COS433/Math 473: Cryptography

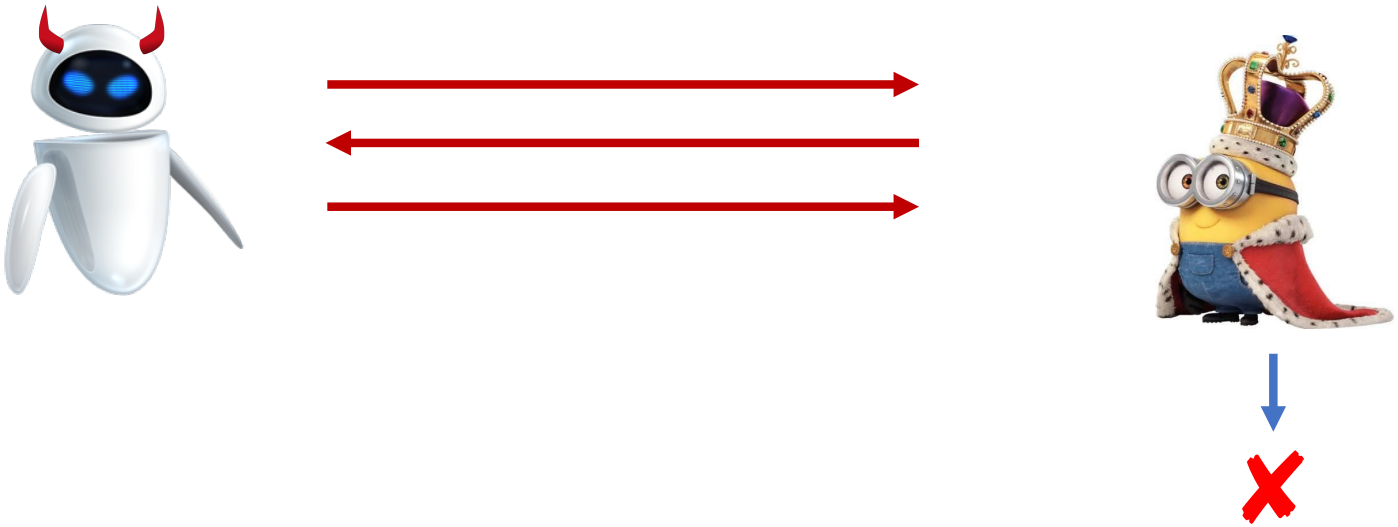
Mark Zhandry

Princeton University

Spring 2017

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

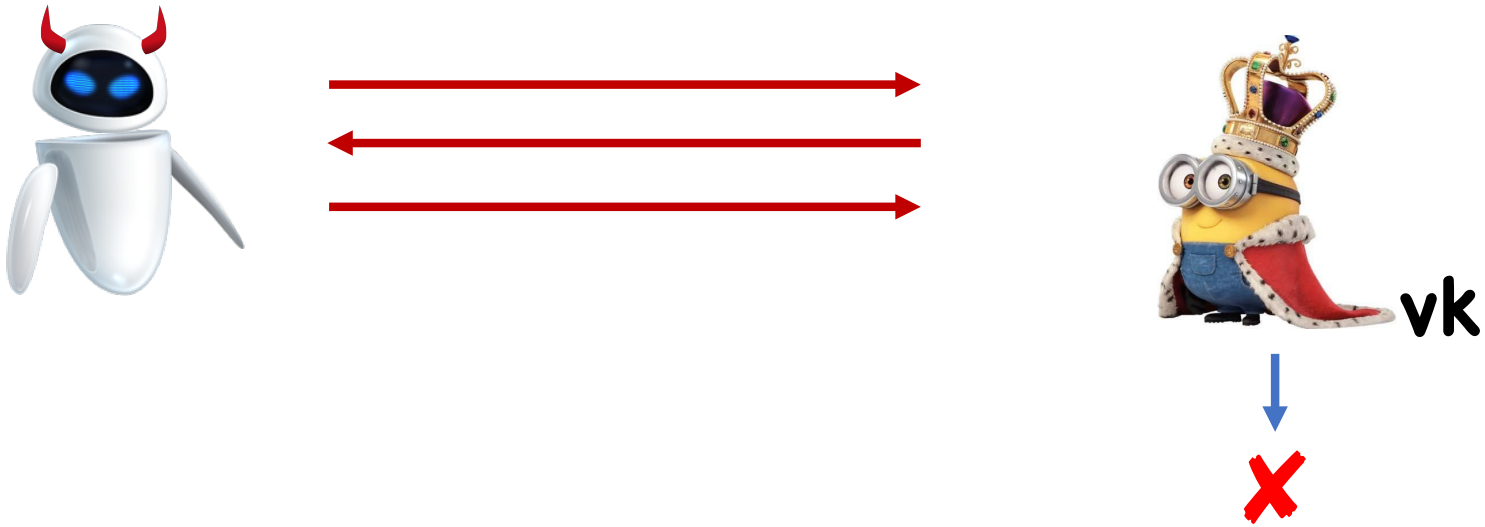
Types of Identification Protocols

Secret key:

Public Key:

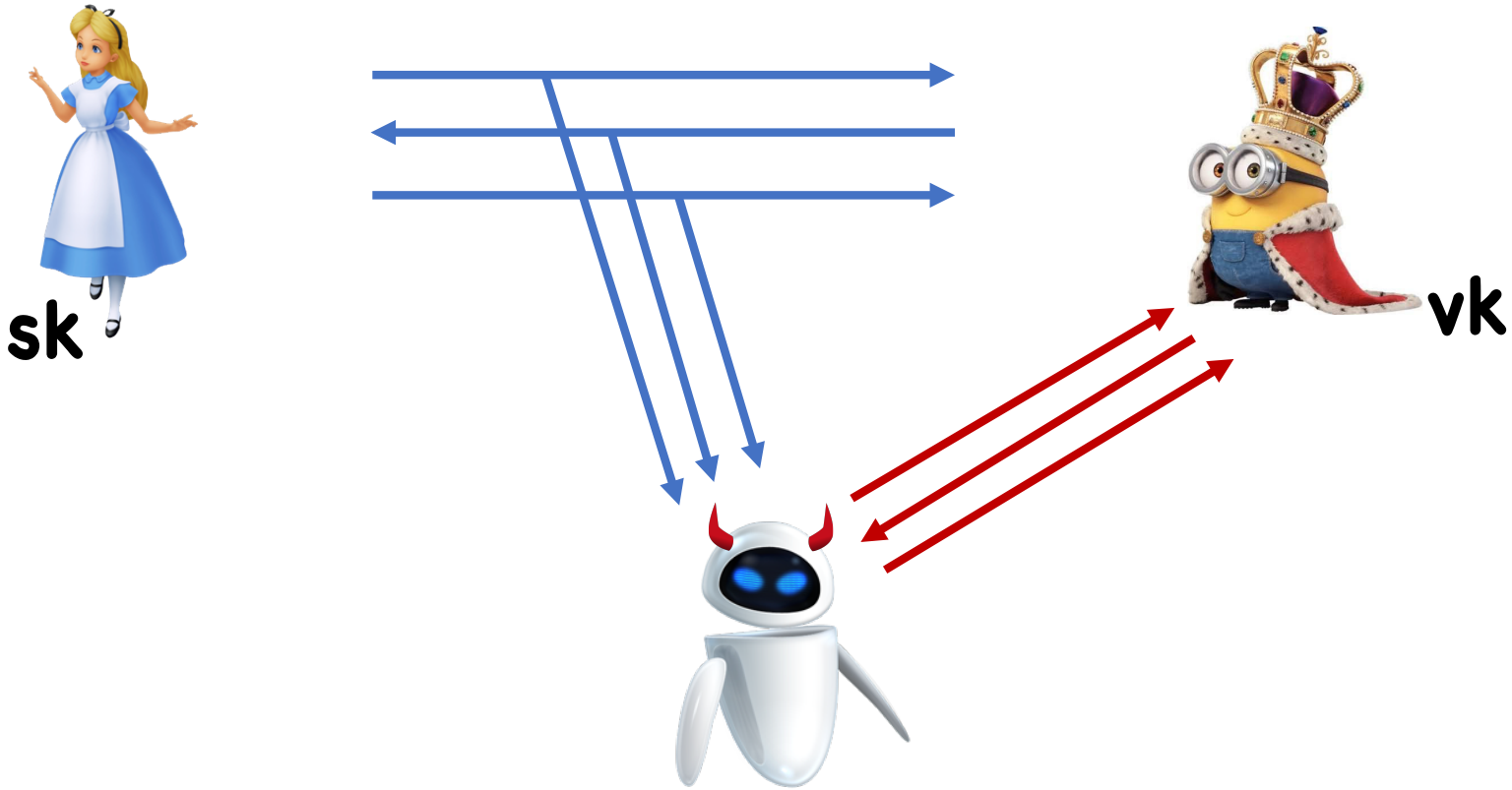
Types of Attacks

Direct Attack:



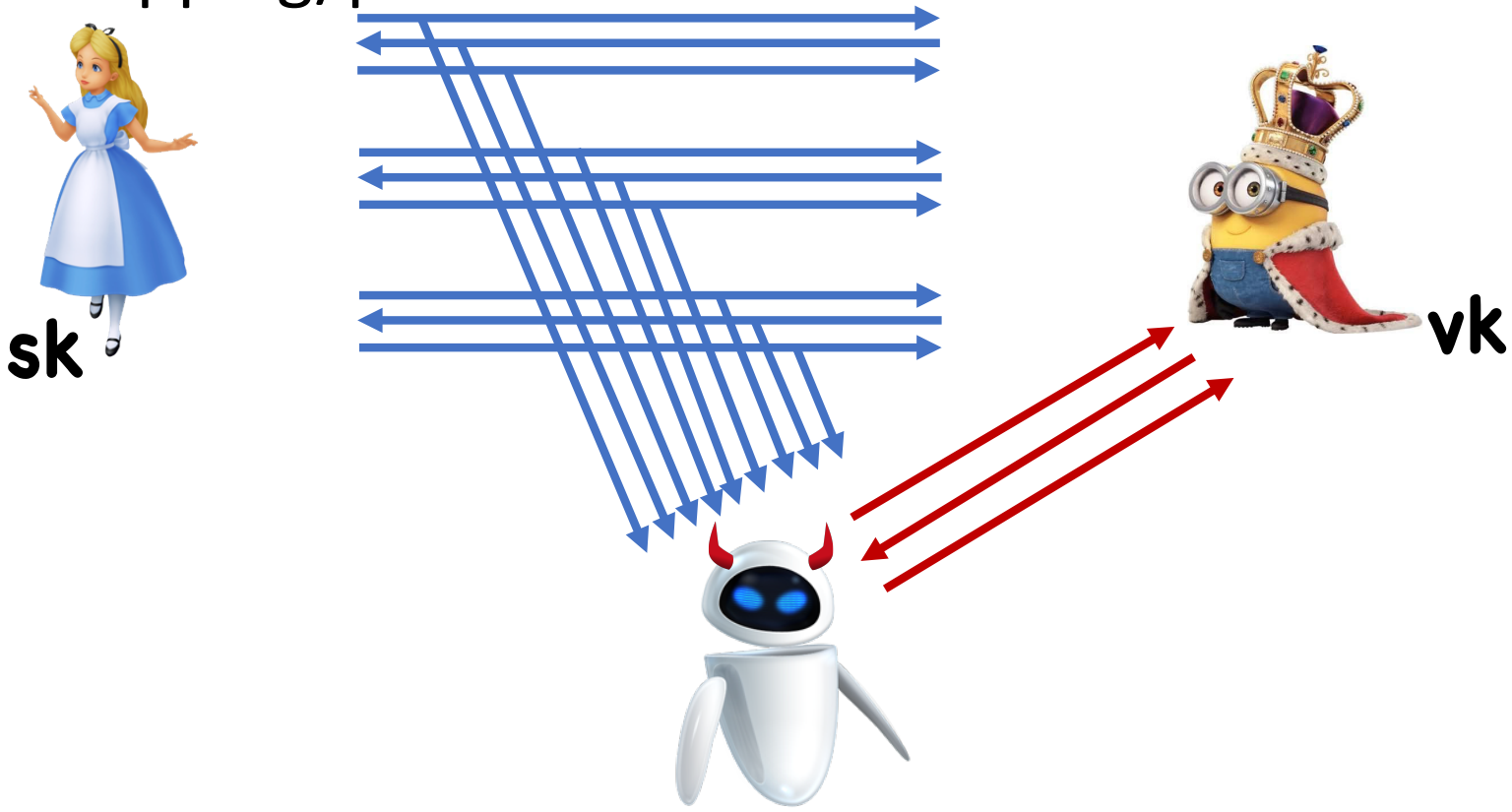
Types of Attacks

Eavesdropping/passive:



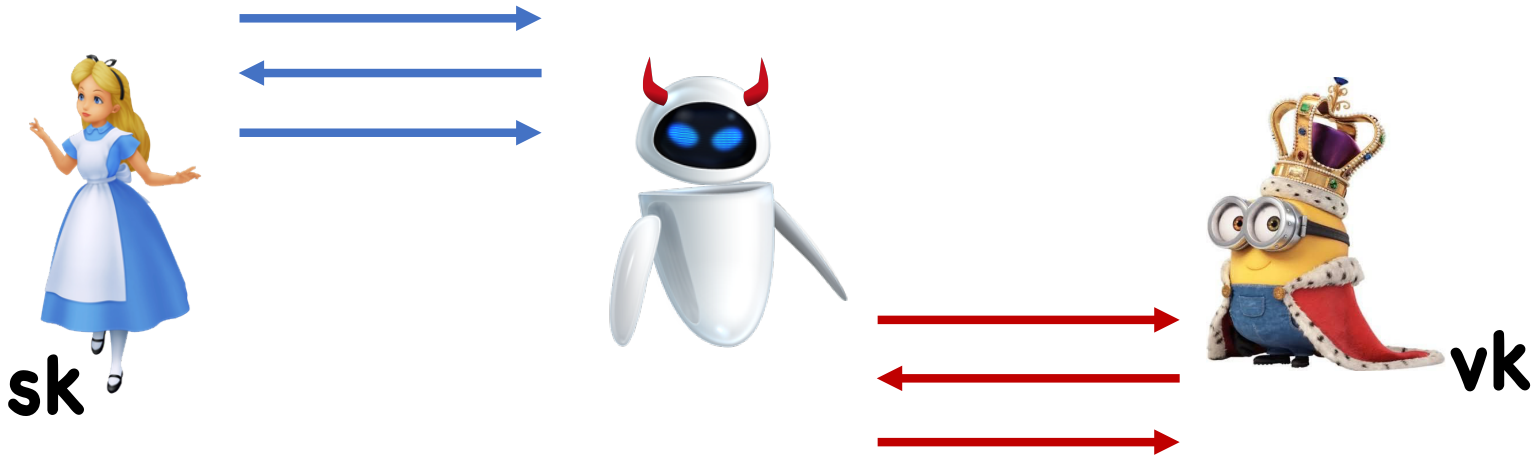
Types of Attacks

Eavesdropping/passive:



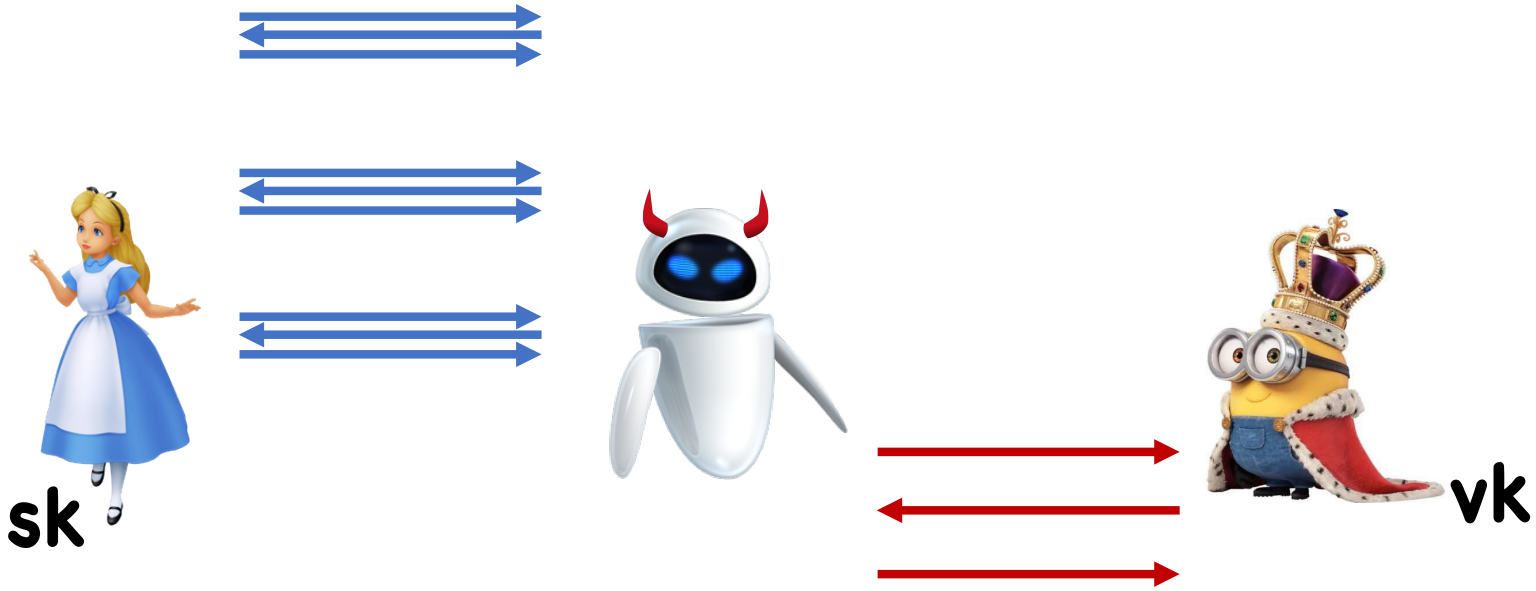
Types of Attacks

Man-in-the-Middle/Active:



Types of Attacks

Man-in-the-Middle/Active:



Basic Password Protocol

Never ever (ever ever…) use

sk=pwd

sk

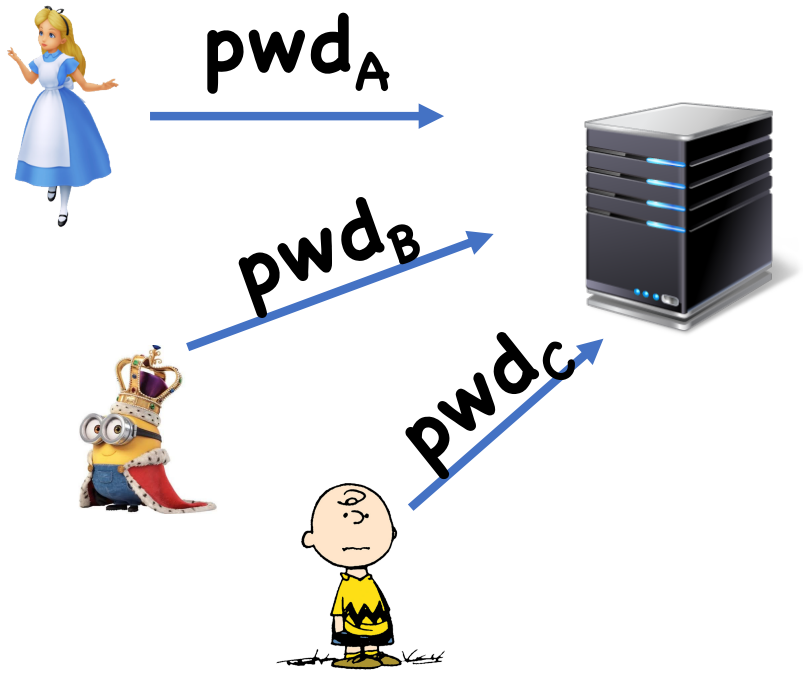
vk=pwd

sk == vk?

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:

pw

MOTHERBOARD

NEWS

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

WIRED SUBSCRIBE

BRIAN BARRETT SECURITY 05.31.16 4:58 PM

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

WIRED SUBSCRIBE

ANDY GREENBERG SECURITY 09.22.16 12:15 PM

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

WIRED SUBSCRIBE

LILY HAY NEWMAN SECURITY 08.31.16 1:10 PM

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

WIRED SUBSCRIBE

LILY HAY NEWMAN SECURITY 12.14.16 6:27 PM

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

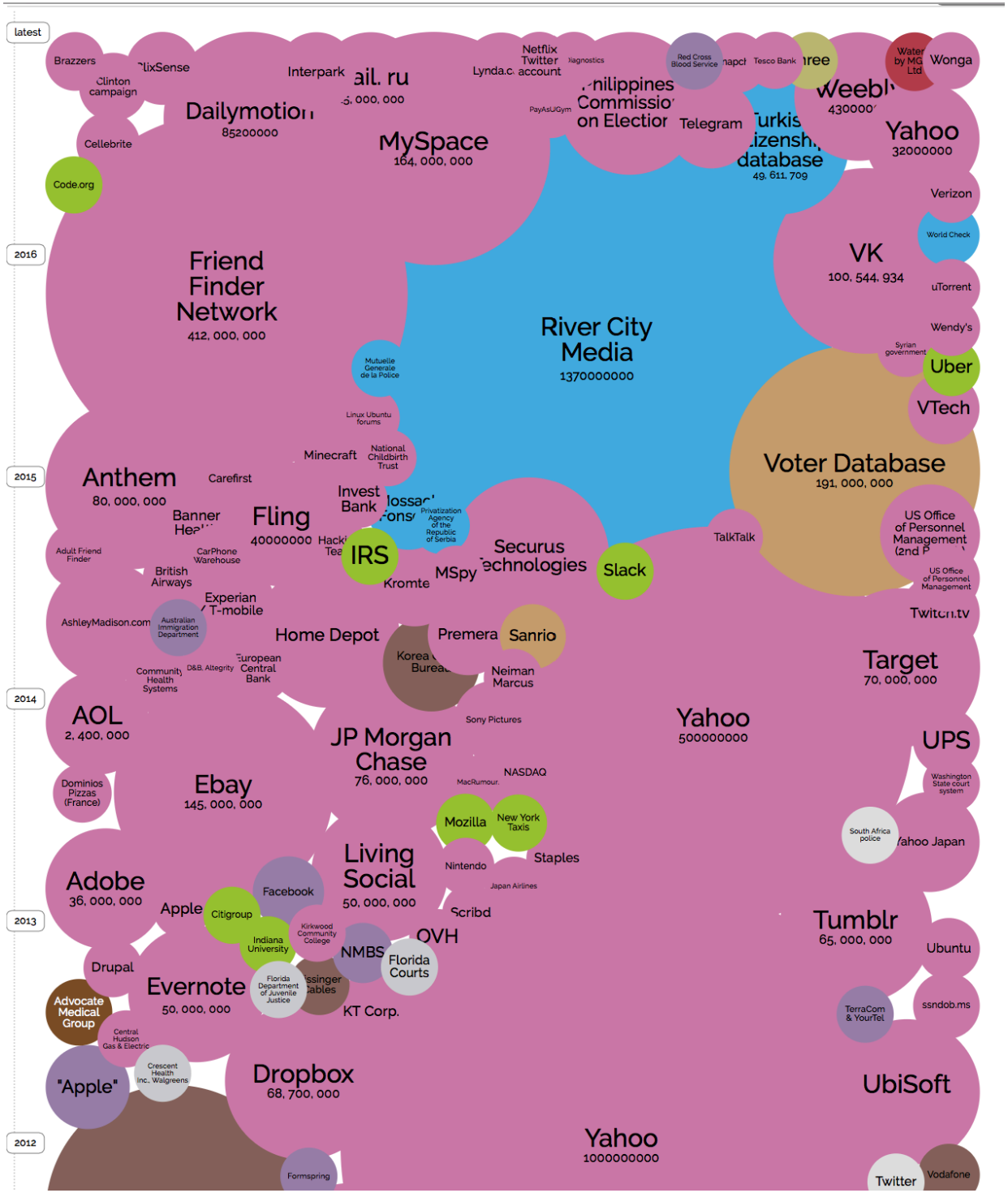
sign in search jobs US edition

theguardian

home › tech US politics world opinion sports soccer arts lifestyle fas all

Tumblr

More than 65m Tumblr emails for sale on the darknet


latest
2016
2015
2014
2013
2012
Brazzers
Clinton campaign
Interpark
Dailymotion
85200000
Netflix
Twitter account
Lynda.com
Red Cross Blood Service
Tesco Bank
Water by MG Ltd
Wonga
Weebly
430000
Cellebrite
MySpace
164, 000, 000
Telegram
Turkish citizenship database
49, 611, 709
Yahoo
32000000
Code.org
Verizon
World Check
Friend Finder Network
412, 000, 000
River City Media
1370000000
VK
100, 544, 934
uTorrent
Wendy's
Syrian government
Uber
VTech
Mutuelle Generale de la Police
Linux Ubuntu forums
Minecraft
National Childbirth Trust
Voter Database
191, 000, 000
Anthem
80, 000, 000
Carefirst
Banner
Fling
40000000
Invest Bank
Privatization Agency of the Republic of Serbia
US Office of Personnel Management (2nd P
Adult Friend Finder
CarPhone Warehouse
IRS
Securus Technologies
Slack
TalkTalk
British Airways
MSpy
Kromtech
US Office of Personnel Management
Experian
T-mobile
Twitch.tv
AshleyMadison.com
Australian Immigration Department
Home Depot
Premera
Sanrio
Target
70, 000, 000
Community Health Systems
D&B
Altegrity
European Central Bank
Neiman Marcus
AOL
2, 400, 000
Sony Pictures
Yahoo
500000000
UPS
JP Morgan Chase
76, 000, 000
Dominios Pizzas (France)
Ebay
145, 000, 000
NASDAQ
MacRumour.
Washington State court system
Mozilla
New York Taxis
South Africa police
Yahoo Japan
Staples
Nintendo
Japan Airlines
Living Social
50, 000, 000
Adobe
36, 000, 000
Facebook
Apple
Citigroup
Scribd
Tumblr
65, 000, 000
Kirkwood Community College
Indiana University
NMBS
OVH
Ubuntu
Drupal
Florida Courts
Florida Department of Juvenile Justice
Evernote
50, 000, 000
KT Corp.
TerraCom & YourTel
ssndob.ms
Advocate Medical Group
Central Hudson Gas & Electric
Crescent Health Inc., Walgreens
"Apple"
Dropbox
68, 700, 000
UbiSoft
Yahoo
1000000000
Formspring
Twitter
Vodafone

Slightly Better Version

STILL never ever (ever ever…) use

Let **H** be a hash function

sk=pwd

sk

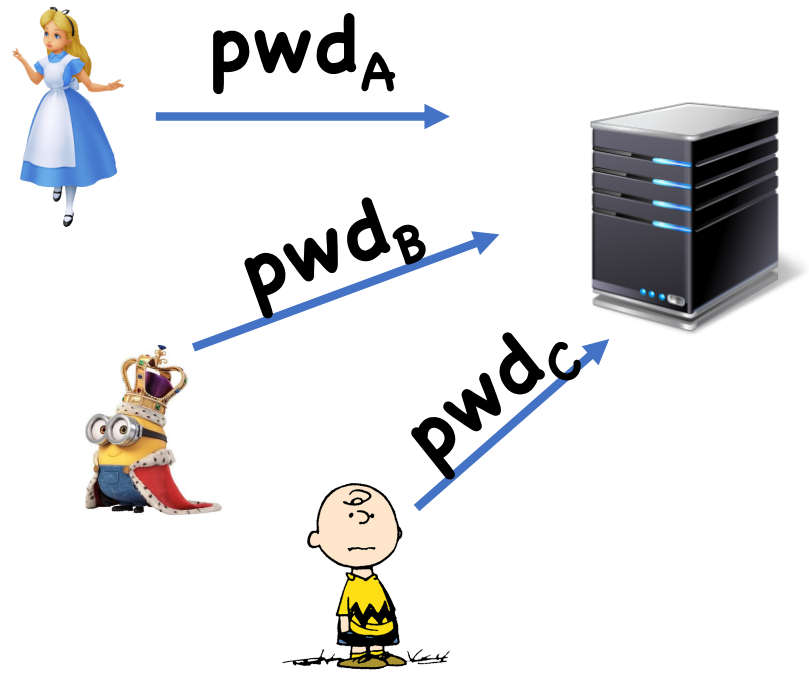
vk=H(pwd)

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 Attack

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 Attack

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 Attack

Now consider what happens when adversary gets entire hashed password database

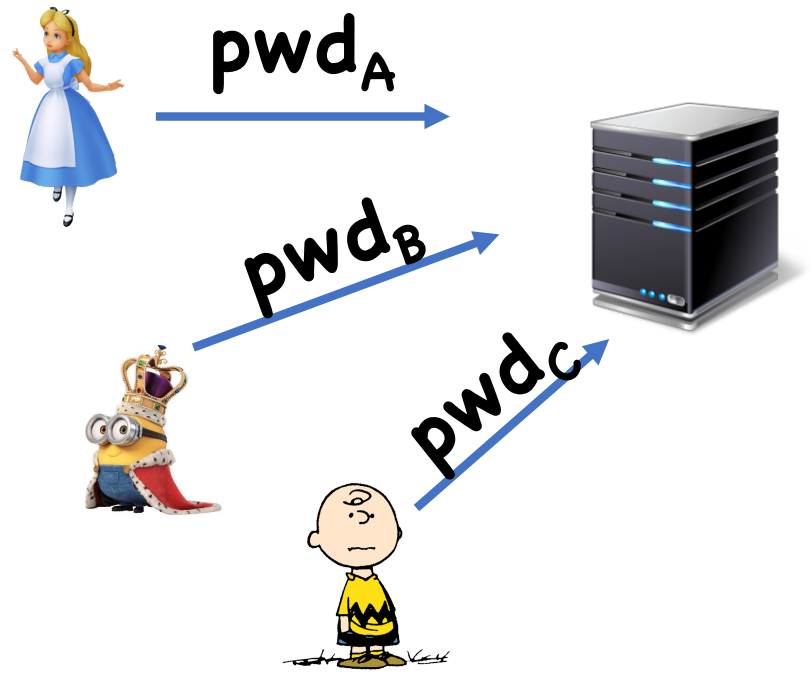
- Hash dictionary once: ***O*(|D|)**
- Index dictionary by hashes
- Lookup each database entry in dictionary: **O(|L|)**

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)**

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 fast 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

What Hash Function to Use

Example: Scrypt

- Slow hash function, and memory requirement is as good as possible (proportional to run time)
- Problem: memory access pattern depends on password
  - Local attack can potentially learn access pattern
  - Turns out this can eliminate the need for memory in attacks

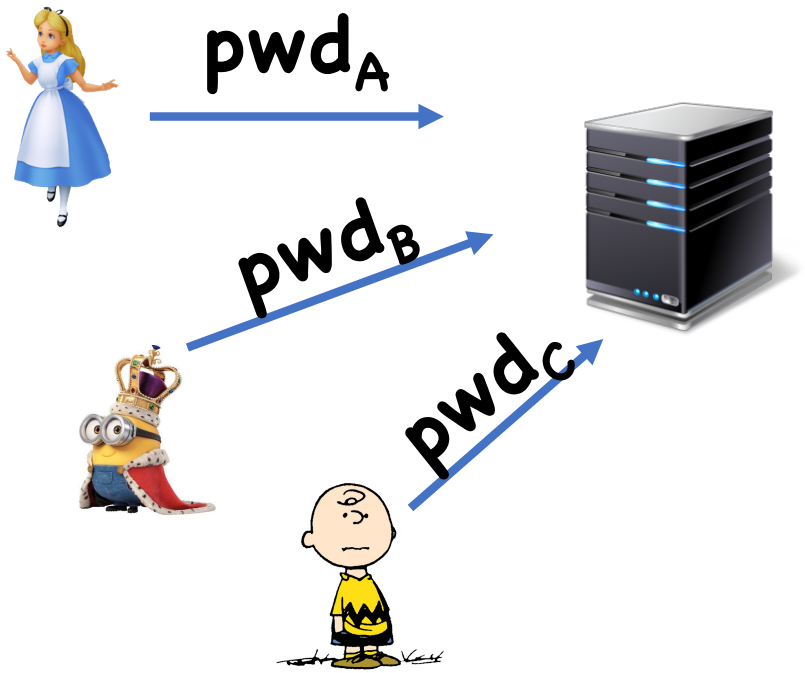
What Hash Function to Use

Instead, want data-independent memory hard function (iMHF)

- Ex: Argon2i

To date, no known practical iMHF with optimal memory requirements

Encrypt Passwords?



