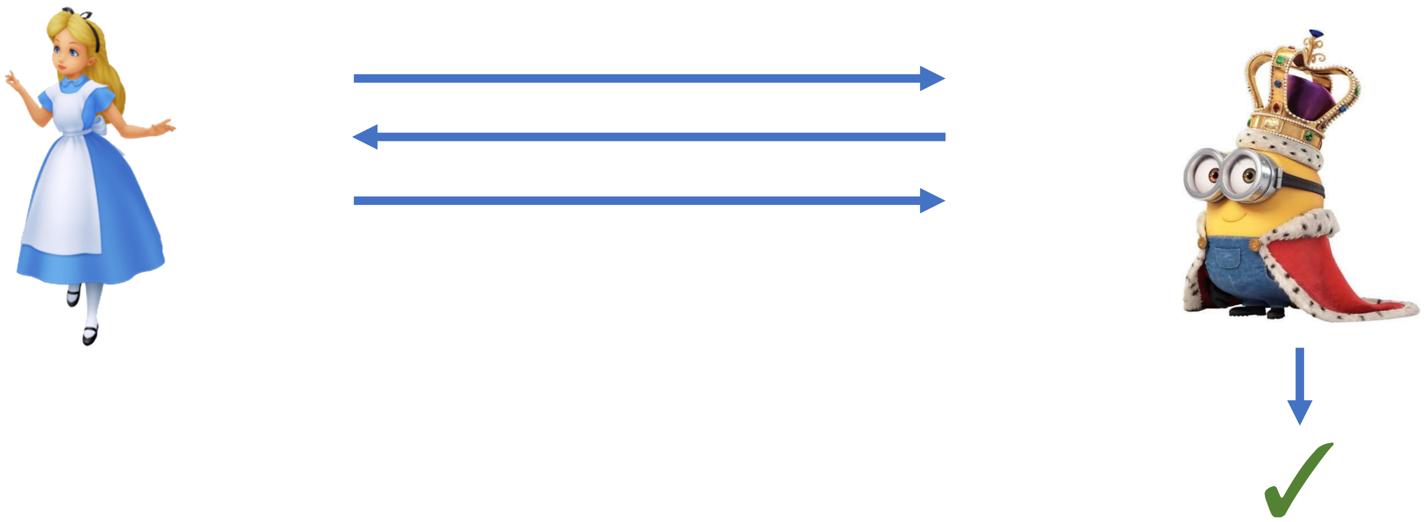
# COS433/Math 473: Cryptography

Mark Zhandry

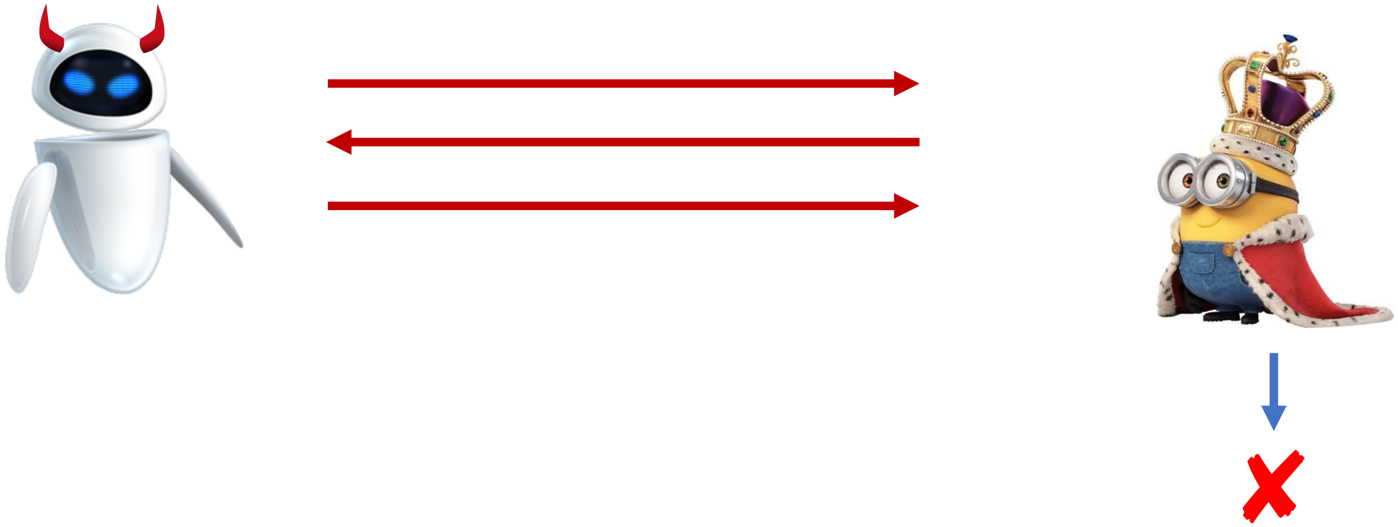
Princeton University

Spring 2018

# Identification



# Identification



# Identification

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

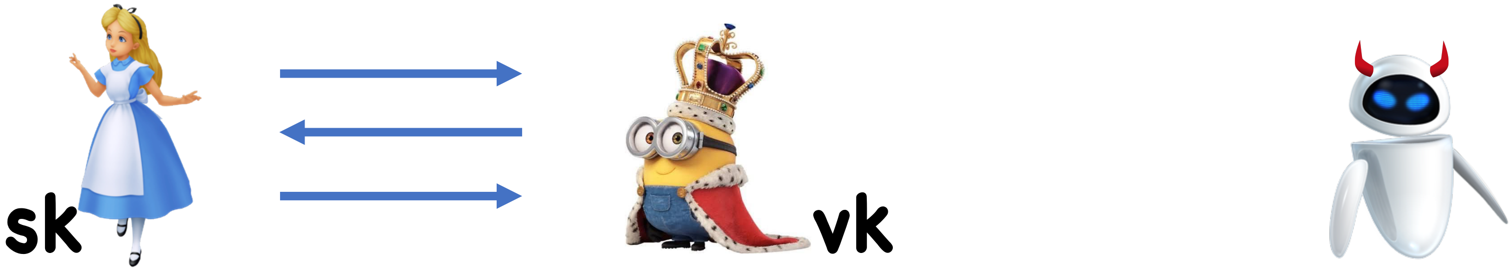
Typical factors:

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

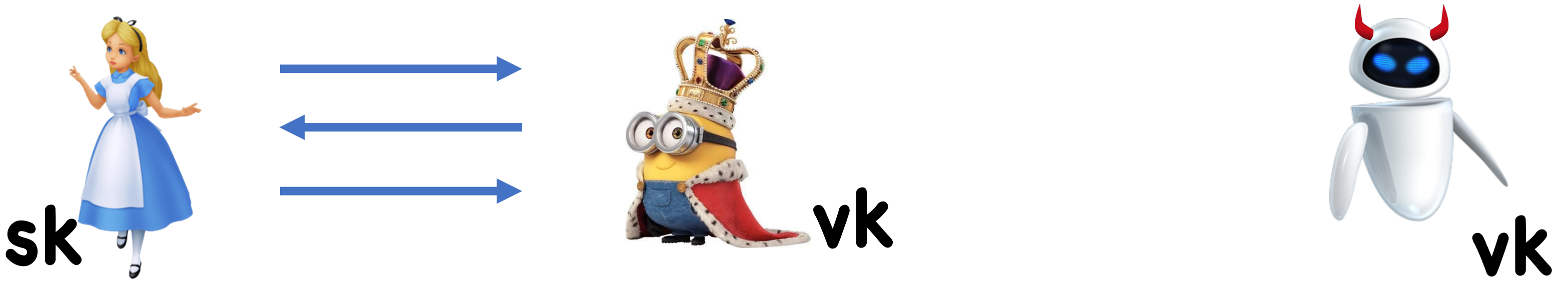
Today

# Types of Identification Protocols

Secret key:

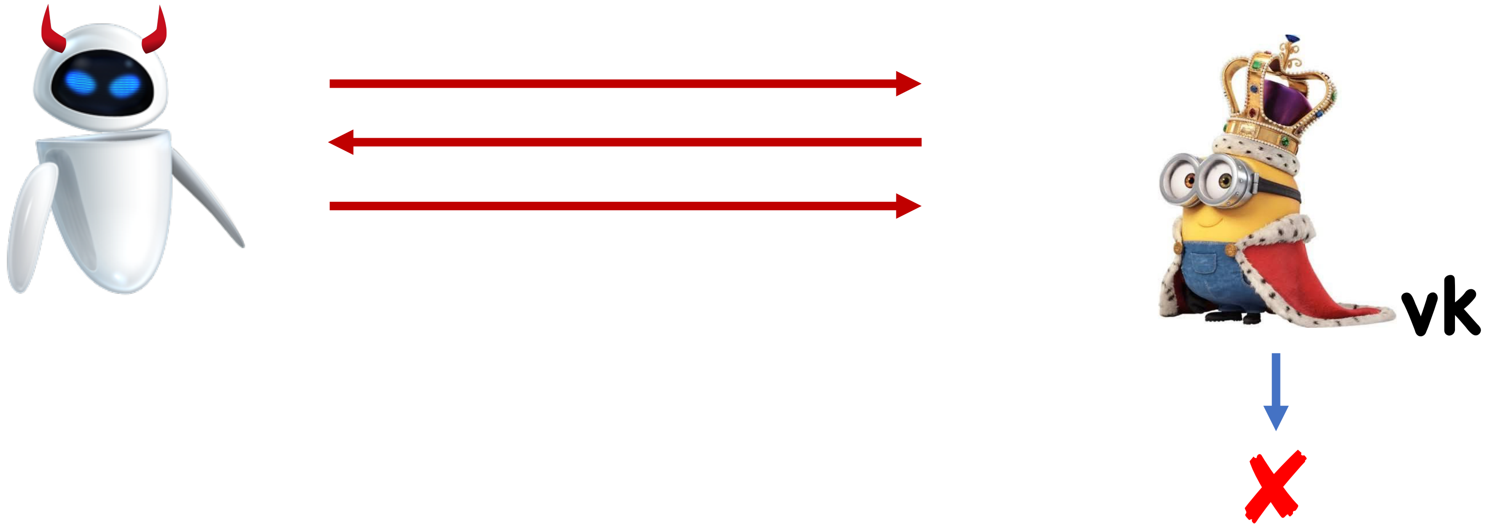


Public Key:



