

CS459/659

Privacy, Cryptography, Network and Data Security

Public Key Cryptography (RSA)

Cryptography Organization

Symmetric

Ciphers

Hash
Functions

Message
Auth. codes

PRFs

C

Stream

Block

M-of-Op

Improvements

Asymmetric

PKE

Digital
Signatures

Key
Exchange

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 ?

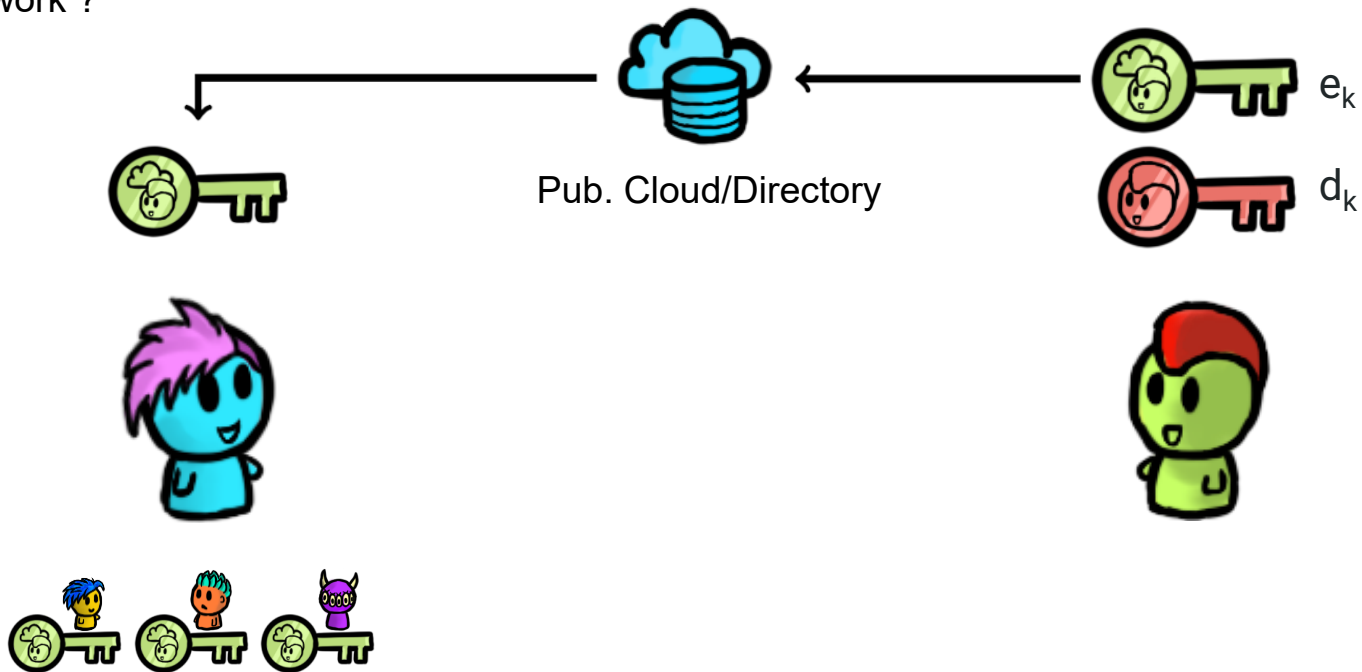


key pair (e_k, d_k)



