Lecture 6: Public Key Encryption, Homomorphic Encryption

Ivan Oleynikov

Chalmers University

November 18, 2022

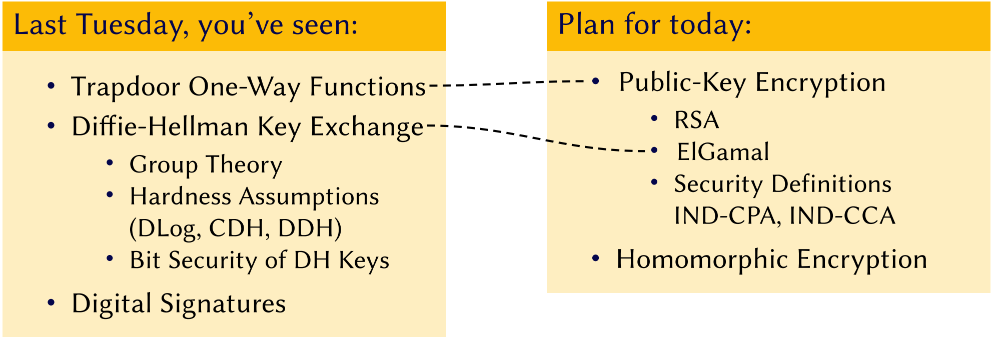
These slides were updated after the lecture. We fixed some typos and added a few new slides (marked with † in the title).
Recommended reading for this lecture:

- Boneh & Shoup, A Graduage Couse in Applied Cryptography:
  sections 11.2, 11.3, 1.5, 12.1, 12.2, 12.6.2.
- Golwasser & Bellare, Lecture Notes on Cryptography:
  sections 7.1, 7.2.2, 7.2.3.

Announcements

- Today is the deadline for assignment Bonus-1.
- Last Wednesday we fixed an error in HA2.
  Make sure the PDF you're reading is up to date.

Agenda



Last Tuesday, you've seen:

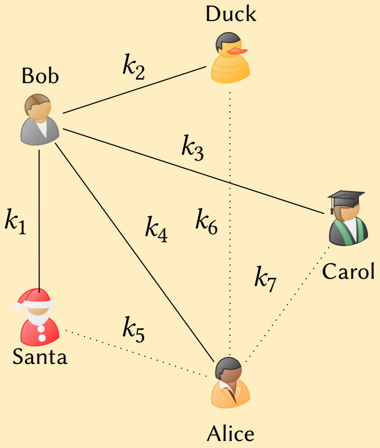
- Trapdoor One-Way Functions
- Diffie-Hellman Key Exchange
  - Group Theory
  - Hardness Assumptions (DLog, CDH, DDH)
  - Bit Security of DH Keys
- Digital Signatures

Plan for today:

- Public-Key Encryption
  - RSA
  - ElGamal
  - Security Definitions IND-CPA, IND-CCA
- Homomorphic Encryption

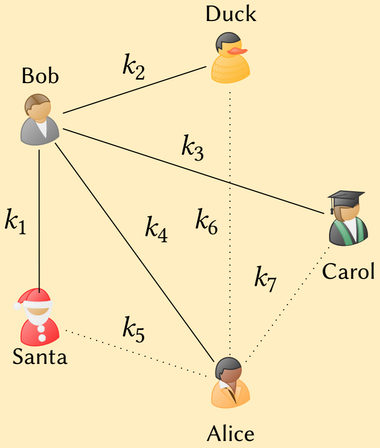
Symmetric Key Distribution Issue

How many keys do we need so everyone can talk to each other?



Symmetric Key Distribution Issue

How many keys do we need so everyone can talk to each other?



- 4 keys for Bob + 3 keys for Alice + 2 keys for ...
- $\frac{n(n-1)}{2}$ keys for n parties
- Public-Key Encryption, on the other hand, solves this with only n keys!

Public-Key Encryption

Public-Key Encryption

A Public-Key Encryption is $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$, three probabilistic poly-time algorithms (PPT) s.t.

