

CMSC/Math 456: Cryptography (Fall 2022)

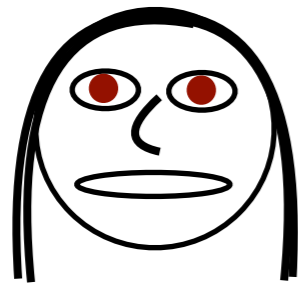
Lecture 16

Daniel Gottesman

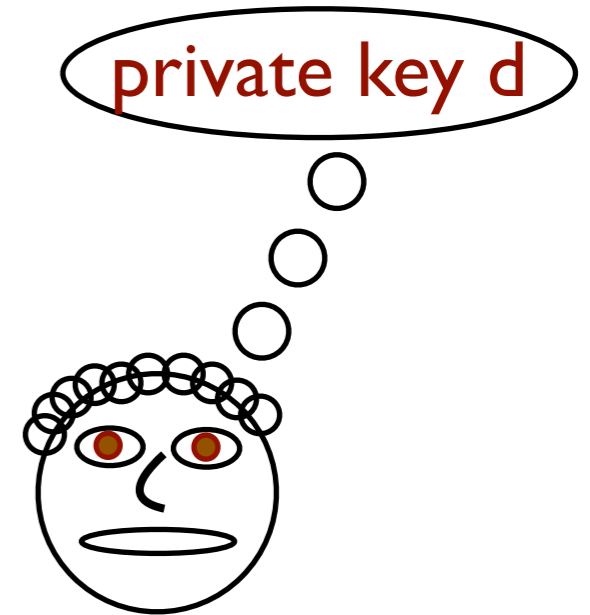
Administrative

There will be no problem set this week.

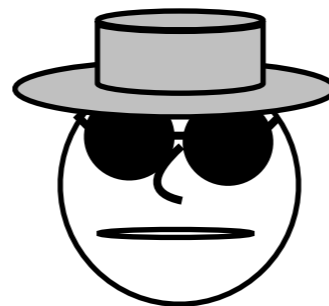
Public Key Encryption



Alice



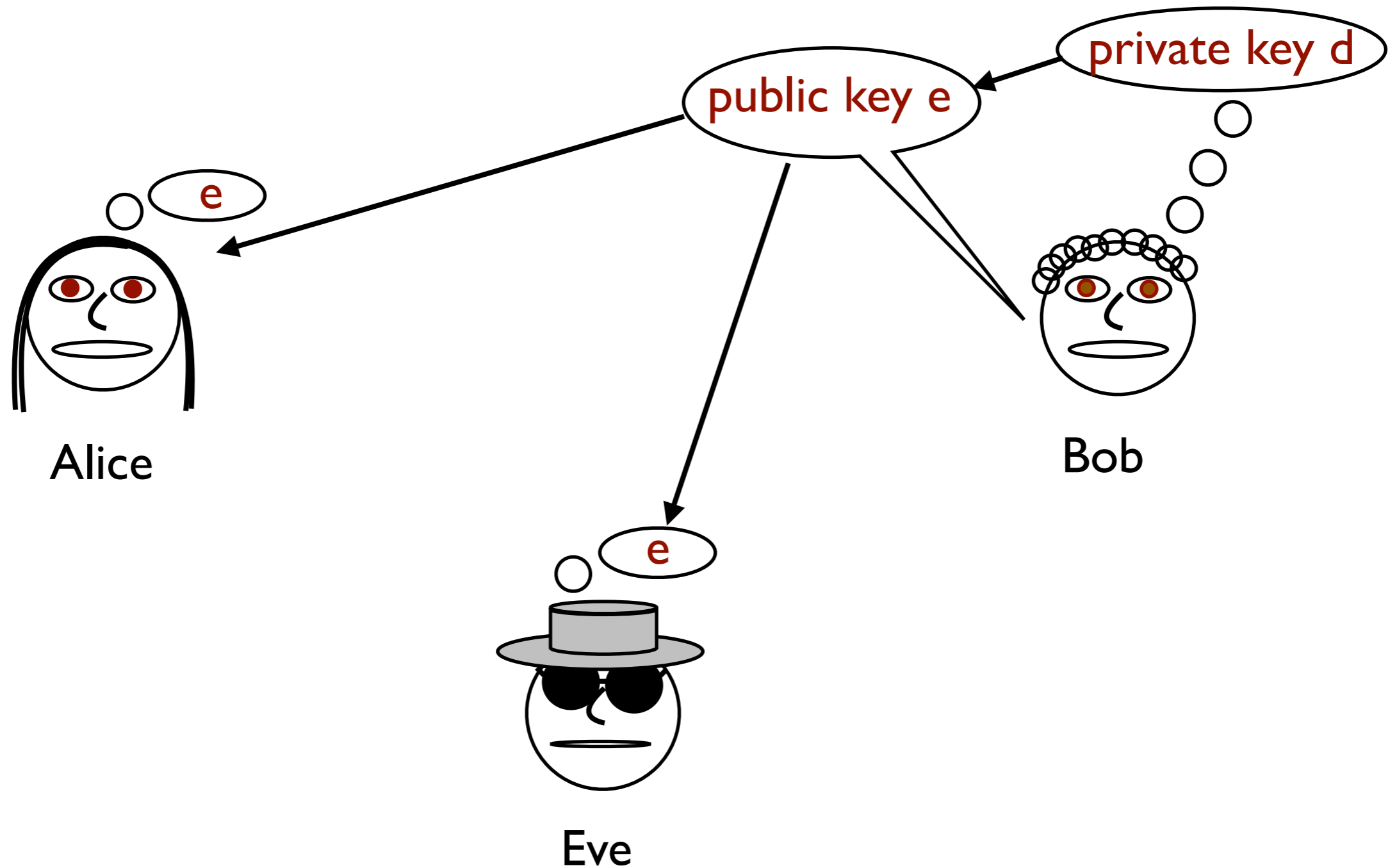
Bob



Eve

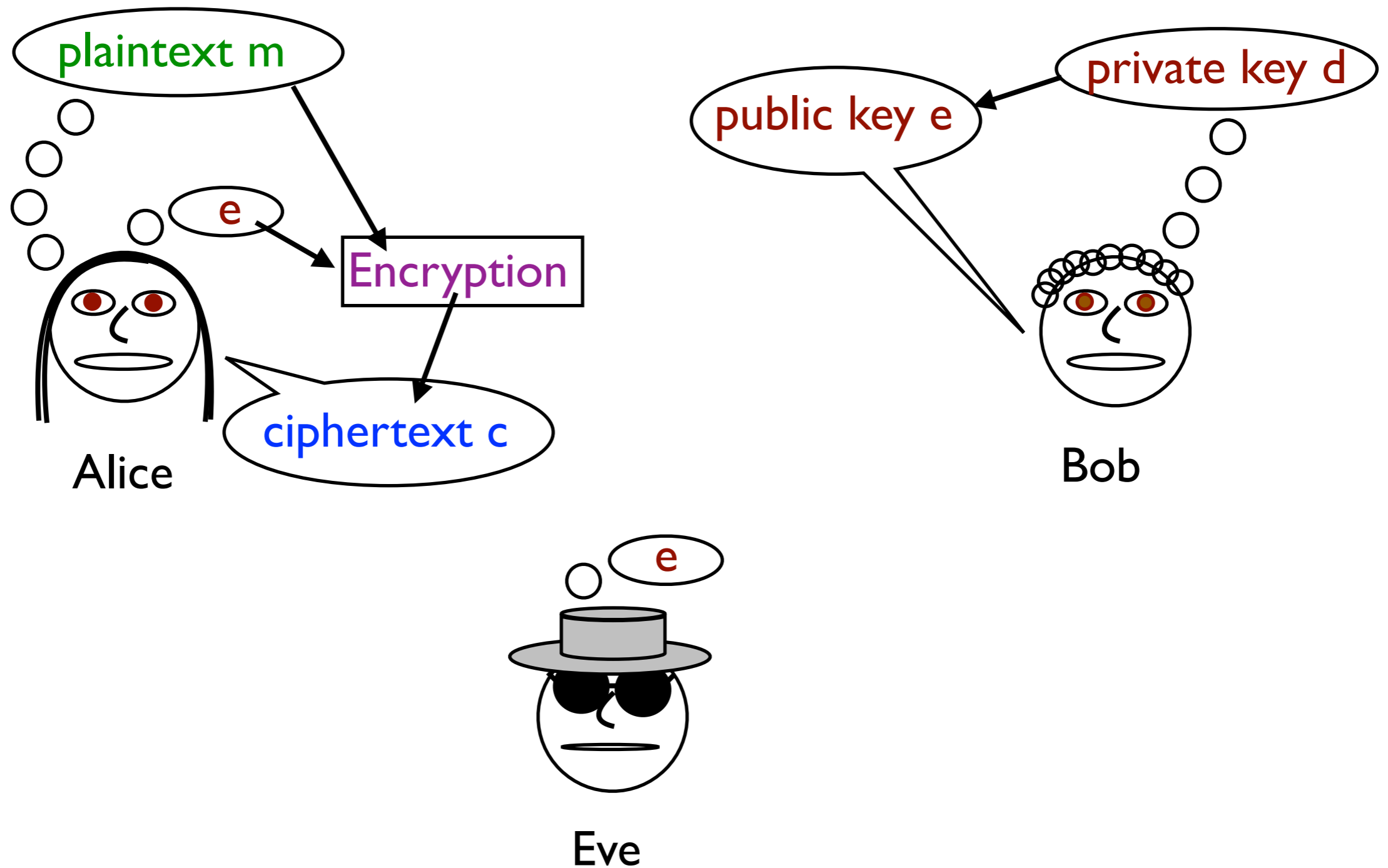
Public-key encryption is an **asymmetric** protocol.