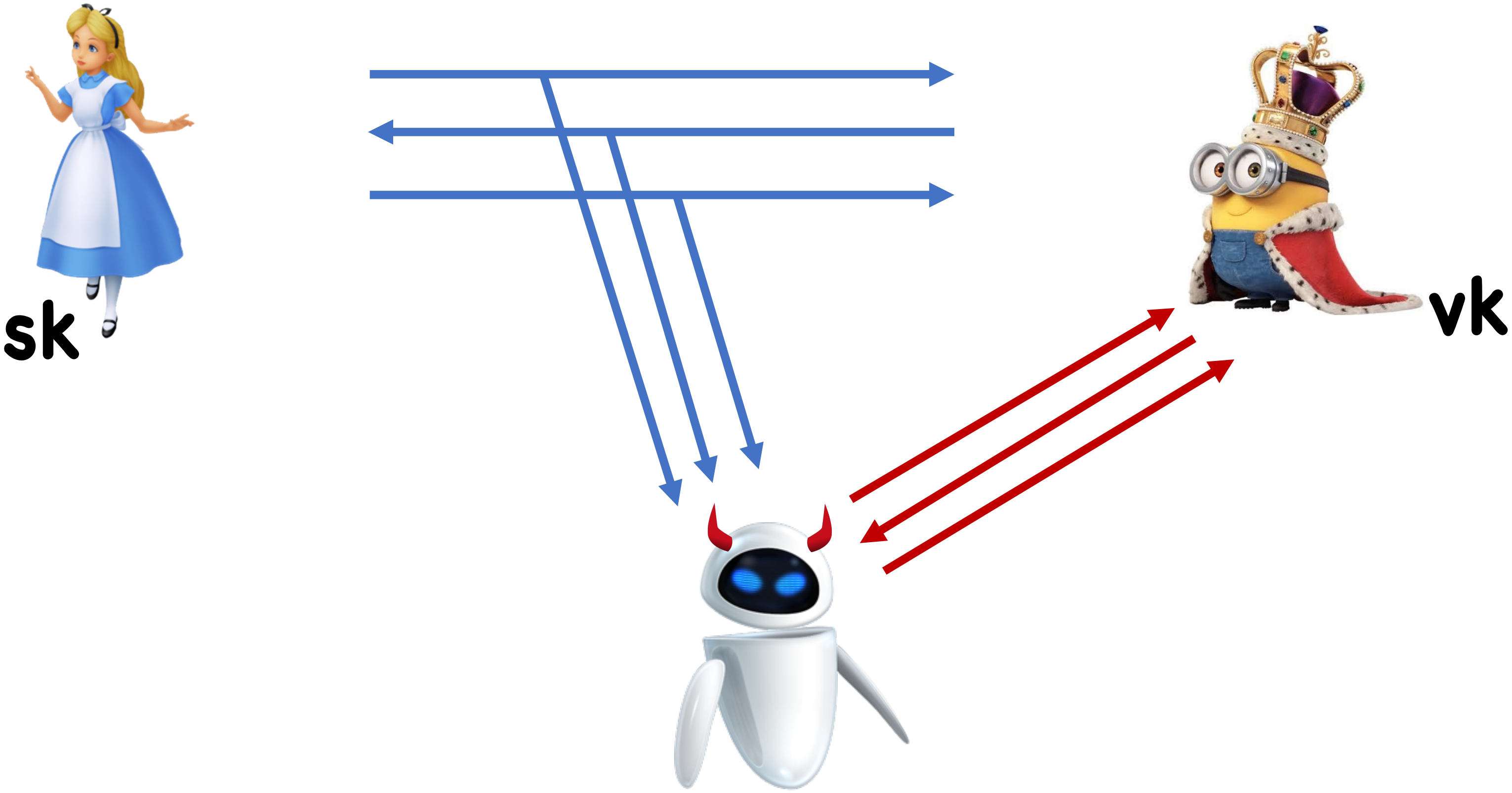
# Types of Attacks

## Direct Attack:



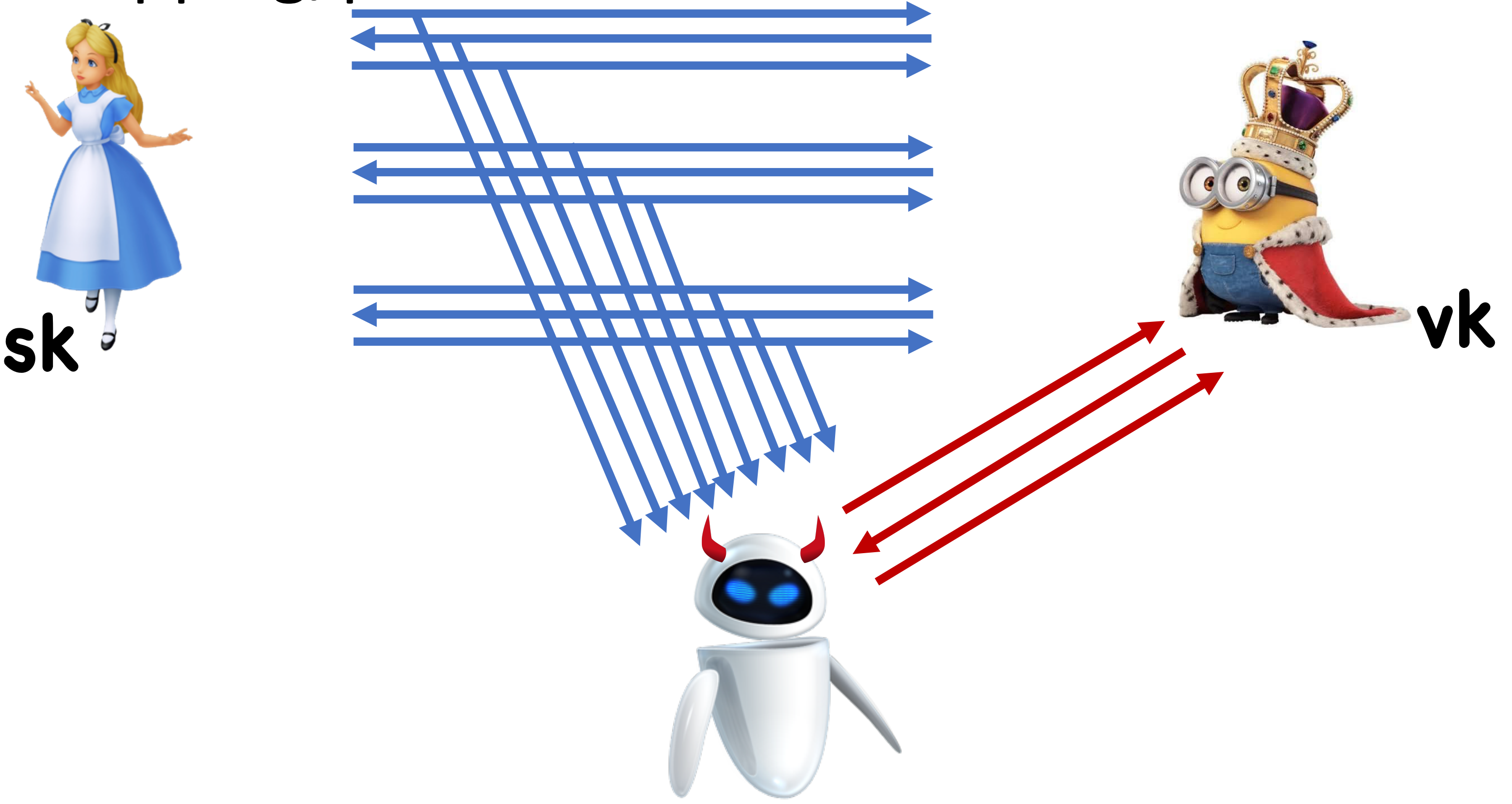
# Types of Attacks

Eavesdropping/passive:



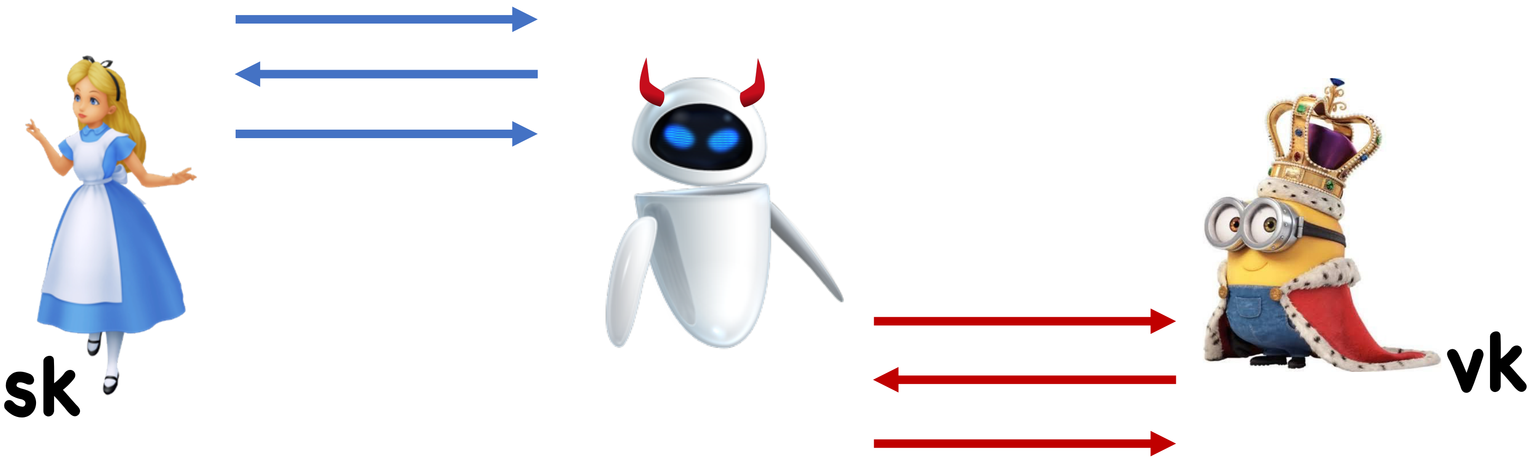
# Types of Attacks

Eavesdropping/passive:



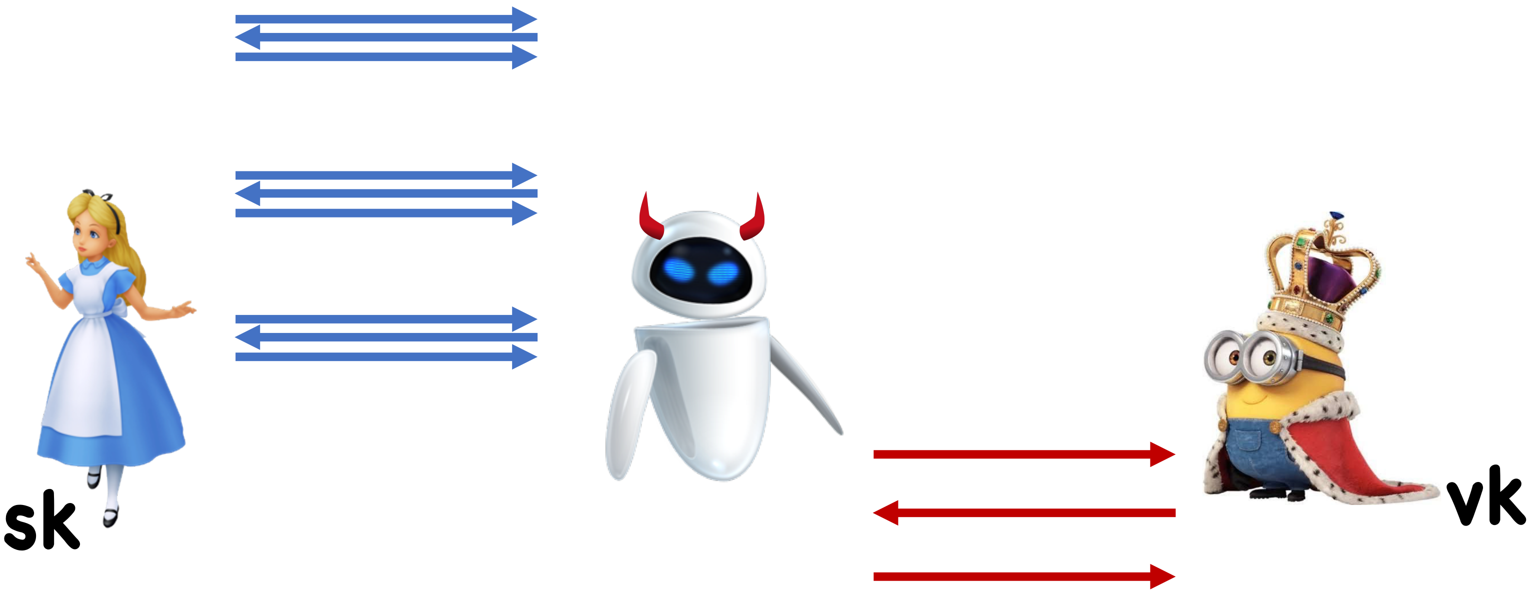
# Types of Attacks

Man-in-the-Middle/Active:



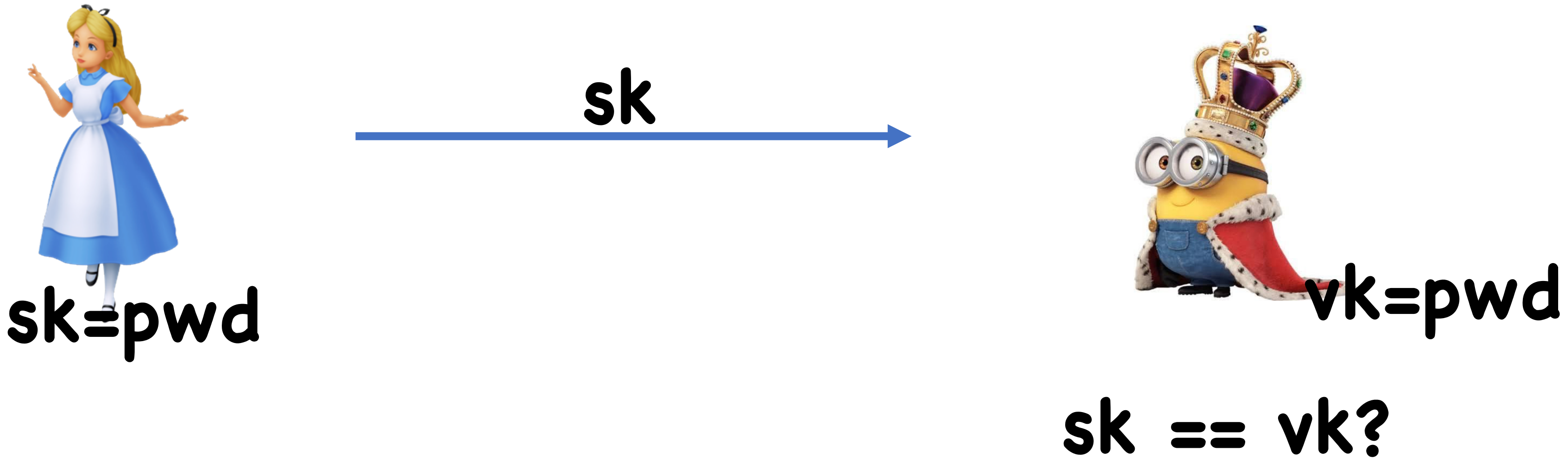
# Types of Attacks

Man-in-the-Middle/Active:



# Basic Password Protocol

Never ever (ever ever…) use



sk

sk=pwd

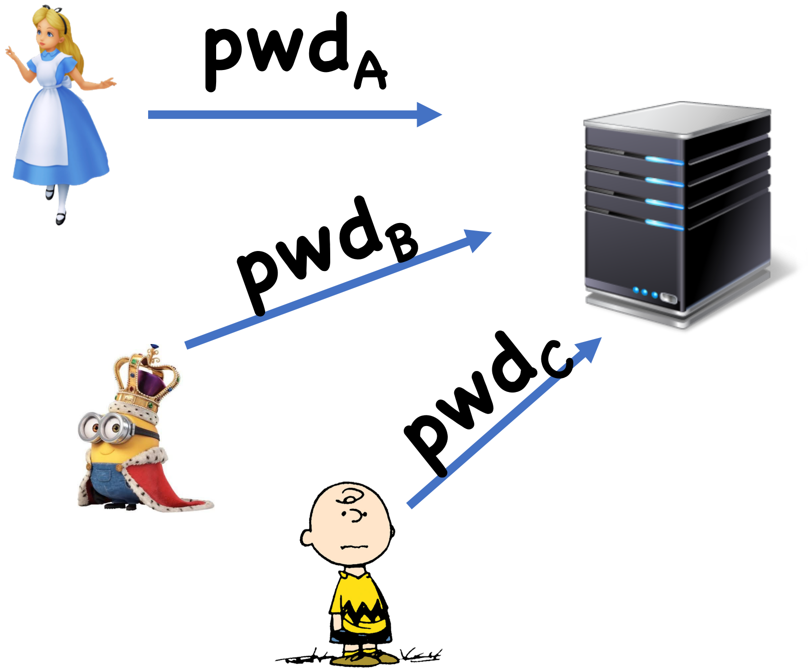
vk=pwd

sk == vk?

# Problem with Basic Pwd Protocol

**vk** must be kept secret at all costs

Issue:

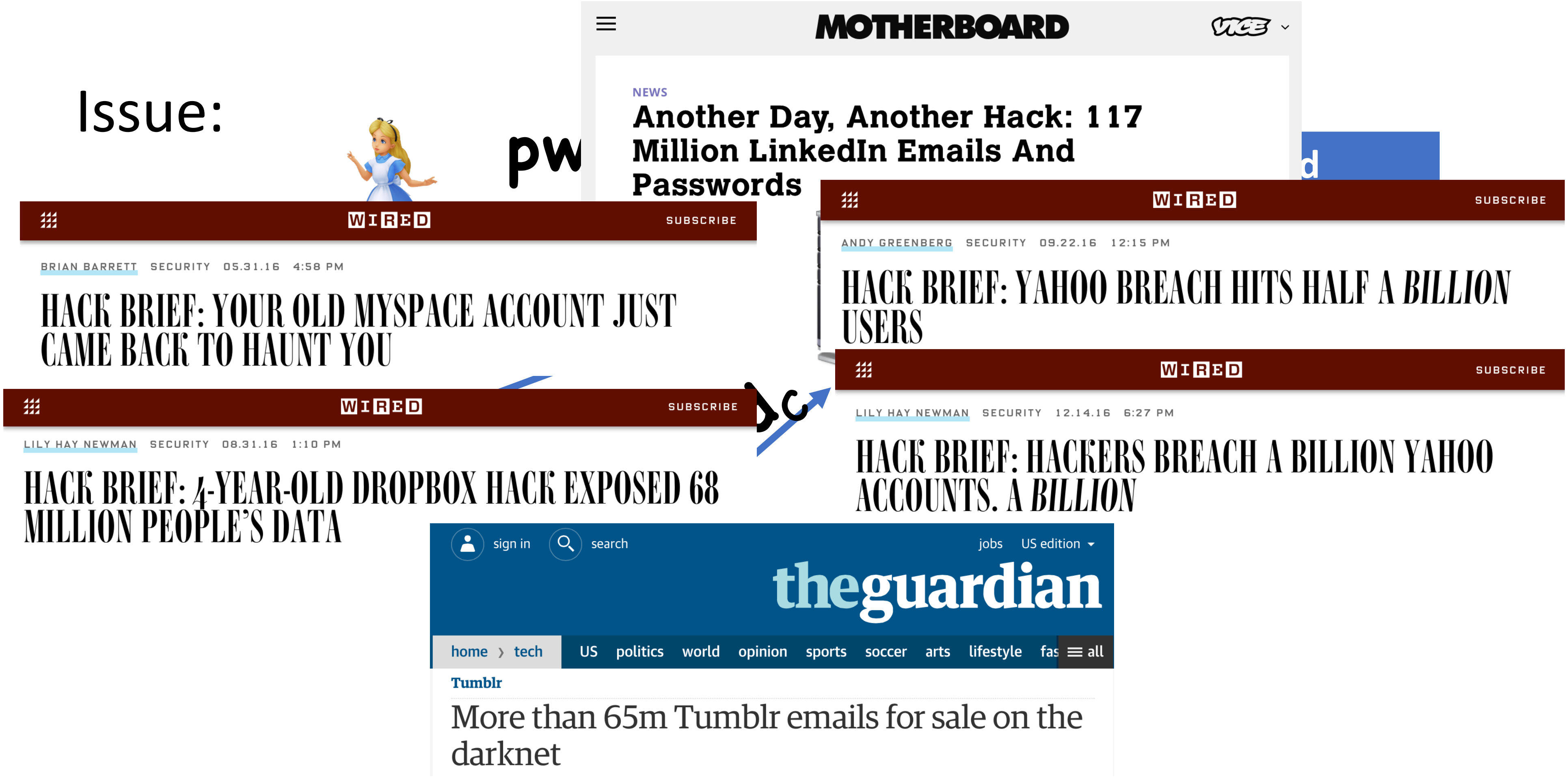


| User | Pwd |
|---|---|
| Alice | $pwd_A$ |
| Bob | $pwd_B$ |
| Charlie | $pwd_C$ |
| … | … |

# Problem with Basic Pwd Protocol

**vk** must be kept secret at all costs

Issue:



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*

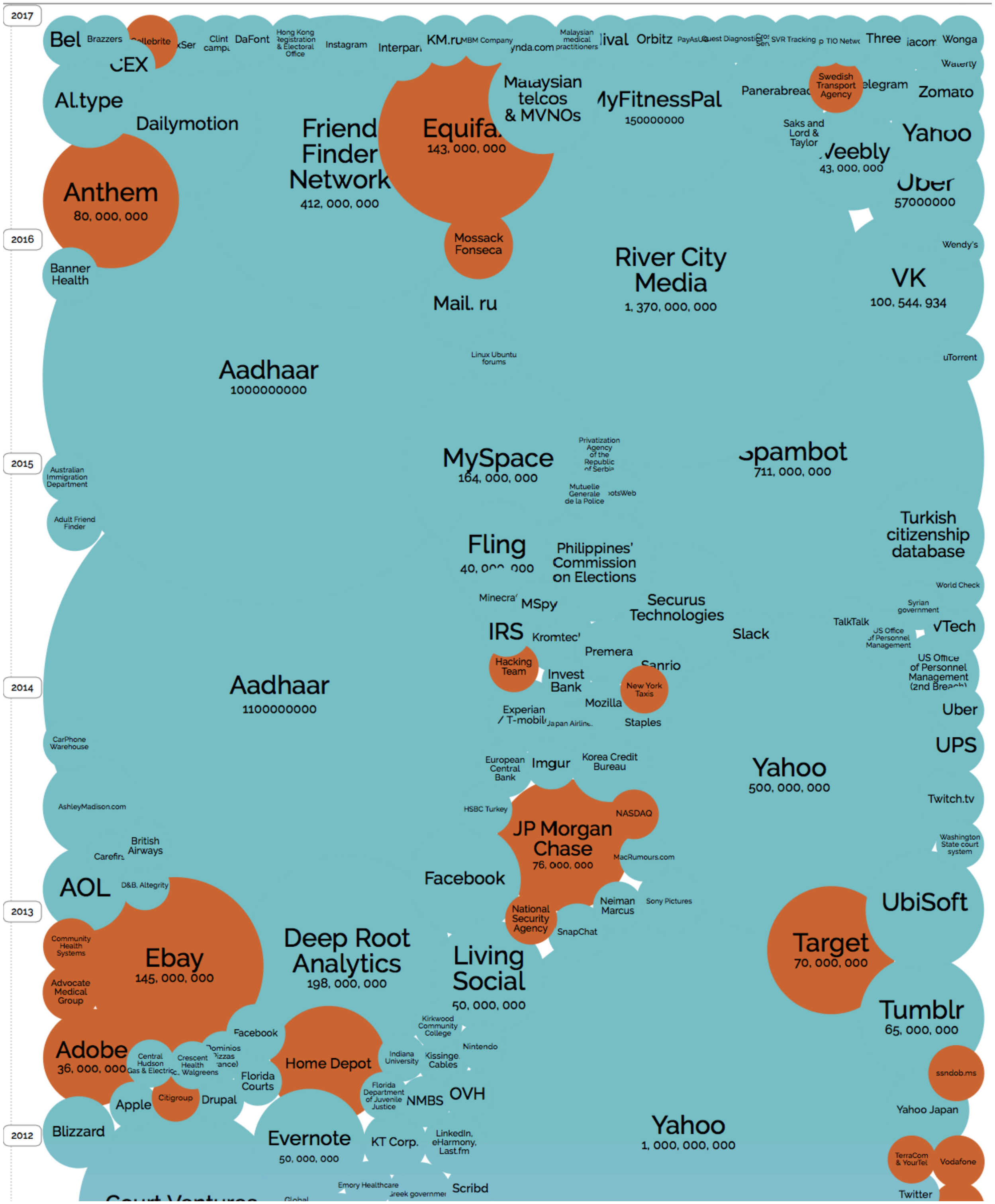
sign in search jobs US edition

theguardian

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

Tumblr

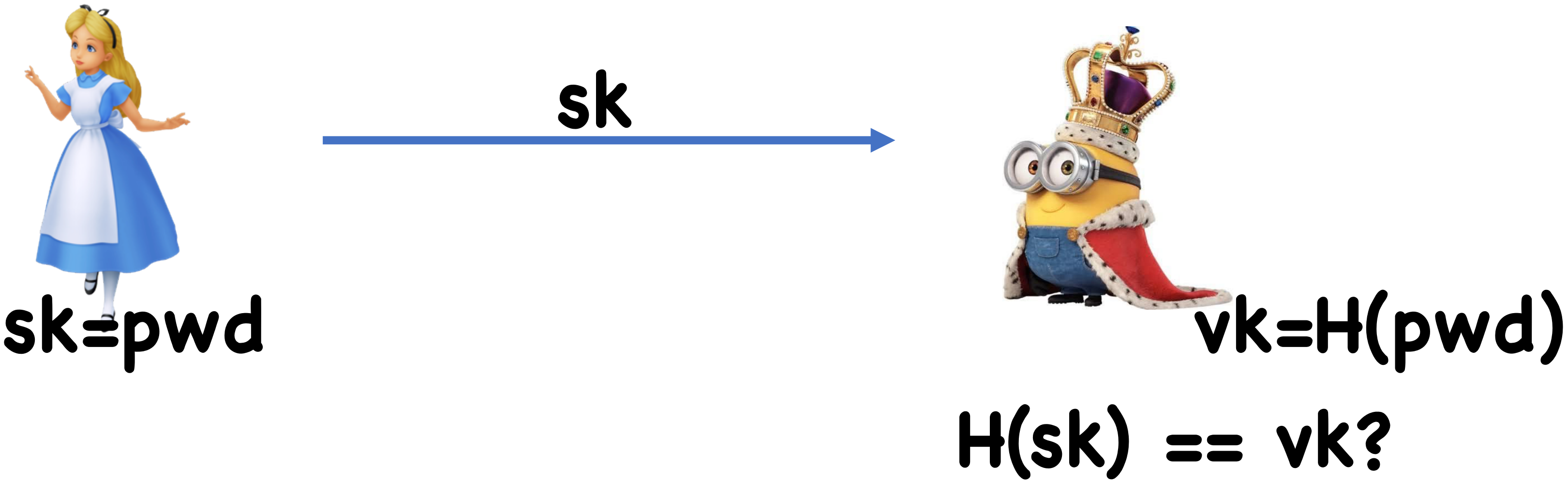
More than 65m Tumblr emails for sale on the darknet


2017
2016
2015
2014
2013
2012
Bel
Brazzers
Cellebrite
xSer
Clint
campa
DaFont
Hong Kong
Registration
& Electoral
Office
Instagram
Interpark
KM.ru
MBM Company
ynda.com
Malaysian
medical
practitioners
Rival
Orbitz
PayAsUGym
Quest Diagnostics
Cros
Serv
SVR Tracking
p TIO Netwo
Three
iacom
Wonga
CEX
AI.type
Dailymotion
Friend
Finder
Network
412, 000, 000
Equifax
143, 000, 000
Malaysian
telcos
& MVNOs
MyFitnessPal
150000000
Panerabread
Swedish
Transport
Agency
Telegram
Waterly
Zomato
Saks and
Lord &
Taylor
Weebly
43, 000, 000
Yahoo
Uber
57000000
Anthem
80, 000, 000
Mossack
Fonseca
Banner
Health
Mail. ru
River City
Media
1, 370, 000, 000
Wendy's
VK
100, 544, 934
Aadhaar
1000000000
Linux Ubuntu
forums
uTorrent
Privatization
Agency
of the
Republic
of Serbia
MySpace
164, 000, 000
Mutuelle
Generale
de la Police
BotsWeb
Spambot
711, 000, 000
Australian
Immigration
Department
Adult Friend
Finder
Fling
40, 000, 000
Philippines'
Commission
on Elections
Turkish
citizenship
database
World Check
Minecraft
MSpy
Securus
Technologies
Syrian
government
TalkTalk
US Office
of Personnel
Management
VTech
IRS
Kromtech
Slack
Premera
Hacking
Team
Sanrio
US Office
of Personnel
Management
(2nd Breach)
Invest
Bank
New York
Taxis
Aadhaar
1100000000
Mozilla
Uber
Experian
/ T-mobile
Japan Airlines
Staples
CarPhone
Warehouse
UPS
European
Central
Bank
Imgur
Korea Credit
Bureau
Yahoo
500, 000, 000
AshleyMadison.com
HSBC Turkey
NASDAQ
Twitch.tv
JP Morgan
Chase
76, 000, 000
British
Airways
Washington
State court
system
Carefirst
MacRumours.com
AOL
D&B, Altegrity
Facebook
Neiman
Marcus
Sony Pictures
UbiSoft
National
Security
Agency
SnapChat
Community
Health
Systems
Ebay
145, 000, 000
Deep Root
Analytics
198, 000, 000
Living
Social
50, 000, 000
Target
70, 000, 000
Advocate
Medical
Group
Tumblr
65, 000, 000
Kirkwood
Community
College
Facebook
Nintendo
Dominios
Pizzas
(France)
Adobe
36, 000, 000
Central
Hudson
Gas & Electric
Crescent
Health
Inc., Walgreens
Home Depot
Indiana
University
Kissinger
Cables
ssndob.ms
Florida
Courts
Florida
Department
of Juvenile
Justice
NMBS
OVH
Apple
Citigroup
Drupal
Yahoo Japan
Blizzard
Evernote
50, 000, 000
KT Corp.
LinkedIn,
eHarmony,
Last.fm
Yahoo
1, 000, 000, 000
TerraCom
& YourTel
Vodafone
Emory Healthcare
Greek government
Scribd
Twitter
Court Ventures
Global

# Slightly Better Version

STILL never ever (ever ever…) use

Let **H** be a hash function



sk

sk=pwd

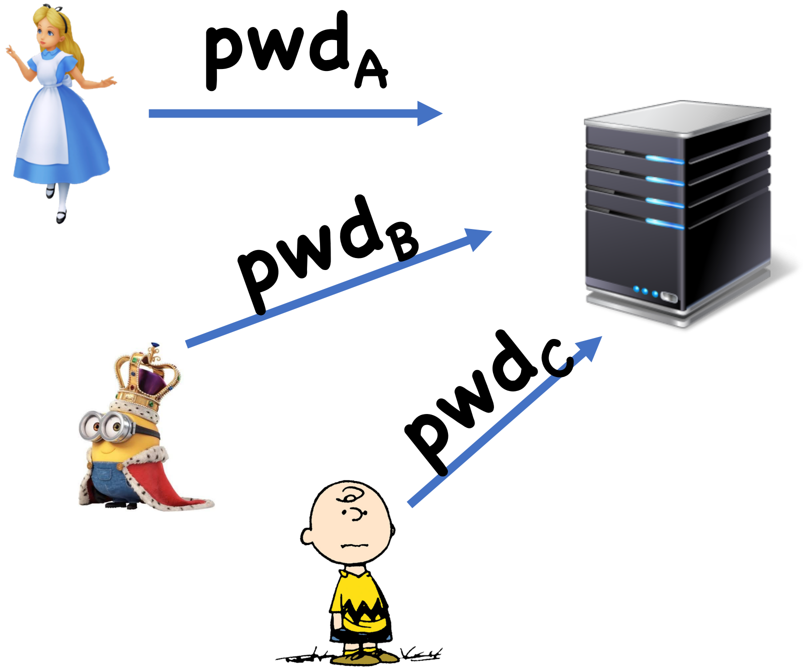
vk=H(pwd)

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

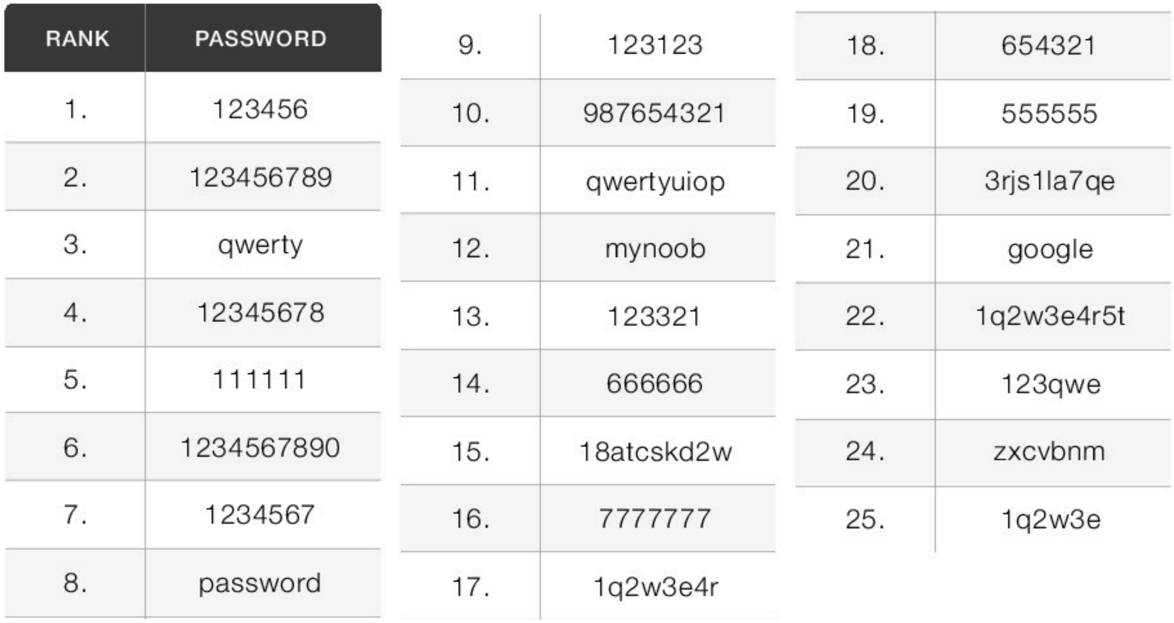
- Therefore, adversary needs to invert hash function to break protocol

