COS433/Math 473: Cryptography

Mark Zhandry

Princeton University

Spring 2020

Announcements

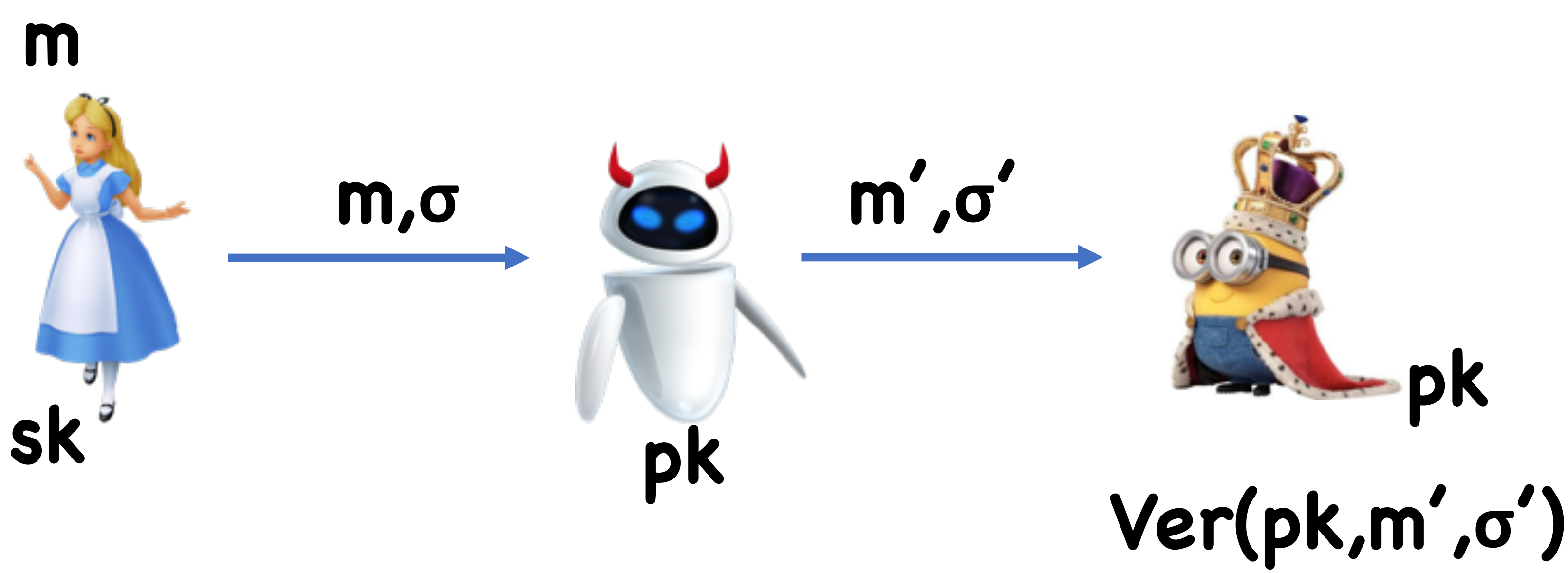
HW6 Due April 23rd
HW7 Due April 30th

Previously on COS 433…

Digital Signatures

(aka public key MACs)

Message Integrity in Public Key Setting



Goal: If Eve changed **m**, Bob should reject

Digital Signatures

Algorithms:

- **Gen() → (sk,pk)**
- **Sign(sk,m) → σ**
- **Ver(pk,m,σ) → 0/1**

Correctness:

Pr[Ver(pk,m,Sign(sk,m))=1: (sk,pk)←Gen()] = 1

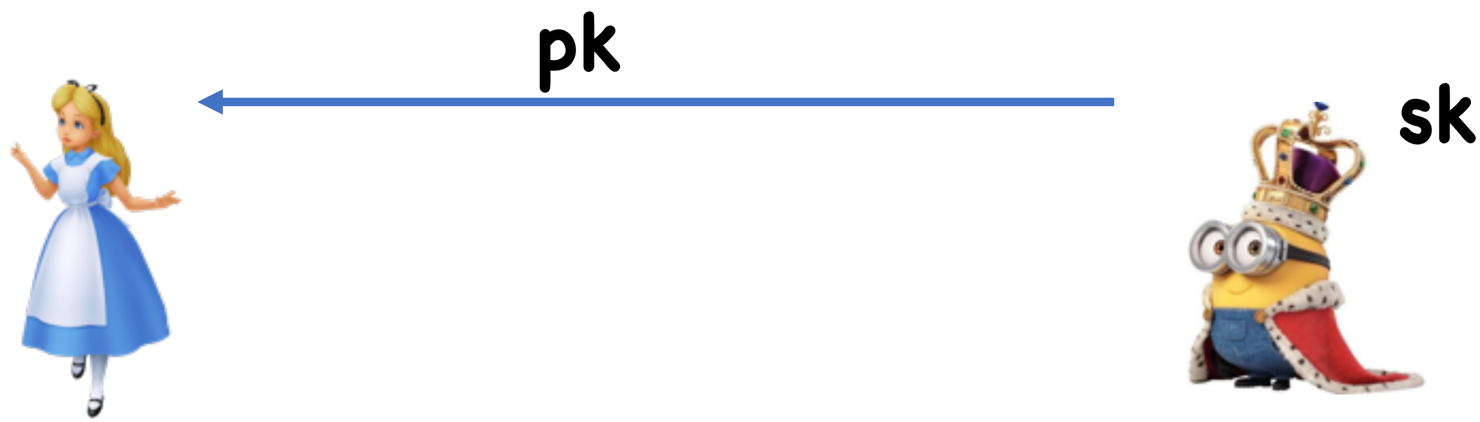
Security Notions?

Much the same as MACs, except adversary gets verification key

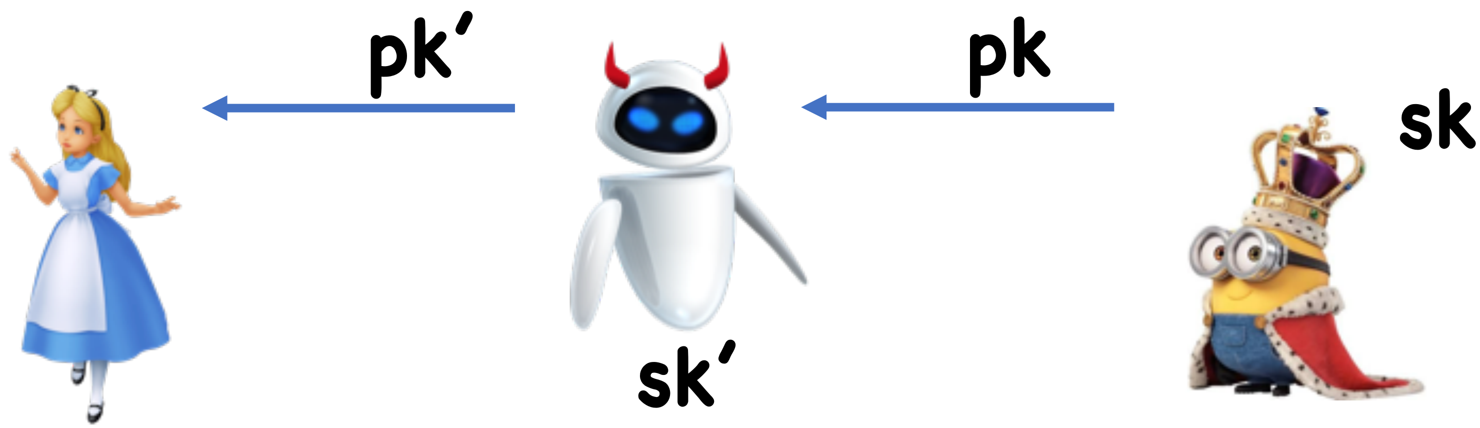
Today

The Public Key Infrastructure

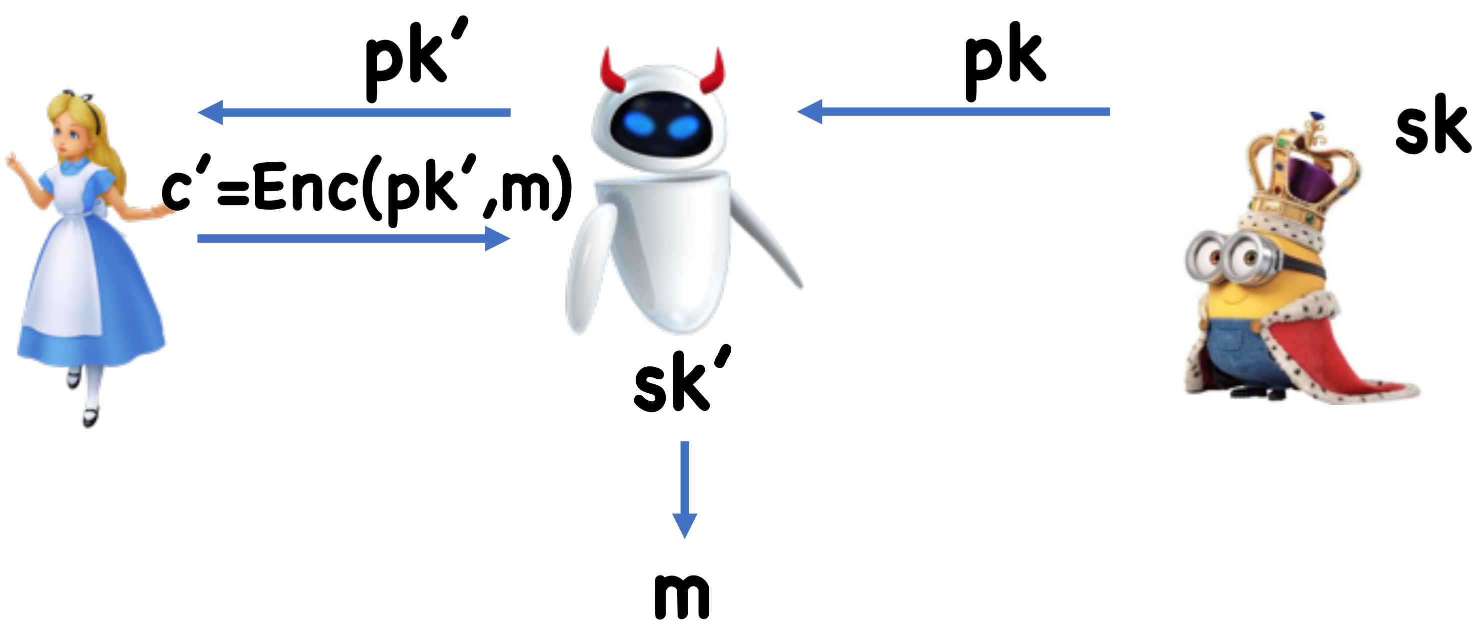
Digital Signatures and the Public Key Infrastructure