Public Key Encryption



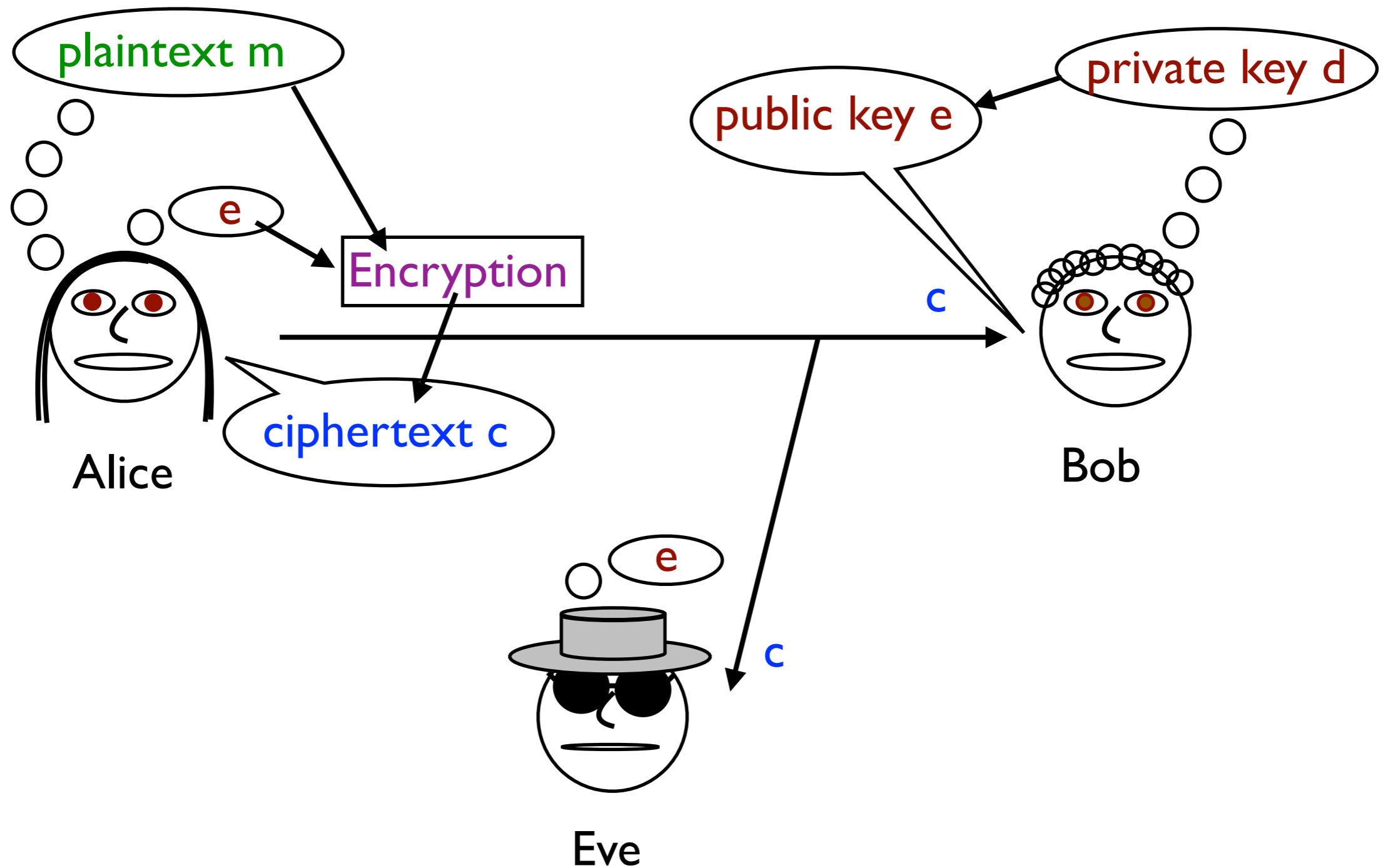
Public-key encryption is an **asymmetric** protocol.

Public Key Encryption



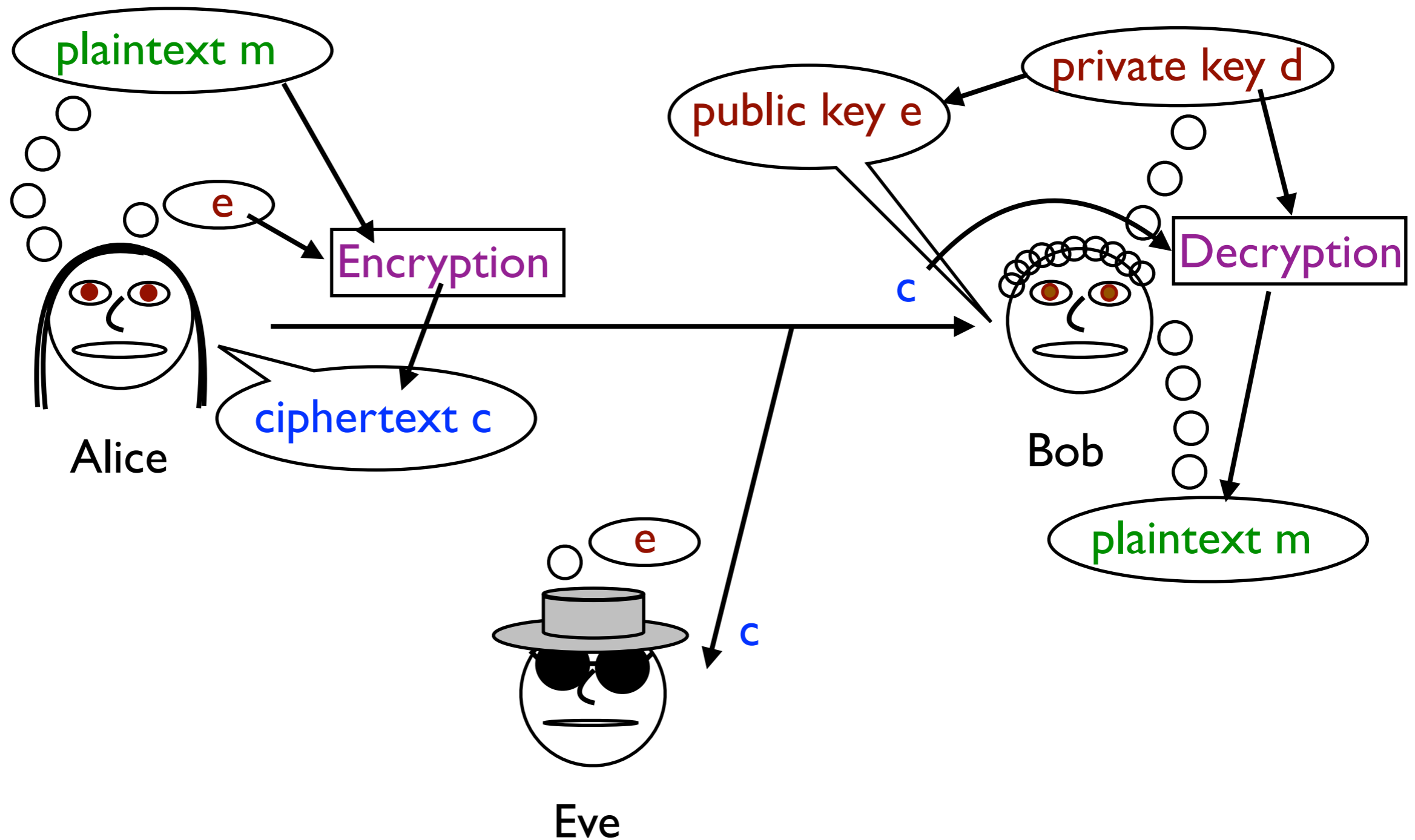
Public-key encryption is an **asymmetric** protocol.