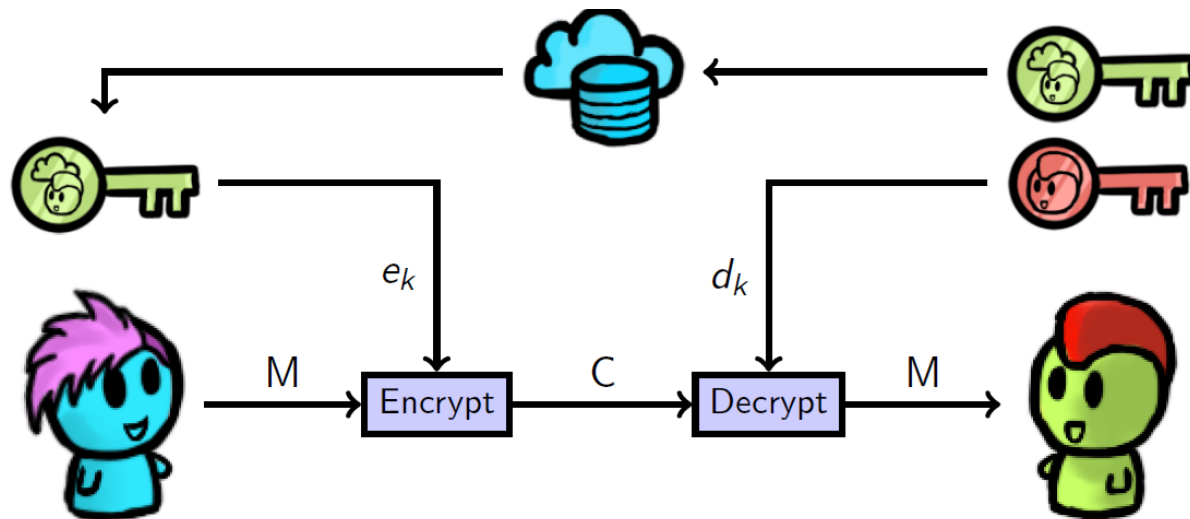
Public-key Encryption

How does it work ?



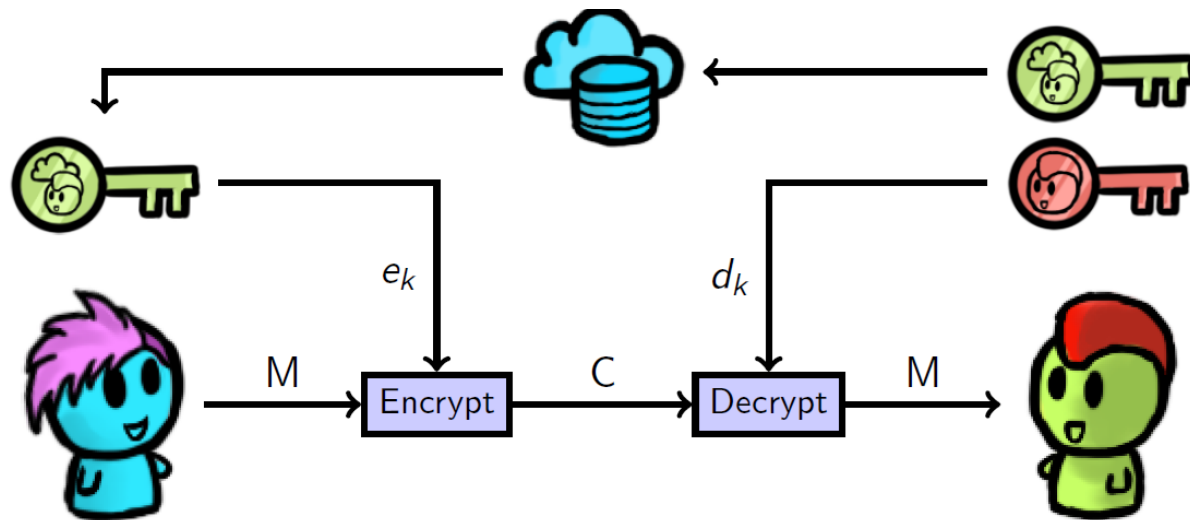
Public-key Encryption

How does it work ?