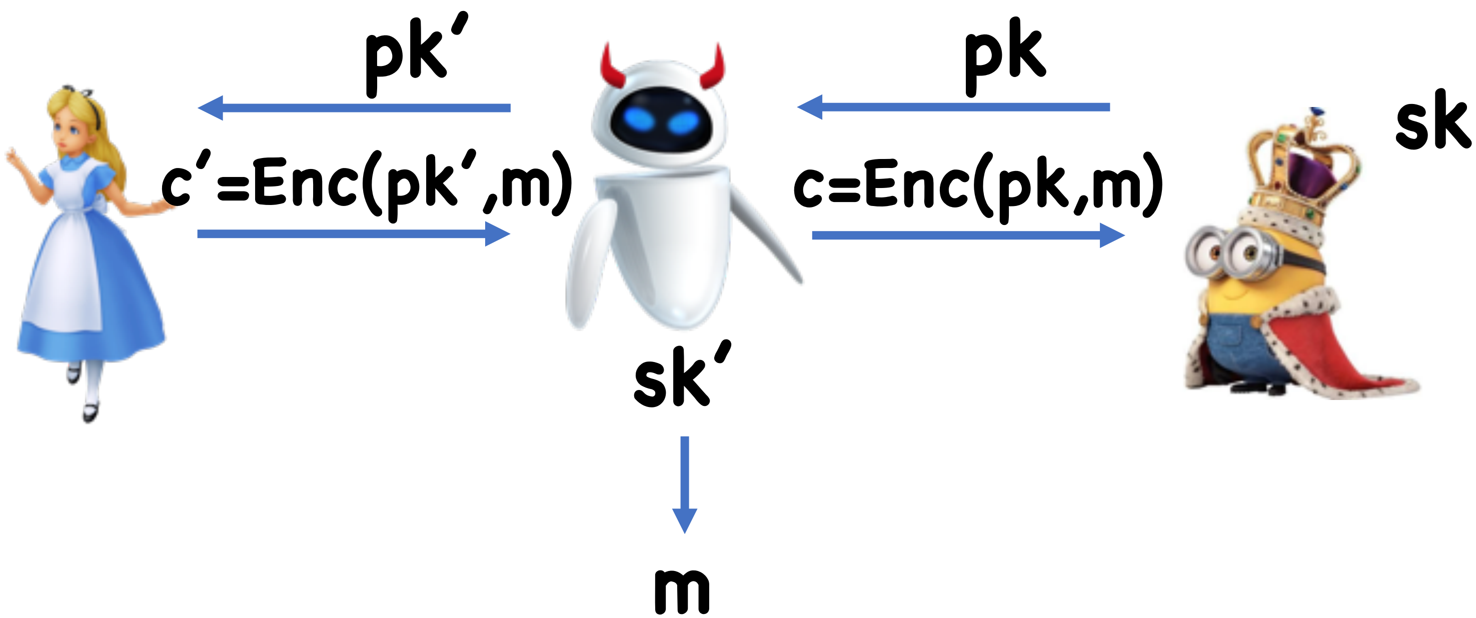
Digital Signatures and the Public Key Infrastructure



Digital Signatures and the Public Key Infrastructure



Digital Signatures and the Public Key Infrastructure

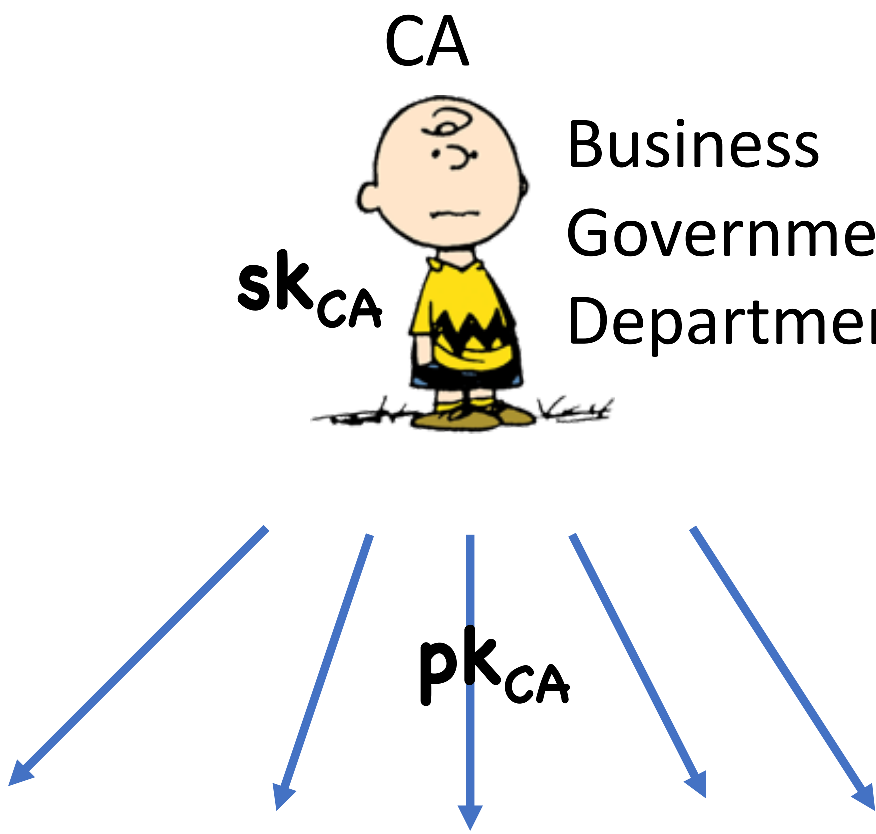


Takeaway

Need some authenticated channel to ensure distribution of public keys

But how to authenticate channel in the first place without being able to distribute public keys?

Solution: Certificate Authorities

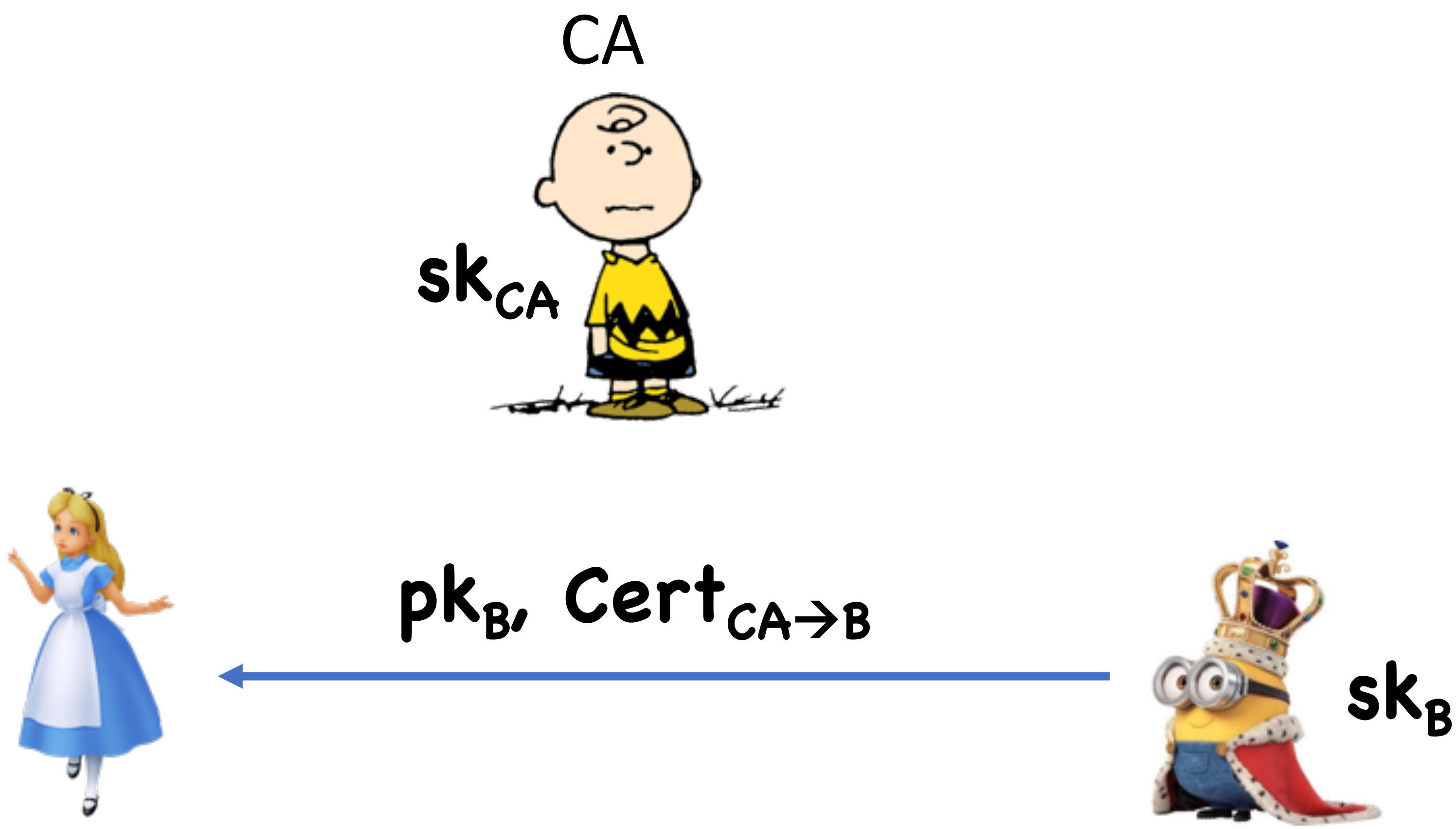


Business

Government Agency

Department within company

Solution: Certificate Authorities



$Cert_{CA \to B}$=Sign(sk_{CA},"Bob's public key is pk_B")

Solution: Certificate Authorities

Bob is typically some website

- Obtains **Cert** by, say, sending someone in person to CA with pk_B
- Only needs to be done once

If Alice trusts CA, then Alice will be convinced that pk_B belongs to Bob

Alice typically gets pk_{CA} bundled in browser

Limitations

Everyone must trust same CA

- May have different standards for issuing certs

Single point of failure: if $\mathbf{sk}_{CA}$ is compromised, whole system is compromised

Single CA must handle all verification

Multiple CAs

There are actually many CA’s, CA_1, CA_2,…

Bob obtains cert from all of them, sends all the certs with his public key

As long as Alice trusts one of the CA’s, she will be convinced about Bob’s public key

Certificate Chaining

CA issues $\textbf{Cert}_{CA \rightarrow B}$ for Bob

Bob can now use his signing key to issue $\textbf{Cert}_{B \rightarrow D}$ to Donald

Donald can now prove his public key by sending $(\textbf{Cert}_{CA \rightarrow B}, \textbf{Cert}_{B \rightarrow D})$

- Proves that CA authenticated Bob, and Bob authenticated Donald

Certificate Chaining

For Bob to issue his own certificates, a standard cert should be insufficient

- CA knows who Bob is, but does not trust him to issue certs on its behalf

Therefore, Bob should have a stronger cert:

$\mathbf{Cert}_{CA\rightarrow B}$**=Sign(**$\mathbf{sk}_{CA}$**,"**Bob's public key is $\mathbf{pk}_B$ and he can issue certificates on behalf of CA**")**

Certificate Chaining

One root CA

Many second level CAs CA_1, CA_2,…

- Each has $\mathbf{Cert}_{CA \rightarrow CA_i}$

Advantage: eases burden on root

Disadvantage: now multiple points of failure

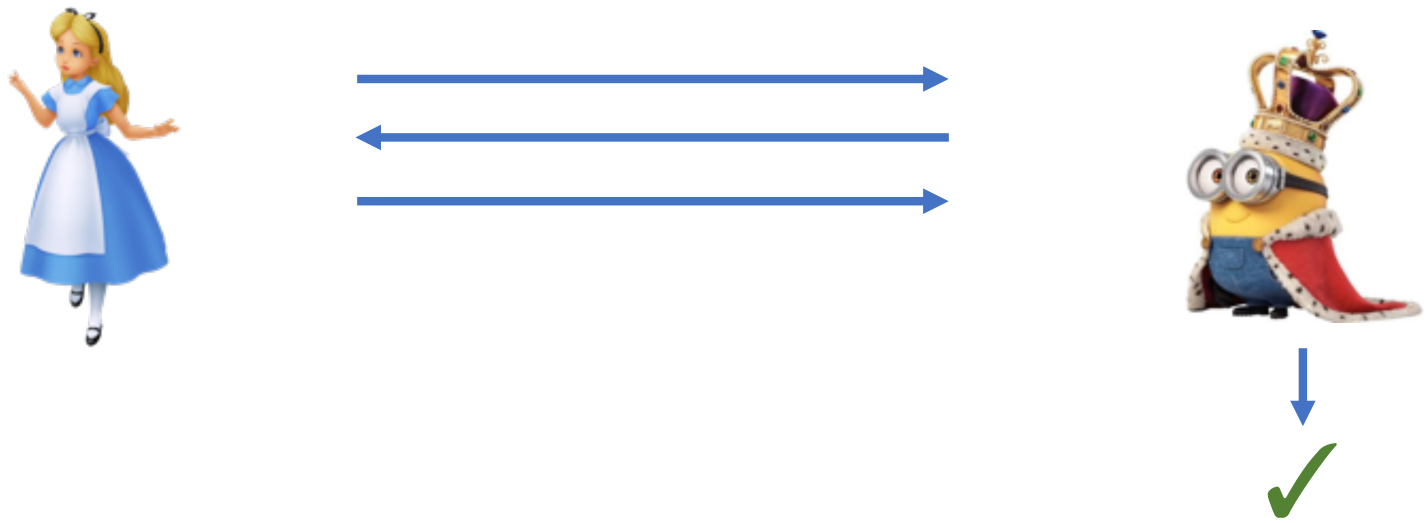
Invalidating Certificates

Sometimes, need to invalidate certificates
- Private key stolen
- User leaves company
- Etc

