CS 114:
Network Security

Lecture 4 - Hashes and Message Authentication

Prof. Daniel Votipka
Spring 2023

(some slides courtesy of Prof. Patrick McDaniel and Prof. Micah Sherr)

Administrivia

- Homework 0 grades are up, everyone did great!
- Homework 1, part 1 due Feb. 2nd at 11:59pm
  - Updated output to provide more information
  - Incorrect output formatting (don't add new lines; use sys.stdout.write(), not print())
  - "Address already in use" error means you didn't close your socket correctly

We have another amazing TA!

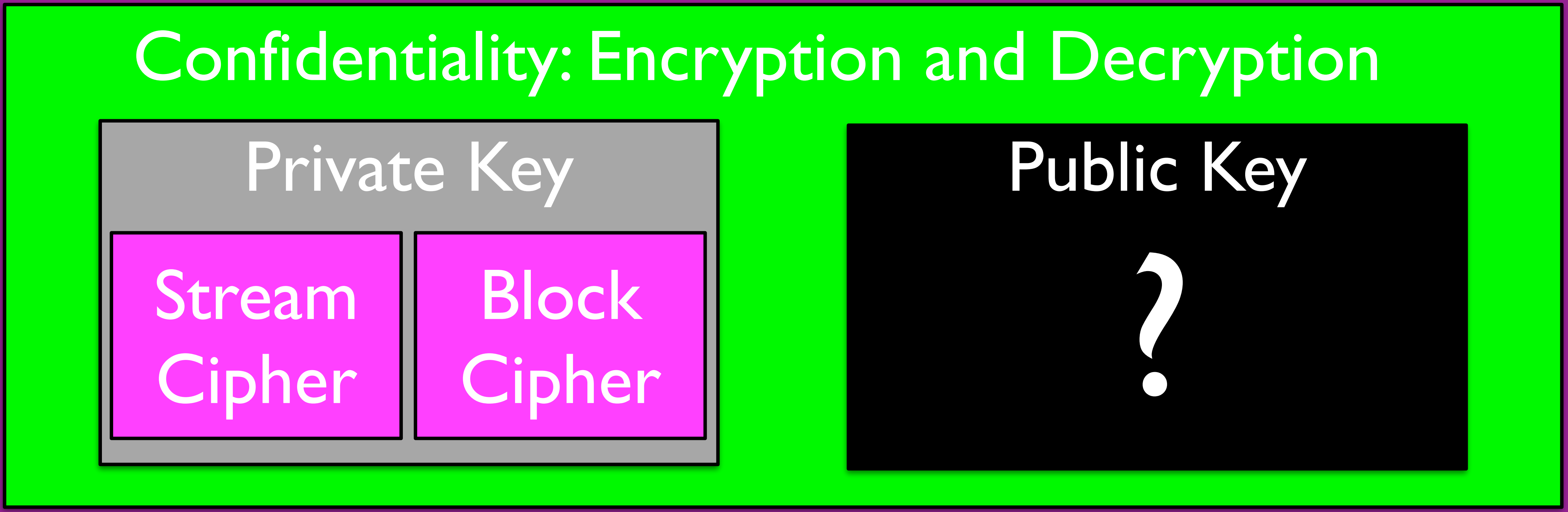
Andrew Vu