Public-key Encryption

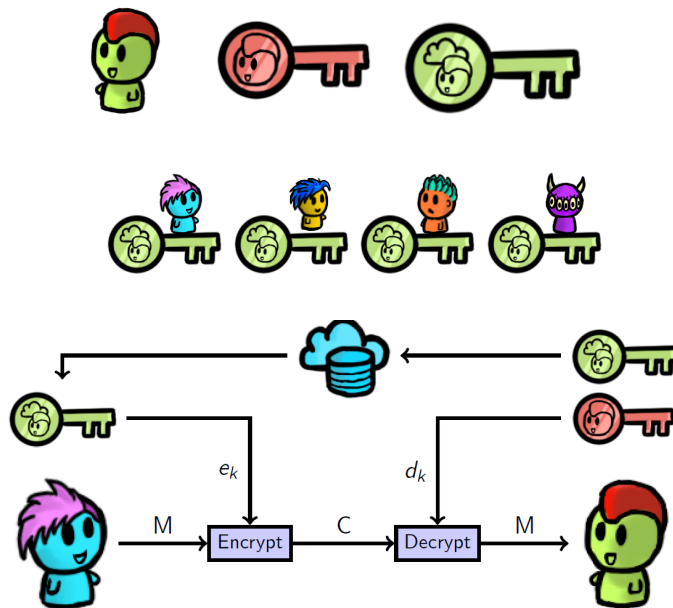
How does it work ?



- ✓ 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

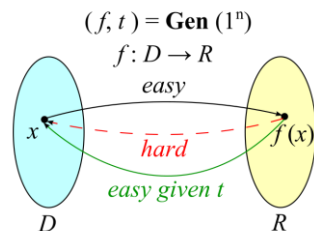


vs.



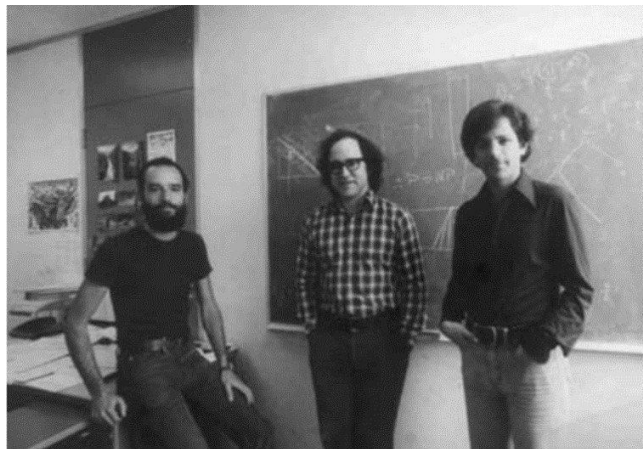
Thus, we require so called “one-way” functions for this.

**But because of decryption,
we need a “Trapdoor”**



Textbook RSA

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



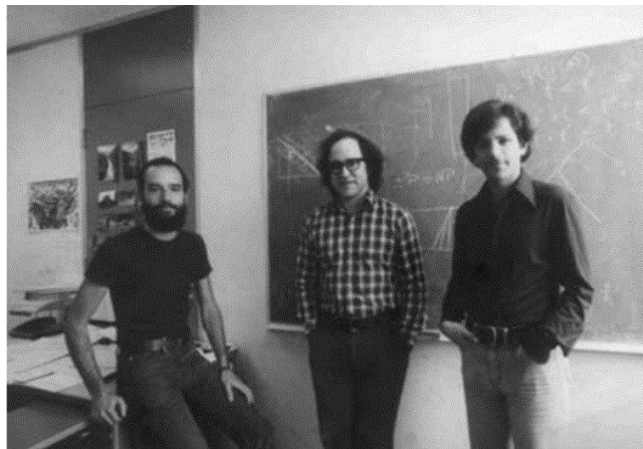
Left to right: Ron Rivest, Adi Shamir, and Leonard Adleman.