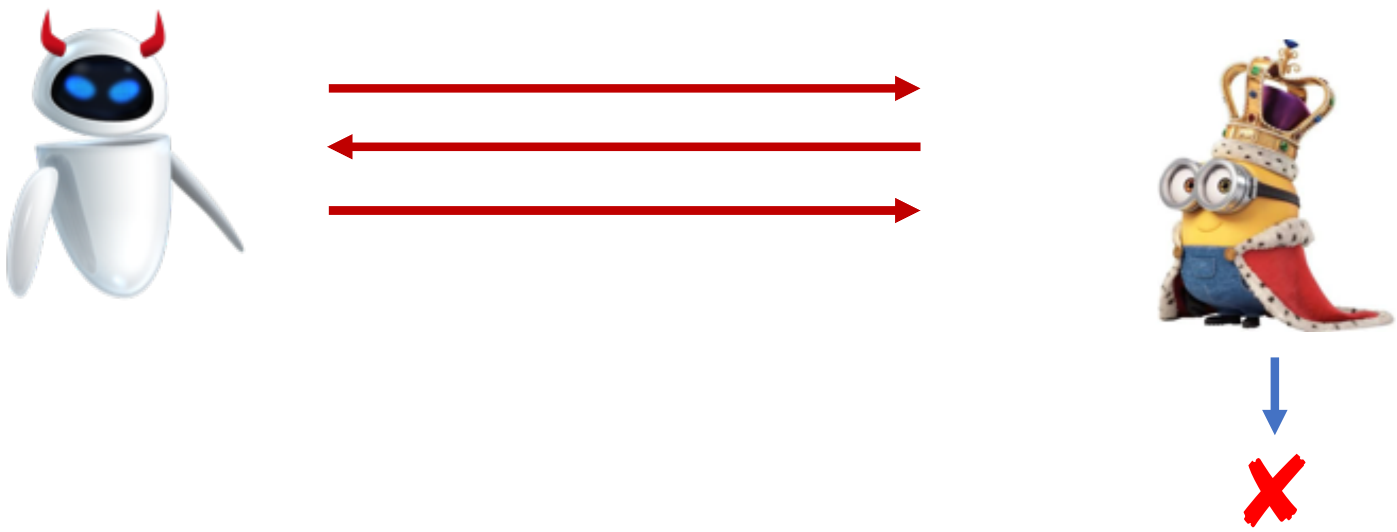
Options:
- Expiration
- Explicit revocation

Identification Protocols

Identification



Identification



Identification

To identify yourself, you need something the adversary doesn't have

Typical factors:

- What you **are**: biometrics (fingerprints, iris scans,…)
- What you **have**: Smart cards, SIM cards, etc
- What you **know**: Passwords, PINs, secret keys

Today

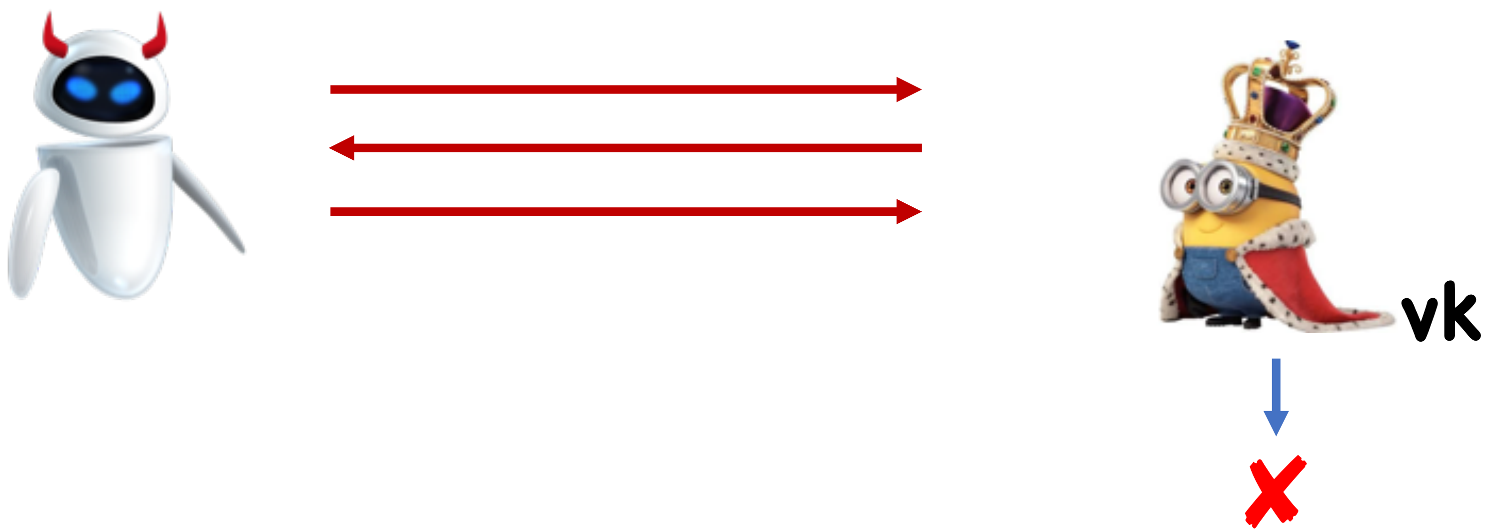
Types of Identification Protocols

Secret key:

Public Key:

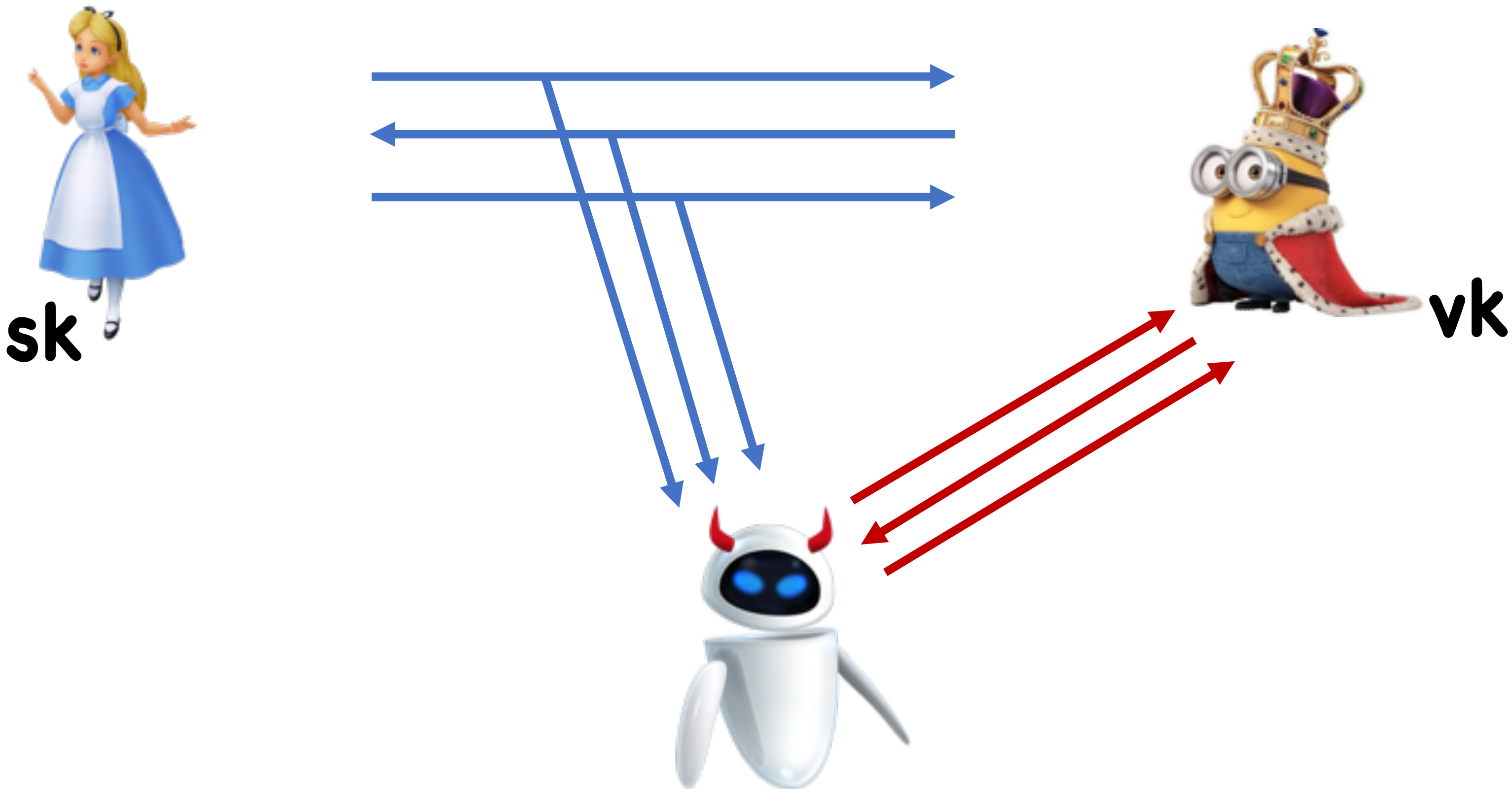
Types of Attacks

Direct Attack:



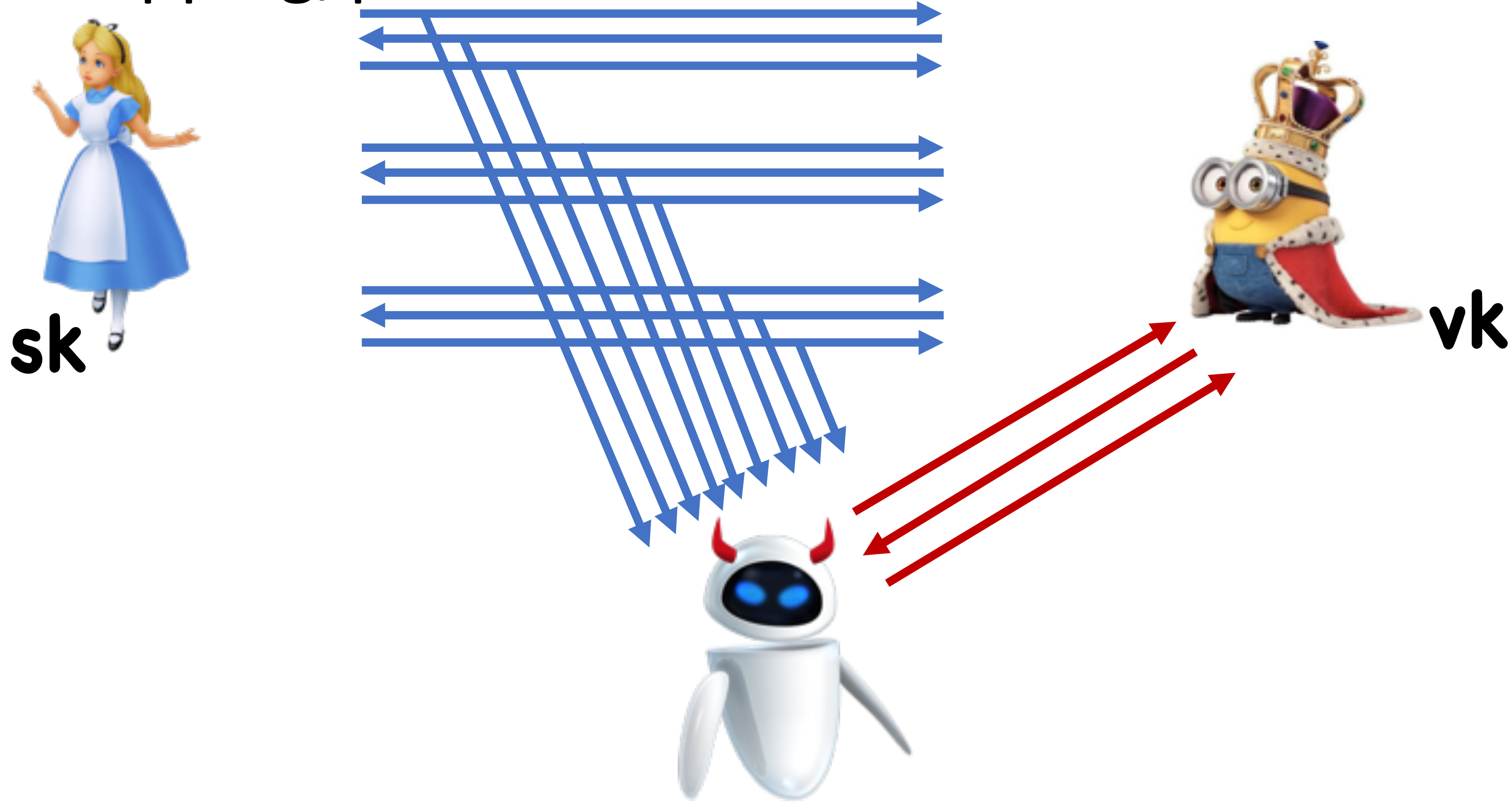
Types of Attacks

Eavesdropping/passive:



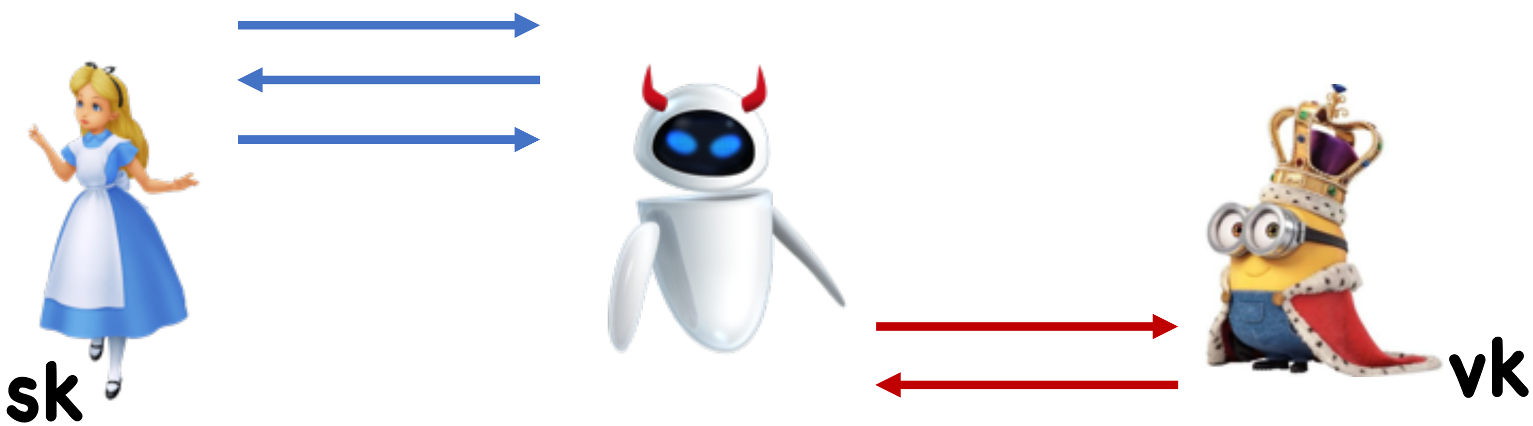
Types of Attacks

Eavesdropping/passive:



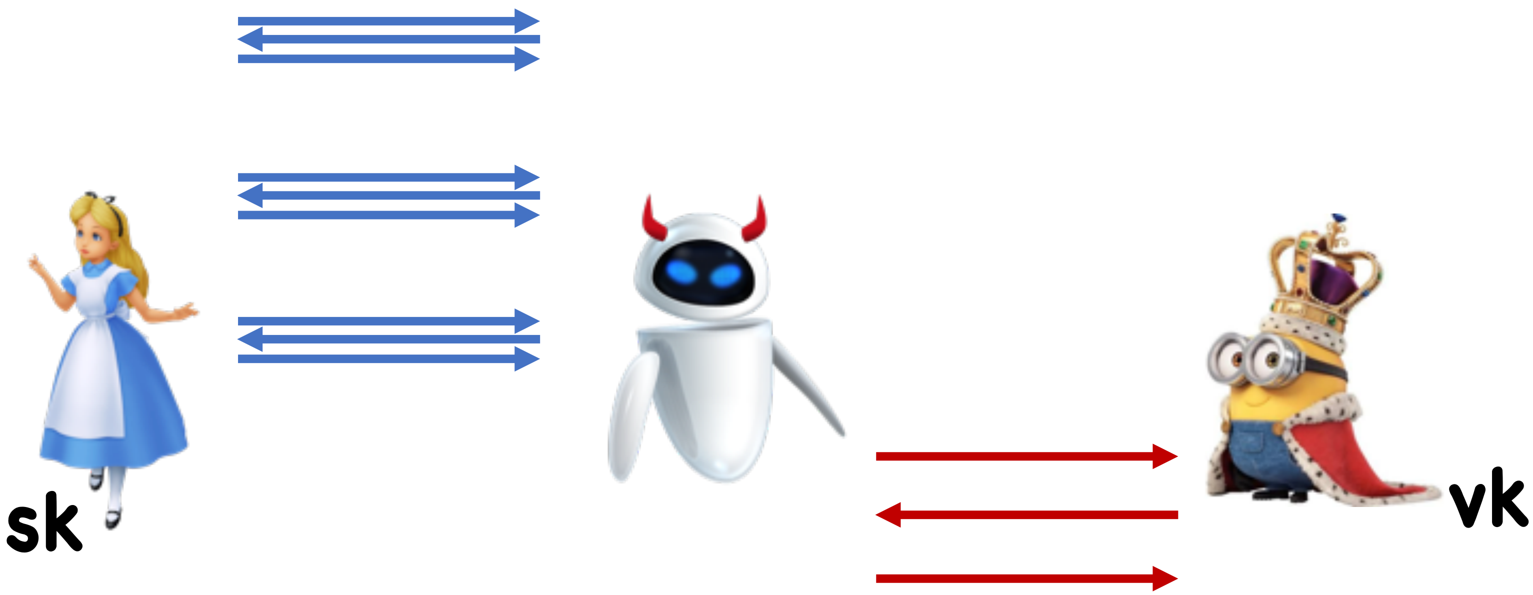
Types of Attacks

Man-in-the-Middle/Active:



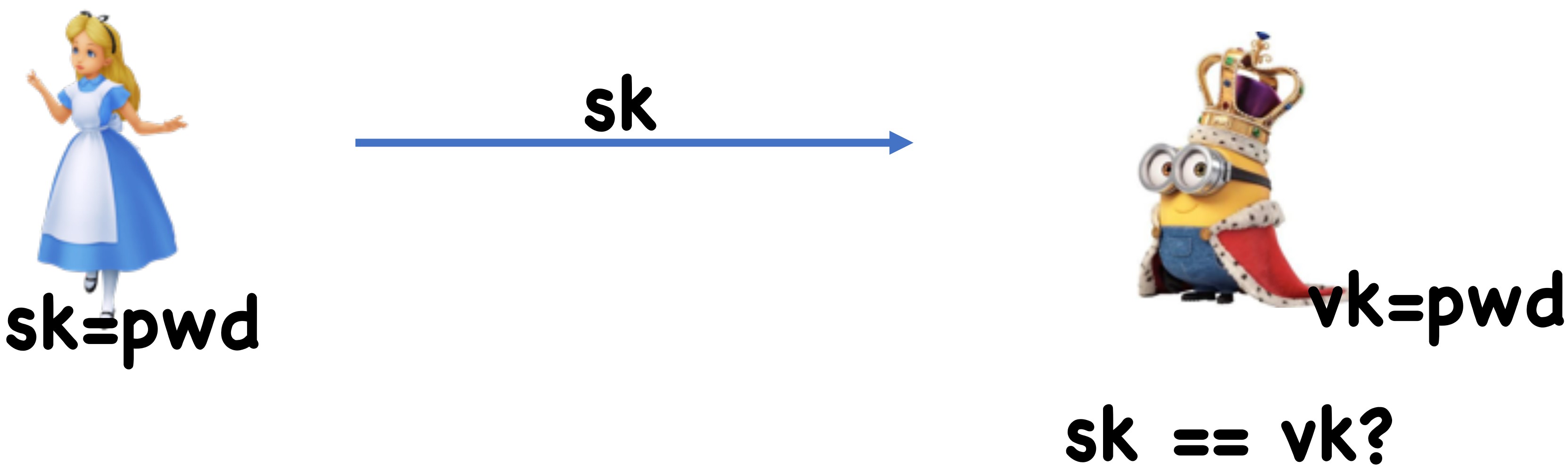
Types of Attacks

Man-in-the-Middle/Active:



Basic Password Protocol

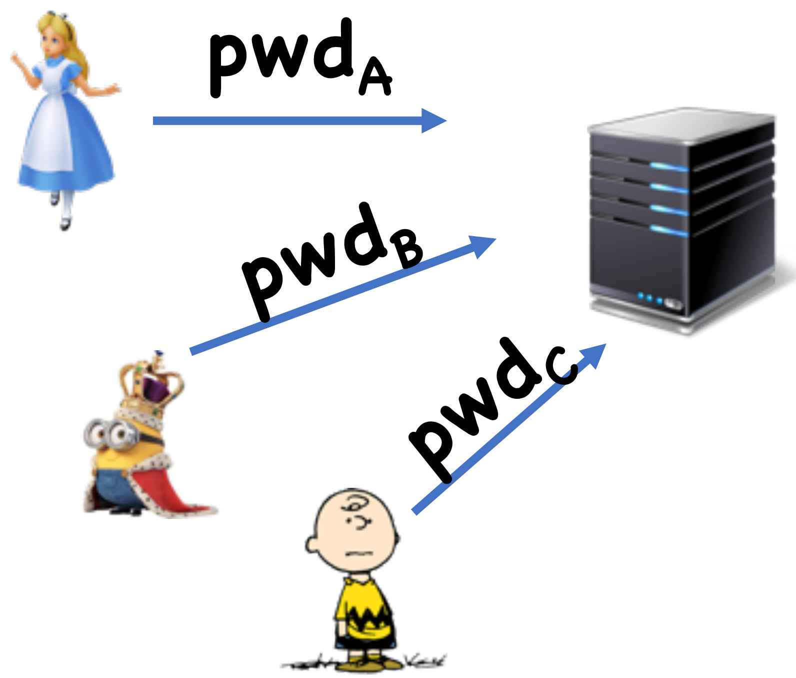
Never ever (ever ever…) use



Problem with Basic Pwd Protocol

vk must be kept secret at all costs

Issue:



User	Pwd
Alice	pwd_A
Bob	pwd_B
Charlie	pwd_C
…	…

Problem with Basic Pwd Protocol

vk must be kept secret at all costs

Issue:

MOTHERBOARD

NEWS

Another Day, Another Hack: 117 Million LinkedIn Emails And Passwords

WIRED

BRIAN BARRETT SECURITY 05.31.16 4:58 PM

HACK BRIEF: YOUR OLD MYSPACE ACCOUNT JUST CAME BACK TO HAUNT YOU

WIRED

LILY HAY NEWMAN SECURITY 08.31.16 1:10 PM

HACK BRIEF: 4-YEAR-OLD DROPBOX HACK EXPOSED 68 MILLION PEOPLE'S DATA

WIRED

ANDY GREENBERG SECURITY 09.22.16 12:15 PM

HACK BRIEF: YAHOO BREACH HITS HALF A *BILLION* USERS

WIRED

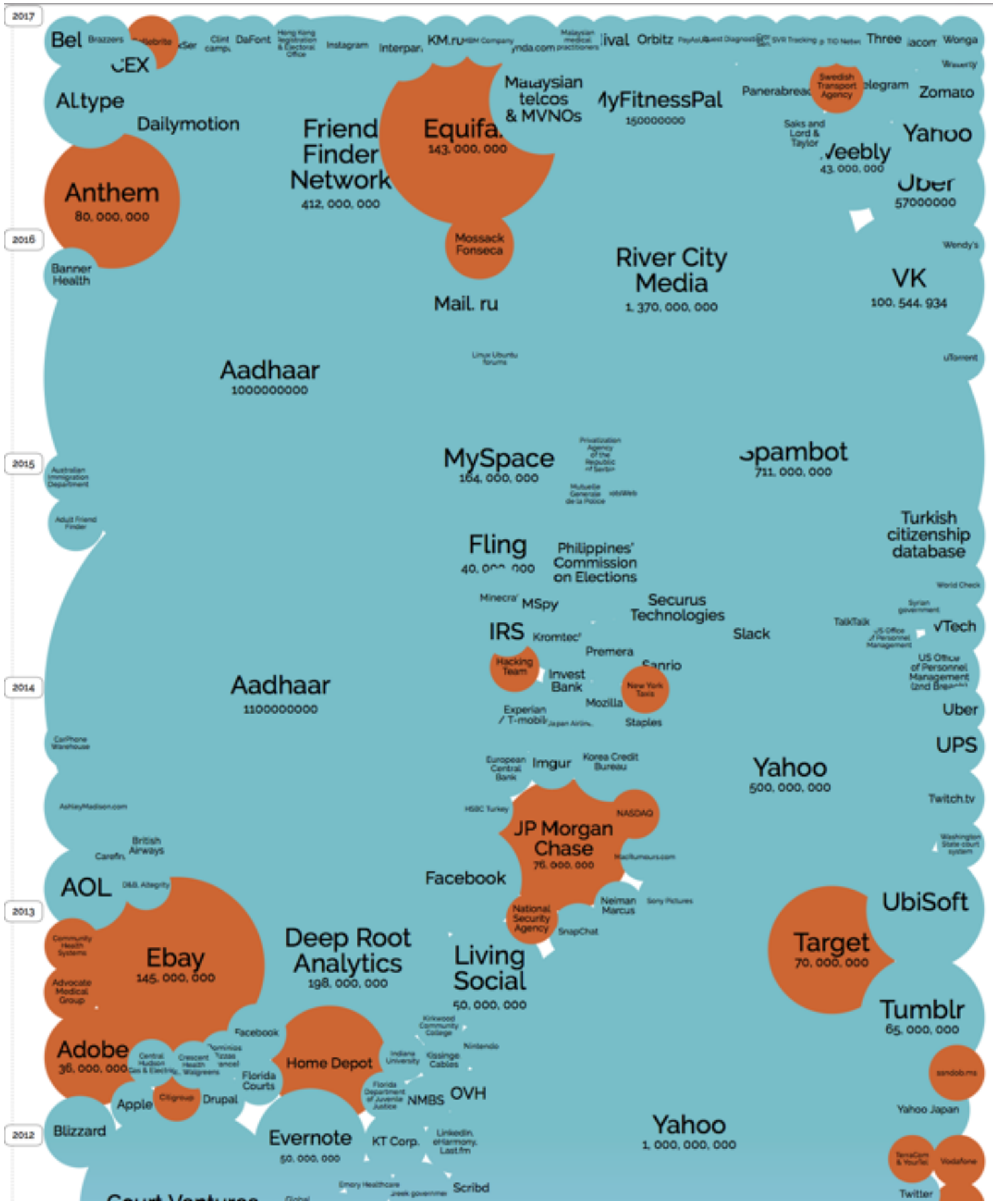
LILY HAY NEWMAN SECURITY 12.14.16 6:27 PM

HACK BRIEF: HACKERS BREACH A BILLION YAHOO ACCOUNTS. A *BILLION*

theguardian

Tumblr

More than 65m Tumblr emails for sale on the darknet



2017
2016
2015
2014
2013
2012
Bel
Instagram
DaFont
KM.ru
Orbitz
Three
Wonga
Altype
Dailymotion
Malaysian telcos & MVNOs
Panerabread
Zomato
Friend Finder Network
412, 000, 000
Equifa
143, 000, 000
Weebly
43, 000, 000
Yahoo
Uber
57000000
Anthem
80, 000, 000
Mossack Fonseca
Banner Health
River City Media
1, 370, 000, 000
VK
100, 544, 934
Mail. ru
Aadhaar
1000000000
MySpace
164, 000, 000
711, 000, 000
Turkish citizenship database
Fling
Philippines' Commission on Elections
MSpy
Securus Technologies
IRS
Slack
vTech
Premera
Hacking Team
Invest Bank
Aadhaar
1100000000
Mozilla
Staples
Uber
UPS
Imgur
Korea Credit Bureau
Yahoo
500, 000, 000
Twitch.tv
JP Morgan Chase
76, 000, 000
NASDAQ
Facebook
British Airways
AOL
National Security Agency
Neiman Marcus
SnapChat
UbiSoft
Ebay
145, 000, 000
Deep Root Analytics
198, 000, 000
Living Social
50, 000, 000
Target
70, 000, 000
Tumblr
65, 000, 000
Facebook
Adobe
36, 000, 000
Home Depot
Florida Courts
OVH
NMBS
Apple
Drupal
Nintendo
Yahoo Japan
Blizzard
Evernote
50, 000, 000
KT Corp.
Yahoo
1, 000, 000, 000
Vodafone
Twitter
Scribd

Slightly Better Version

STILL never ever (ever ever…) use

Let **H** be a hash function

sk

sk=pwd

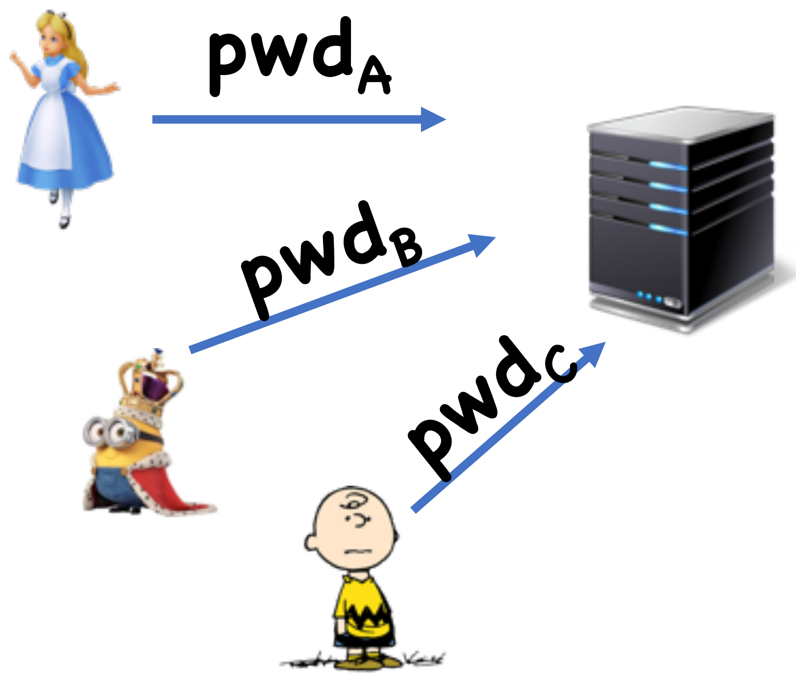
vk=H(pwd)

H(sk) == vk?

Slightly Better Version

STILL never ever (ever ever…) use

Let **H** be a hash function



User	Pwd
Alice	$H(pwd_A)$
Bob	$H(pwd_B)$
Charlie	$H(pwd_C)$
…	…

Slightly Better Version

STILL never ever (ever ever…) use

Advantage of hashing:

- Now if pwd database is leaks, adversary only gets hashes passwords

- For identification protocol, need actual password

- Therefore, adversary needs to invert hash function to break protocol

- Presumed hard

Weak Passwords

Data from 10M passwords leaked in 2016:

17%

RANK	PASSWORD
1.	123456
2.	123456789
3.	qwerty
4.	12345678
5.	111111
6.	1234567890
7.	1234567
8.	password
9.	123123
10.	987654321
11.	qwertyuiop
12.	mynoob
13.	123321
14.	666666
15.	18atcskd2w
16.	7777777
17.	1q2w3e4r
18.	654321
19.	555555
20.	3rjs1la7qe
21.	google
22.	1q2w3e4r5t
23.	123qwe
24.	zxcvbnm
25.	1q2w3e

50% of available passwords

https://blog.keepersecurity.com/2017/01/13/most-common-passwords-of-2016-research-study/

Weak Passwords

Of course, pwds that have been leaked are likely the particularly common ones

Even so, 360M pwds covers about 25% of all users

Online Dictionary Attacks

Suppose attacker gets list of usernames

Attacker tries logging in to each with **pwd** = ‘123456’

5-17% of accounts will be compromised

Online Dictionary Attacks

How to slow down attacker?

- Lock out after several unsuccessful attempts
  - Honest users may get locked out too

- Slow down response after each unsuccessful attempt
  - 1s after 1st, 2s after 2nd, 4s after 3rd, etc

Offline Dictionary Attacks

Suppose attacker gets hashed password **vk = H(pwd)**

Attack:

- Assemble dictionary of 360M common passwords
- Hash each, and check if you get **vk**
- If so, you have just found **pwd**!

On modern hardware, takes a few seconds to recover a a passwords 25% of the time

Offline Dictionary Attacks

Now consider what happens when adversary gets entire hashed password database

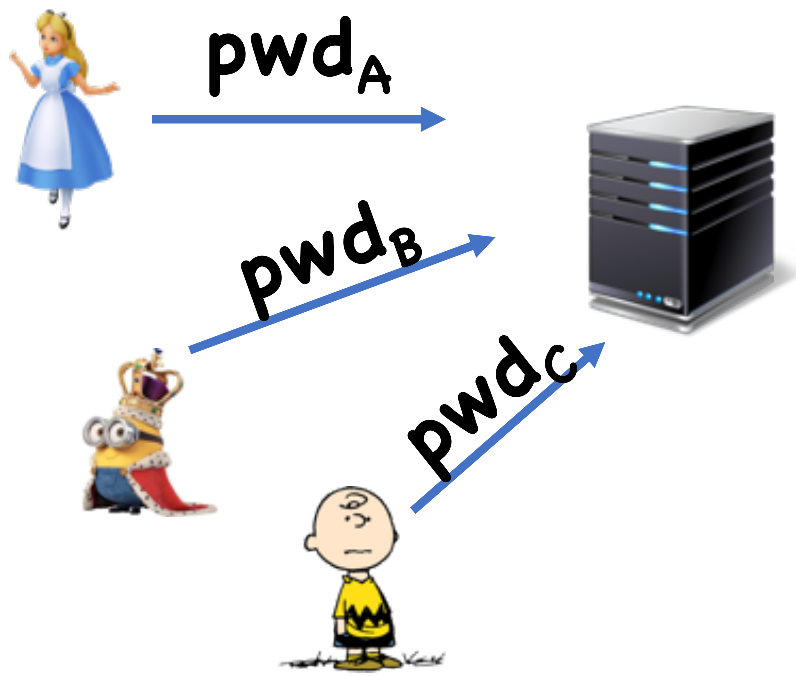
- Hash dictionary once: $O(|D|)$
- Index dictionary by hashes
- Lookup each database entry in dictionary: $O(|L|)$

To get 25% of passwords takes $O(|D|+|L|)$ time

- Amortize cost of hashing dictionary over many passwords

Salting

Let **H** be a hash function



s_i random

User	Salt	Pwd
Alice	s_A	$H(s_A,pwd_A)$
Bob	s_B	$H(s_B,pwd_B)$
Charlie	s_C	$H(s_C,pwd_C)$
…	…	…

Salting

Salt length? Enough to make each user's salt unique
- At least 64 bits

Salting kills amortization:
- To recover Alice's key, adversary must hash entire dictionary with s_A
- To recover Bob's key, adversary must hash entire dictionary with s_B
- Must hash entire dictionary again for each user

  Running time: $O(|D| \times |L|)$

Unique Passwords

Different websites may employ different standards for password security

- Some may store passwords in clear, some may hash without salt, some may salt

If you use the same password at a bank (high security) and your high school reunion (low security), could end up with your password stolen

Unique Passwords

Solutions:

- Password managers

- Salt master password to generate website-specific password (e.g. pwdhash):

  Master password: **pwd**
  Pwd for abcdefg.com: **H(**abcdefg.com,**pwd)**

My Personal Favorite

Stanford PwdHash

PwdHash generates theft-resistant passwords. The PwdHash browser extension invisibly generates these passwords when it is installed in your browser. You can activate this protection by pressing F2 before you type your password, or by choosing passwords that start with @@. If you don't want to install PwdHash on your computer, you can generate the passwords right here.

- Visit the Stanford project website.
- Install PwdHash for Firefox. It has been ported to Chrome and Opera.
- Read the USENIX Security Symposium 2005 paper (PDF).
- This site and plugin are no longer under active development and the code is available for use. See individual files for license details.

Site Address

http://www.example.com

Site Password

Hashed Password

Press Generate | Generate

Version 0.8 (more versions)
Tip: You can save this page to disk.