- 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 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

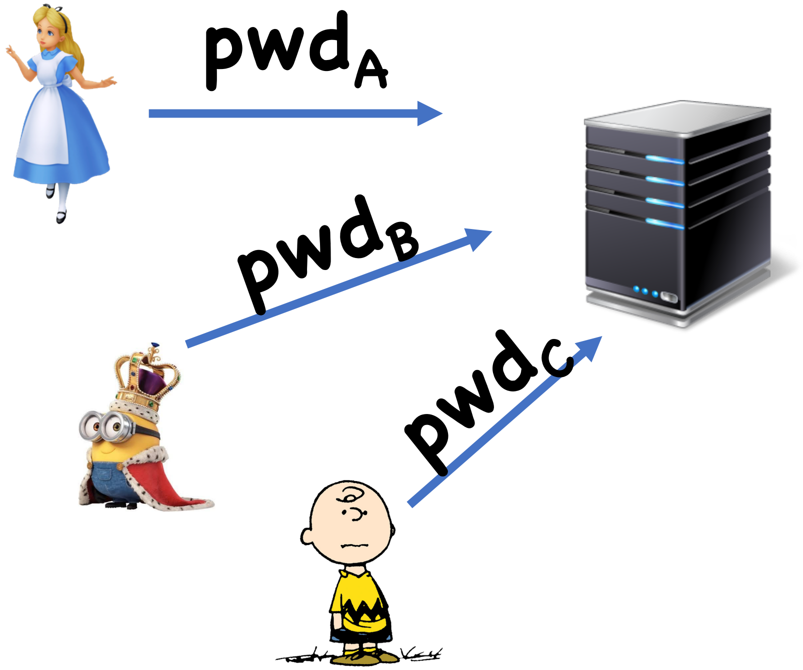
- Hash dictionary once: **$O(|D|)$**
- Index dictionary by hashes
- Lookup each database entry in dictionary: **$O(|L|)$**

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 $\mathbf{s_A}$
- To recover Bob's key, adversary must hash entire dictionary with $\mathbf{s_B}$
- Must hash entire dictionary again for each user

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

# 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

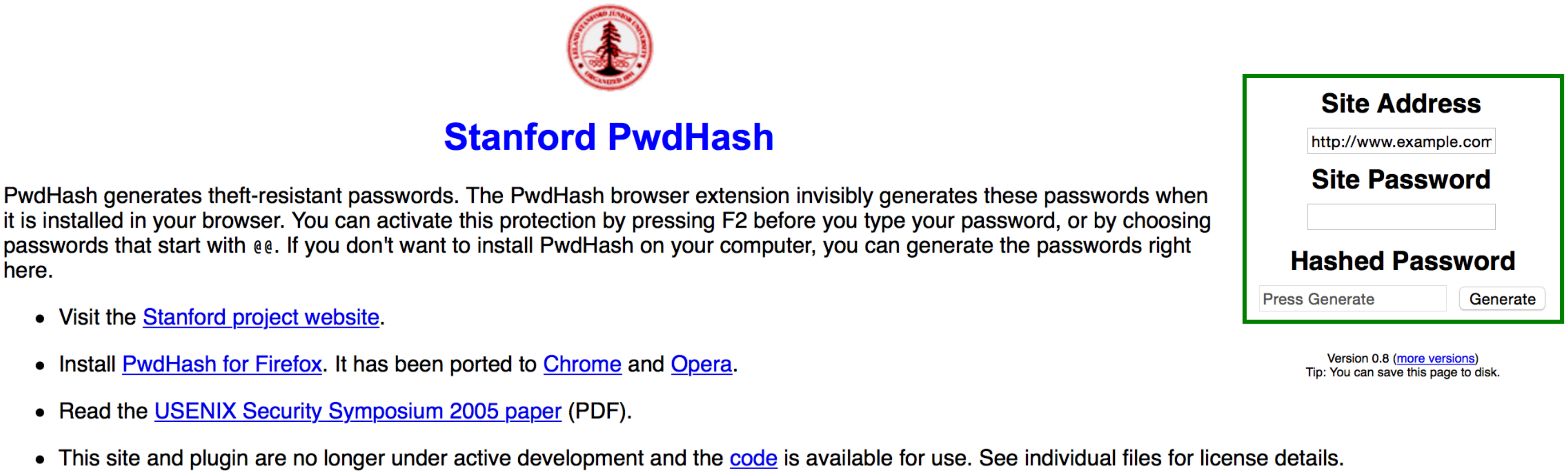
Solutions:

- 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**

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

# 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

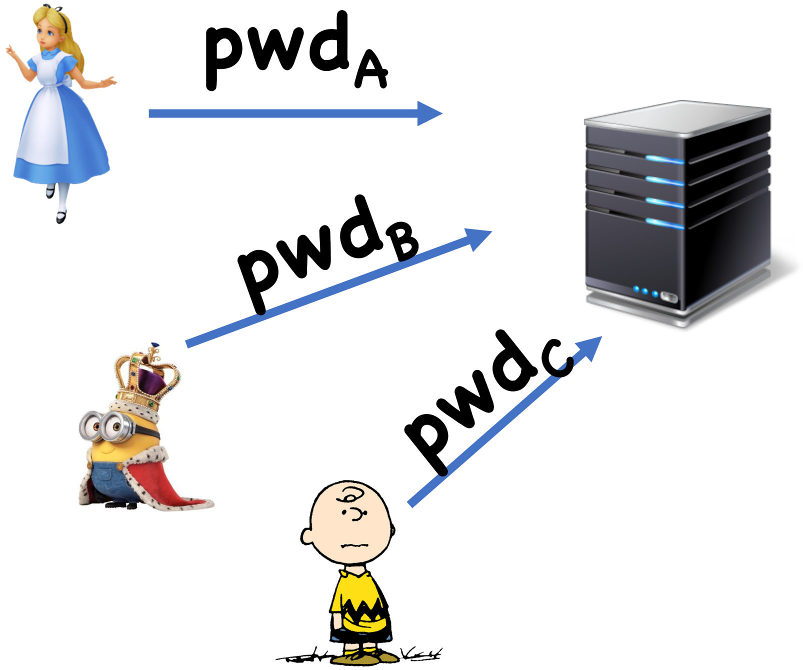
# What Hash Function to Use

Instead, often 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?

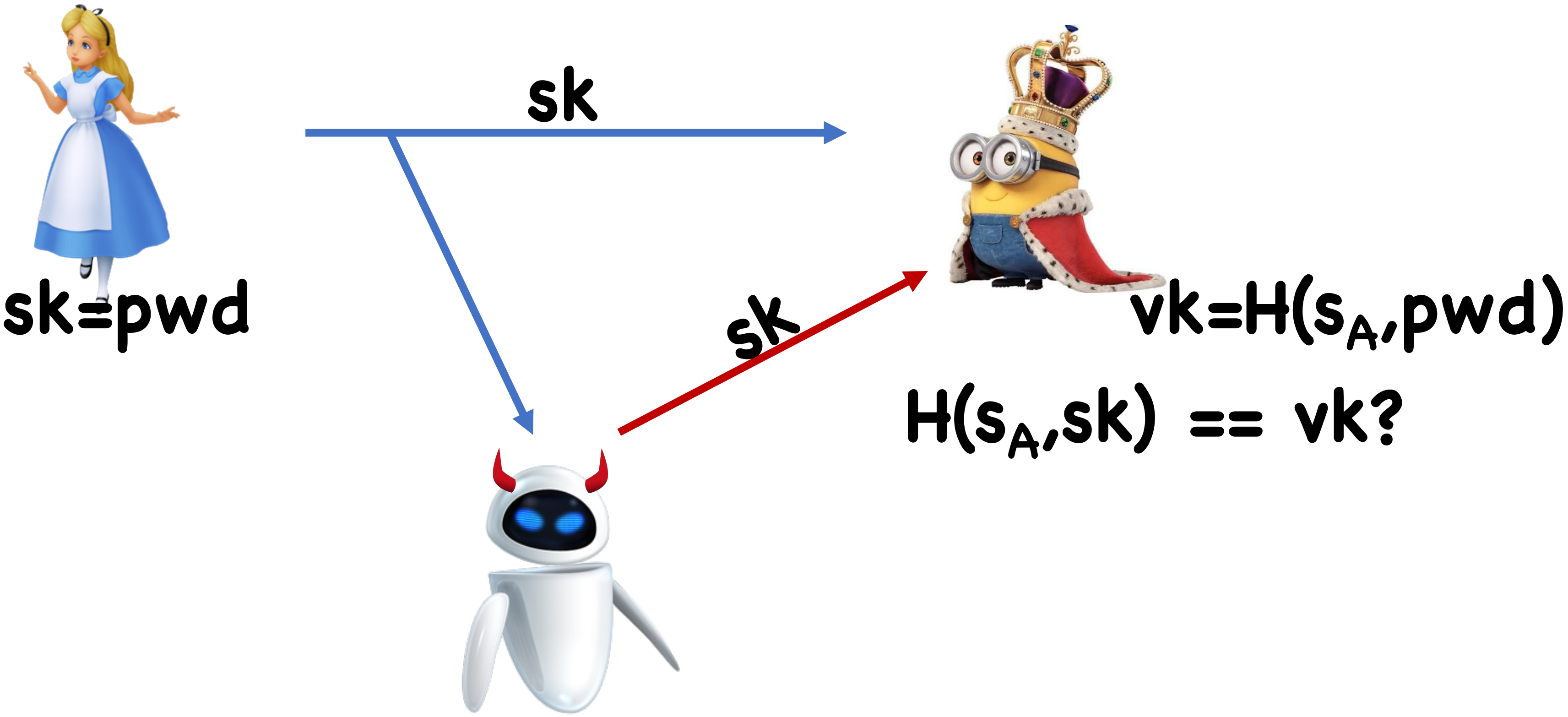
Again, never ever (ever ever….) use

- 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



sk

sk=pwd

sk

vk=H($s_A$,pwd)

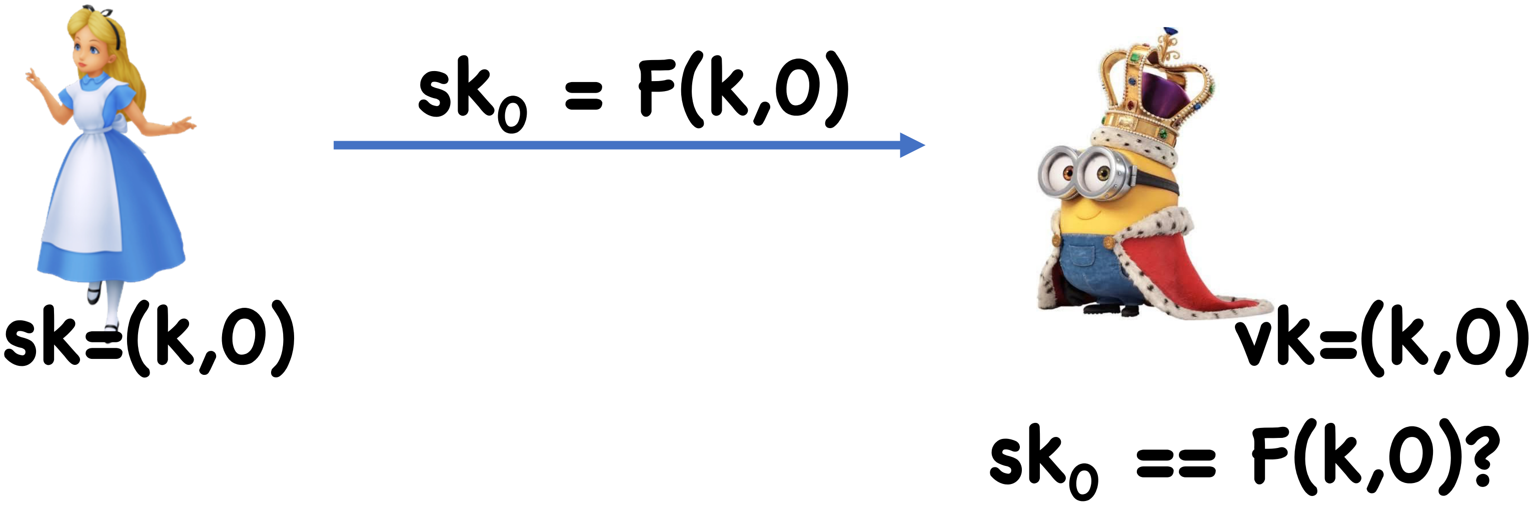
H($s_A$,sk) == vk?