OH: W/F 12-1pm, room 359

Guest Lecture

- Ariana Miran, UCSD (4/11)
    - Hack for Hire
    - https://arianamirian.com/

Crypto



Confidentiality: Encryption and Decryption

Private Key

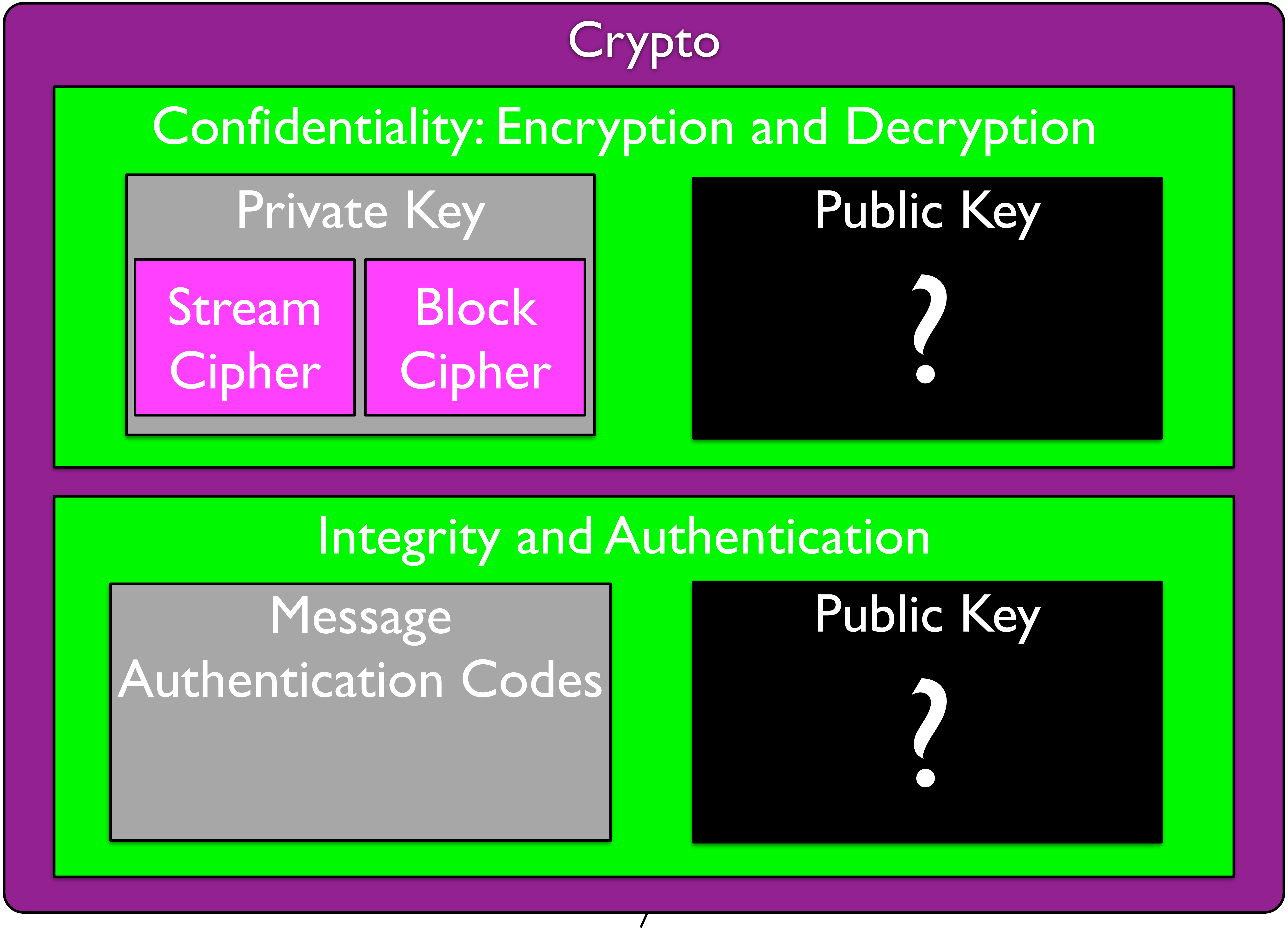
- Stream Cipher
- Block Cipher

Public Key

?

What encryption does and does not

- Does:
  - confidentiality
- Doesn’t do:
  - data integrity
  - source authentication
- **Need:** ensure that data is not altered and is from an authenticated source



Crypto

Confidentiality: Encryption and Decryption

Private Key

Stream Cipher

Block Cipher

Public Key

?

