CS1699: Blockchain Technology and Cryptocurrency

2. Public-Key Cryptography

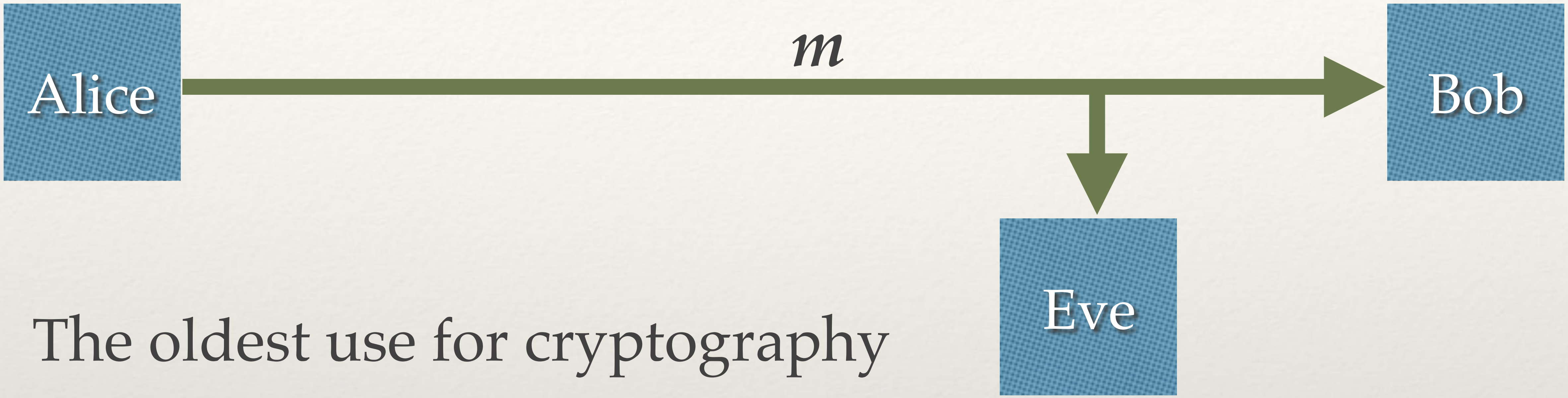
Bill Laboon

Cryptocurrency

- "Cryptographic currency": it behooves us to understand cryptography to really understand how a ***crypto***currency works
- The cryptographic primitives mentioned in chapter 1 assume some basic knowledge of cryptography

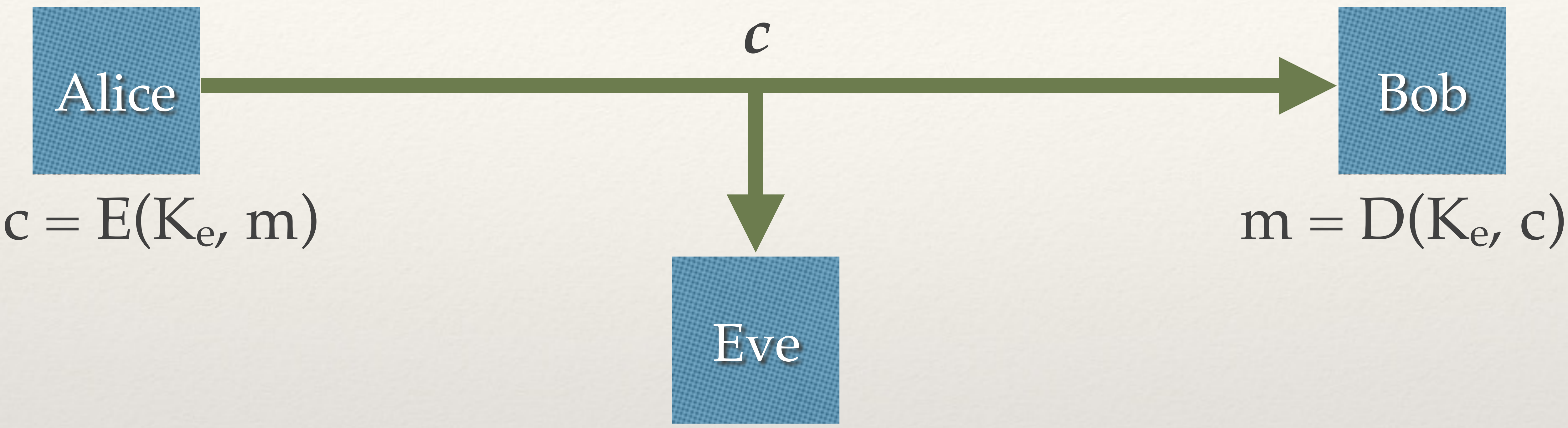
Much of the information on these slides is covered in-depth in Cryptography Engineering: Design, Principles, and Practical Applications by Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno.