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

Encrypt Passwords?

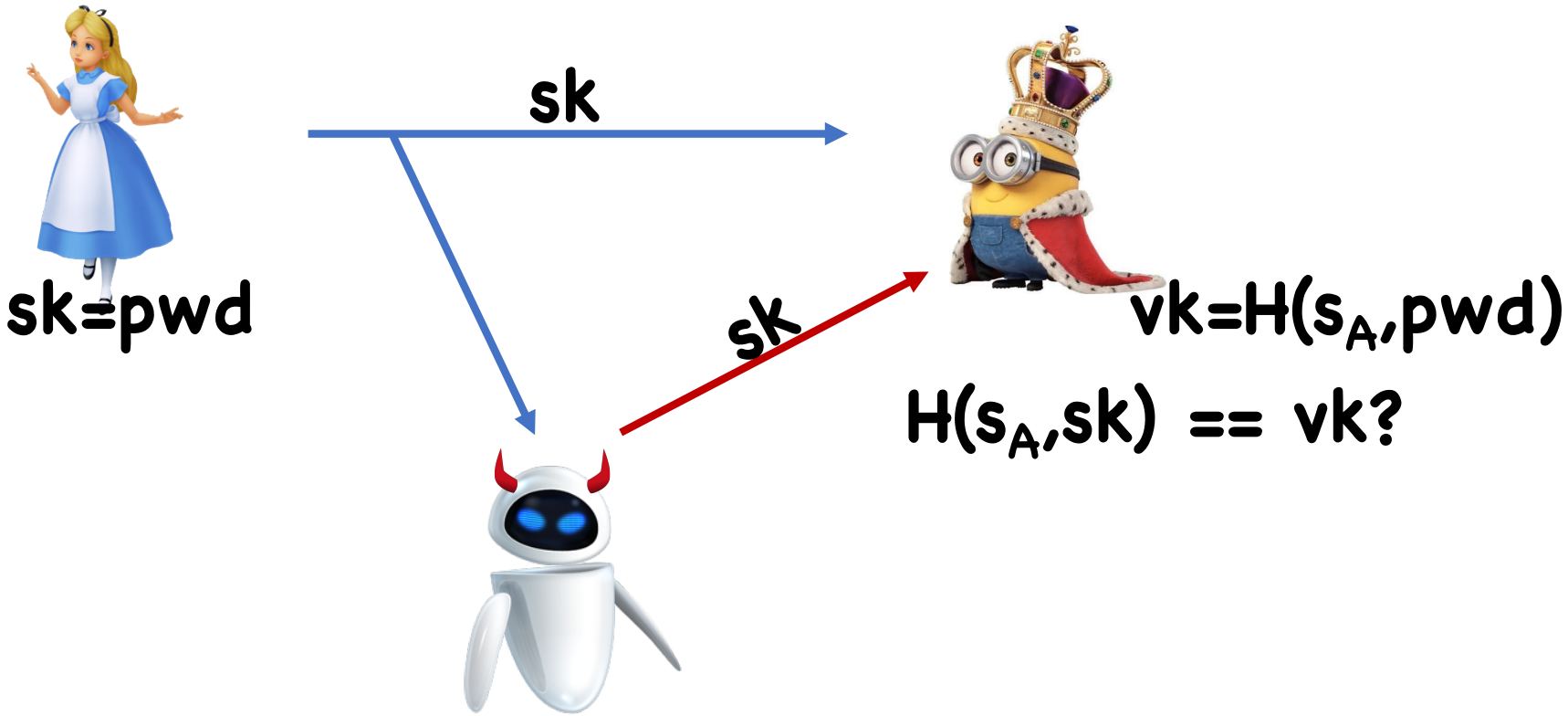
Again, never ever (ever ever….) use

- 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=(k,0)

sk_0 = F(k,0)

vk=(k,0)

sk_0 == F(k,0)?

One-time Passwords

Let **F** be a PRF

sk=(k,1)

$sk_1 = F(k,1)$

vk=(k,1)

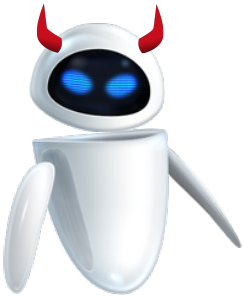
$sk_1 == F(k,1)?$

One-time Passwords

Let **F** be a PRF

sk=k,0

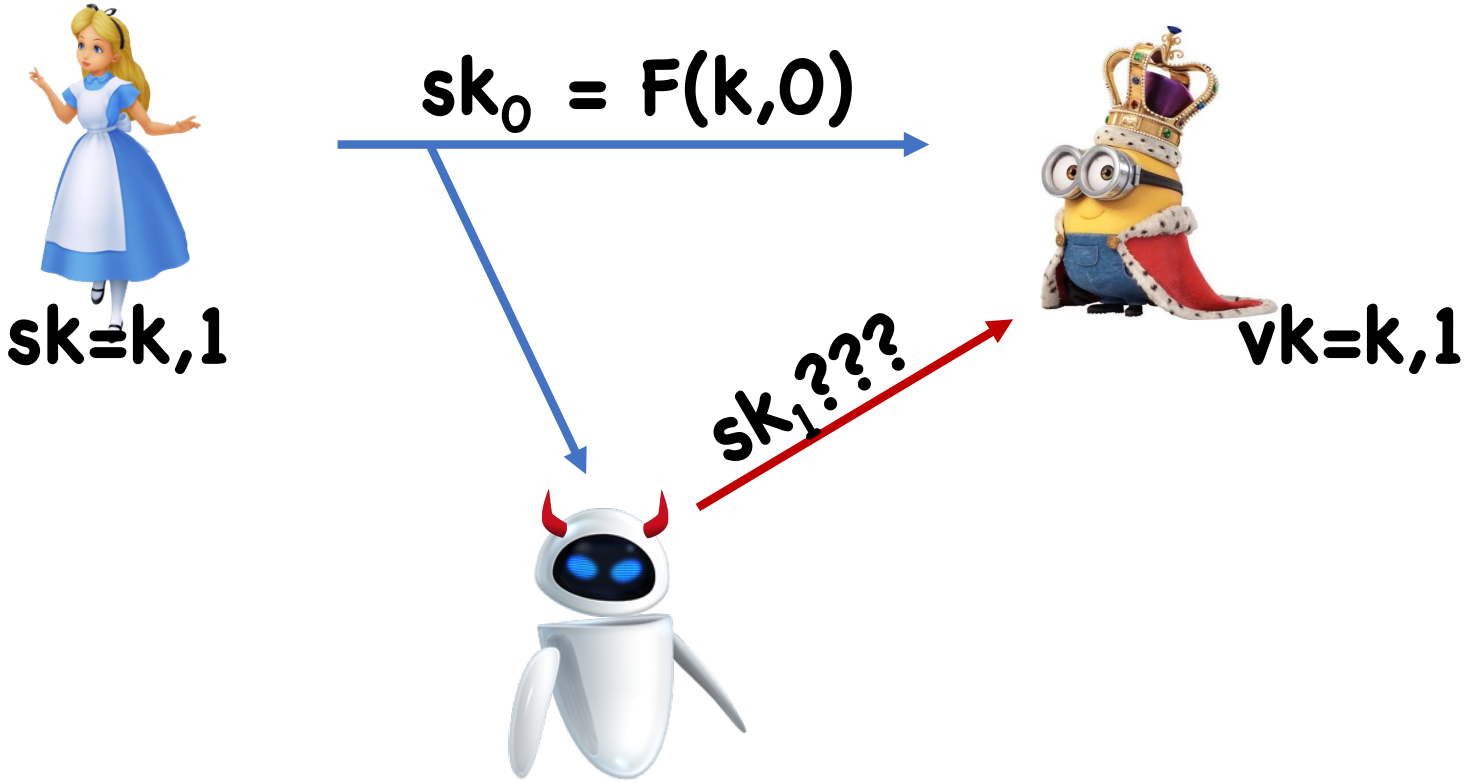
sk_0 = F(k,0)