Integrity and Authentication

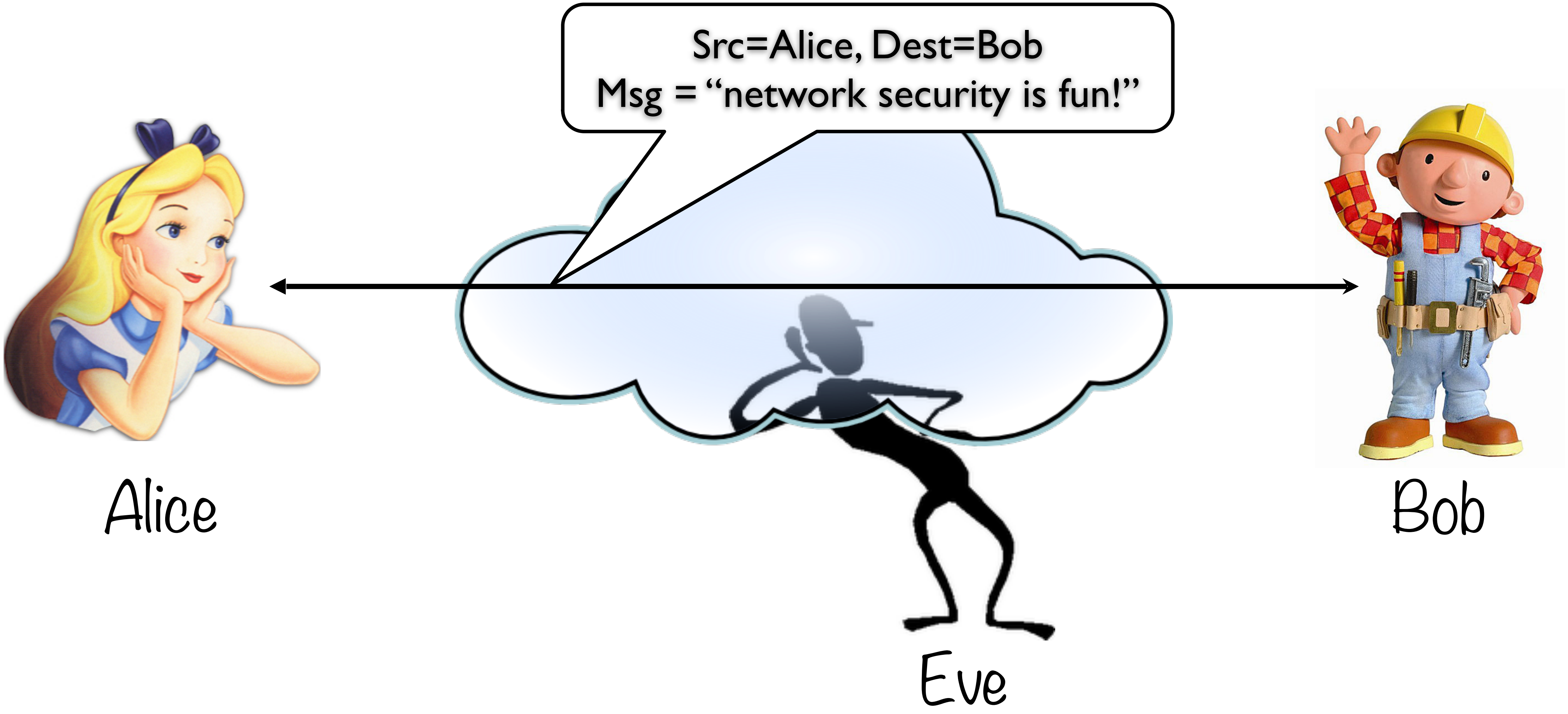
Message Authentication Codes

Public Key

?

Message Authentication Codes

Principals



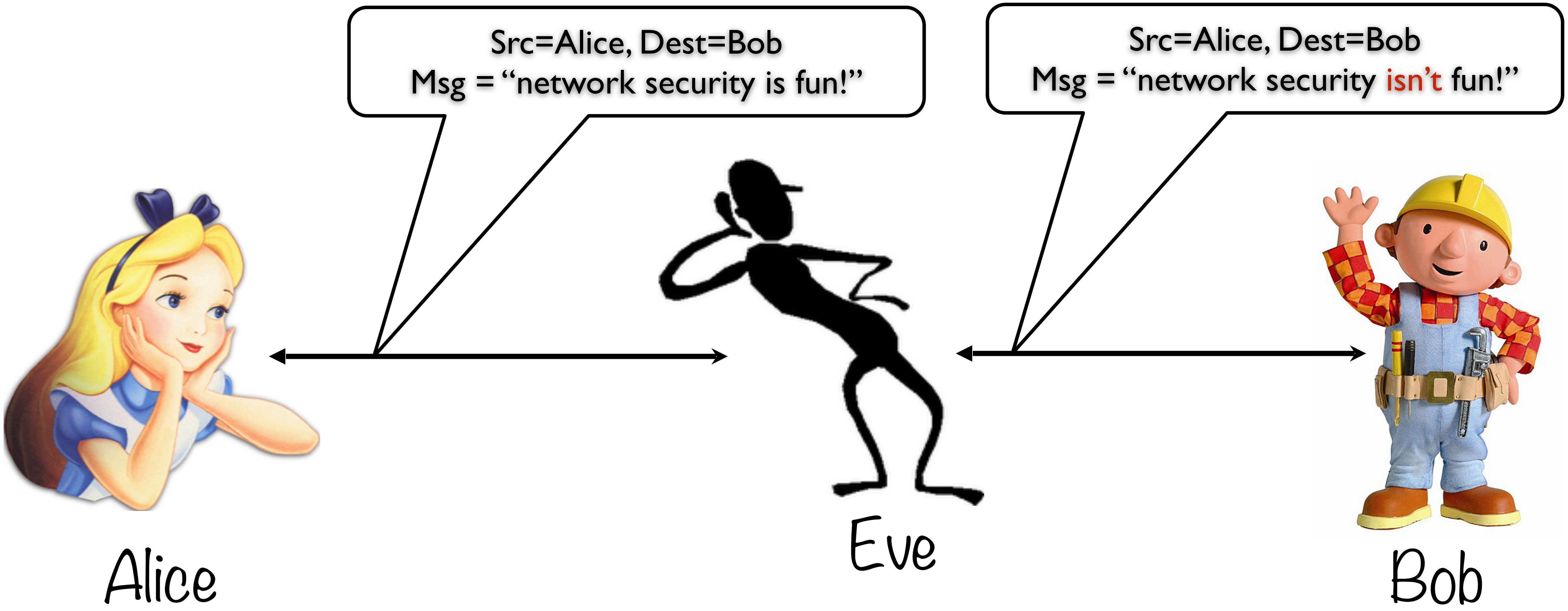
Src=Alice, Dest=Bob
Msg = "network security is fun!"

Alice

Bob

Eve

Man-in-the-Middle (MitM) attack



Src=Alice, Dest=Bob
Msg = "network security is fun!"

Src=Alice, Dest=Bob
Msg = "network security isn't fun!"