Encryption



- The oldest use for cryptography
- Alice wants to send a message, m, to Bob, but Eve can read everything that is sent over the communication channel between them.
- How can Alice send m to Bob without Eve understanding the message?

Symmetric-Key Encryption



- Assume Alice and Bob share the same secret key, K_e .
- Alice can encrypt text using encryption function $c = E(K_e, m)$ send ciphertext c over the unencrypted channel
- Bob can decrypt using function $m = D(K_e, c)$
- Eve only ever sees c, not m, which is useless to her without K_e , even if she knows the decryption algorithm D

Symmetric-Key Encryption Using a Caesar Cipher

- One of the simplest ciphers around, the Caesar cipher simply converts a letter to a number, then given a key K_e, adds K_e to convert plaintext to ciphertext modulo the length of the alphabet l, then adds $-K_e$ modulo l to convert ciphertext to plaintext.
- For example, assume a letter translation function $t(x)$, where A = 1, B = 2… Z=26. Assume $K_e = 5$.
- $E(K_e, m)$ can be defined for each character as $c = (t(x) + K_e)$
- $D(K_e, c)$ can be defined for each character as $m = (t(x) - K_e)$

Symmetric-Key Encryption Using a Caesar Cipher

- Message m = ATTACKATDAWN, key $K_e = 5$
- For each character, apply encryption function $E(K_e, m)$
- Ciphertext c = FYYFHPFYIFBS

Ruby code to follow along:

```
Ke = 5
m = "ATTACKATDAWN"
m.chars.map { |x| x.ord - 64 }.map { |x| (x + Ke) % 26}.map { |x| (x + 64).chr }.join
```

Symmetric-Key Decryption Using a Caesar Cipher

- Ciphertext c = FYYFHPFYIFBS, key $K_e = 5$
- For each character, apply encryption function $D(K_e, m)$
- Message m = ATTACKATDAWN

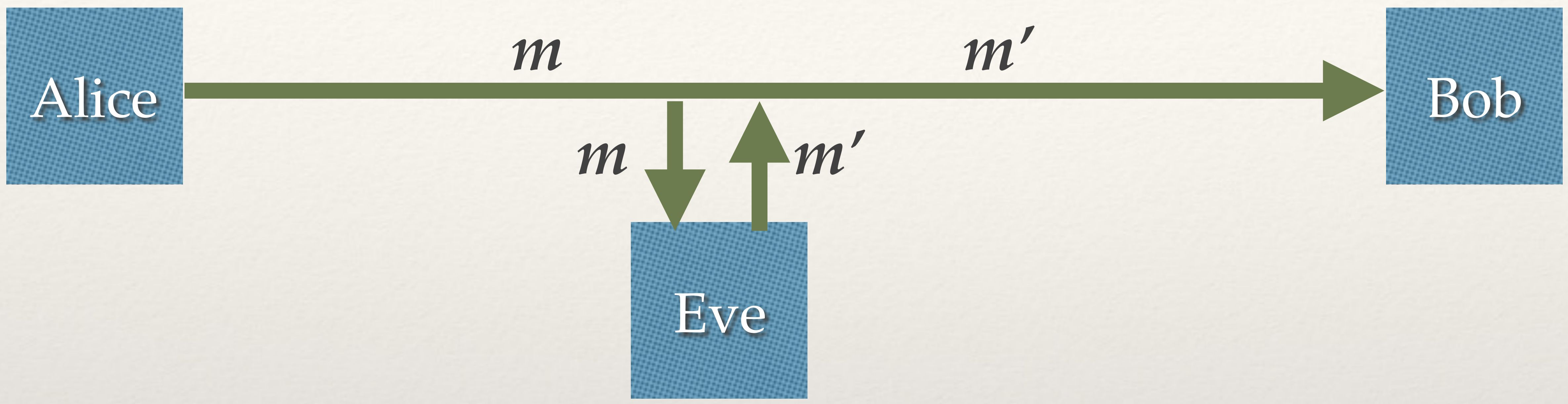
Ruby code to follow along:

```
Ke = 5
c = "FYYFHPFYIFBS"
c.chars.map { |x| x.ord - 64 }.map { |x| (x - Ke) % 26}.map { |x| (x + 64).chr }.join
```