- $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KeyGen}(1^n)$ is key generation, produces secret key and public key
- $c \leftarrow \mathrm{Enc}(\mathrm{pk}, m)$ is encryption
- $m' = \mathrm{Dec}(\mathrm{sk}, c)$ is decryption

Notation: $m \in \mathscr{M}$, $c \in \mathscr{C}$.
Correctness property: $m = m'$.

Public-Key Encryption

Public-Key Encryption

A Public-Key Encryption is $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$, three probabilistic poly-time algorithms (PPT) s.t.

- $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KeyGen}(1^n)$ is key generation, produces secret key and public key
- $c \leftarrow \mathrm{Enc}(\mathrm{pk}, m)$ is encryption
- $m' = \mathrm{Dec}(\mathrm{sk}, c)$ is decryption

Notation: $m \in \mathscr{M}$, $c \in \mathscr{C}$.
Correctness property: $m = m'$.

Trapdoor One-Way Function

Three algorithms[a] $(\mathrm{KeyGen}, \mathrm{F}, \mathrm{I})$ s.t.

- $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KeyGen}(1^n)$
- $y = \mathrm{F}(\mathrm{pk}, x)$
- $x' = \mathrm{I}(\mathrm{sk}, y)$

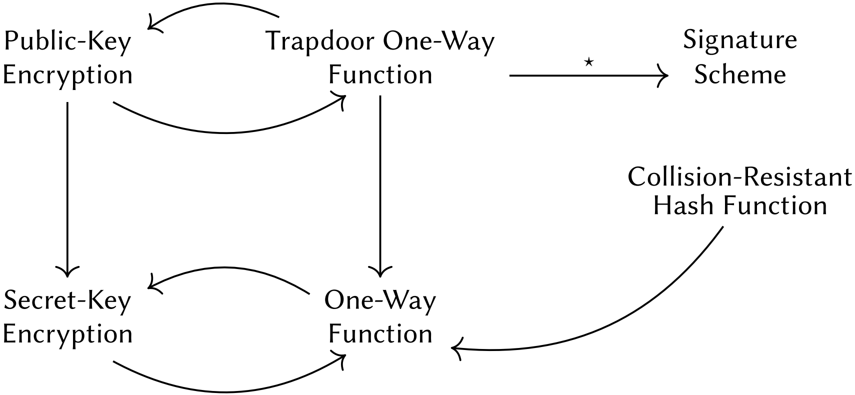
It's guarranteed that $\mathrm{F}(\mathrm{pk}, x') = y$, but not guarranteed that $x = x'$.

What's the difference?

[a] KeyGen is probabilistic, other two are deterministic

Why should I care about Trapdoor One-Way Functions?

Simpler than other constructions. Help cryptographers analyze cryptographic assumptions.



Public-Key Encryption

Trapdoor One-Way Function

Signature Scheme

Collision-Resistant Hash Function

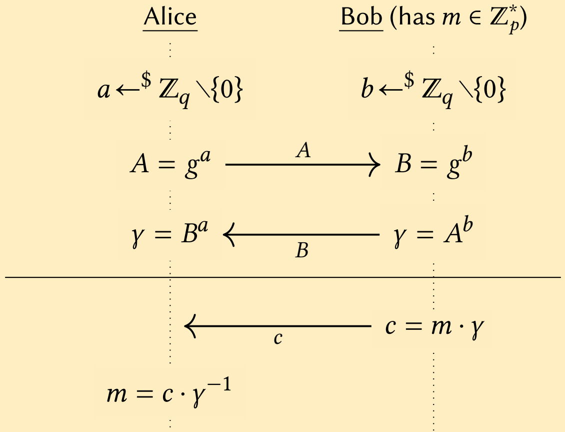
Secret-Key Encryption

One-Way Function

⋆ true with some caveats; Elena will present this next week

Recap: Encrypting a Message with Diffie-Hellman

Establish a key + transmit a secret message



Alice

Bob (has $m \in \mathbb{Z}_p^*$)