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:

$$7 \bmod 5 = 2$$

$$12 \bmod 5 = 2$$

$$7 \equiv 12 \bmod 5$$

(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: $\text{Pub}_{\text{Key}} = (e, n)$

Textbook RSA

- Decryption:

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

Decryption relies on number d satisfying $e * d = 1 \bmod \varphi(n)$,
s.t. $m^{ed} \bmod n = m^1 \bmod n = m$

- In other words, d is the multiplicative inverse of $e \bmod \varphi(n)$ *

Private key: $\text{Priv}_{\text{Key}} = d$ (other numbers can be discarded)

* $\varphi(n) = (p-1)(q-1)$, where p and q are random primes

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

- Pick two random primes **p** and **q** such that $p \cdot q = n$
- 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 $\varphi(n)$)
 - $\mathbf{e} \cdot \mathbf{d} = 1 \bmod \varphi(n)$
 - 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 $n = p \cdot q$
3. “Choose” value e and find d such that
 - $(m^e)^d \equiv m \pmod{n}$
4. **Public key:** (e, n)
5. **Private key:** d
6. Encryption: $C = m^e \pmod{n}$
7. Decryption: $m = C^d \pmod{n}$

- ✓ 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:
 $m=\underline{20}$

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

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

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

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



Nice!

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:
 $m=\underline{20}$

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:

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



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

Size of message on textbook RSA

- Overview:

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



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**?

Attack idea:

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

Parameters:

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

- $c = 5274$

Attacking RSA (Bad primes)



Why can't we find d ?

WARNING: Factoring is hard, but...

Attack idea:

- Factor n to obtain p and q
- Obtain $\phi(n)$
- From $\phi(n)$ and e , generate d just like Alice would

Parameters:

- $p=53, q=101, n=5353$
- $\phi(n) = (53-1).(101-1) = 5200$
- $e=139$
- $d=1459$
- $c=5274$

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

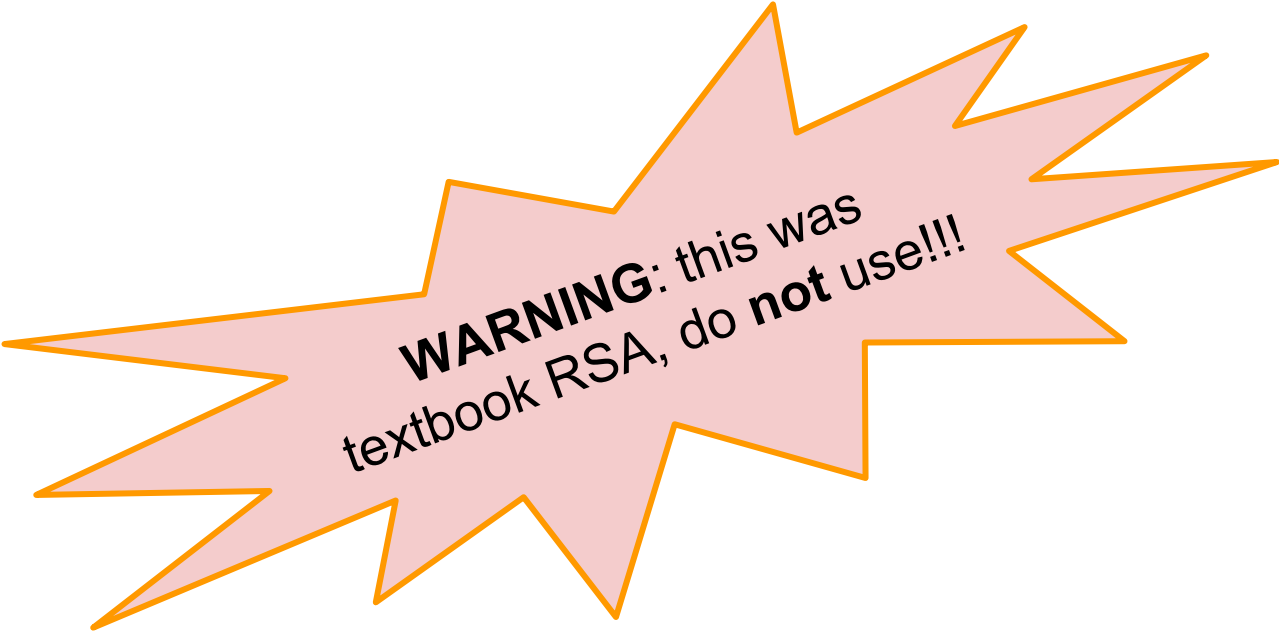


Attacking “bad primes”

- 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
 - $n = p^r q^s$ 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 $C_1 = \text{Enc}_e(m_1)$.
- Compute $C_2 = \text{Enc}_e(m_2)$.
- Compute $m = \text{Dec}_d(C_1 * C_2)$. 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.