Alice

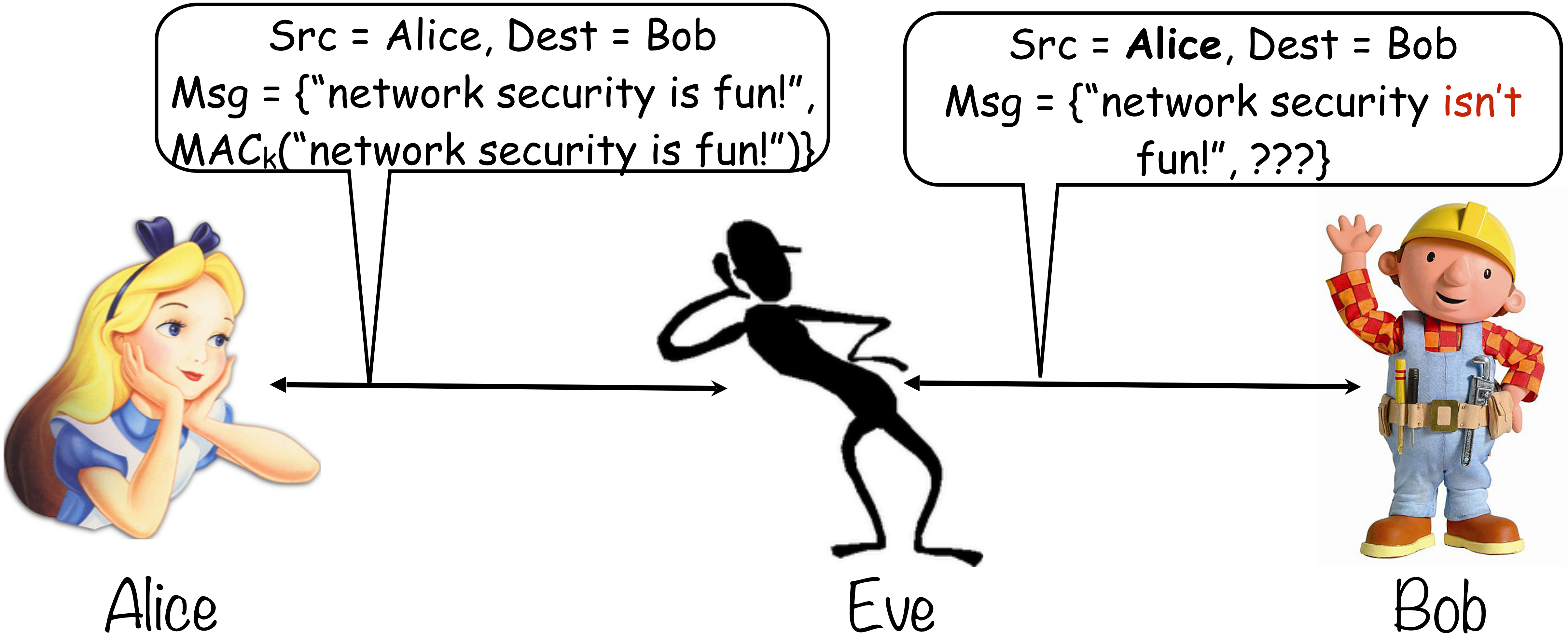
Eve

Bob

Message Authentication Codes (MACs)

- MACs provide message **integrity** and **authenticity**
- $MAC_K(M)$ – use symmetric encryption to produce short sequence of bits that depends on both the message (M) and the key (K)
- MACs should be resistant to **existential forgery**: Eve should not be able to produce a valid MAC for a message M' without knowing K

Message Integrity/Authenticity



Src = Alice, Dest = Bob
Msg = {“network security is fun!”, MAC_k(“network security is fun!”)}