$a \overset{\$}{\leftarrow} \mathbb{Z}_q \setminus \{0\}$

$b \overset{\$}{\leftarrow} \mathbb{Z}_q \setminus \{0\}$

$$A = g^a \xrightarrow{A} B = g^b$$

$$\gamma = B^a \xleftarrow[B]{} \gamma = A^b$$

$$\xleftarrow[c]{} c = m \cdot \gamma$$

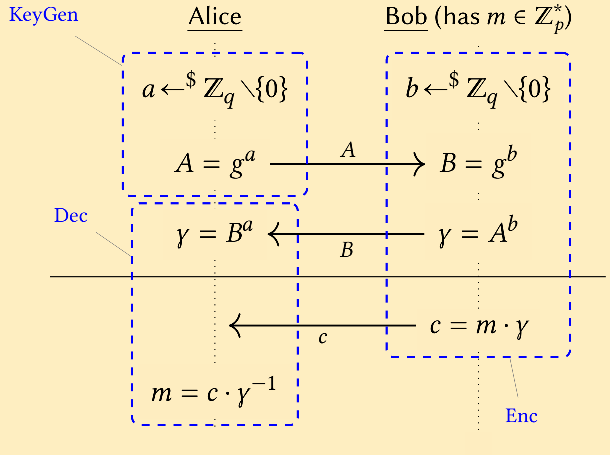
$$m = c \cdot \gamma^{-1}$$

Parameters

- p is a large prime
- g is an element of $\mathbb{Z}_p^*$ which generates a cyclic (multiplicative) subgroup of order q

Recap: Encrypting a Message with Diffie-Hellman

Establish a key + transmit a secret message



KeyGen

Alice

Bob (has $m \in \mathbb{Z}_p^*$)

$a \xleftarrow{\$} \mathbb{Z}_q \setminus \{0\}$

$b \xleftarrow{\$} \mathbb{Z}_q \setminus \{0\}$

$A = g^a \xrightarrow{A} B = g^b$

Dec

$\gamma = B^a \xleftarrow{B} \gamma = A^b$

$\xleftarrow{c} c = m \cdot \gamma$

$m = c \cdot \gamma^{-1}$

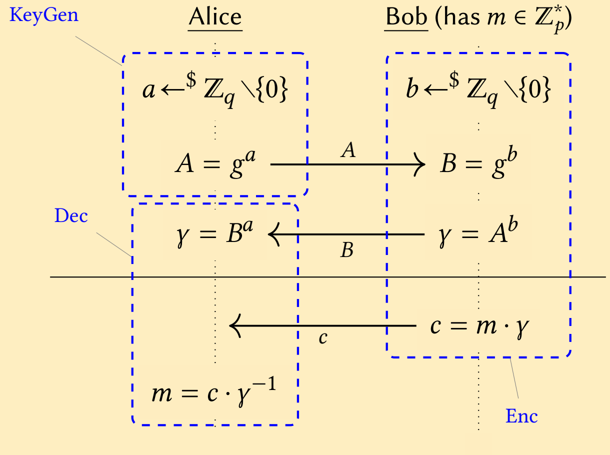
Enc

Parameters

- p is a large prime
- g is an element of $\mathbb{Z}_p^*$ which generates a cyclic (multiplicative) subgroup of order q

Recap: Encrypting a Message with Diffie-Hellman

Establish a key + transmit a secret message



KeyGen

Alice

Bob (has $m \in \mathbb{Z}_p^*$)

$a \xleftarrow{\$} \mathbb{Z}_q \setminus \{0\}$

$b \xleftarrow{\$} \mathbb{Z}_q \setminus \{0\}$

$A = g^a \xrightarrow{A} B = g^b$

Dec

$\gamma = B^a \xleftarrow[B]{} \gamma = A^b$

$\xleftarrow[c]{} c = m \cdot \gamma$

$m = c \cdot \gamma^{-1}$

Enc

Parameters

- p is a large prime
- g is an element of $\mathbb{Z}_p^*$ which generates a cyclic (multiplicative) subgroup of order q