Public Key Encryption



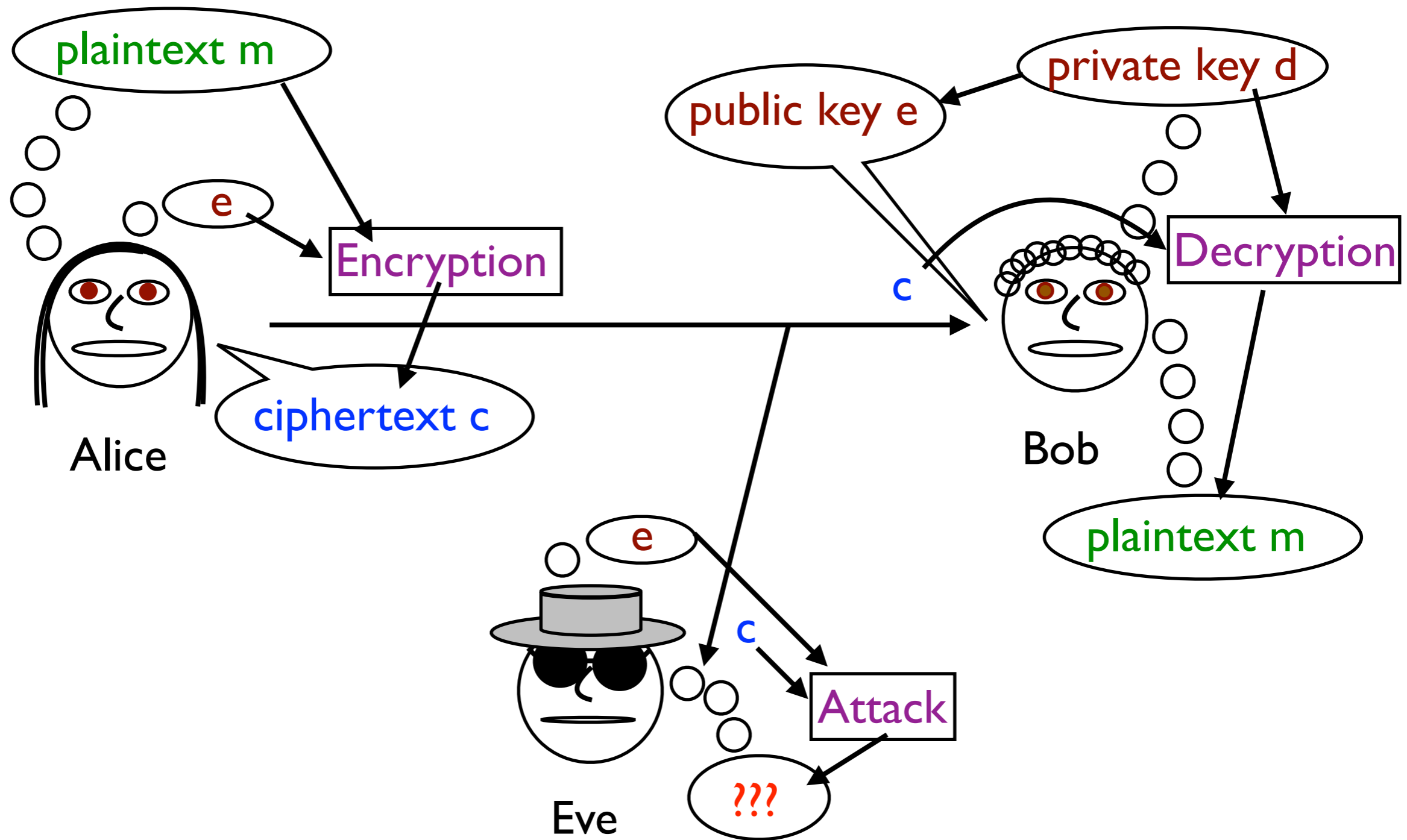
Public-key encryption is an **asymmetric** protocol.

Public Key Encryption



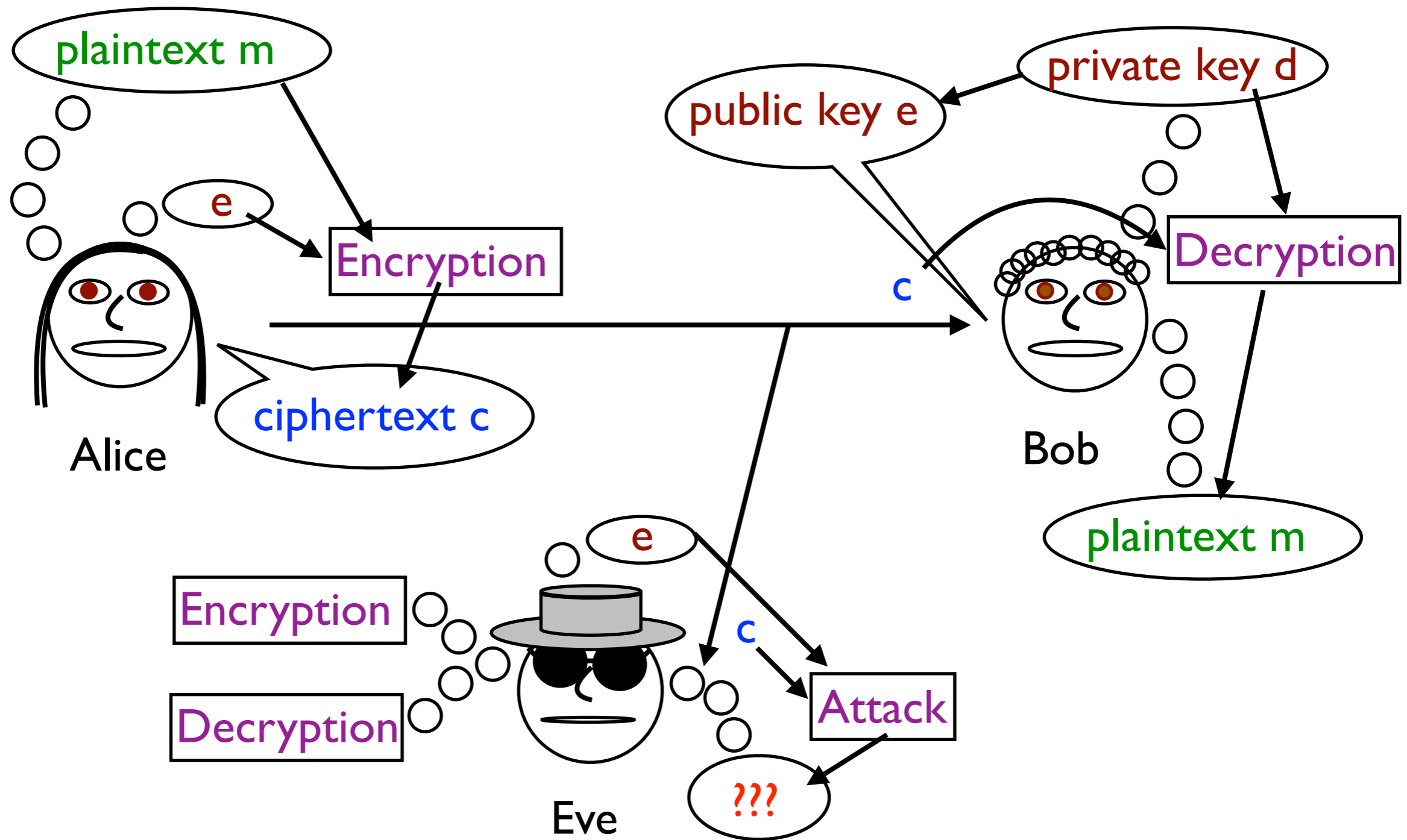
Public-key encryption is an **asymmetric** protocol.

Public Key Encryption



Public-key encryption is an **asymmetric** protocol.

Public Key Encryption



Public-key encryption is an **asymmetric** protocol.

Definition of Public Key Encryption

Definition: A **public-key encryption protocol** is a set of three probabilistic polynomial-time algorithms (**Gen**, **Enc**, **Dec**).

Gen is the **key generation algorithm**. It takes as input **s**, the **security parameter**, and outputs a **public key, private key pair** $(e, d) \in \{0,1\}^* \times \{0,1\}^*$.

Enc is the **encryption algorithm**. It takes as input **e** and a **plaintext or message** $m \in \{0,1\}^*$ and outputs a **ciphertext** $c \in \{0,1\}^*$.

Dec is the **decryption algorithm**. It takes as input **d** and **c** and outputs some $m' \in \{0,1\}^*$.

The encryption protocol is **correct** if

$$Dec(d, Enc(e, m)) = m$$

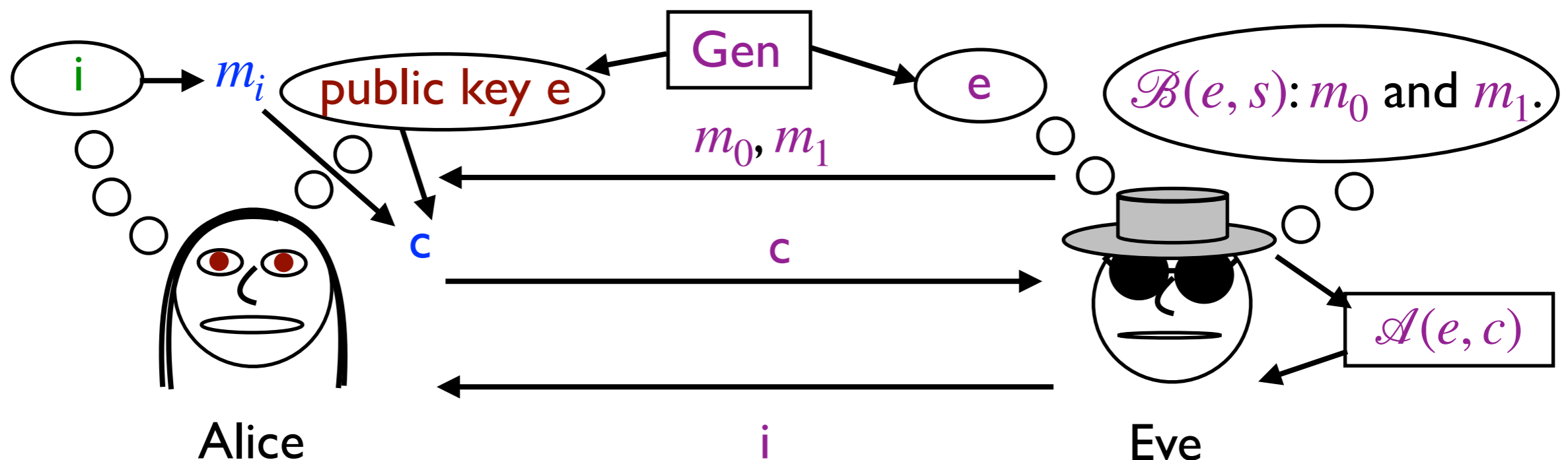
Note: **Gen** here is much more complex than for private-key encryption and doesn't just generate random bit strings.