# 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

$sk = (k,1)$

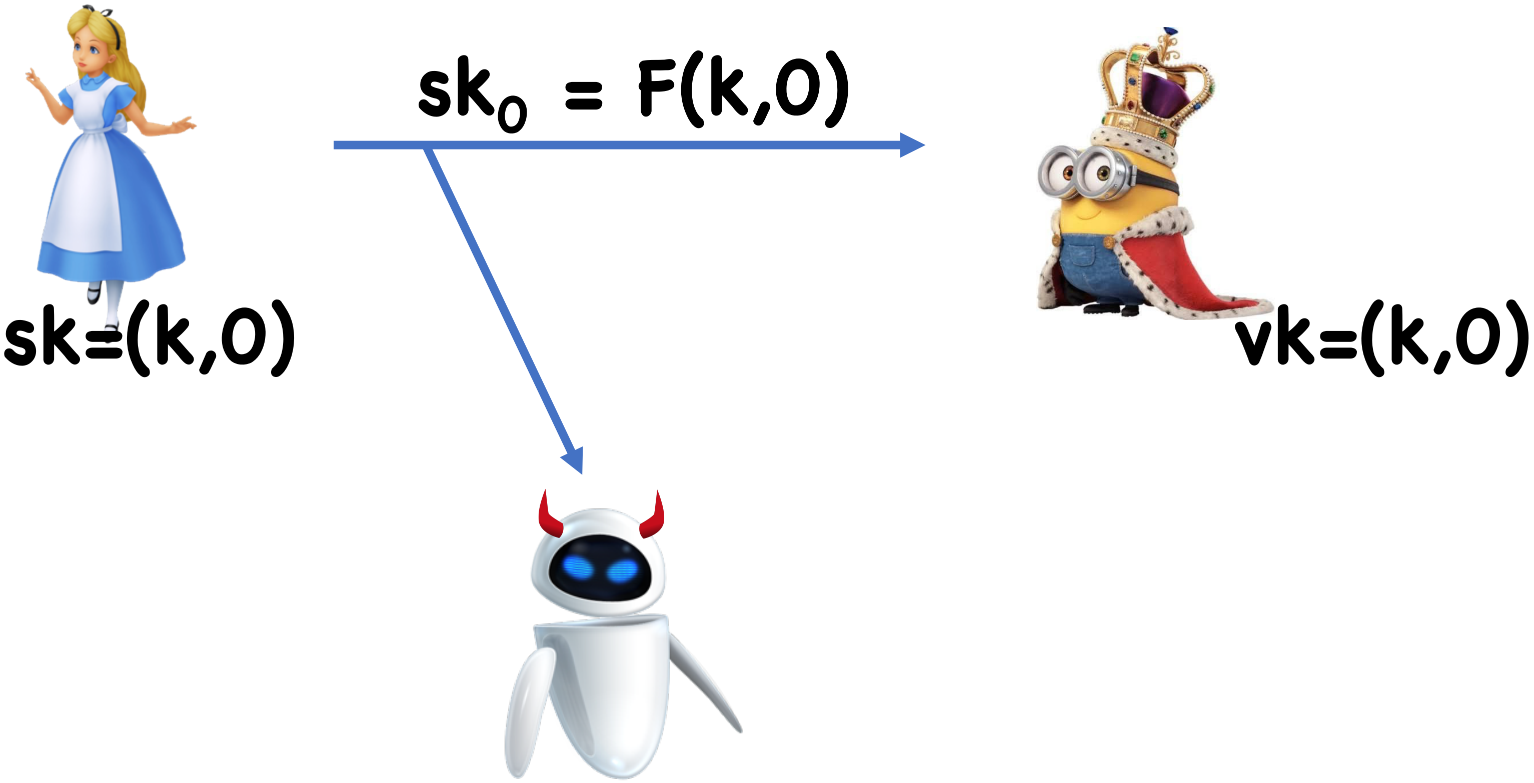
$sk_1 = F(k,1)$ →

$vk = (k,1)$

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

# One-time Passwords

Let **F** be a PRF



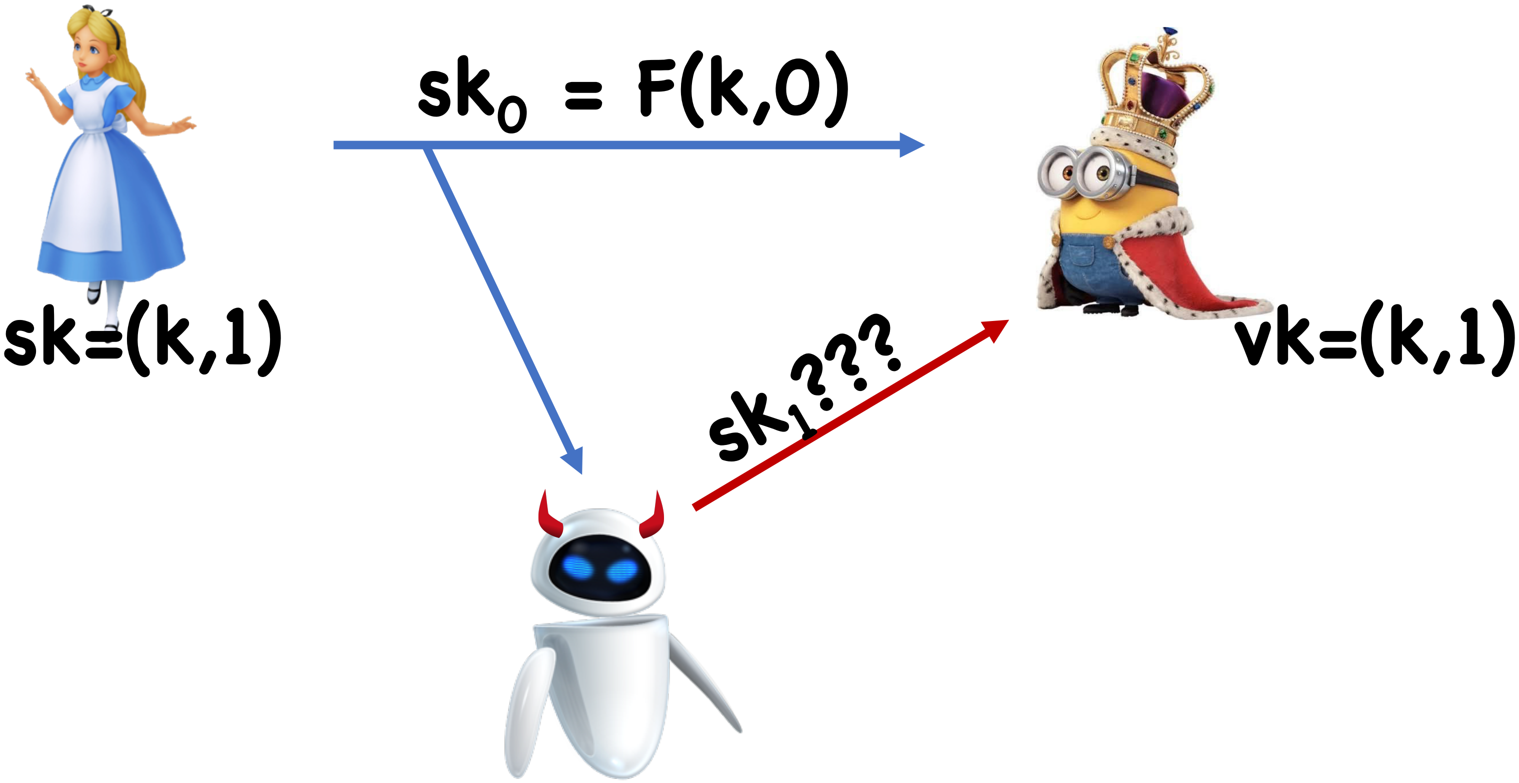
$sk_0 = F(k,0)$

$sk=(k,0)$

$vk=(k,0)$

# One-time Passwords

Let **F** be a PRF



$sk_0 = F(k,0)$

$sk=(k,1)$

$vk=(k,1)$

$sk_1???$

# 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**

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

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

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

# S/Key

$sk=k$

$sk_0 = H^{n-1}(k)$ →

$vk_0=H^n(k)$

$H(sk_0) == vk_0?$

# S/Key

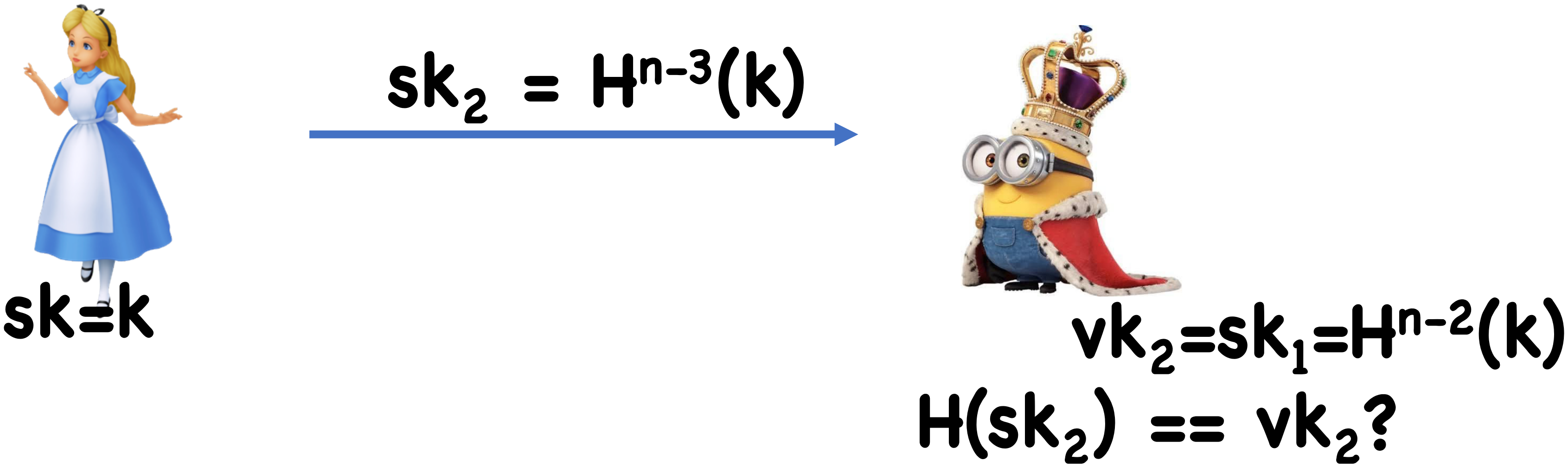
$sk=k$

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

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

$H(sk_1) == vk_1$?