Kerckhoff's Principle

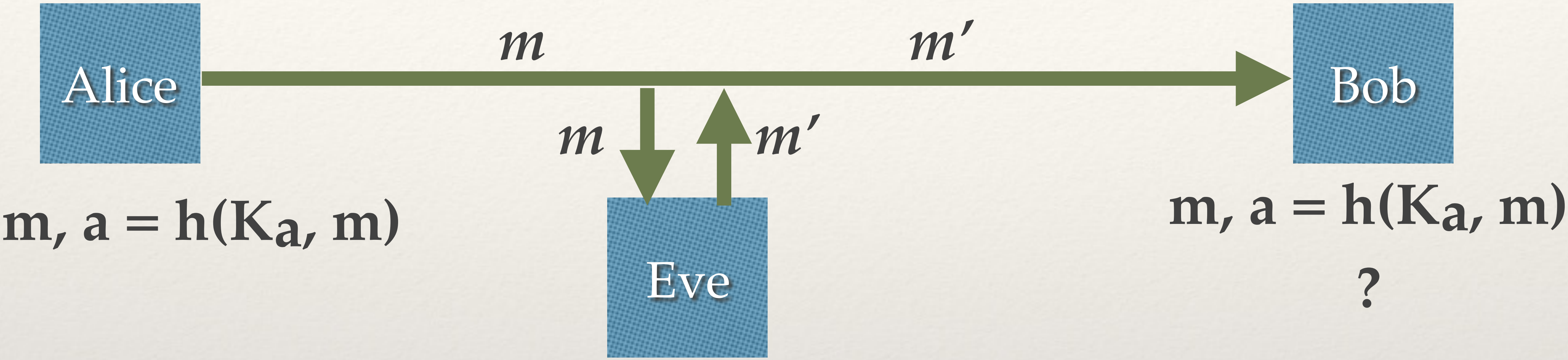
- "The security of a system must depend only on the secrecy of the key, and not the secrecy of the algorithm."
- Security through obscurity is not a valid defense!
- How could you break our Caesar cipher, even if you don't know K_e?

Authentication via Encryption



- Let us assume that Eve has gained the ability to modify traffic in-transit, or send a different message entirely
- How can Bob verify that Alice sent the received message (i.e., that it is message m from Alice and not m' from Eve)?