In ElGamal terms…

- a is Alice's private key
- A is Alice's public key
- (B, c) is the ciphertext that encrypts Bob's message m for Alice

ElGamal, More Formally

KeyGen(1^n)

```
Pick $p$, $g$ and $q$ so that $|q| \geq n$
$a \xleftarrow{\$} \mathbb{Z}_q \setminus \{0\}$    // secret key
$A = g^a$    // public key
return $(A, a)$
```

Dec$_a$((B, c))

```
$\gamma = B^a$
return $c \cdot \gamma^{-1}$
```

Enc$_A$(m)

```
$b \xleftarrow{\$} \mathbb{Z}_q \setminus \{0\}$
$B = g^b$
$\gamma = A^b$
$c = \gamma \cdot m$
return $(B, c)$
```

Textbook RSA

$\mathrm{KeyGen}(1^n)$

```
1 Pick $n$-bit long primes $p$ and $q$
2 $N = pq$
3 Pick $e$ so that $\gcd(e, \Phi(N)) = 1$
4 $d = e^{-1} \mod \Phi(N)$
5 $\mathrm{pk} = (N, e)$, $\mathrm{sk} = (N, d)$
6 return $(\mathrm{pk}, \mathrm{sk})$
```

$\mathrm{Enc}_{(N,e)}(m)$

```
1 $c = m^e \mod N$
2 return $c$
```

$\mathrm{Dec}_{(N,d)}(c)$

```
1 $m = c^d \mod N$
2 return $m$
```

Textbook RSA

KeyGen(1^n)

1. Pick n-bit long primes p and q
2. $N = pq$
3. Pick e so that $\gcd(e, \Phi(N)) = 1$
4. $d = e^{-1} \mod \Phi(N)$
5. $\mathrm{pk} = (N, e)$, $\mathrm{sk} = (N, d)$
6. return $(\mathrm{pk}, \mathrm{sk})$

$\mathrm{Dec}_{(N,d)}(c)$

1. $m = c^d \mod N$
2. return m

$\mathrm{Enc}_{(N,e)}(m)$

1. $c = m^e \mod N$
2. return c

Correctness

$$\begin{aligned} c^d \mod N &= (m^e)^d \mod N \\ &= m \mod N \end{aligned}$$

Number Theory for RSA and ElGamal[†]

Euler's Theorem

Let $M > 0$ and $x \in \mathbb{Z}_M^*$, then $x^{\Phi(M)} = 1 \mod M$.
Corollary: $x^{\Phi(M)-1} = x^{-1} \mod M$.

Euler's totient function Φ

- $\Phi(M)$ is the number of values in $\mathbb{Z}_M$ which are co-prime with M. In other words, $\Phi(M) = |\mathbb{Z}_M^*|$.
- If we know that M is a product of distinct primes, $M = p_1 p_2 \dots p_k$, we can compute $\Phi(M) = (p_1 - 1)(p_2 - 1) \dots (p_k - 1)$.
- For example, $\Phi(p) = p - 1$ and $\Phi(pq) = (p - 1)(q - 1)$ when p and q are primes.

Bézout's identity

Let x and y be positive integers, and $d = \gcd(x, y)$. Then there exist integers (possibly negative) a and b such that $xa + yb = d$.

Extended Euclidean Algorithm

The coefficients a and b can be computed by given x and y using Extended Euclidean Algorithm[a].

[a] See the lecture notes, as well as `https://en.wikipedia.org/wiki/Extended_Euclidean_algorithm`

Finding multiplicative inverses modulo[†]

Suppose we are given M and $x \in \mathbb{Z}_M^*$. We want to find x^{-1} s.t. $x^{-1} \cdot x = 1 \mod M$. This is needed, for example, to find d in RSA, and also in ElGamal to invert γ. How do we do this?

