA is not IND-CPA Secure

1. Eve produces two plaintexts, m_0 and m_1 

Textbook RSA is not IND-CPA Secure

1. Eve produces two plaintexts, m_0 and m_1 

2. “Challenger” encrypts an m as $c^* = m_b^e \pmod{n}$, secret b 

Textbook RSA is not IND-CPA Secure

1. Eve produces two plaintexts, m_0 and m_1 
2. “Challenger” encrypts an m as $c^* = m_b^e \pmod n$, secret b 
3. Eve’s goal? Determine $b \in \{0,1\}$

Textbook RSA is not IND-CPA Secure

1. Eve produces two plaintexts, m_0 and m_1 

2. “Challenger” encrypts an m as $c^* = m_b^e \pmod n$, secret b 

3. Eve’s goal? Determine $b \in \{0,1\}$

4. Sooo, Eve computes $c = m_1^e \pmod n$

   If $c^* = c$ then Eve knows $m_b = m_1$
   If $c^* \neq c$ then Eve knows $m_b = m_0$

Textbook RSA is not IND-CPA Secure

1. Eve produces two plaintexts, m_0 and m_1 

2. "Challenger" encrypts an m as $c^* = m_b^e \pmod{n}$, secret b 

3. Eve's goal? Determine $b \in \{0,1\}$
4. Sooo, Eve computes $c = m_1^e \pmod{n}$

   If $c^* = c$ then Eve knows $m_b = m_1$
   If $c^* \neq c$ then Eve knows $m_b = m_0$

Adversaries and their Goals

Adversaries and their Goals

You’ve assumed my goal is the secret/private key…

…but less ambitious goals can be very effective…

Adversaries and their Goals

Goal 1: Total Break

- Win the Symmetric key K
- Win Bob's private key k_b
- Can decrypt any c_i *for:*

$$c_i = Enc_K(m)$$

or

$$c_i = Enc_{kb}(m)$$

- All messages using compromised k revealed
- Unless **detected** game over

Goal 2: Partial Break

- Decrypt **a ciphertext c** (without the key)
- Learn **some** specific information about a message *m* from c

Needs to occur with non-negligible probability.

- **Some (or a)** message revealed

Goal 3: Distinguishable Ciphertexts

- *Pr {learn b ∈ {0,1}}* exceeds ½
- Distinguish between Enc(m_1) and Enc(m_2)

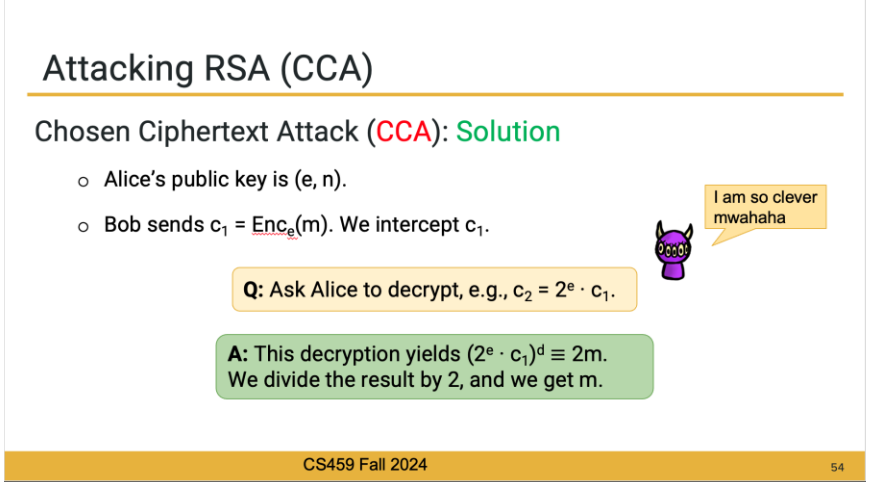
  or

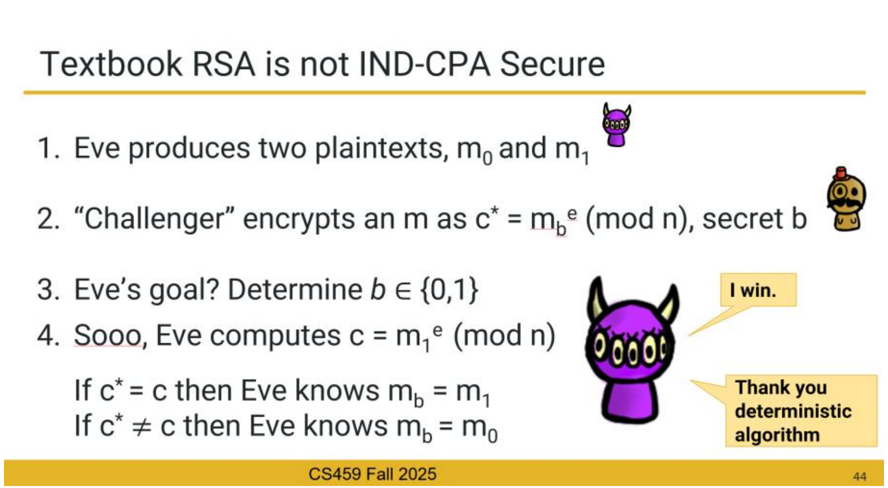
  between Enc(m) and Enc(random string)