Adversaries and their Goals



**You've assumed
my goal is the
secret/private
key...**

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



**You've assumed
my goal is the
secret/private
key...**



**...but less ambitious
goals can be very
effective...**



We better figure this out.

Yup.



Goal 1: Total Break



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

$$c_i = \text{Enc}_K(m)$$

or

$$c_i = \text{Enc}_{k_b}(m)$$



- 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



- $\Pr \{ \text{learn } b \in \{0,1\} \}$ exceeds $\frac{1}{2}$
- Distinguish between $\text{Enc}(m_1)$ and $\text{Enc}(m_2)$
or
between $\text{Enc}(m)$ and $\text{Enc}(\text{random string})$



- The ciphertexts are leaking small/some information...



First Step

- In order to prove security, one needs a definition what secure „is“
- In cryptography one defines the weakest possible attack and shows that it is infeasible

What is an attack?

- An attack is a game („interface“) between an attacked system and an attacker
- The attacker is given certain „functions“, which the attacker may (almost) arbitrarily call
- The attacker tries to solve some given „puzzle“

Attacks in Public-Key Cryptography

- Chosen Plaintext Attack (CPA):
 - The attacker has the public key and may encrypt arbitrary plaintexts
- Chosen Ciphertext Attack (CCA):
 - The attacker may in addition decrypt certain ciphertexts (using the interface – without the private key)

Interface

1. Phase:

(IND-CPA)
(IND-CCA1)

getPubKey()
decrypt(c)

Challenge

$c_f = \text{getCipherText}(x_1, x_2)$
random plaintext index $f \in \{1, 2\}$

2. Phase

(IND-CCA2)

decrypt($c \neq c_f$)

Output of the attacker

f'

Successful Attack

Advantage: $\text{Adv} = \Pr[f' = f] - \frac{1}{2}$

Success: $\text{Adv} \geq 1/\text{poly}(k)$

poly arbitrary positive polynomial

k key length

Non-negligible function



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 $c = \text{Enc}_e(m)$. 
- 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) . A blue character representing Alice and a key icon.
- Bob sends secret message m , encrypted as $c = \text{Enc}_e(m)$. A green character representing Bob and a scroll with a lock icon.
- We intercept c . A purple devil character representing Eve and a scroll with a lock icon.
- 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 $c_1 = \text{Enc}_e(m)$. We intercept c_1 .



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



I am so clever
mwahaha

Attacking RSA (CCA)

Chosen Ciphertext Attack (CCA): Solution

- Alice's public key is (e, n) .  

- Bob sends $c_1 = \text{Enc}_e(m)$. 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 .

I am so clever
mwahaha



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

Attacking RSA (CCA)

Chosen Ciphertext Attack (CCA): Solution

- Alice's public key is (e, n) .  

- Bob sends $c_1 = \text{Enc}_e(m)$. 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 .

I am so clever
mwahaha



- ✓ Textbook RSA is **vulnerable** against chosen ciphertext attacks (among other things)
- ✓ We can fix this with **padding techniques** (RSA-OAEP), Alice must ensure that padding of decrypted ciphertext is still valid and should not return plaintext otherwise

Textbook RSA is not IND-CPA Secure

IND-CPA: **I**ndistinguishability 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 $m_b = m_0$



I win.

Thank you
deterministic
algorithm

Semantic Security of RSA

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

Attacking RSA (CCA)

Chosen Ciphertext Attack (CCA): Solution

- Alice's public key is (e, n) .
- Bob sends $c_1 = \text{Enc}_e(m)$. 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 .



I am so clever
mwahaha

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 $m_b = m_0$



I win.