What Hash Function to Use

In LindedIn leak (using Sha1), 90% of passwords were recovered within a week

Problem: Sha1 is very fast!

To make hashing harder, want hash function that is just slow enough to be unnoticeable to user

What Hash Function to Use

Examples: PBKDF2, bcrypt

- Iterate hash function many times:

$$H'(x) = H(H(H(\ldots.H(x)\ldots.)))$$

- Set #iterations to get desired hashing time

Still problem:

- Adversary may have special purpose hardware

  $\Rightarrow$ Can eval much faster than you can (50,000x)

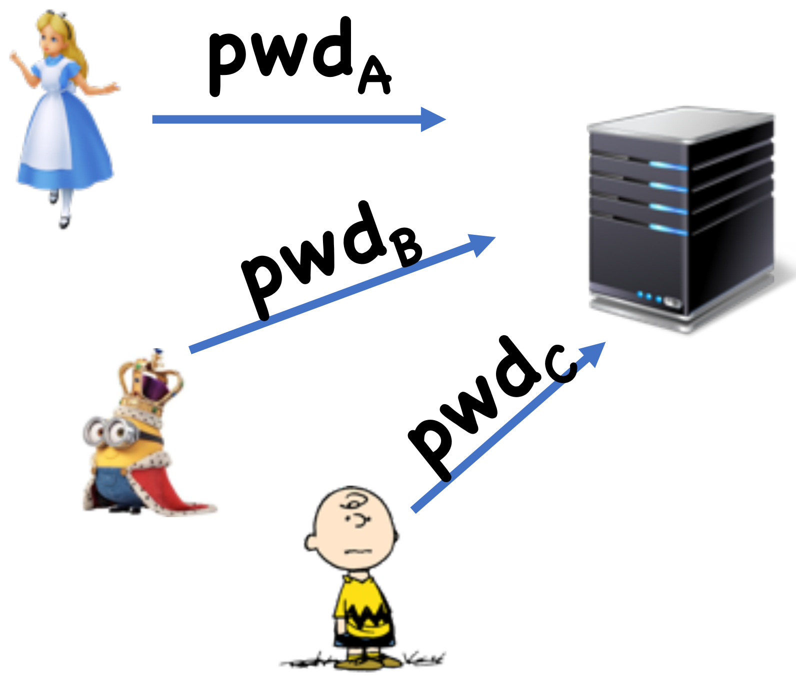
What Hash Function to Use

Memory-hard functions: functions that require a lot of memory to compute

- As far as we know, no special purpose memory
- Attacker doesn't gain advantage using special purpose hardware

Examples: Scrypt, Argon2i

Encrypt Passwords?



User	Pwd
Alice	**Enc(k,pwd_A)**
Bob	**Enc(k,pwd_B)**
Charlie	**Enc(k,pwd_C)**
…	…

Encrypt Passwords?

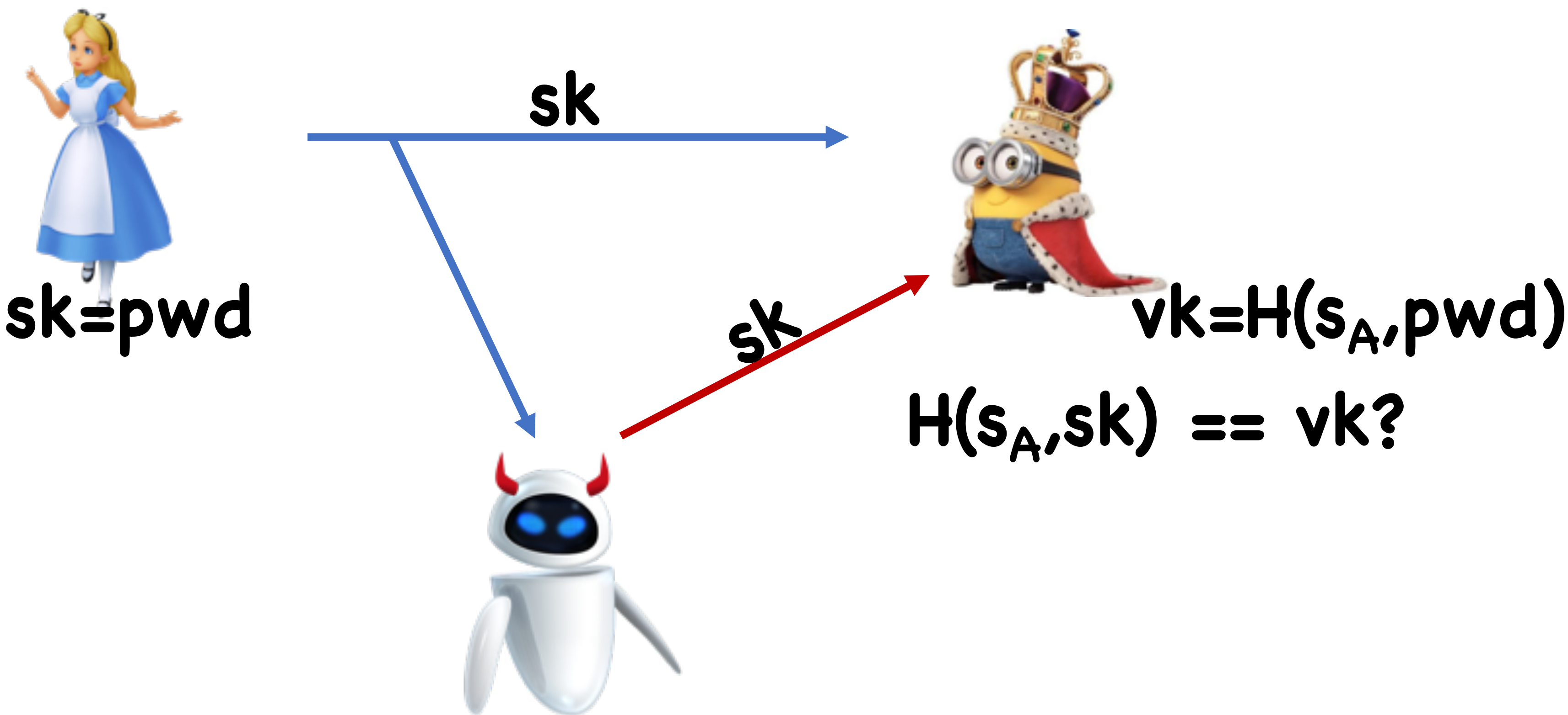
Again, never ever (ever ever….) use

- To check password, need to decrypt
- Must store decryption key **k** somewhere
- What if **k** is stolen?

Need to use one-way mechanism

- With hash function, not even server can recover password

Security Against Eavesdropping



Security Against Eavesdropping

One solution: update **sk,vk** after every run

One-time Passwords

Let **F** be a PRF

sk_0 = F(k,0)

sk=(k,0)

vk=(k,0)

sk_0 == F(k,0)?

One-time Passwords

Let **F** be a PRF

$sk_1 = F(k,1)$

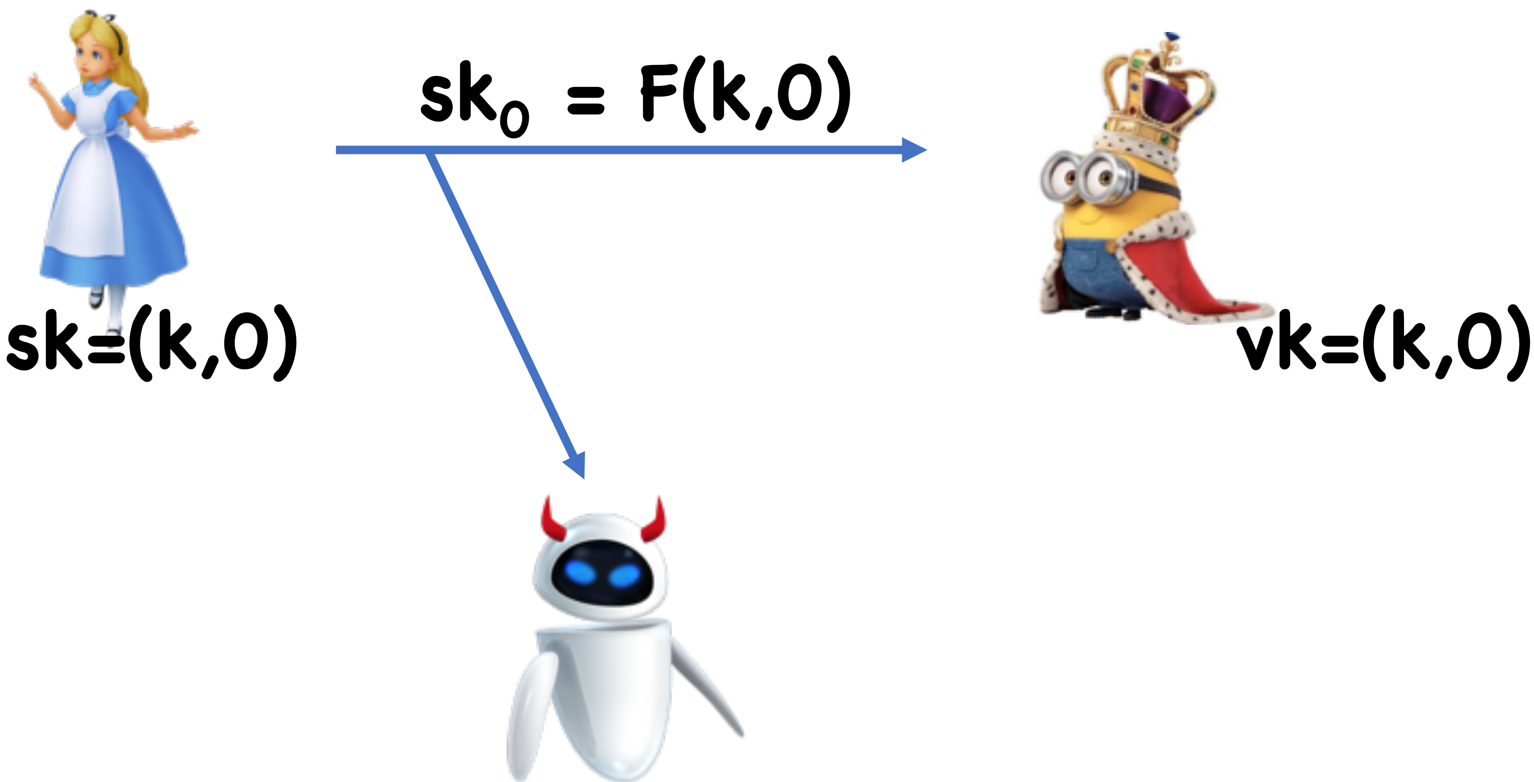
sk=(k,1)

vk=(k,1)

$sk_1 == F(k,1)$?