Public Key Security

Definition: A public-key encryption protocol $(\text{Gen}, \text{Enc}, \text{Dec})$ with security parameter s has **indistinguishable encryptions in the presence of an eavesdropper (is EAV-secure)** if, for any pair of messages m_0 and m_1 chosen by the adversary (using efficient algorithm $\mathcal{B}(e, s)$) and for any efficient attack $\mathcal{A}(e, c)$,

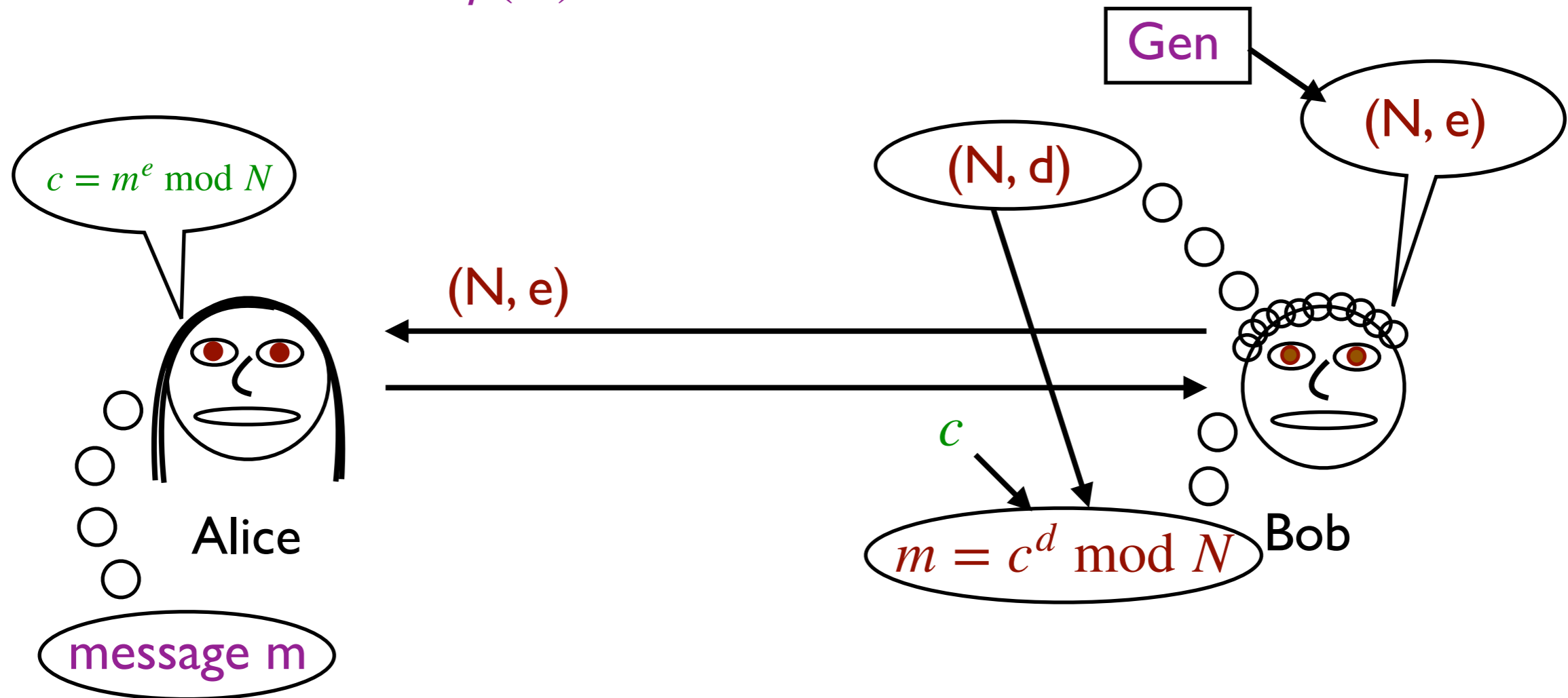
$$|\Pr_{(e,d)}(\mathcal{A}(e, \text{Enc}(e, m_0)) = 1) - \Pr_{(e,d)}(\mathcal{A}(e, \text{Enc}(e, m_1)) = 1)| \leq \epsilon(s)$$

for negligible $\epsilon(s)$ and probability taken over valid public key, private key pairs (e,d) and randomness of Enc , \mathcal{A} , and \mathcal{B} .



RSA Encryption (Picture)

In RSA, Gen creates a public key (N, e) and a private key (N, d) with $ed = 1 \pmod{\varphi(N)}$.



“RSA” stands for the inventors: Rivest, Shamir, and Adleman

RSA Encryption (Gen, Enc, Dec)

Gen: Generate two random primes p and q which are s bits long. Let $N = pq$. Choose $e, d \in \mathbb{Z}_N^*$ such that $ed = 1 \pmod{\varphi(N)}$. The **public key** is (N, e) and the **private key** is (N, d) .

Enc: Given message m and public key (N, e) . The ciphertext is $c = m^e \pmod{N}$.

Dec: Given ciphertext c and private key (N, d) . The decrypted message is $m' = c^d \pmod{N}$.

Example:

Gen: Let $p = 11$, $q = 17$. Then $N = 187$ and $\varphi(N) = (p - 1)(q - 1) = 10 \cdot 16 = 160$. Let $e = 3$; then $d = 107$ (so $3 \cdot 107 = 1 \pmod{160}$).

Enc: Let $m = 113$. Then $c = 113^3 \pmod{187} = 5$.

Dec: $m' = 5^{107} \pmod{187} = 113$.

Gen Algorithm Analyzed

- Why pick p and q instead of picking N directly?

To pick e and d , we need to know $\varphi(N)$, and to find $\varphi(N)$, we need to know the prime factorization of N . **Factoring seems hard**, so we pick the prime factors directly.

- How do we pick e and d ?

We can pick e in any convenient way. Then use Euclid's algorithm to find its multiplicative inverse **mod** $\varphi(N)$.

- Eve knows e . Can't she also apply Euclid's algorithm to find e ?

Only if she knows $\varphi(N)$. But to compute $\varphi(N)$, it seems she needs to know the prime factorization of N . And **factoring seems hard**.

Why is RSA Correct?

Why does $\text{Dec}(d, \text{Enc}(e, m)) = m$?

Recall the **Euler-Fermat theorem**: $x^{\varphi(N)} = 1 \pmod N$.

Then

$$\text{Dec}(d, \text{Enc}(e, m)) = m^{ed} \pmod N$$

But

$$ed = 1 + k\varphi(N)$$

so

$$\begin{aligned}\text{Dec}(d, \text{Enc}(e, m)) &= m^{1+k\varphi(N)} \pmod N \\ &= m \cdot (m^{\varphi(N)})^k \pmod N \\ &= m \cdot 1^k \pmod N\end{aligned}$$

Hardness of Factoring

Factoring a product of two large primes seems to be hard.

Definition: Given a security parameter s , let $P(N)$ be a probability distribution over $(2s)$ -bit numbers. We say that **factoring is average-case hard for $P(N)$** if for **any polynomial time algorithm \mathcal{A}** , for random N chosen according to $P(N)$,

$$\Pr(\mathcal{A}(N) \text{ succeeds}) \leq \epsilon(s)$$

for $\epsilon(s)$ a negligible function, where we say $\mathcal{A}(N)$ **succeeds** if $\mathcal{A}(N) = p$ with p a non-trivial factor of N : $1 < p < N$ and $p \mid N$.

When N is chosen to be the product of two random s -bit primes, it appears that factoring is hard for the resulting distribution.

Factoring and RSA

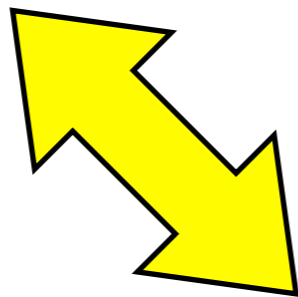
If Eve can factor N , she can break RSA:

Eve factors $N = pq$ and computes $\varphi(N) = (p - 1)(q - 1)$.