# 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?

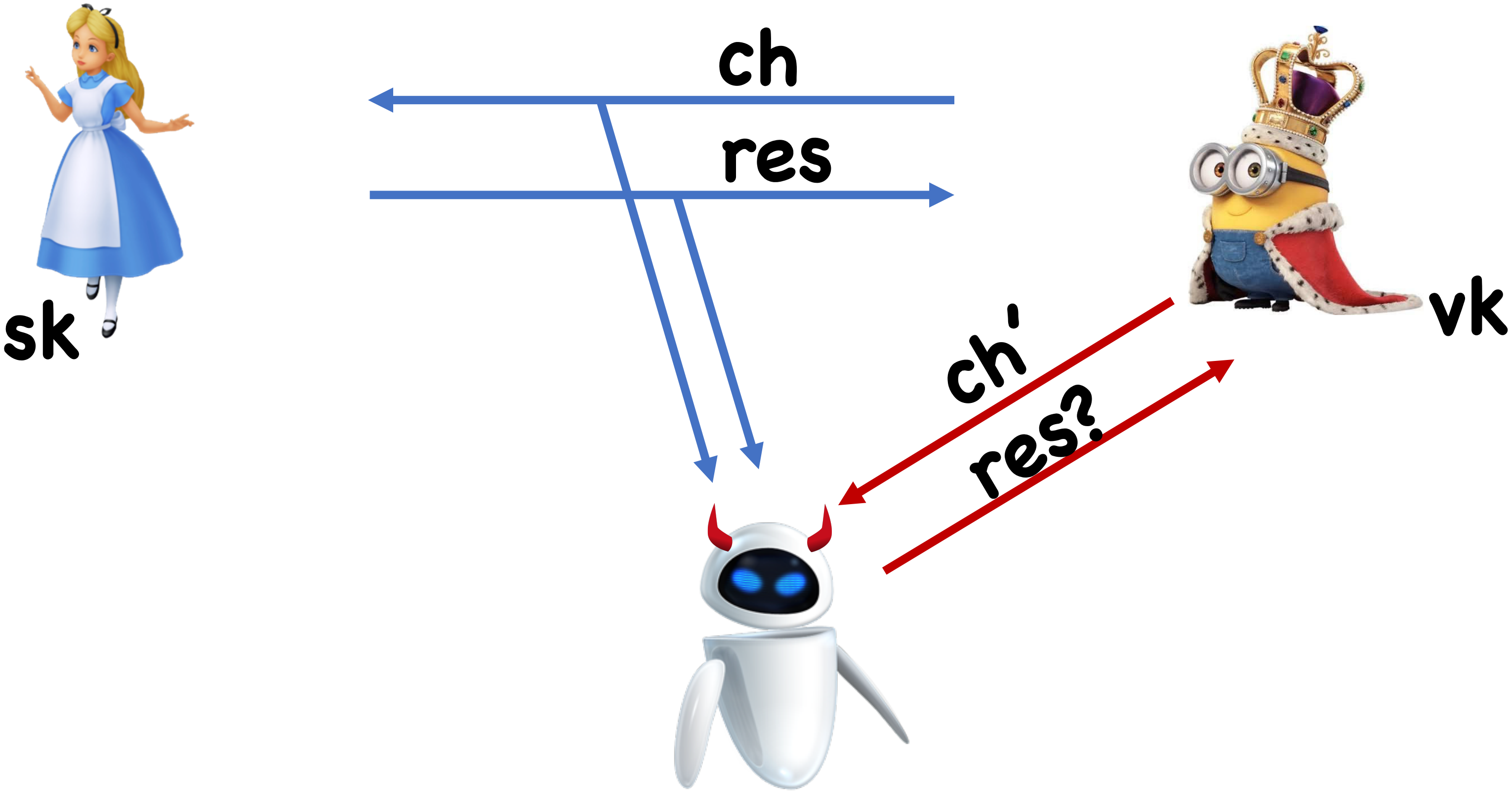
So far, all schemes secure against eavesdropping are stateful