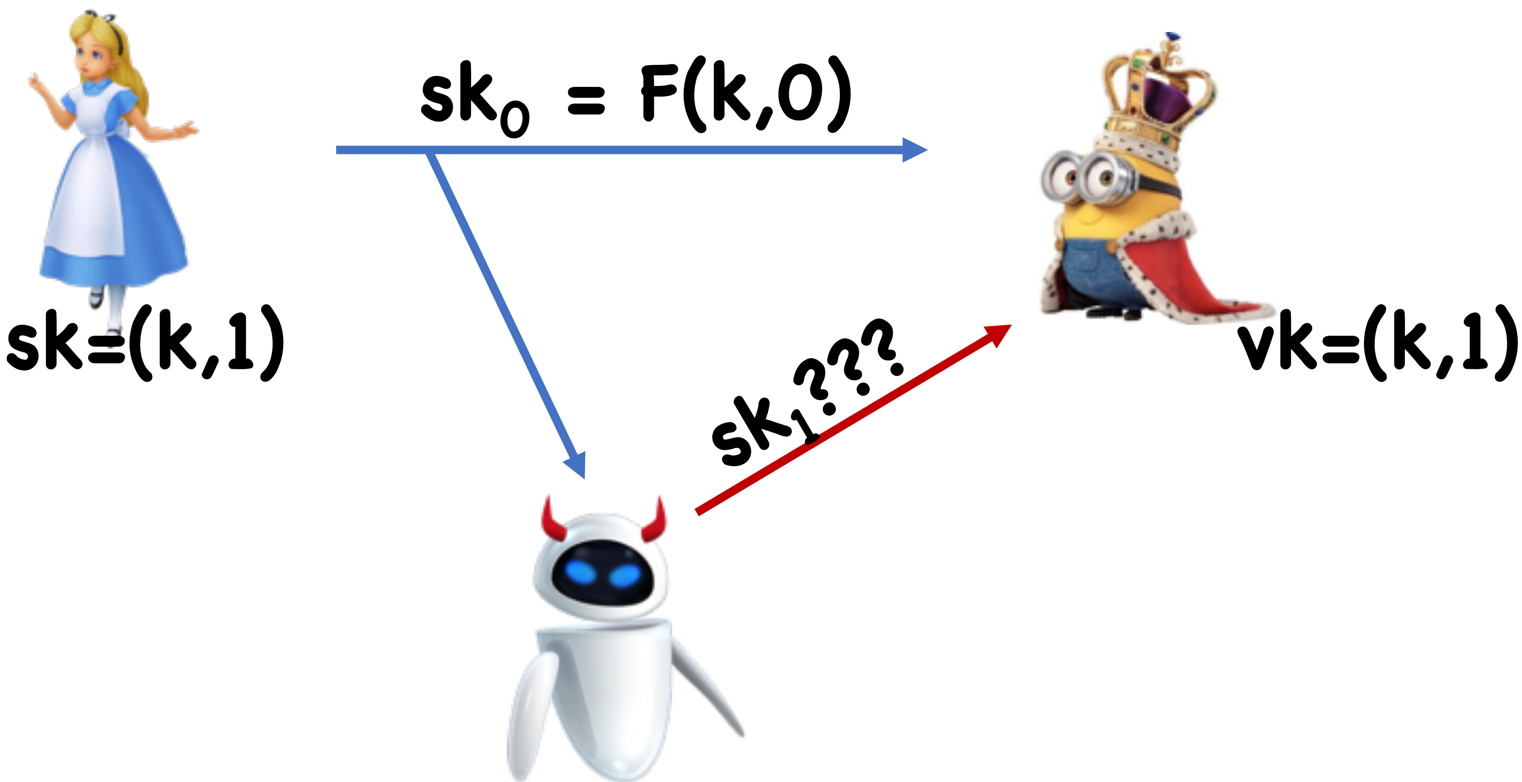
One-time Passwords

Let **F** be a PRF



One-time Passwords

Let **F** be a PRF



One-time Passwords

Advancing state:

- Time based (e.g. every minute, day, etc)
- User Action (button press)

Must allow for small variation in counter value

- Clocks may be off, user may accidentally press button

Stateless Schemes?

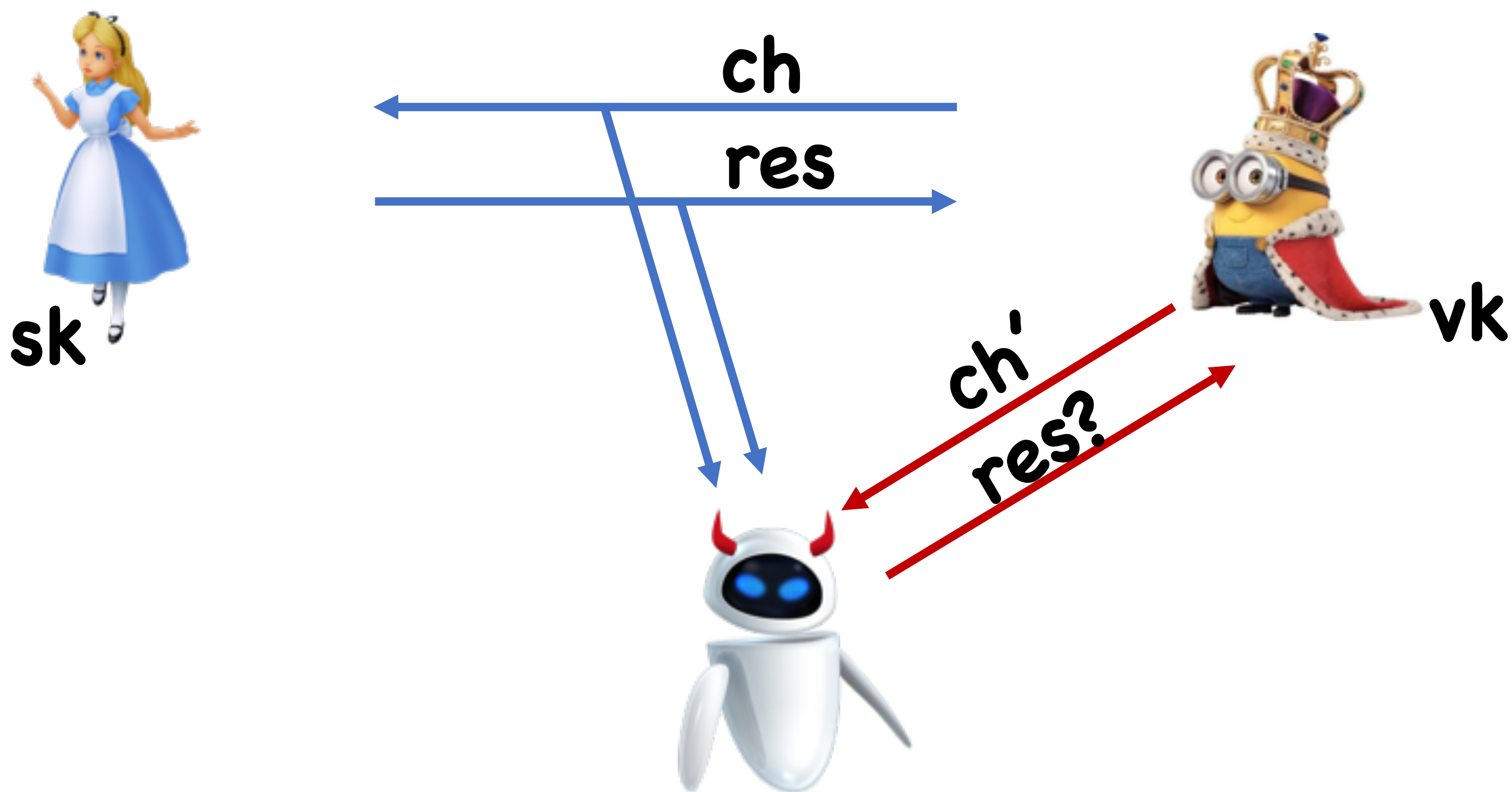
So far, all schemes secure against eavesdropping are stateful

Easy theorem: any one-message stateless ID protocol is insecure if the adversary can eavesdrop