Using Euclid's algorithm, she computes d given e and $\varphi(N)$.

Theorem: Given N , if Eve knows d and e , she has an efficient probabilistic algorithm to factor N .

Factoring N



Learning the private key for RSA

But perhaps it is possible to break RSA without learning the private key

Is RSA Secure?

Vote: Is RSA as I've presented it CPA-secure? (Yes/No/Unknown)

Is RSA Secure?

Vote: Is RSA as I've presented it CPA-secure? (Yes/No/Unknown)

Answer: No.

This version of RSA (**Plain RSA**) is **deterministic**: It cannot be CPA-secure.

If Eve is trying to determine which of two messages was sent, she can just try encrypting each of them using the public key. In Plain RSA, the message m_i always corresponds to ciphertext $c_i = m_i^e \bmod N$, so she can easily find i by matching $c = c_i$.

Is RSA Secure?

Vote: Is RSA as I've presented it CPA-secure? (Yes/No/Unknown)

Answer: No.

This version of RSA (**Plain RSA**) is **deterministic**: It cannot be CPA-secure.

If Eve is trying to determine which of two messages was sent, she can just try encrypting each of them using the public key. In Plain RSA, the message m_i always corresponds to ciphertext $c_i = m_i^e \bmod N$, so she can easily find i by matching $c = c_i$.

We need to add a random element to **Enc**.

Padded RSA

One option to add randomness to RSA is to **pad** the message with random bits:

Gen remains the same.

Enc: Given message m , generate ℓ -bit random r and let $\tilde{m} = r||m$ (the concatenation r followed by m). Then the ciphertext $c = \tilde{m}^e \bmod N$.

Dec: Given c , compute $\tilde{m}' = c^d \bmod N$. Discard the first ℓ bits of \tilde{m}' to get m' .

Example:

Gen: $N = 187$, $e = 3$, and $d = 107$ as before.

Enc: Let $m = 15 = 1111_2$. Pad using $r = 110_2$. Then $\tilde{m} = 1101111_2 = 111$ and $c = 111^3 \bmod 187 = 100$.

Dec: $\tilde{m}' = 100^{107} \bmod 187 = 111$ and $m' = 15$.

Security of Padded RSA

If $\ell = \lfloor \log_2 N \rfloor - 1$, Padded RSA is CPA-secure assuming security of an RSA assumption: It is hard to find x such that $x^\ell = y \pmod N$.

But this implies just 1-bit messages.

Shorter padding is not known to be secure (at least without a stronger assumption), but is plausibly secure as long as ℓ is not too short.

RSA PKCS #1 v1.5 is a variant of padded RSA where some of the padded bits are random and some have fixed values. With a good choice of parameters, it is probably CPA-secure in the same sense as Padded RSA.

Picking e

How do we pick e ?

- e must be relatively prime to $\varphi(N)$ so that it has an inverse $\text{mod } \varphi(N)$.
- Otherwise, any e works.
- But if e and m are both small ($m^e < N$), decrypting becomes a problem over \mathbb{Z} (which is easy) instead of over \mathbb{Z}_N (which is hard).
- Might as well pick an e so that it is easy to calculate $m^e \text{ mod } N$.
 - Using repeated squaring, the # of terms to multiply together is the # of 1's in the binary representation of e .
 - So pick e with weight 2, e.g., $e=3$.
 - Or maybe a larger e of the form $2^a + 1$ to avoid the possible small e and small m attack.

Algorithms for Factoring

A factoring algorithm breaks RSA ... so how hard is factoring?

Similar to discrete log:

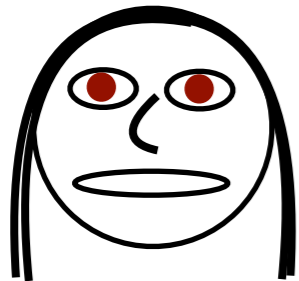
- When $N=pq$ and $p-1$ is a product of small primes, Pollard's $p-1$ algorithm factors efficiently.
- For general N , the number field sieve runs in time $2^{O((\log N)^{1/3}(\log \log N)^{2/3})}$ (apparently). This is sub-exponential.
- **Except:** A quantum computer can efficiently factor arbitrary N .

So recommended key lengths for secure variants of RSA are 2048 bits and higher.

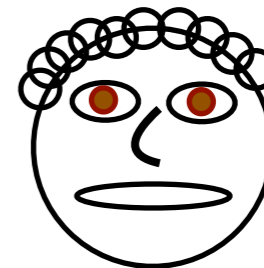
KEM/DEM

Recall that a KEM creates and sends a random key string to someone whose public key you have.

That key then becomes the key for a private-key encryption system, which is here called a **data-encapsulation mechanism (DEM)**



Alice

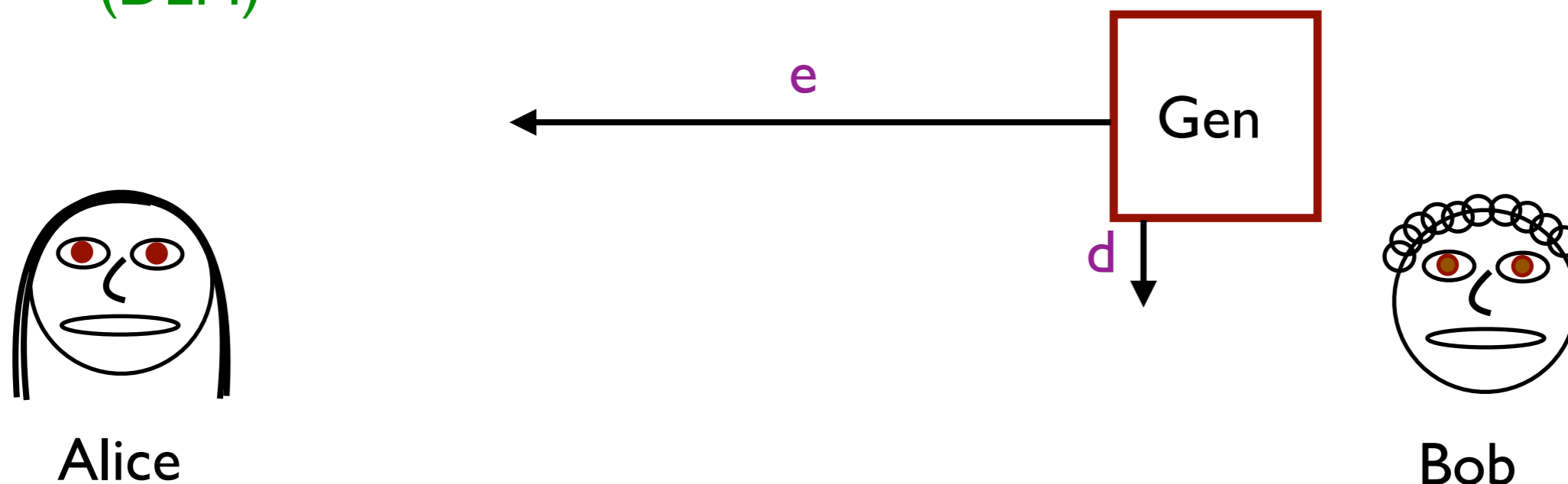


Bob

KEM/DEM

Recall that a KEM creates and sends a random key string to someone whose public key you have.

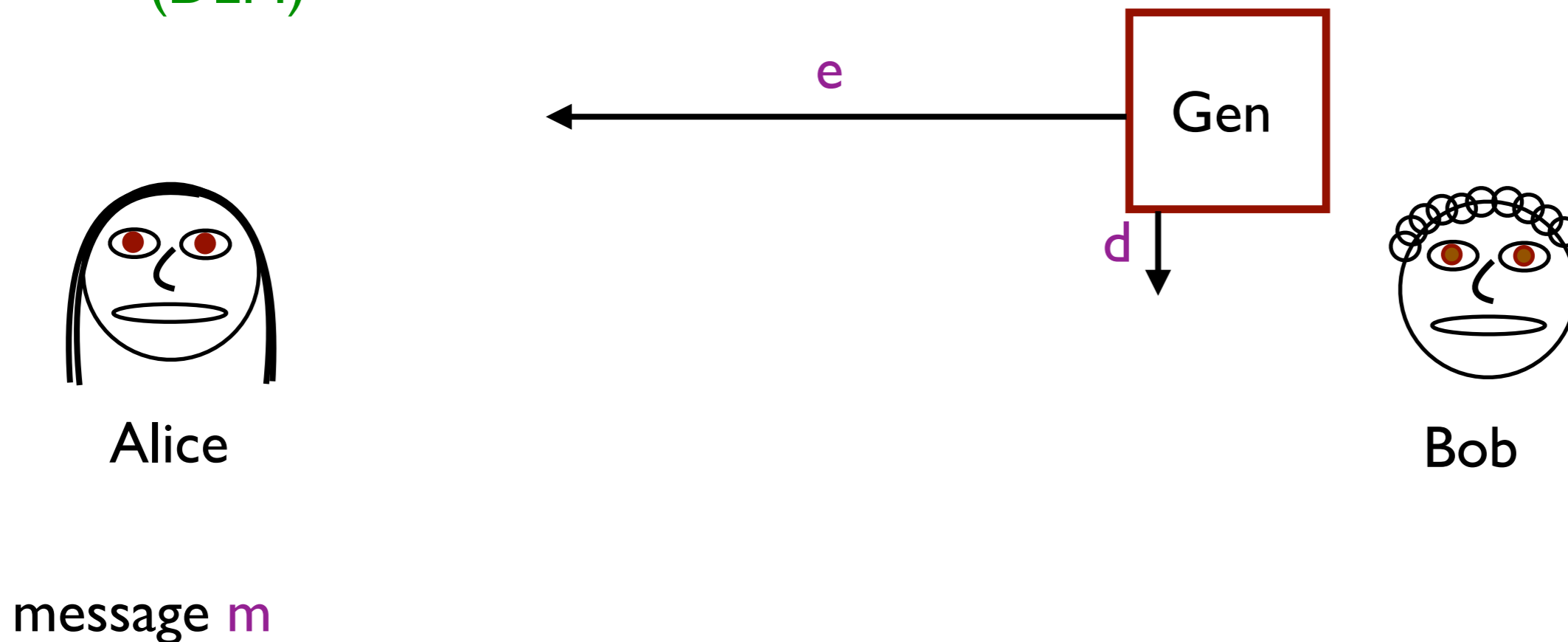
That key then becomes the key for a private-key encryption system, which is here called a **data-encapsulation mechanism (DEM)**