vk=k,0

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

S/Key

Allow for **vk** to be public

sk = random string **k**

$$\mathbf{vk} = H^n(k) := \underbrace{H(H(H(\ldots H(x)\ldots)))}_{\mathbf{n}\text{ times}}$$

$$\mathbf{sk}_i = H^{n-i-1}(k)$$

$$\mathbf{vk}_i = H^{n-i}(k)$$

S/Key

$sk_0 = H^{n-1}(k)$

sk=k

$vk_0=H^n(k)$

$H(sk_0) == vk_0?$

S/Key

$sk = k$

$sk_1 = H^{n-2}(k)$

$vk_1 = sk_0 = H^{n-1}(k)$

$H(sk_1) == vk_1$?

S/Key

$sk = k$

$sk_2 = H^{n-3}(k)$

$vk_2 = sk_1 = H^{n-2}(k)$

$H(sk_2) == vk_2$?

S/Key

Now **vk** can be public

However, after **n** runs, need to reset

Stateless Schemes?

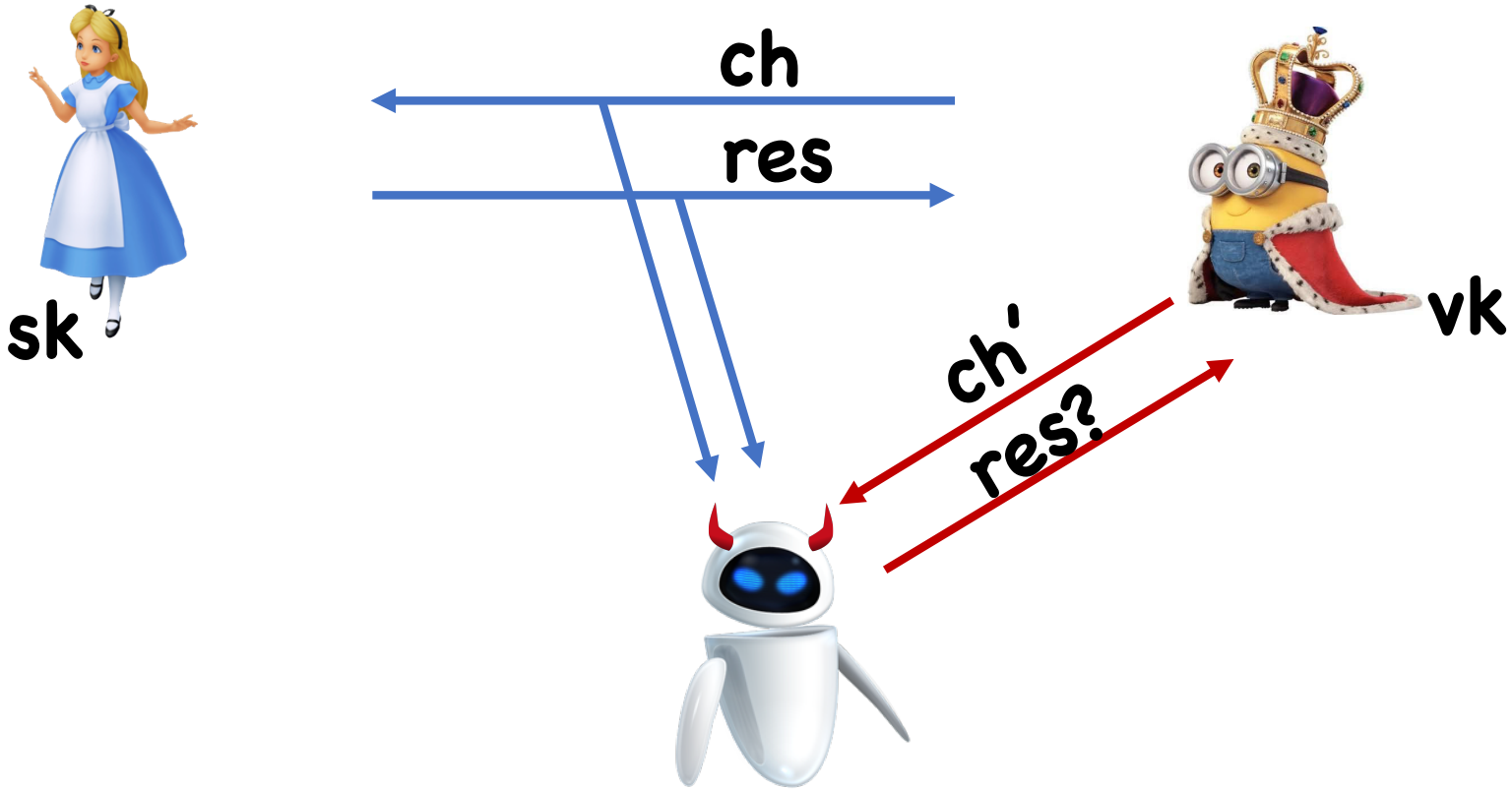
So far, all schemes secure against eavesdropping are stateless