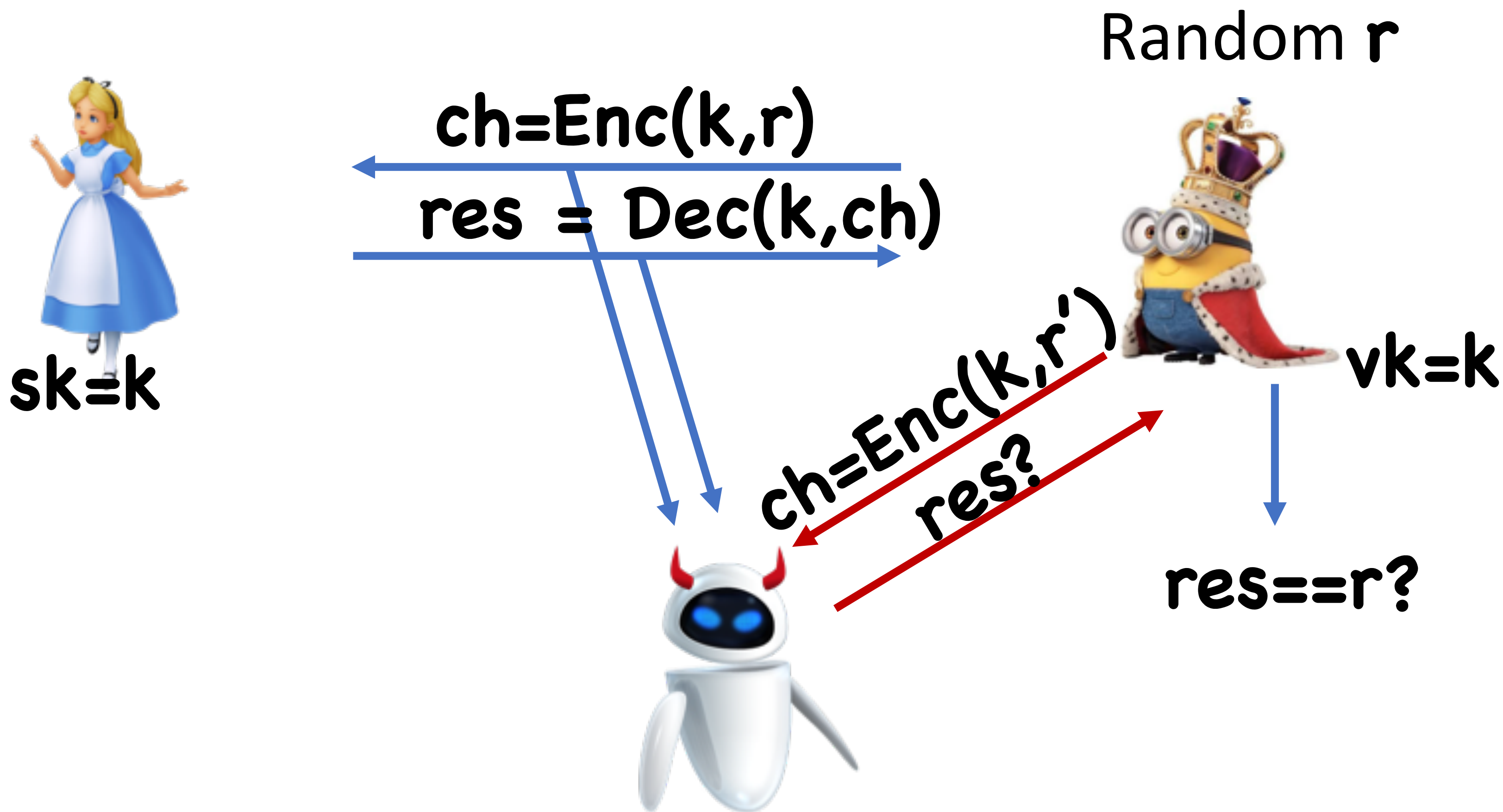
- Simply replay message

If want stateless scheme, instead want at least two messages

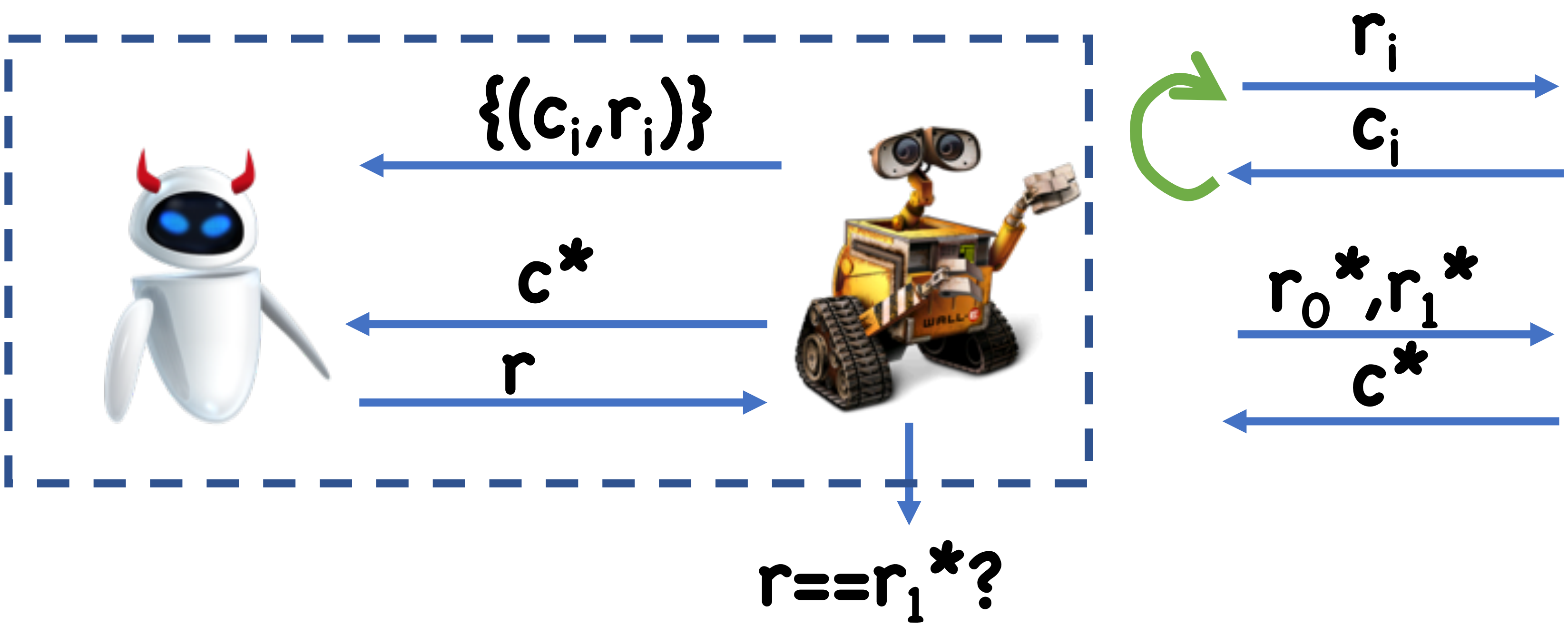
Challenge-Response



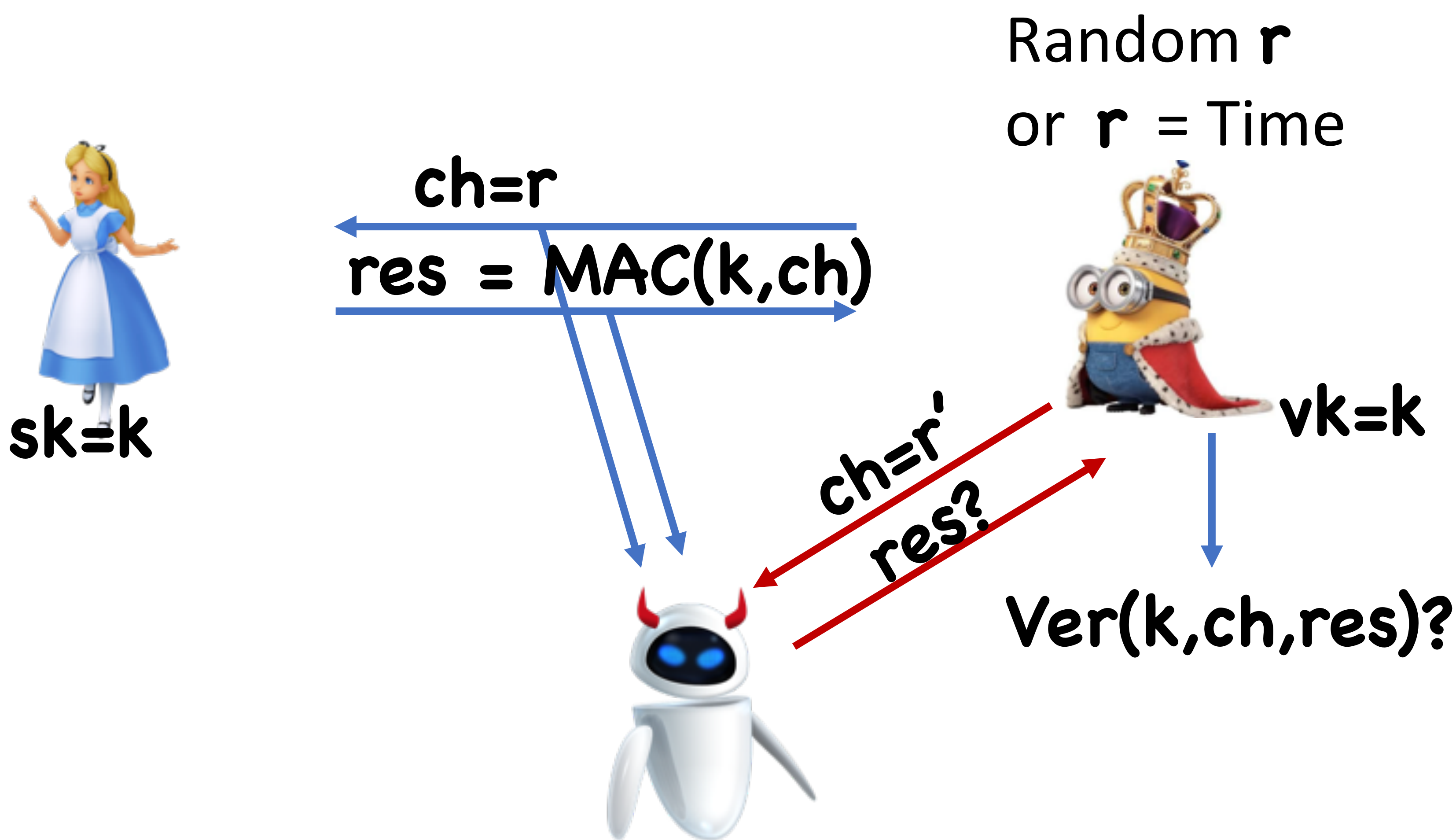
C-R Using Encryption



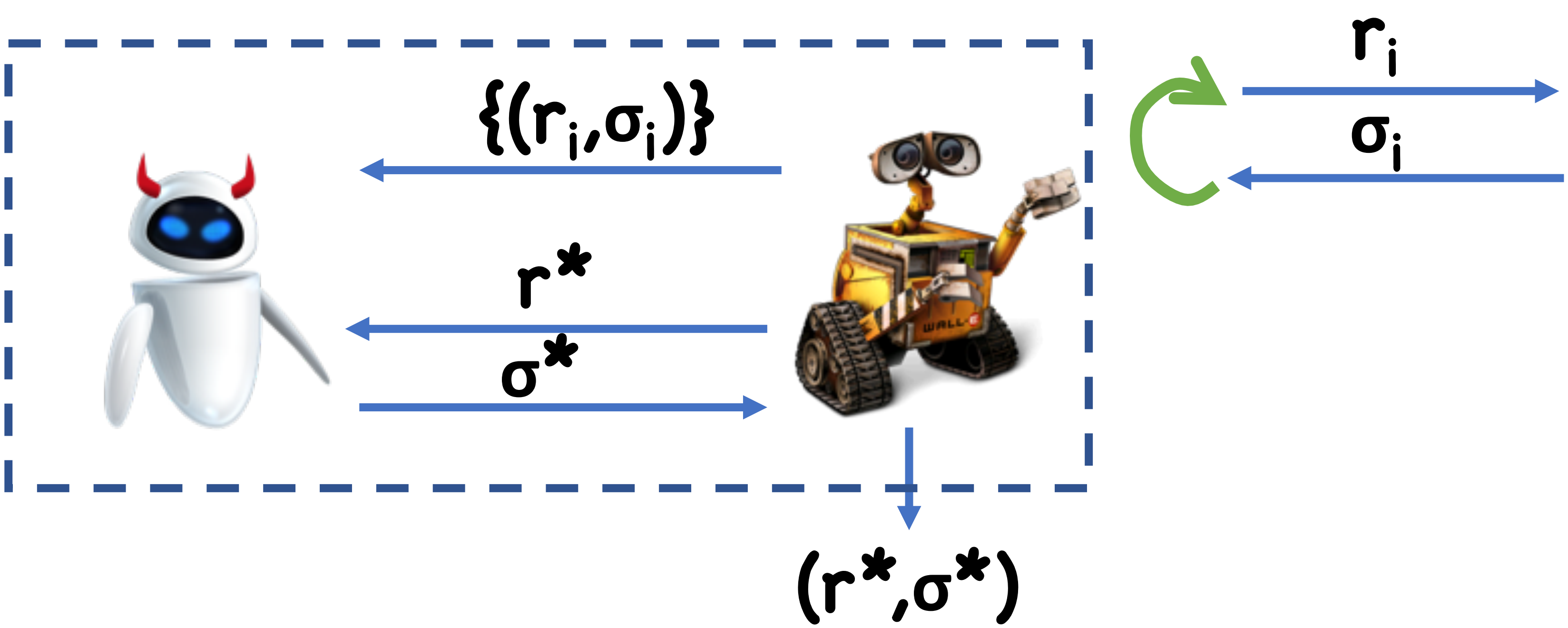
Theorem: If **(Enc,Dec)** is a CPA-secure secure SKE/PKE scheme, then the C-R protocol is a secret key/public key identification protocol secure against eavesdropping attacks



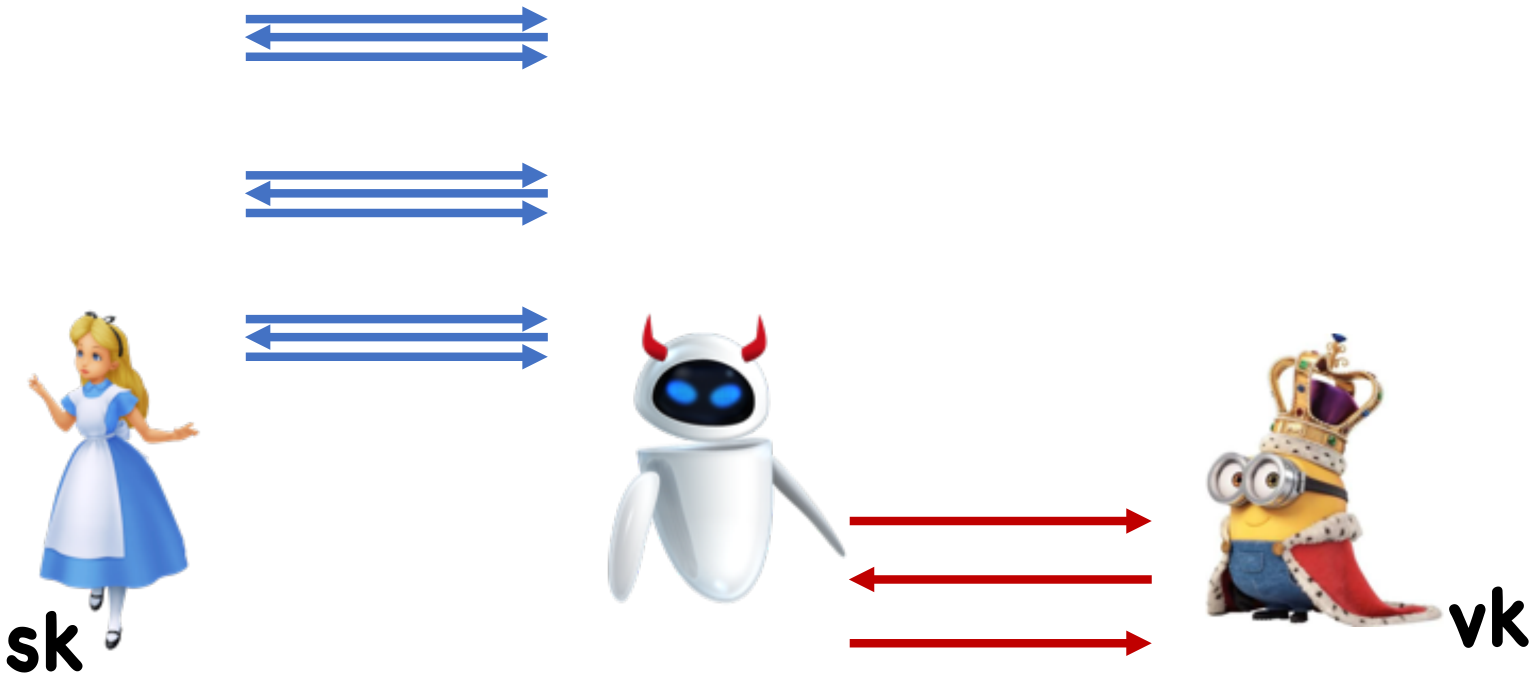
C-R Using MACs/Signatures



Theorem: If **(MAC,Ver)** is a CMA-secure secure MAC/Signature scheme, then the C-R protocol is a secret key/public key identification protocol secure against eavesdropping attacks



Active Attacks



Active Attacks

For enc-based C-R, CPA-secure is insufficient

- Instead need CCA-security (lunch-time sufficient)

For MAC/Sig-based C-R, CMA-security is sufficient

Announcements

HW6 Due April 23rd
HW7 Due April 30th