KEM/DEM

Recall that a KEM creates and sends a random key string to someone whose public key you have.

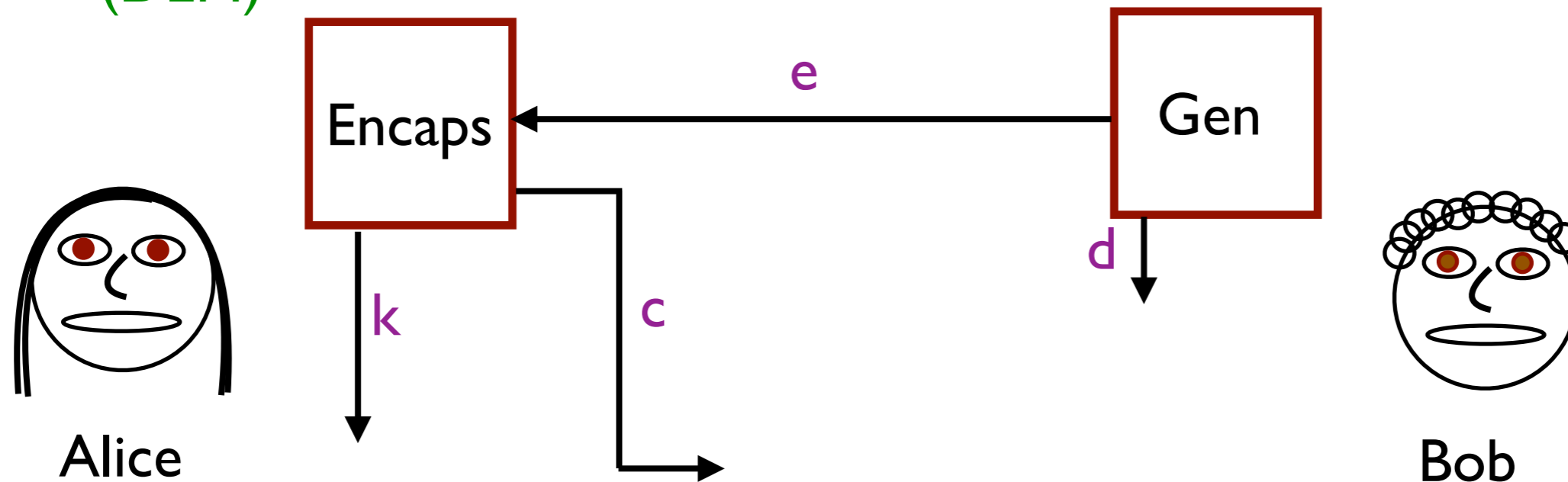
That key then becomes the key for a private-key encryption system, which is here called a **data-encapsulation mechanism (DEM)**



KEM/DEM

Recall that a KEM creates and sends a random key string to someone whose public key you have.

That key then becomes the key for a private-key encryption system, which is here called a **data-encapsulation mechanism (DEM)**

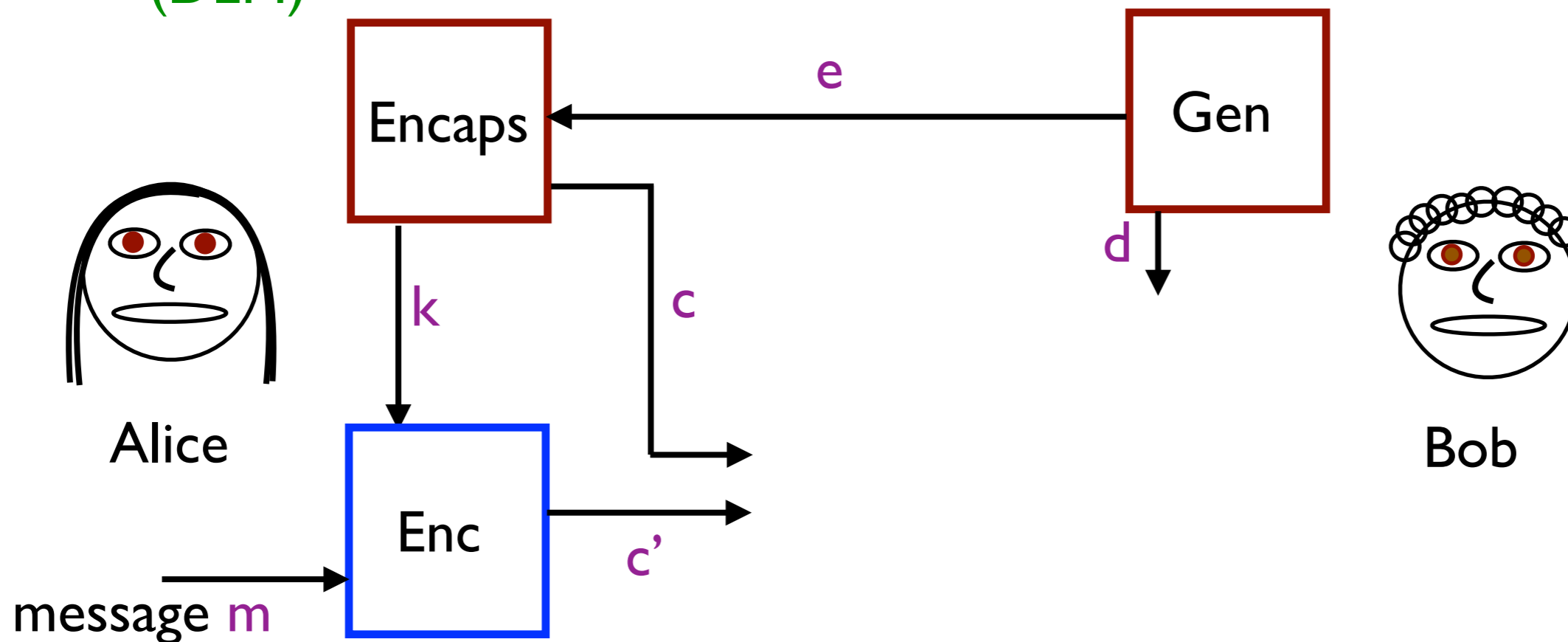


message m

KEM/DEM

Recall that a KEM creates and sends a random key string to someone whose public key you have.