Src = **Alice**, Dest = Bob
Msg = {“network security isn’t fun!”, ???}

Alice

Eve

Bob

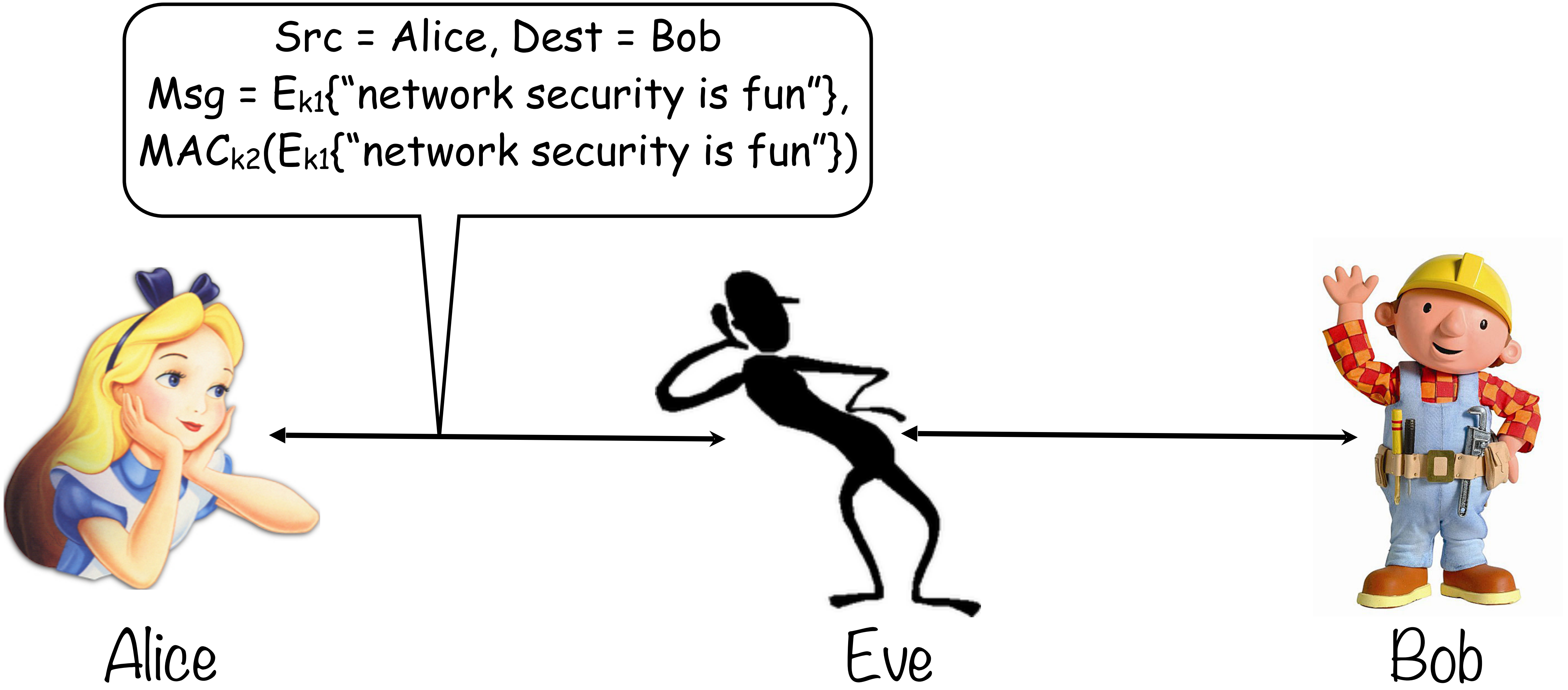
Without knowledge of *k*, Eve can’t compute a valid MAC for her forged message!

Message Authentication Codes (MACs)

- MACs provide message **integrity** and **authenticity**
- $MAC_K(M)$ – use symmetric encryption to produce short sequence of bits that depends on both the message (M) and the key (K)
- MACs should be resistant to **existential forgery**: Eve should not be able to produce a valid MAC for a message M' without knowing K
- To provide confidentiality, authenticity, and integrity of a message, Alice sends
  - MAC-then-Encrypt: $E_K(M,MAC_K(M))$ where $E_K(X)$ is the encryption of X using key K; or
  - Encrypt-then-MAC: $E_K(M),MAC_K(E_K(M))$

    or
  - Encrypt-and-MAC: $E_K(M),MAC_K(M)$
  - Proves that M was encrypted (confidentiality) by someone who knew K (authenticity) and hasn't been changed (integrity)

Encryption + Message Integrity/Authenticity



Src = Alice, Dest = Bob
Msg = E_{k1}\{"network security is fun"\},
MAC_{k2}(E_{k1}\{"network security is fun"\})

**Without knowing *k1*,
Eve can't read Alice's message.**

Without knowing *k2*, Eve can't compute a valid MAC for her forged message!

Message Authentication Codes (MACs)

- MACs provide message **integrity** and **authenticity**
- $\text{MAC}_K(M)$ – use symmetric encryption to produce short sequence of bits that depends on both the message (M) and the key (K)
- MACs should be resistant to **existential forgery**: Eve should not be able to produce a valid MAC for a message M' without knowing K
- To provide confidentiality, authenticity, and integrity of a message, Alice sends
  - MAC-then-Encrypt: $E_K(M,\text{MAC}_K(M))$ where $E_K(X)$ is the encryption of X using key K; or
  - Encrypt-then-MAC: $E_K(M),\text{MAC}_K(E_K(M))$ ← Best option

    or
  - Encrypt-and-MAC: $E_K(M),\text{MAC}_K(M)$
  - Proves that M was encrypted (confidentiality) by someone who knew K (authenticity) and hasn't been changed (integrity)

Crypto

Confidentiality: Encryption and Decryption

- Private Key
  - Stream Cipher
  - Block Cipher
- Public Key
  - ?

Integrity and Authentication

- Message Authentication Codes
  - Crypto Hash
- Public Key
  - ?