Authentication via Symmetric-Key Cryptography

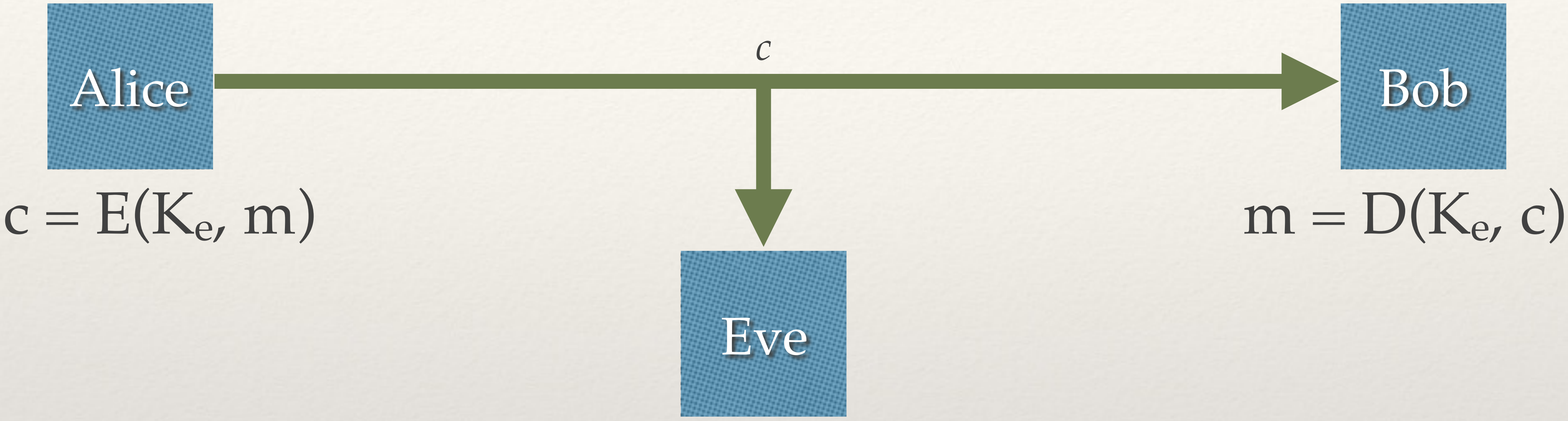


- Assume Alice and Bob share another secret key, K_a and know an authentication function $h(K_a, m)$
- Alice sends the message *m* along with a message authentication code (MAC), *a*
- When Bob receives *m*, if $h(K_a, m)$ does not return a, he will know that it was not "signed" with key K_a

Combining Symmetric-Key Encryption and Authentication

- You can combine encryption and authentication (by including the MAC a inside the ciphertext c) and be able to:
  - Prove that a message came from a specific person
  - Prevent others from reading that message

Symmetric-Key Encryption Weakness

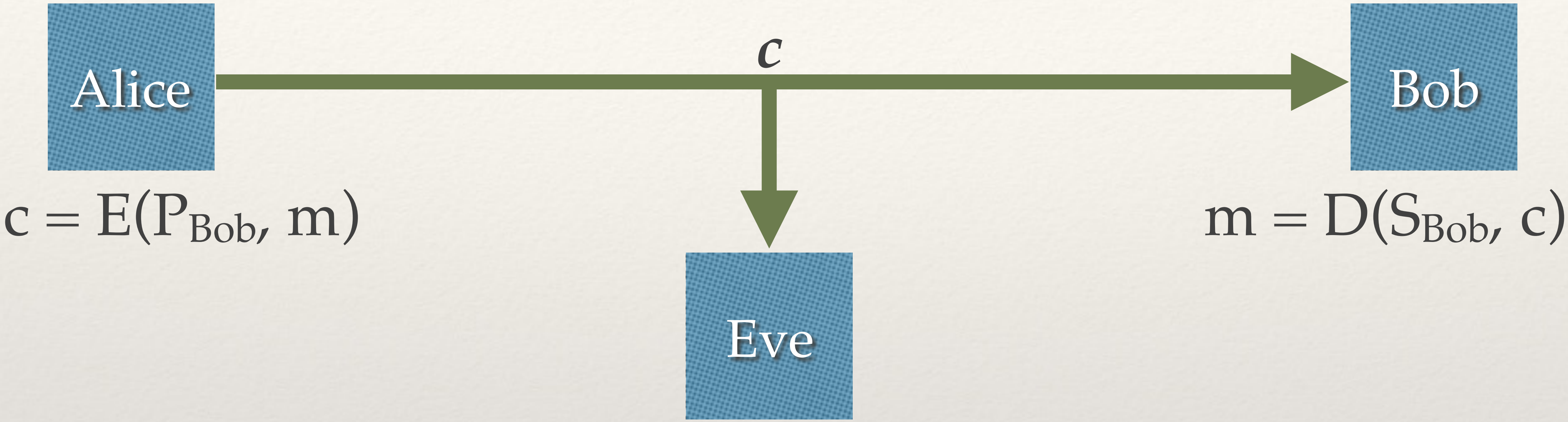


- How do Alice and Bob share keys K_e or K_a? They will need a separate, secure channel.
- But if they have a separate, secure channel, why use an insecure one? (Note: there are actual reasons!)