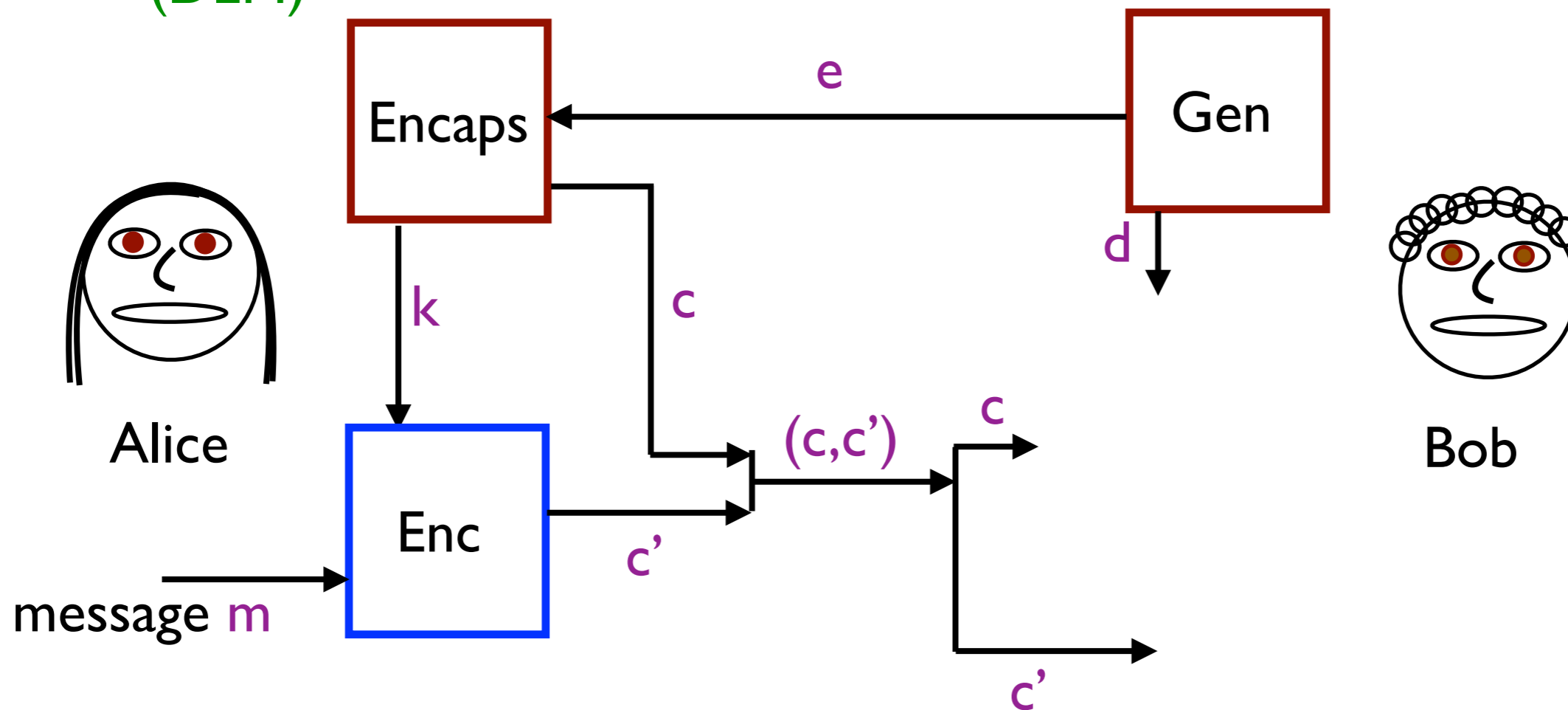
That key then becomes the key for a private-key encryption system, which is here called a **data-encapsulation mechanism (DEM)**



KEM/DEM

Recall that a KEM creates and sends a random key string to someone whose public key you have.

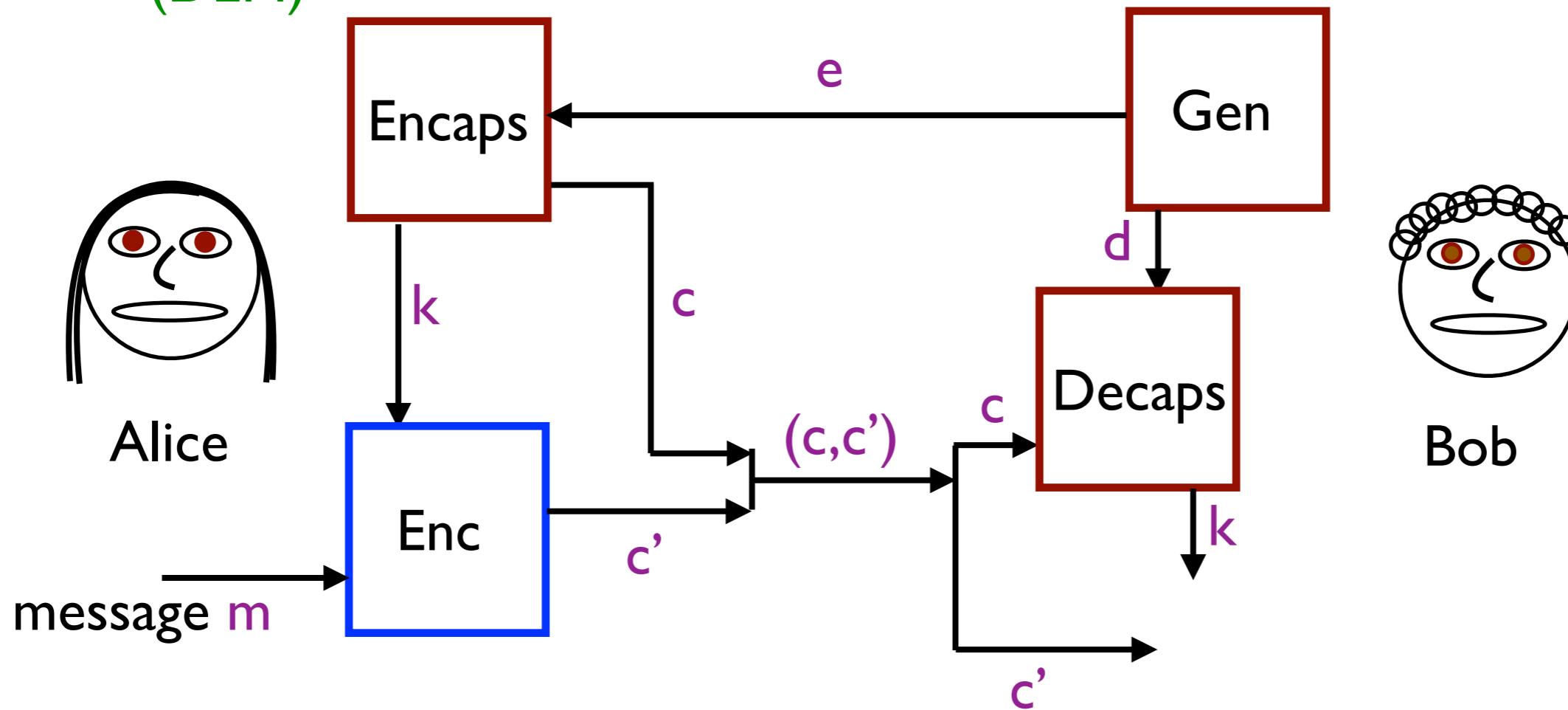
That key then becomes the key for a private-key encryption system, which is here called a **data-encapsulation mechanism (DEM)**



KEM/DEM

Recall that a KEM creates and sends a random key string to someone whose public key you have.

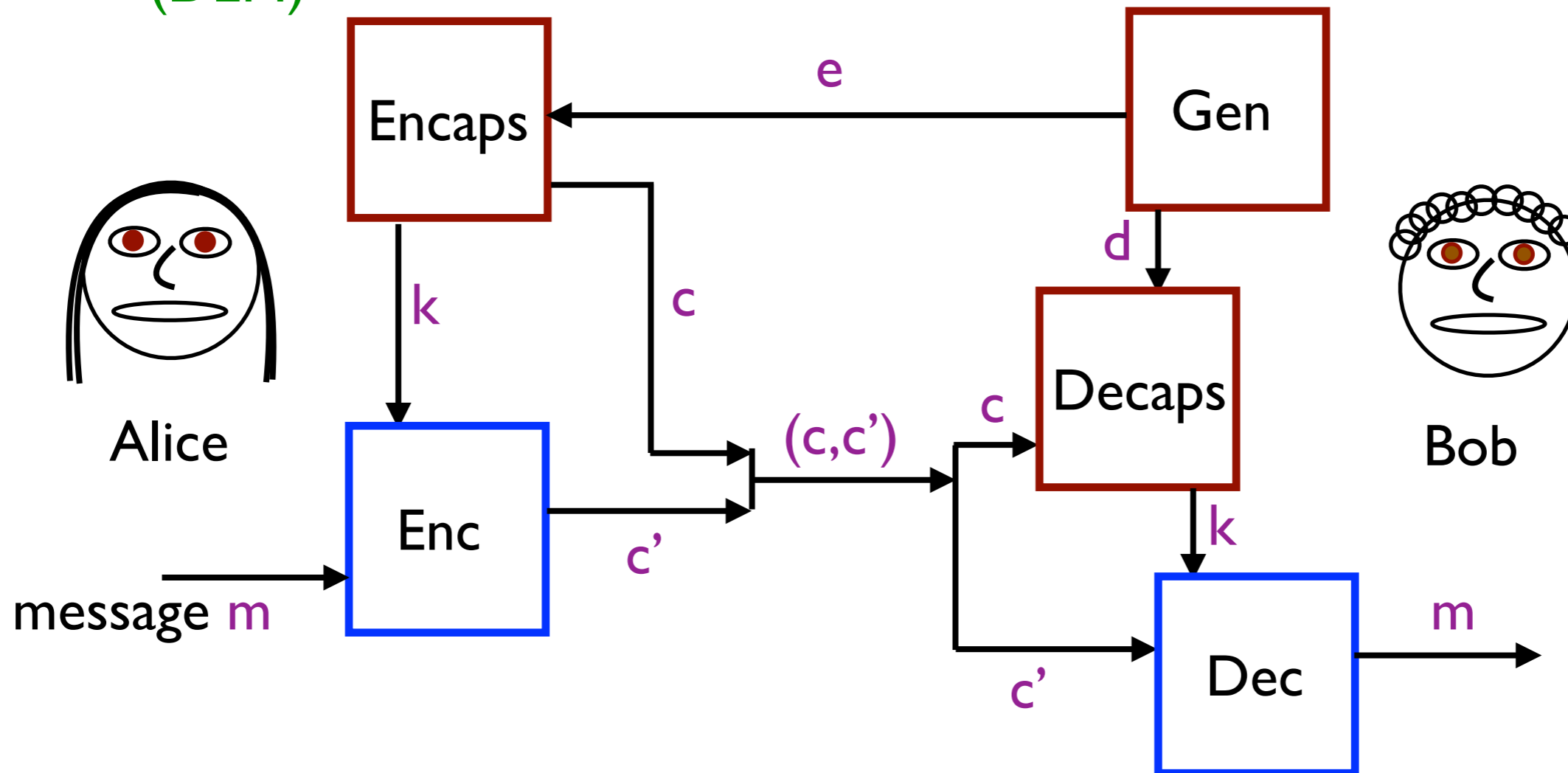
That key then becomes the key for a private-key encryption system, which is here called a **data-encapsulation mechanism (DEM)**



KEM/DEM

Recall that a KEM creates and sends a random key string to someone whose public key you have.