Cryptographic Hash

Cryptographic Hash Functions

- **Hash function** `h`: deterministic one-way function that takes as input an arbitrary message `M` (sometimes called a *preimage*) and returns as output `h(M)`, a small fixed length *hash* (sometimes called a *digest*)
- Hash functions should have the following two properties:
  - *compression*: reduces arbitrary length string to fixed length hash
  - *ease of computation*: given message `M`, `h(M)` is easy to compute

Cryptographic Hash Functions

- Properties of good cryptographic hash functions:
  - **preimage resistance:** given digest y, computationally infeasible to find preimage x' such that h(x')=y
  - **2nd-preimage resistance:** given preimage x, computationally infeasible to find preimage x' such that h(x)=h(x')
  - **collision resistance:** computationally infeasible to find preimages i,j such that h(i)=h(j)

Demo

Hash functions are usually fairly inexpensive

(i.e., compared with public key cryptography)

```
[dvotipka@NotLinux 05:30 PM] ~> openssl speed sha
Doing sha1 for 3s on 16 size blocks: 4470649 sha1's in 3.00s
Doing sha1 for 3s on 64 size blocks: 3442313 sha1's in 2.99s
Doing sha1 for 3s on 256 size blocks: 2040819 sha1's in 3.00s
Doing sha1 for 3s on 1024 size blocks: 773189 sha1's in 3.00s
Doing sha1 for 3s on 8192 size blocks: 114222 sha1's in 3.00s
...
Doing sha512 for 3s on 16 size blocks: 2849624 sha512's in 2.99s
Doing sha512 for 3s on 64 size blocks: 2837564 sha512's in 3.00s
Doing sha512 for 3s on 256 size blocks: 1281416 sha512's in 3.00s
Doing sha512 for 3s on 1024 size blocks: 481337 sha512's in 3.00s
Doing sha512 for 3s on 8192 size blocks: 71397 sha512's in 3.00s
OpenSSL 1.0.0d 8 Feb 2011
built on: Tue Feb 15 16:03:54 EST 2011
options:bn(64,64) rc4(ptr,char) des(idx,cisc,16,int) aes(partial) idea(int) blowfish(idx)
compiler: /usr/bin/gcc-4.2 -fPIC -fno-common -DOPENSSL_PIC -DZLIB -DOPENSSL_THREADS
-D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -arch x86_64 -O3 -DL_ENDIAN -DMD32_REG_T=int -Wall
The 'numbers' are in 1000s of bytes per second processed.
type             16 bytes     64 bytes    256 bytes   1024 bytes   8192 bytes
sha1             23843.46k    73681.62k   174149.89k   263915.18k   311902.21k
sha256           18572.85k    47224.32k    89395.29k   115009.19k   125728.09k
sha512           15248.82k    60534.70k   109347.50k   164296.36k   194961.41k
```

How do we use crypto to make a MAC?

- $MAC_K(M) = h(M|K)$
  - Only computable if you know K
  - Any change in data will cause change in hash

Birthday Attack

- **Birthday Paradox:** chances that 2+ people share birthday in group of 23 > 50%.
- General formulation
  - function f() whose output is uniformly distributed over H possible outputs
  - Number of experiments Q(H) until we find a collision is approximately:

$$Q(H) \approx \sqrt{\frac{\pi}{2} H}$$

  - E.g.,

$$Q(365) \approx \sqrt{\frac{\pi}{2} 365} = 23.94$$

- Why is this relevant to hash sizes?

Some common cryptographic hash functions

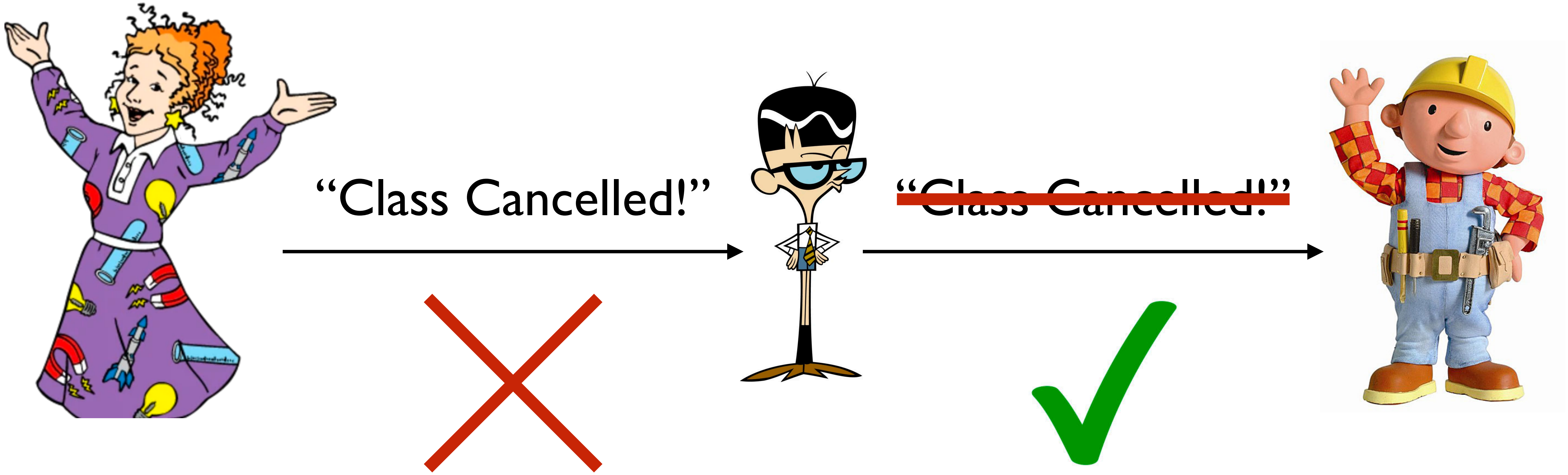
- MD5 (128-bit digest) [don’t use this]
- SHA-1 (160-bit digest) [don’t use this]

- SHA-256 (256-bit digest)
- SHA-512 (512-bit digest)
- ...