Asymmetric Key (aka Public-Key) Encryption

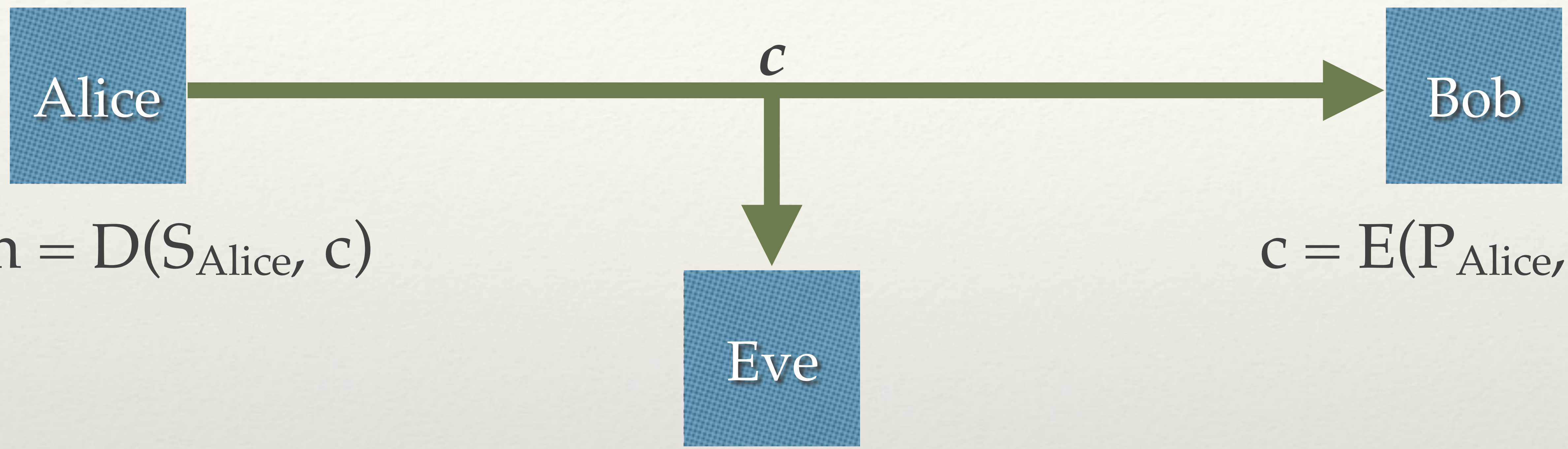
- Two different keys: one to encrypt, *P*, a separate one to decrypt, *S*
- *P* is your public key - anyone can encrypt a message to you with it
- *S* is your private key - you can use it along with P to decrypt a message (as its name implies, you should keep it private!)

Public-Key Encryption Fundamentals



- Bob tells the world about P_{Bob} but keeps S_{Bob} secret
- If Alice wants to communicate with Bob, she can encrypt a message m to ciphertext c with function $E(P_{Bob}, m)$
- The only way to DECRYPT the message is with function $D(S_{Bob}, c)$ which requires knowledge of S_{Bob}

Secure Communication Without Secure Channels



$m = D(S_{Alice}, c)$

$c = E(P_{Alice}, m)$

- Bob can also communicate with Alice by using Alice's public key, P_{Alice}
- Alice and Bob can communicate over an insecure channel, even if all communication is over that channel
- Eve will only ever see ciphertext c

Diffie, W. and M.E. Hellman. "New Directions in Cryptography", IEEE Transactions on Information Theory, Vol 22, No 6, November 1976. https://ee.stanford.edu/~hellman/publications/24.pdf

Public/Private Key Generation

- Public/private keys share a relationship - they are not just two random values
- Bitcoin uses ECDSA (Elliptic Curve Digital Signature Algorithm) to generate your key pair, which we will delve into later this semester
- There are other possibilities (such as RSA)
- For now, just know the concept

Efficiency

- Turns out asymmetric encryption itself is extremely inefficient compared to symmetric encryption
- Most modern systems use a hybrid approach
  - Step 1: Establish a secure communications channel using asymmetric encryption
  - Step 2: Share a symmetric encryption key
  - Step 3: Use symmetric encryption for further communication

Public Key Infrastructure (PKI)

- How do I know that P_{Bob} is actually Bob's key and not Eve's? Very important for websites/users - make sure you are on amazon.com and not amazoon.com!
- Various approaches: certificate authorities, web of trust, SPKI, even some blockchain-based approaches like Emercoin and Civic
- In Bitcoin, no built-in PKI - you have the key, you have the bitcoin associated with that account
- "Not your keys, not your coins"

One-Way Functions

- A **one-way function** (or **trapdoor function**) is a function $y = f(x)$ where, given y, it is computationally infeasible to calculate x, but given x, it is possible to calculate y
- That is, while $y = f(x)$ is (relatively) easy to calculate, its inverse $x = f'(y)$, is much more difficult, or even impossible, to calculate

One-Way Function Example

- Assume function f is simply the "square" function
- Calculating y = f(27) is relatively simple, even by hand: 27 * 27 = 729
- However, calculating its inverse ("square root") x = f'(729) is much more difficult
- Obviously, computers have harder algorithms than square/square root (Bitcoin makes special use of SHA256, which we will cover later) but the idea is the same

Proving a One-Way Function

- One-way functions (especially implemented as hashes) will be very important to Bitcoin
- Useful for proving that a hard computation has been done without revealing interior details or secret values