When we can factorize M

- Works in ElGamal, where $M = p$ is prime.
- Use Euler's Theorem:

$$\gamma^{-1} = \gamma^{\Phi(p)-1} = \gamma^{p-2} \mod p.$$

- Together with Binary Exponentiation algorithm.

```
# Returns (x ** power) % m
def exp(x, power, m):
  if power == 0:
    return 1
  else if power % 2 == 0:
    return exp(x, power // 2, m) ** 2 % m
  else
    return x * exp(x, power - 1, m) % m
```

When we can't factorize M

- We can't apply the Euler's Theorem trick to RSA: we have $x = e$ and $M = \Phi(N) = (p-1)(q-1)$. Computing $\Phi((p-1)(q-1))$ requires factorizing $(p-1)(q-1)$ and we don't know how to do that.
- In this case we will have to use Bézout's identity, and Extended Euclidean Algorithm...

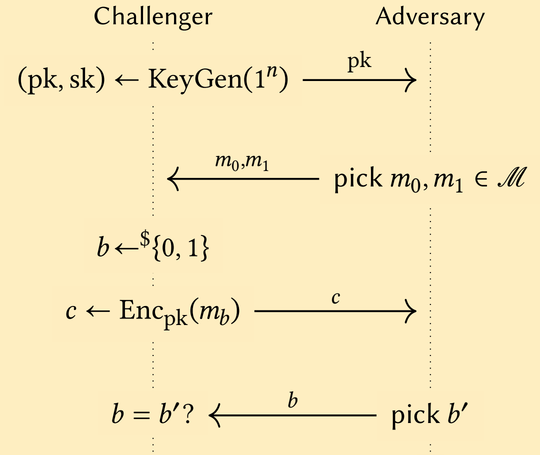
Back to Bézout...

We know that $\gcd(x, M) = 1$. Bézout identity tells us that there exist a and b s.t. $xa + Mb = 1$. Then $xa + Mb \mod M = xa \mod M = 1 \mod M$. So $a = x^{-1} \mod M$.

- Use Extended Euclidean Algorithm to find a and b, return $a = x^{-1}$.

Security Notions: IND-CPA

IND-CPA Game



Challenger

Adversary

$(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KeyGen}(1^n) \xrightarrow{\mathrm{pk}}$

$\xleftarrow{m_0, m_1}$ pick $m_0, m_1 \in \mathscr{M}$

$b \xleftarrow{\$} \{0, 1\}$

$c \leftarrow \mathrm{Enc}_{\mathrm{pk}}(m_b) \xrightarrow{c}$

$b = b'? \xleftarrow{b}$ pick b'

Adversary wins if $b = b'$.

Definition

(KeyGen, Enc, Dec) is Indistinguishable under Chosen-Plaintext Attack (IND-CPA) if

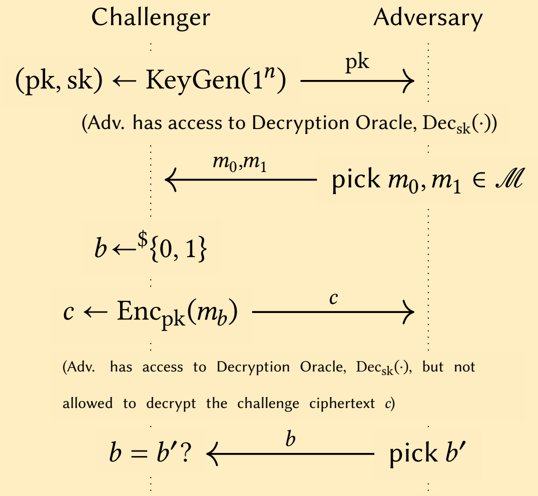
$$\Pr[b = b'] = 1/2 + \mathrm{negl}(n).$$

Remarks

- Since Adversary knows the pk, it can encrypt any messages of its choice at any time.
- Deterministic encryption can't be IND-CPA secure. Why?