Using hashes as authenticators

Using hashes as authenticators

- Consider the following scenario
  - Prof. Frizzle has not decided if she will cancel the next lecture.
  - When she does decide, she communicates to Bob the student through Mandark, her evil TA.
  - Prof. Frizzle does not trust Mandark to deliver the message.
  - She does not care if Bob shows up to a cancelled class, but she does not want students to not show up if the class hasn't been cancelled



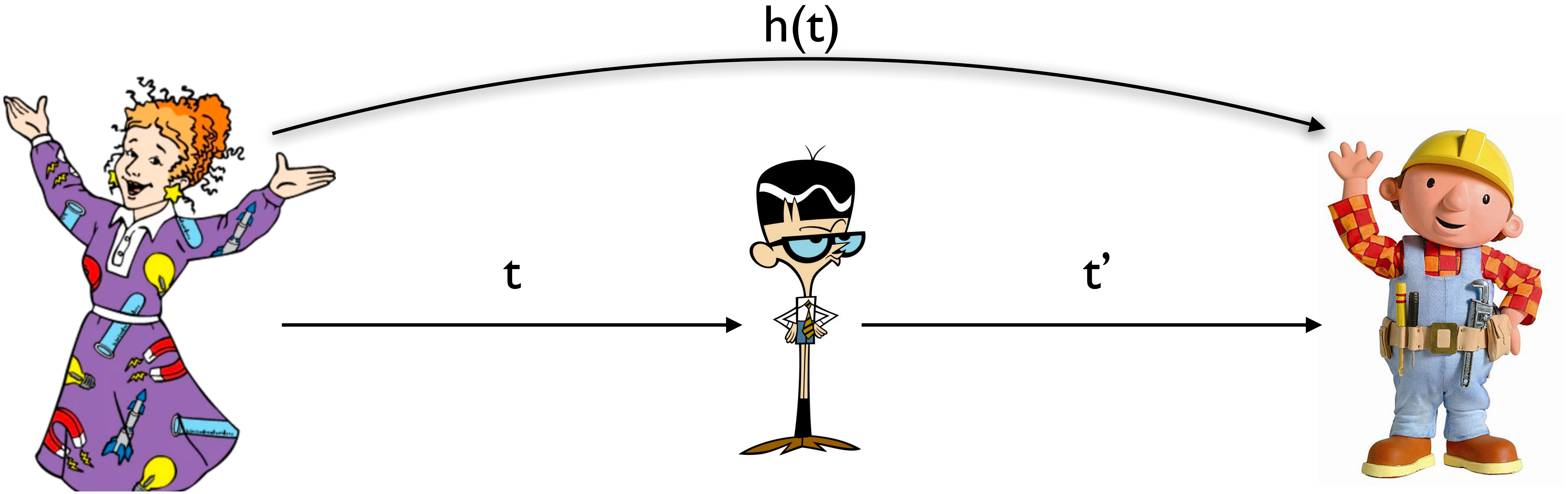
“Class Cancelled!”

✗

~~“Class Cancelled!”~~

✓

Using hashes as authenticators



h(t)

t

t’

- Prof. Frizzle and Bob use the following protocol:
  - Prof. Frizzle invents a secret t
  - Prof. Frizzle gives Bob h(t), where h() is a crypto hash function
  - If she cancels class, she gives t to Mandark to give to Bob
  - If she does not cancel class, she does nothing
  - If Bob receives the token t, he knows that Prof. Frizzle sent it

Hash chain

- Now, consider the case where Prof. Frizzle wants to do the same protocol, only for all 26 classes (the semester)
- Prof. Frizzle and Bob use the following protocol:

1. Prof. Frizzle invents a secret t
2. She gives Bob $H^{26}(t)$, where $H^{26}()$ is 26 repeated uses of $H()$.
3. If she cancels class on day d, she gives $H^{(26-d)}(t)$ to Mandark, e.g.,

   If cancels on day 1, she gives Mandark $H^{25}(t)$

   If cancels on day 2, she gives Mandark $H^{24}(t)$

   …….

   If cancels on day 25, she gives Mandark $H^{1}(t)$

   If cancels on day 26, she gives Mandarks t
4. If Prof. Frizzle does not cancel class, she does nothing

– If Bob receives the token t, he knows that Prof. Pants sent it

Hash Chain (cont.)