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

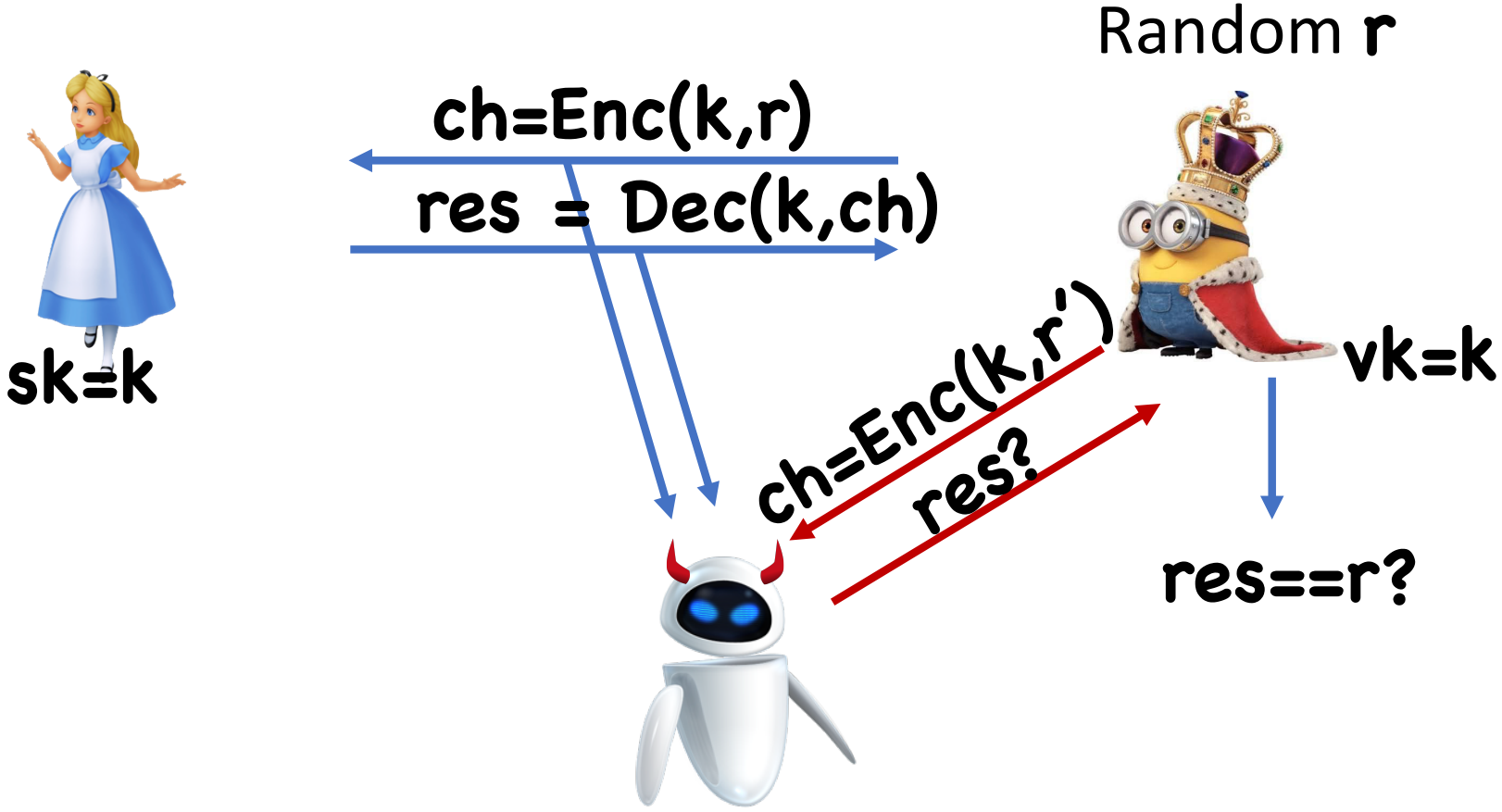
- Simply replay message

If want stateless scheme, instead want at least two messages

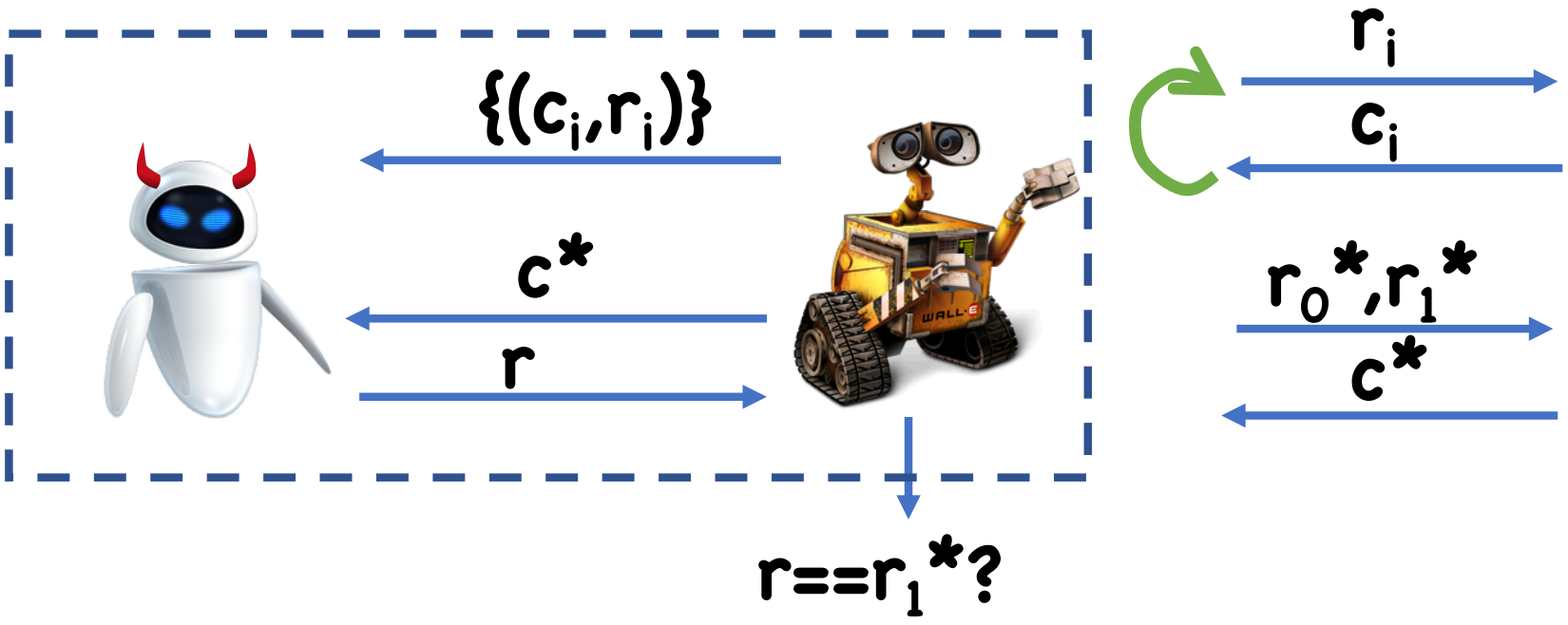
Challenge-Response



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

sk=k

ch=r

res = MAC(k,ch)

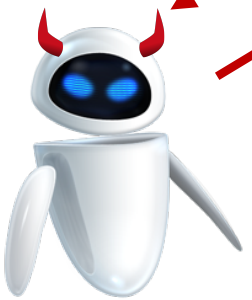
Random **r**

or r = Time

vk=k

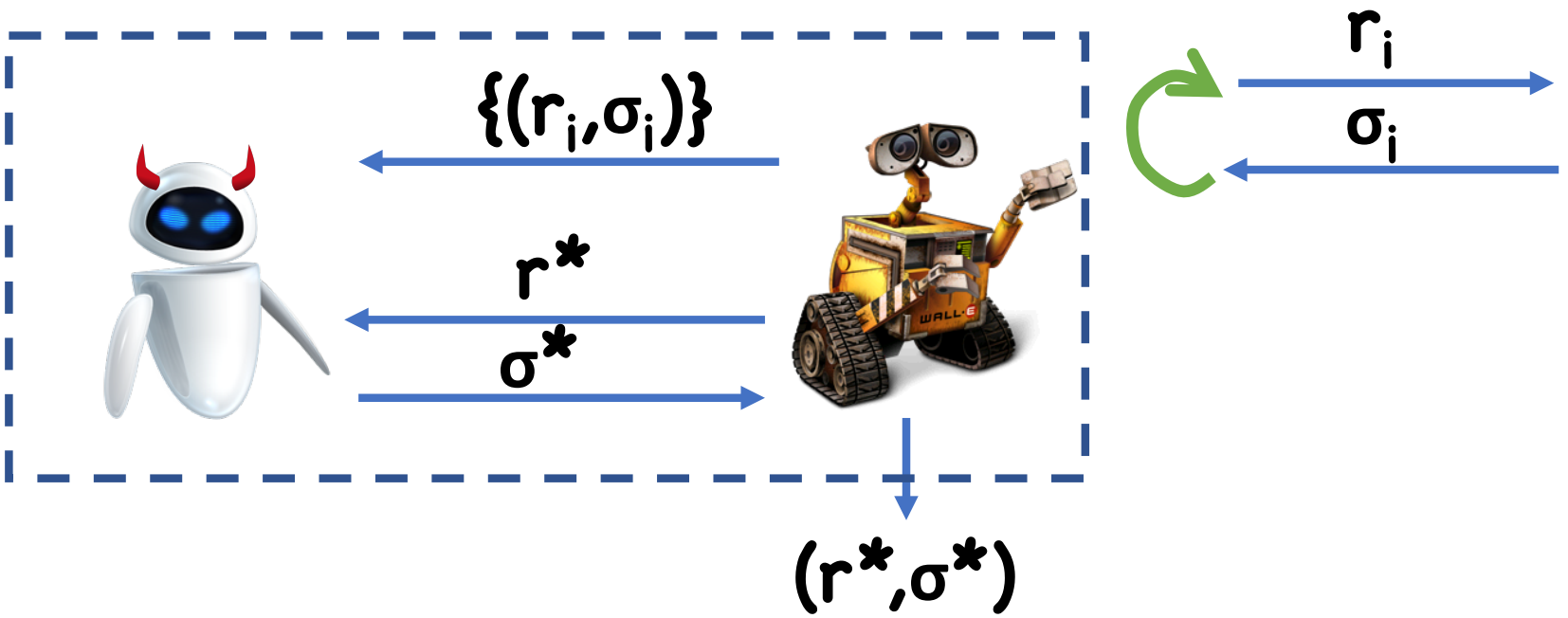
ch=r'

res?