Security Notions: IND-CCA

IND-CCA Game



Challenger | Adversary

$(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KeyGen}(1^n) \xrightarrow{\mathrm{pk}}$

(Adv. has access to Decryption Oracle, $\mathrm{Dec}_{\mathrm{sk}}(\cdot)$)

$\xleftarrow{m_0, m_1}$ pick $m_0, m_1 \in \mathscr{M}$

$b \xleftarrow{\$} \{0, 1\}$

$c \leftarrow \mathrm{Enc}_{\mathrm{pk}}(m_b) \xrightarrow{c}$

(Adv. has access to Decryption Oracle, $\mathrm{Dec}_{\mathrm{sk}}(\cdot)$, but not allowed to decrypt the challenge ciphertext c)

$b = b'? \xleftarrow{b}$ pick b'

Adversary wins if $b = b'$.

Definition

(KeyGen, Enc, Dec) is Indistinguishable under Chosen-Ciphertext Attack (IND-CCA) if

$$\Pr[b = b'] = 1/2 + \mathrm{negl}(n).$$

Remarks

- Since Adversary knows the pk, it can encrypt any messages of its choice at any time.
- Maleable encryption can't be IND-CCA secure. Why?

RSA Optimal Asymmetric Encryption Padding, AOEP

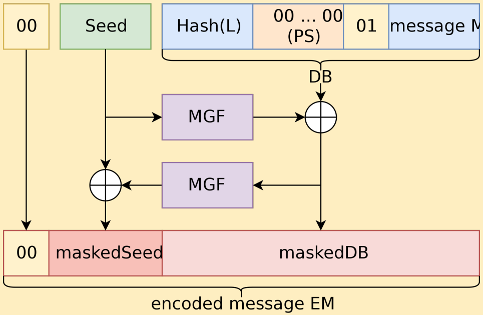
Being IND-CCA secure requires both non-maleability, and randomized encryption. AOEP padding ensures both.

Encoding

- L = public label
- Seed = random seed
- MGF is a special pseudorandom Mask Generation Function

RSA-AOEP is proven to be IND-CCA secure (in the Random Oracle model)

Encoding Diagram



Homomorphic Encryption

Linearly Homomorphic Encryption (LHE)

Public-key encryption $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ with additional algorithms $\oplus_{\mathrm{pk}}$ and $\odot_{\mathrm{pk}}$:

- $\mathrm{Enc}_{\mathrm{pk}}(a) \oplus_{\mathrm{pk}} \mathrm{Enc}_{\mathrm{pk}}(b) = \mathrm{Enc}_{\mathrm{pk}}(a + b)$
- $\mathrm{Enc}_{\mathrm{pk}}(a) \odot_{\mathrm{pk}} b = \mathrm{Enc}_{\mathrm{pk}}(ab)$
  Can't multiply two ciphertexts (note that b is given in plain).

The plaintexts here are interpreted as numbers $\{0, 1, \ldots M - 1\}$ with arithmetic operations on them performed modulo M. M is defined by pk.

Homomorphic Encryption

Linearly Homomorphic Encryption (LHE)

Public-key encryption $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ with additional algorithms $\oplus_{\mathrm{pk}}$ and $\odot_{\mathrm{pk}}$:

- $\mathrm{Enc}_{\mathrm{pk}}(a) \oplus_{\mathrm{pk}} \mathrm{Enc}_{\mathrm{pk}}(b) = \mathrm{Enc}_{\mathrm{pk}}(a+b)$
- $\mathrm{Enc}_{\mathrm{pk}}(a) \odot_{\mathrm{pk}} b = \mathrm{Enc}_{\mathrm{pk}}(ab)$
  Can't multiply two ciphertexts (note that b is given in plain).

The plaintexts here are interpreted as numbers $\{0, 1, \ldots M-1\}$ with arithmetic operations on them performed modulo M. M is defined by pk.

Fully Homomorphic Encryption (FHE)

Same as LHE, but now:

- $\mathrm{Enc}_{\mathrm{pk}}(a) \oplus_{\mathrm{pk}} \mathrm{Enc}_{\mathrm{pk}}(b) = \mathrm{Enc}_{\mathrm{pk}}(a+b)$
- $\mathrm{Enc}_{\mathrm{pk}}(a) \odot_{\mathrm{pk}} \mathrm{Enc}_{\mathrm{pk}}(b) = \mathrm{Enc}_{\mathrm{pk}}(ab)$
  Now we can multiply ciphertexts!