That key then becomes the key for a private-key encryption system, which is here called a **data-encapsulation mechanism (DEM)**



RSA-based KEM

Gen: Pick a (public) **key derivation function** $H(x)$, then as usual for RSA, i.e., generate two random primes p and q which are s bits long. Let $N = pq$. Choose $e, d \in \mathbb{Z}_N^*$ s.t. $ed = 1 \pmod{\varphi(N)}$. The **public key** is (N, e) and the **private key** is (N, d) .

Encaps: Choose random x . The **ciphertext** is $c = x^e \pmod N$ and the **key** is $H(x)$.

Decaps: Given c and d , compute $x' = c^d \pmod N$. Then the key is $H(x')$.

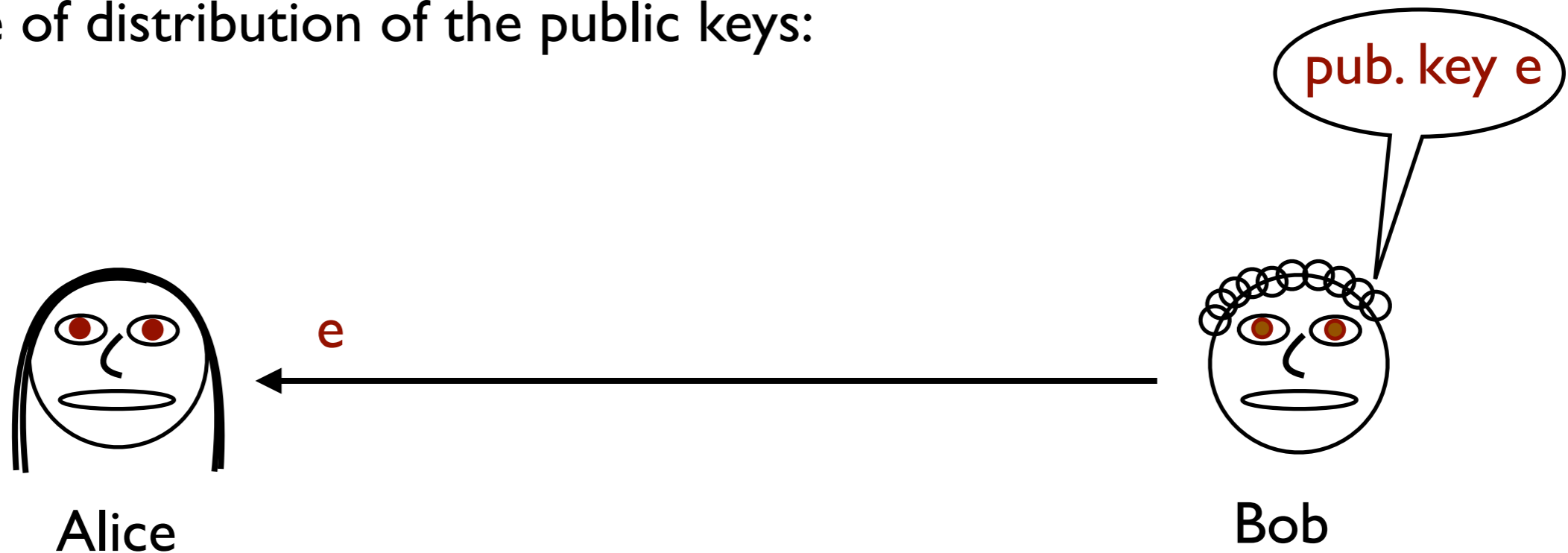
As with Diffie-Hellman-based KEM/DEM, this allows us to combine the strengths of public key and private key cryptography.

This **KEM is secure** given RSA assumption and an assumption on the key derivation function $H(x)$.

How Do We Distribute Public Keys?

Recall that Diffie-Hellman had a man-in-the-middle attack where Eve pretended to be Bob when talking to Alice and vice-versa.

Public key systems have the same sort of problem at the time of distribution of the public keys:

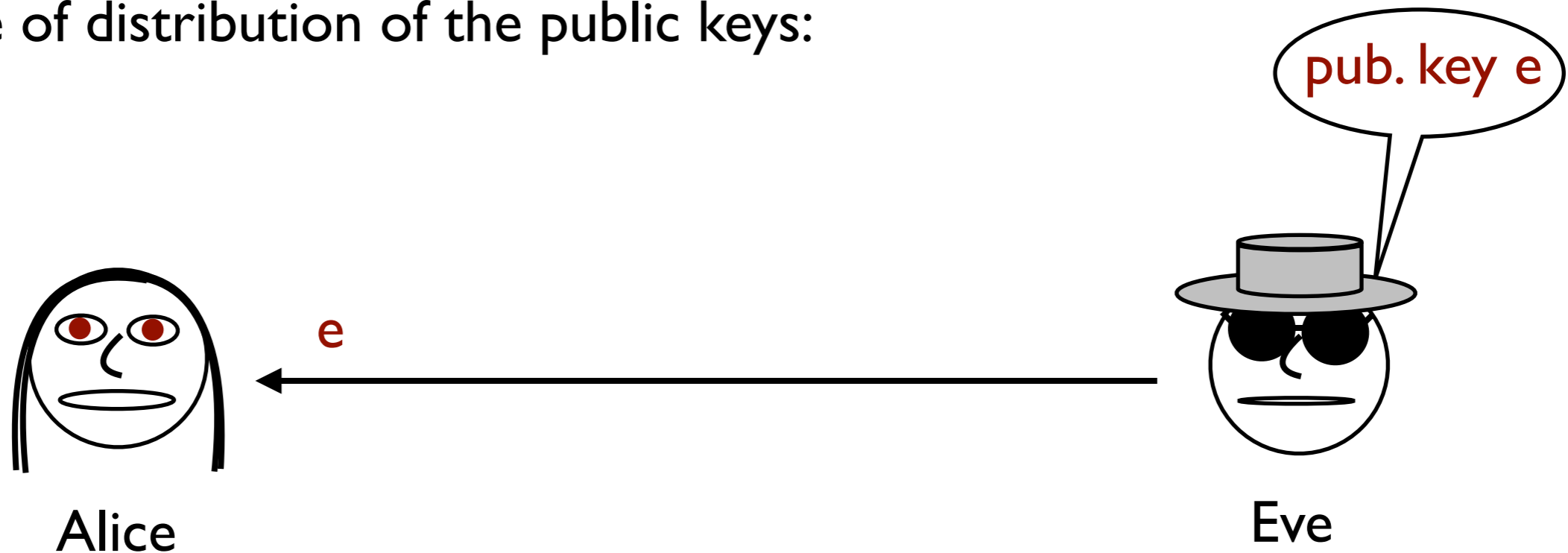


How does Alice know Bob's public key really comes from Bob and not from Eve? Bob needs some way to **authenticate** it.

How Do We Distribute Public Keys?

Recall that Diffie-Hellman had a man-in-the-middle attack where Eve pretended to be Bob when talking to Alice and vice-versa.

Public key systems have the same sort of problem at the time of distribution of the public keys:



How does Alice know Bob's public key really comes from Bob and not from Eve? Bob needs some way to **authenticate** it.

Chosen Ciphertext Attack on RSA

Plain RSA and Padded RSA are vulnerable to another kind of attack with a stronger threat model, a *chosen ciphertext attack*:

Suppose Alice sends Bob the ciphertext $c = \tilde{m}^e \bmod N$.

Eve can easily create $c' = 2^e c = (2\tilde{m})^e \bmod N$.

Why is that a problem? What if Eve sends c' to Alice, who decrypts it and then acts on the message assuming it to be from Bob. By observing Alice, Eve may be able to deduce the plaintext corresponding to c' , namely $2\tilde{m}$. This tells her \tilde{m} .

Authenticating messages will help us deal with this class of attacks as well, by removing Eve's ability to change the message.

To Do List

- **Message authentication** and **digital signatures** to ensure validity of public keys.
- **Hash functions** to serve as key derivation functions.
- **Chosen-ciphertext attacks** (CCA) and CCA-security.

Message authentication, digital signatures, and hash functions all have other applications beyond their usefulness for public key cryptography.