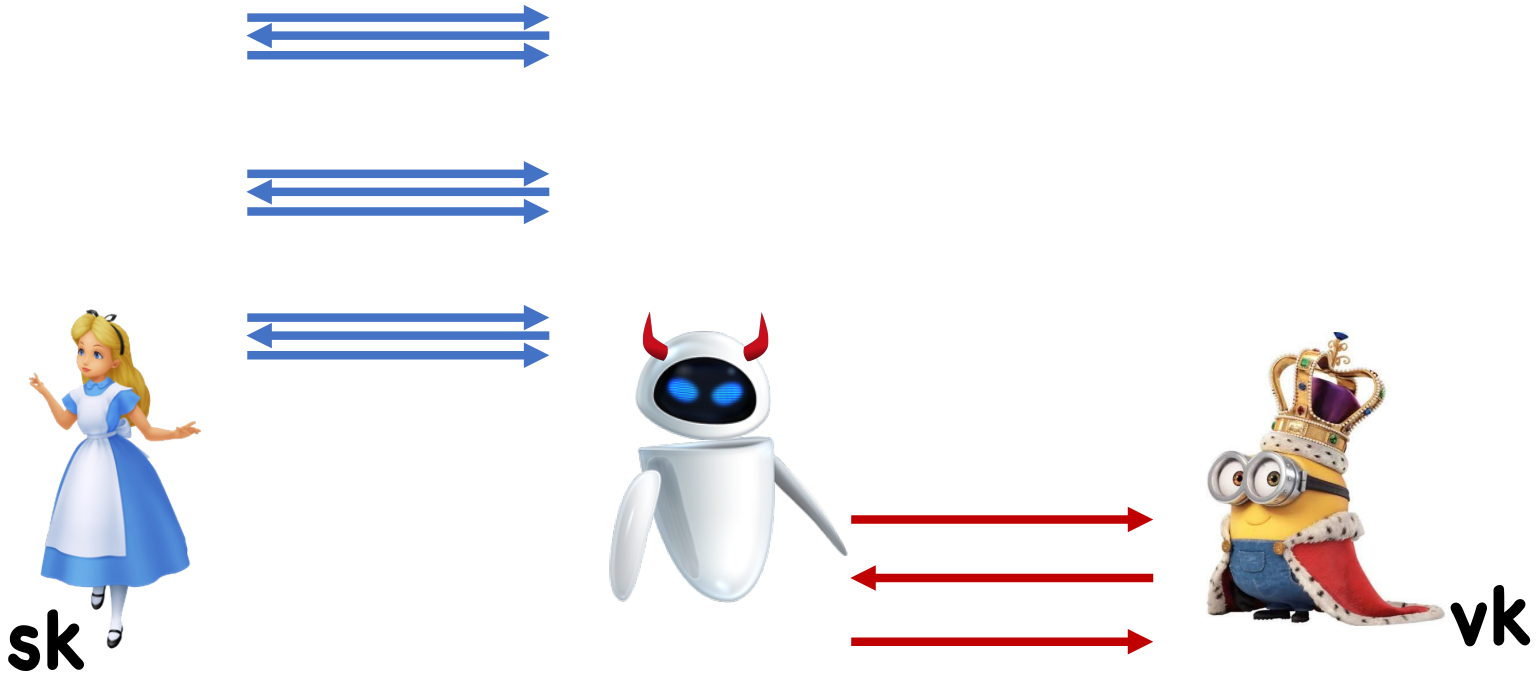


Ver(k,ch,res)?

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

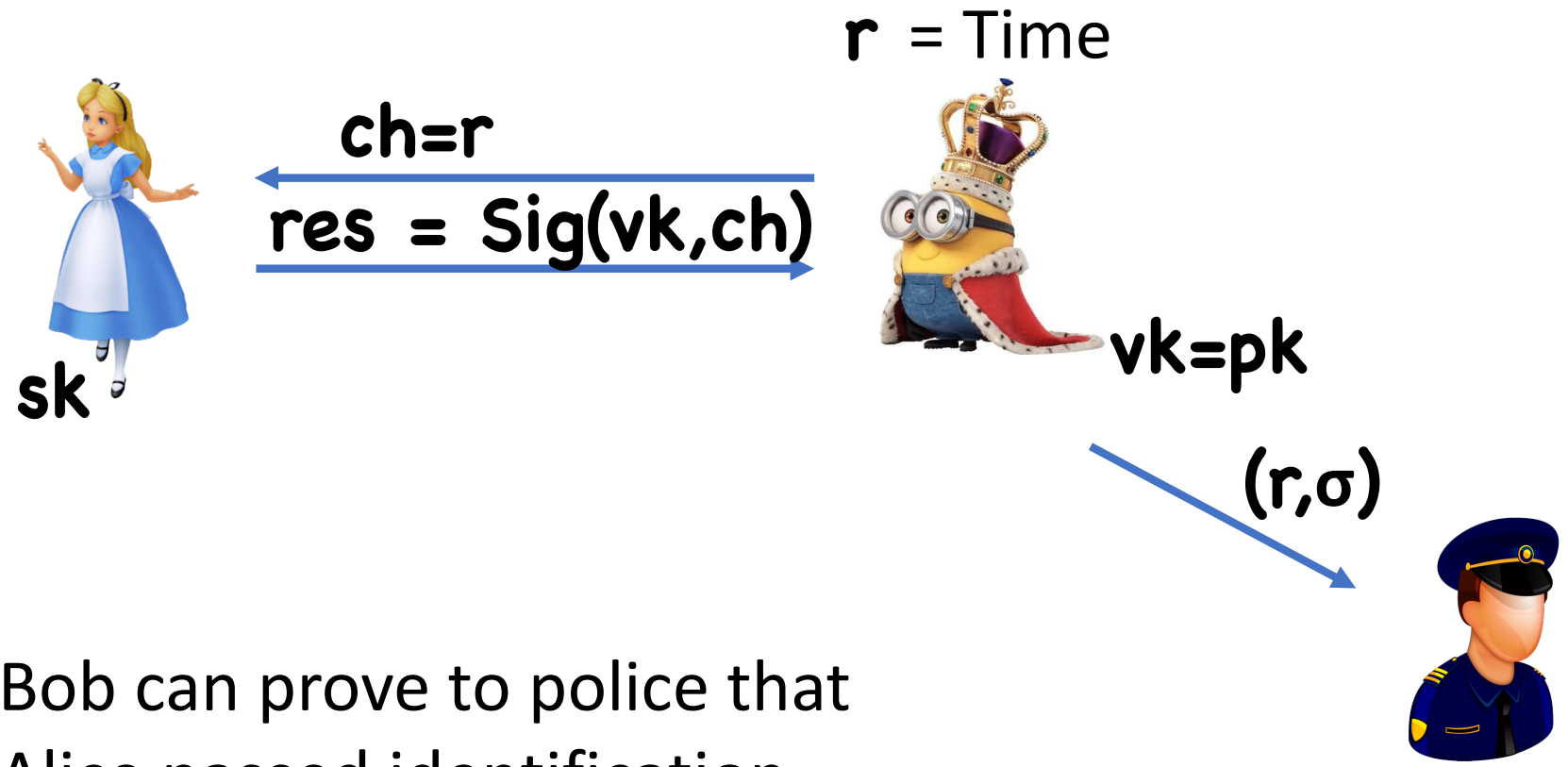
For enc-based C-R, CPA-secure is insufficient

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

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

Non-Repudiation

Consider signature-based C-R



Bob can prove to police that
Alice passed identification

Zero Knowledge

What if Bob could have come up with a valid transcript, without ever interacting with Alice?

- Then Bob cannot prove to police that Alice authenticated

Seems impossible:

- If (public) **vk** is sufficient to come up with valid transcript, why can't an adversary do the same?

Zero Knowledge

Adversary CAN come up with valid transcripts, but Bob doesn't accept transcripts

- Instead, accepts i*nteractions*

Ex: public key Enc-based C-R

- Valid transcript: **(c,r)** where **c** encrypts **r**
- Anyone can come up with a valid transcript
- However, only Alice can generate the transcript for a given **c**

Takeaway: order matters

Next Time

Zero knowledge proofs

- Prove a theorem without revealing how to prove it