- The ciphertexts are leaking small/some information…

Semantic Security of RSA

- We saw CCA against Textbook RSA
- We showed IND-CPA on Textbook RSA





Fix it? Remove Ciphertext Distinguishability

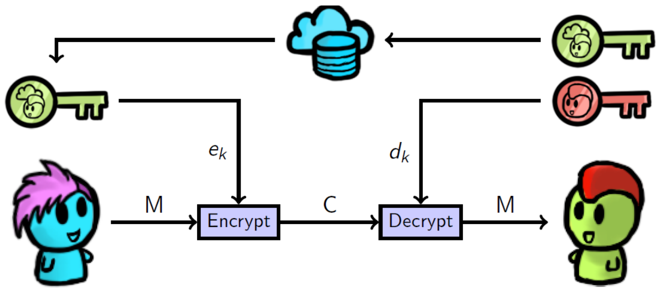
Goal: prove (given comp. assumptions) that no information regarding *m* is revealed in polynomial time by examining $c = Enc(m)$

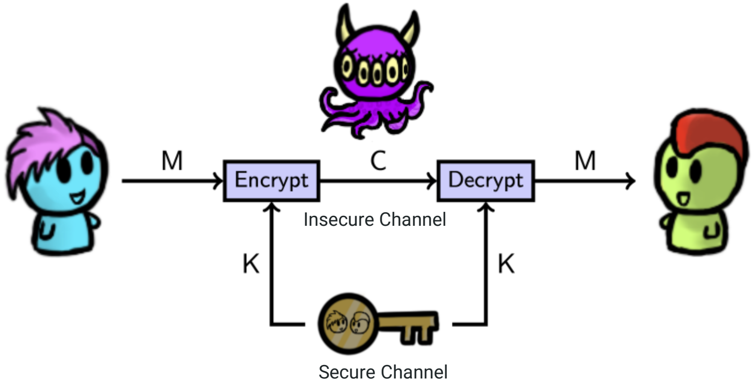
- If Enc( ) is deterministic, fail
- Thus, require some randomization

RSA-OAEP: Optimal Asymmetric Encryption Padding

- Padding contains randomness, use RSA-OAEP in practice

Practicality of Public-Key vs. Symmetric-Key





1. Longer keys
2. Slower
3. Different keys for Enc(m) and Dec(c)

1. Shorter keys
2. Faster
3. Same key for Enc(m) and Dec(c)

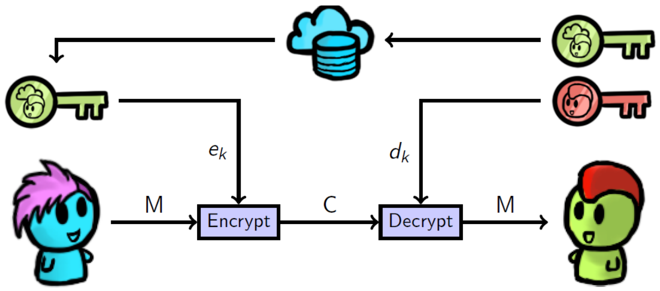
Public-Key Sizes

- Recall that if there are no shortcuts, Eve would have to try 2^{128} iterations in order to read a message encrypted with a 128-bit key

- Unfortunately, all of the public-key methods we know do have shortcuts
  - Eve could read a message encrypted with a 128-bit RSA key with just 2^{33} work, which is easy!
  - Comparison of key sizes for roughly equal strength

AES	RSA	ECC
80	1024	160
116	2048	232
128	2600	256
160	4500	320
256	14000	512

Practicality of Public-Key vs. Symmetric-Key



1. Longer keys
2. Slower
3. Different keys for Enc(m) and Dec(c)

1. Shorter keys
2. Faster
3. Same key for Enc(m) and Dec(c)

What can be done? (Hybrid Cryptography)

We can get the best of both worlds:

- Pick a random "128-bit" key K for a symmetric-key cryptosystem
- Encrypt the large message with the key K (e.g., using AES)

And then…

- Encrypt the key K using a public-key cryptosystem
- Send the encrypted message and the encrypted key to Bob

Hybrid cryptography is used in (many) applications on the internet today

Knowledge Check!

Public: (e_A, d_A)

Secret: K

Public: (e_B, d_B)

Secret: ?

- Enc/Dec functions: $Enc_{key}(*)$, $Dec_{key}(*)$
- Alice wants to send a **large** message *m* to Bob.

Q: How should Alice build the message efficiently? How does Bob recover m?

Knowledge Check!

Public: (e_A, d_A)

Secret: K

Public: (e_B, d_B)

Secret: ?

- Enc/Dec functions: $Enc_{key}(*)$, $Dec_{key}(*)$
- Alice wants to send a **large** message *m* to Bob.

Q: How should Alice build the message efficiently? How does Bob recover m?

FYI: PKE is slow!! We don't want to use it on m.

Knowledge Check!

Public: (e_A, d_A)

Secret: K

Public: (e_B, d_B)

Secret: ?

- Enc/Dec functions: $Enc_{key}(*)$, $Dec_{key}(*)$
- Alice wants to send a **large** message *m* to Bob.

Q: How should Alice build the message efficiently? How does Bob recover m?

A: Alice computes $c_1 = Enc_{eB}(K)$, $c_2 = E_K(m)$ and sends $<c_1||c_2>$.
Bob recovers $K = Dec_{dB}(c_1)$ and then $m = Dec_K(c_2)$

|| denotes concatenation

Knowledge Check!

We know how to "send secret messages", and Eve cannot do anything about it. What else is there to do?

- Mallory can modify our encrypted messages in transit!
- Mallory won't necessarily know what the message says, but can still change it in an undetectable way
  - e.g. bit-flipping attack on stream ciphers
- This is counterintuitive, and often forgotten

Q: How do we make sure that Bob gets the same message Alice sent?

Up next: More Cryptography…