- Why is this protocol secure?
  - On day d, $H^{(26-d)}(t)$ acts as an authenticated value (authenticator) because Mandark could not create t without inverting H() because for any $H^k(t)$ she has $k>(26-d)$
  - That is, Mandark potentially has access to the hash values for all days prior to today, but that provides no information on today's value, as they are all post-images of today's value
  - Note: Mandark can again convince Bob that class is occurring by not delivering $H^{(26-d)}(t)$
  - Chain of hash values are ordered authenticators
- Important that Bob got the original value $H^{26}(t)$ from Prof. Pants directly (was provably authentic)

Prof. Pedantic decides to use SHA256 to **authenticate** messages

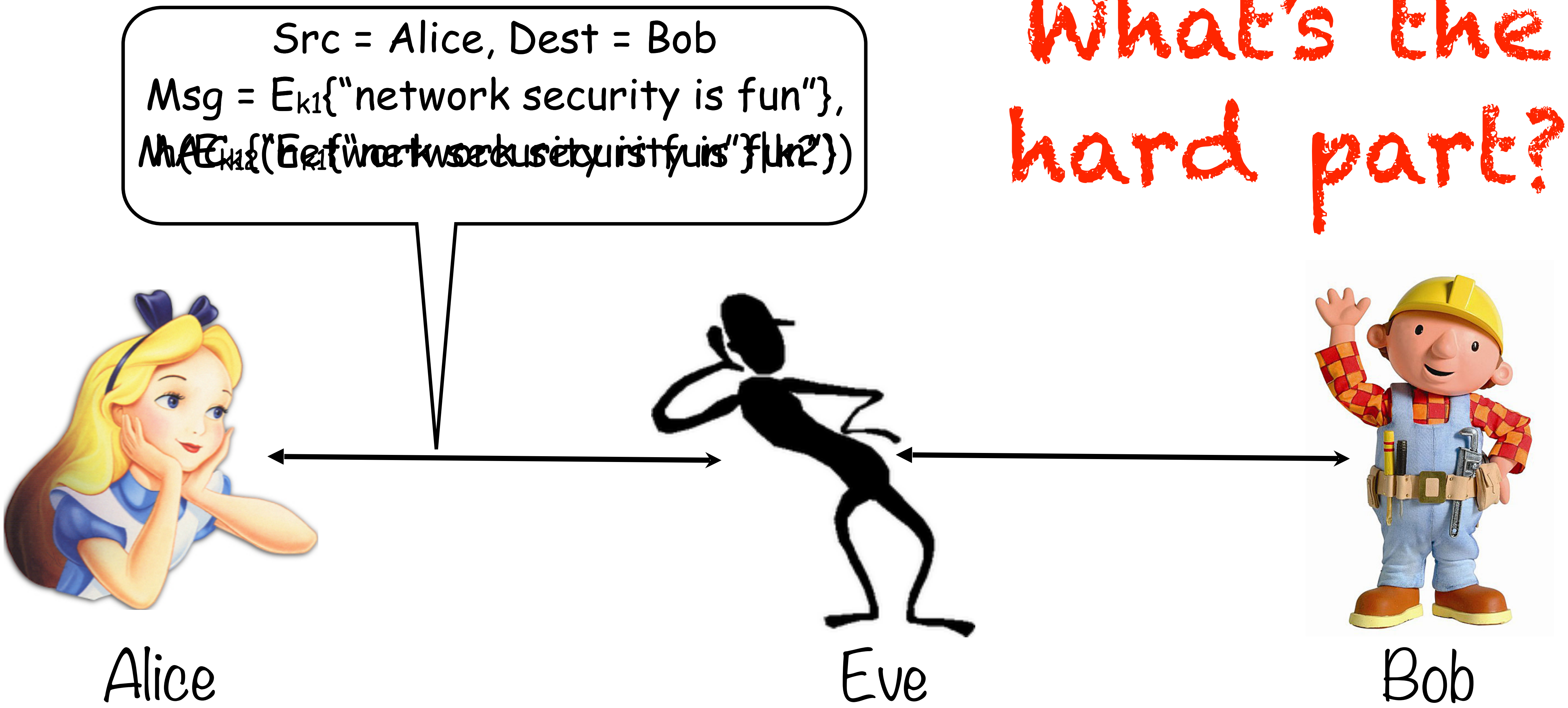
- Protocol:
  - Sender:
    - Input: message `M`
    - Output: `M | SHA256(M)`
  - Receiver:
    - Input: `M | SHA256(M)` (from the Sender)
    - Computes hash over M and checks that it matches value from sender

why is this terrible?
...and how can it be improved?

Let’s Review!

Encryption + Message Integrity/Authenticity

What's the hard part?



Src = Alice, Dest = Bob
Msg = E_{k1}\{"network security is fun"\},
MAC_{k2}(E_{k1}\{"network security is fun"\})
h(E_{k1}\{"network security is fun"\}|k2)

Without knowing *k1*, Eve can't read Alice's message.

Without knowing *k2*, Eve can't compute a valid MAC for her forged message.