FHE schemes often involve heavy computations and large ciphertexts, which limit their applicability.

Both LHE and FHE are maleable, so they can't be IND-CCA secure.

LHE: Familiar Examples

ElGamal

- Let (B_1, c_1) encrypt m_1 with key A
- Let (B_2, c_2) encrypt m_2 with key A
- Suppose Alice is decrypting $(B_1 \cdot B_2, c_1 \cdot c_2)$ with her private key a:

$$\begin{aligned} \gamma &= (B_1 B_2)^a \\ &= B_1^a \cdot B_2^a \\ m &= c_1 c_2 \cdot \gamma^{-1} \\ &= (c_1 B_1^{-a})(c_2 B_2^{-a}) \\ &= m_1 m_2 \end{aligned}$$

LHE: Familiar Examples

ElGamal

- Let (B_1, c_1) encrypt m_1 with key A
- Let (B_2, c_2) encrypt m_2 with key A
- Suppose Alice is decrypting $(B_1 \cdot B_2, c_1 \cdot c_2)$ with her private key a:

$$\begin{aligned} \gamma &= (B_1 B_2)^a \\ &= B_1^a \cdot B_2^a \\ m &= c_1 c_2 \cdot \gamma^{-1} \\ &= (c_1 B_1^{-a})(c_2 B_2^{-a}) \\ &= m_1 m_2 \end{aligned}$$

Textbook RSA

- Let m_1^e and m_2^e be RSA ciphertexts, encrypted for Alice whose private keys is (d, N)
- Let's see what happens if Alice decrypts $m_1^e m_2^e$:

$$\begin{aligned} m &= (m_1^e m_2^e)^d \\ &= (m_1 \cdot m_2)^{ed} \\ &= m_1 m_2 \end{aligned}$$

Why is it linear even though m_1 and m_2 got multiplied?

More on Homomorphic Encryption[†]

Why ElGamal and RSA are only Linearly Homomorphic?

As we've shown on the previous slide, ElGamal and RSA allow multiplying encryptions. We still call them Linearly Homomorphic schemes because the multiplications of numbers there correspond to the group operation, and from group's perspective it can be seen as "addition". Applying this group operation repeatedly would correspond to "multiplication" in group terms.

- For ElGamal, the group is $(Z_p^*, \star)$ with $m_1 \star m_2 = m_1 \cdot m_2 \mod p$.
- For RSA, the group is $(Z_N^*, \star)$ with $m_1 \star m_2 = m_1 \cdot m_2 \mod N$.

These encryption schemes would be Fully Homomorphic if they allowed by given $\text{Enc}_{\text{pk}}(m_1)$ and $\text{Enc}_{\text{pk}}(m_1)$ computing $\text{Enc}_{\text{pk}}(\underbrace{m_1 \star m_1 \star \cdots \star m_1}_{m_2 \text{ times}})$, which is "multiplication" from group's perspective.

Uses of Homomorphic Encryption

LHE and FHE can be used to "outsource" computations to an untrusted party.

- To implement a cloud that can process people's private data without being able to peek on the data.
- To implement some form of Multi-Party Computation where parties jointly perform operations on their data without anyone of them being able to see the data they're operating on. (We'll dicsuss this more in Lecture 9.)

Self-check questions

1. How to exchange (symmetric) keys using Public-Key Encryption? (We built ElGamal from Diffie-Hellman key exchange. This question asks if one could go the other way.)
2. What property of Diffie-Hellman actually helped us convert it to ElGamal cryptosystem? Could one do the same with any key exchange protocol?
3. How to make Trapdoor One-Way Function from Public-Key Encryption (using it as blackbox)?

Copyright & Attribution

- The RSA-OAEP Encoding Diagram was taken from Wikipedia[1] and used under Creative Commons.
- People icons in diagrams were taken from `tikzpeople` LaTeX package[2]
- Font used is Libertinus Sans.

[1] `https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding`

[2] `https://www.ctan.org/pkg/tikzpeople`