Thank you
deterministic
algorithm

Fix it? Remove Ciphertext Distinguishability

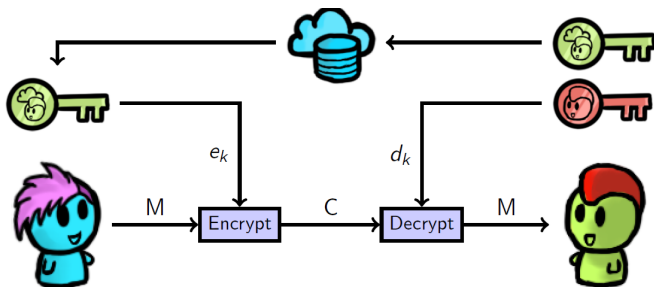
Goal: prove (given comp. assumptions) that no information regarding m is revealed in polynomial time by examining $c = \text{Enc}(m)$

- If $\text{Enc}()$ is deterministic, fail
- Thus, require some randomization

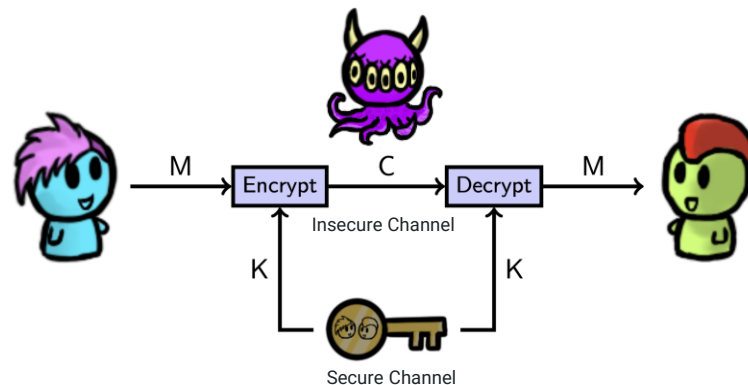
RSA-OAEP: Optimal Asymmetric Encryption Padding

- Padding contains randomness, use RSA-OAEP in practice instead of textbook RSA

Practicality of Public-Key vs. Symmetric-Key



1. Longer keys
2. Slower
3. Different keys for $\text{Enc}(m)$ and $\text{Dec}(c)$



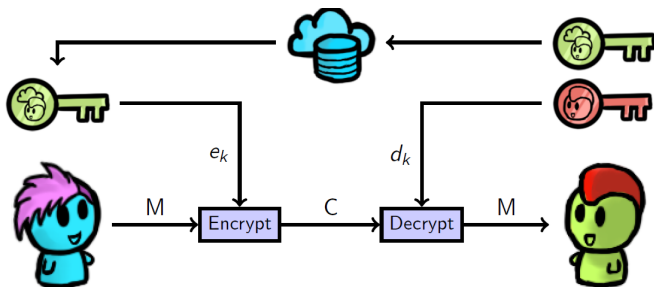
1. Shorter keys
2. Faster
3. Same key for $\text{Enc}(m)$ and $\text{Dec}(c)$

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

<u>AES</u>	<u>RSA</u>	<u>ECC</u>
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 $\text{Enc}(m)$ and $\text{Dec}(c)$



1. Shorter keys
2. Faster
3. Same key for $\text{Enc}(m)$ and $\text{Dec}(c)$

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: $\text{Enc}_{\text{key}}(*), \text{Dec}_{\text{key}}(*)$
- 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: $\text{Enc}_{\text{key}}(*), \text{Dec}_{\text{key}}(*)$
- 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: $\text{Enc}_{\text{key}}(*), \text{Dec}_{\text{key}}(*)$
- 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 $c_1 = \text{Enc}_{e_B}(K)$, $c_2 = E_K(m)$ and sends $\langle c_1 || c_2 \rangle$.
Bob recovers $K = \text{Dec}_{d_B}(c_1)$ and then $m = \text{Dec}_K(c_2)$

$||$ 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...

Symmetric

Ciphers

Hash
Functions

Message
Auth. codes

PRFs

Stream

Block

Asymmetric

PKE

Digital
Signatures

Key
Exchange

RSA

IND-CCA security types