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

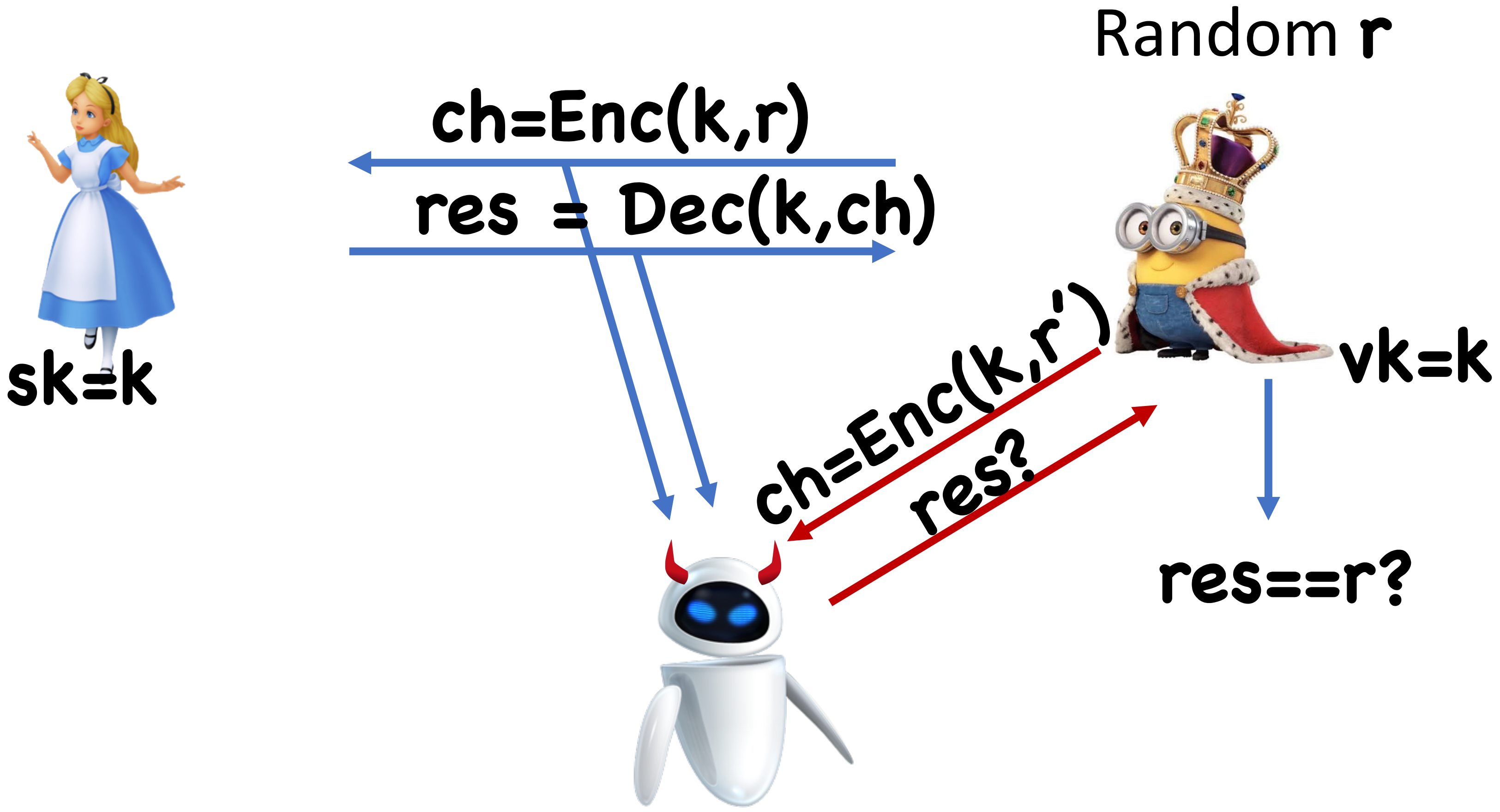
- Simply replay message

If want stateless scheme, instead want at least two messages

# Challenge-Response



# C-R Using Encryption



Random **r**

**ch=Enc(k,r)**

**res = Dec(k,ch)**

**sk=k**

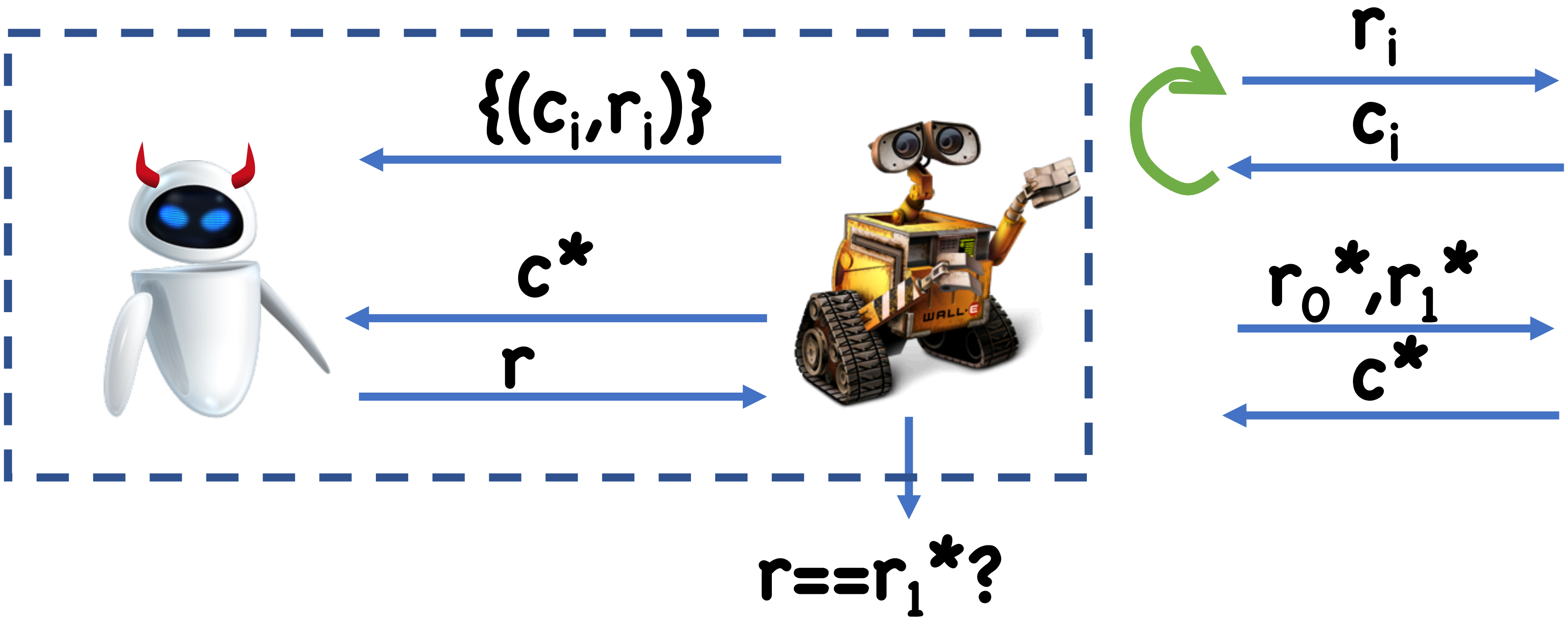
**vk=k**

**ch=Enc(k,r')**

**res?**

**res==r?**

**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



$\{(c_i,r_i)\}$

$c^*$

$r$

$r_i$

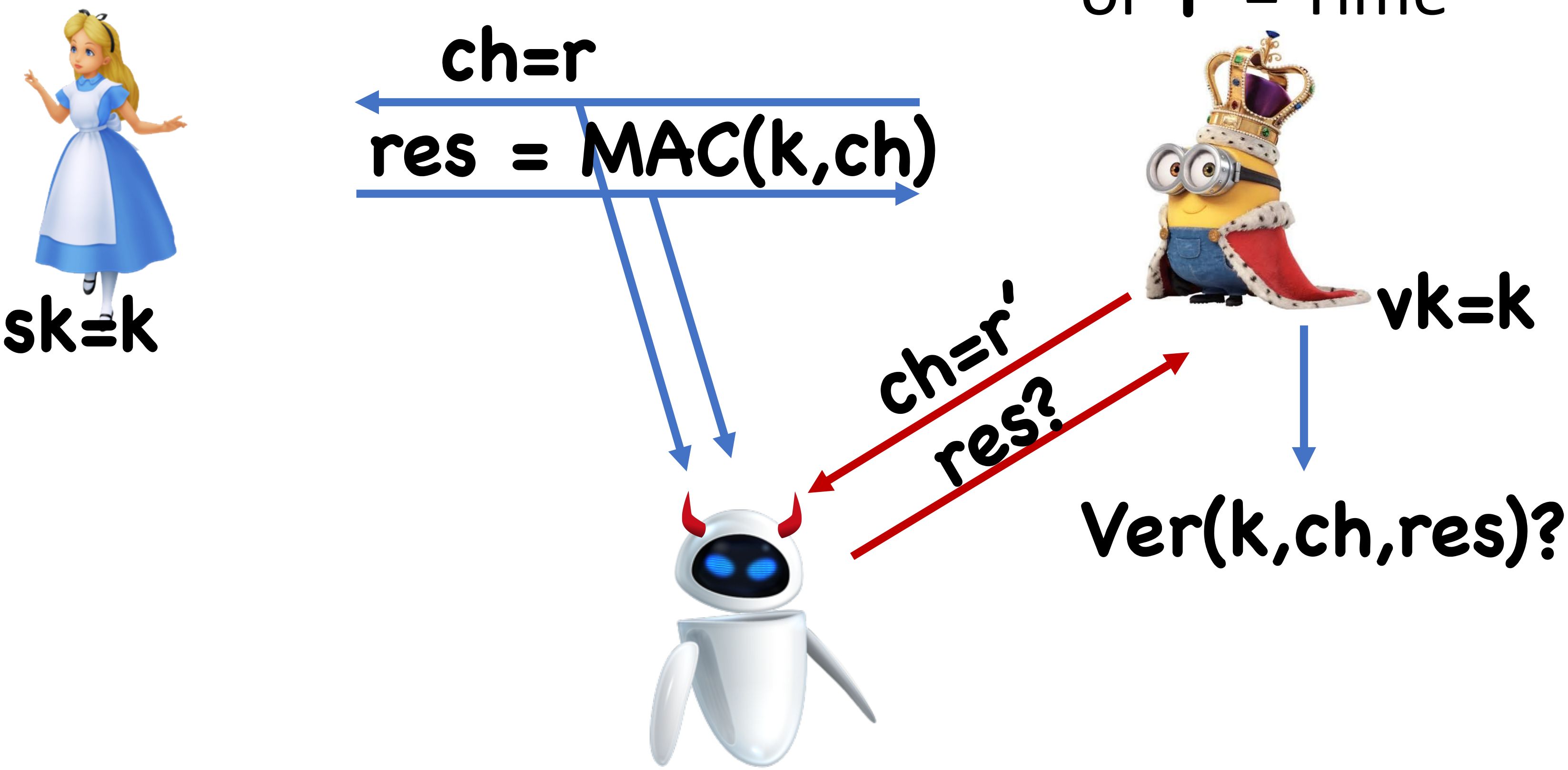
$c_i$

$r_0^*, r_1^*$

$c^*$

$r == r_1^*?$

# C-R Using MACs/Signatures



Random **r**

or **r** = Time

ch=r

res = MAC(k,ch)

sk=k

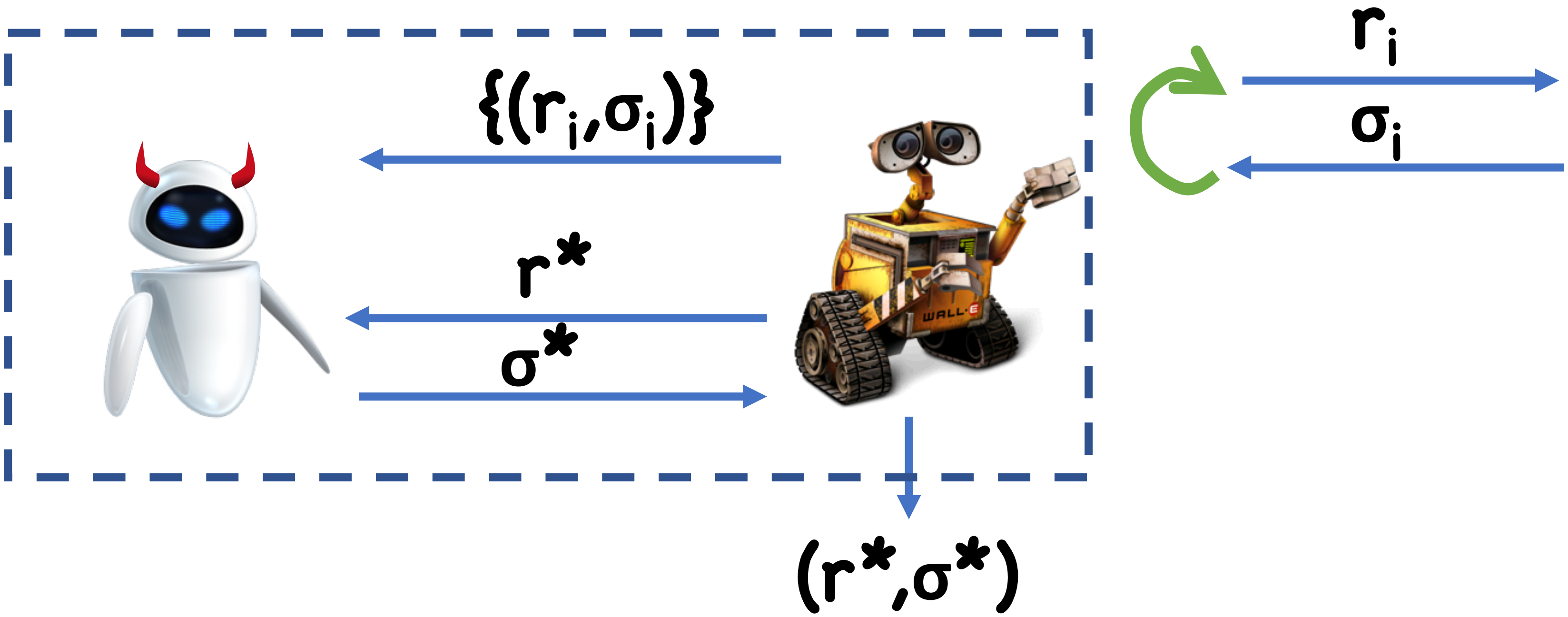
vk=k

ch=r'

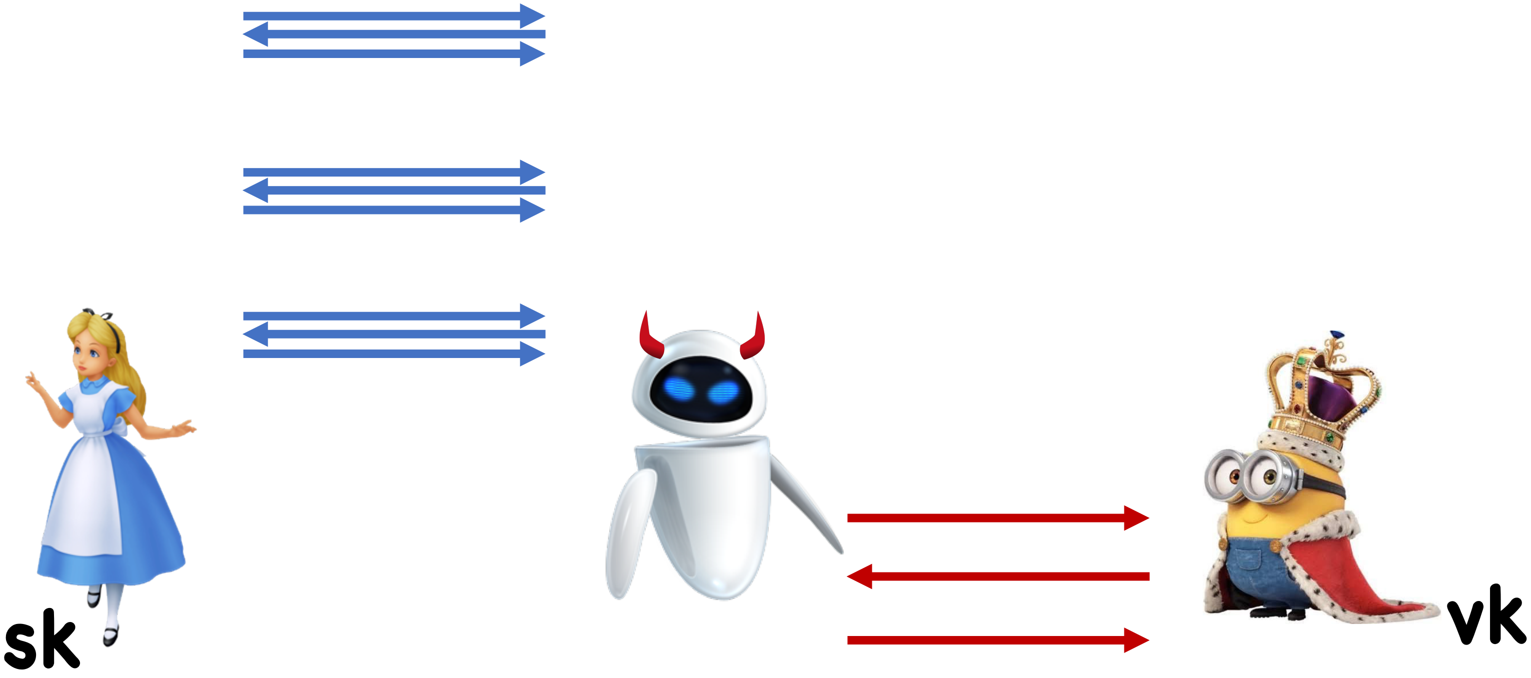
res?

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

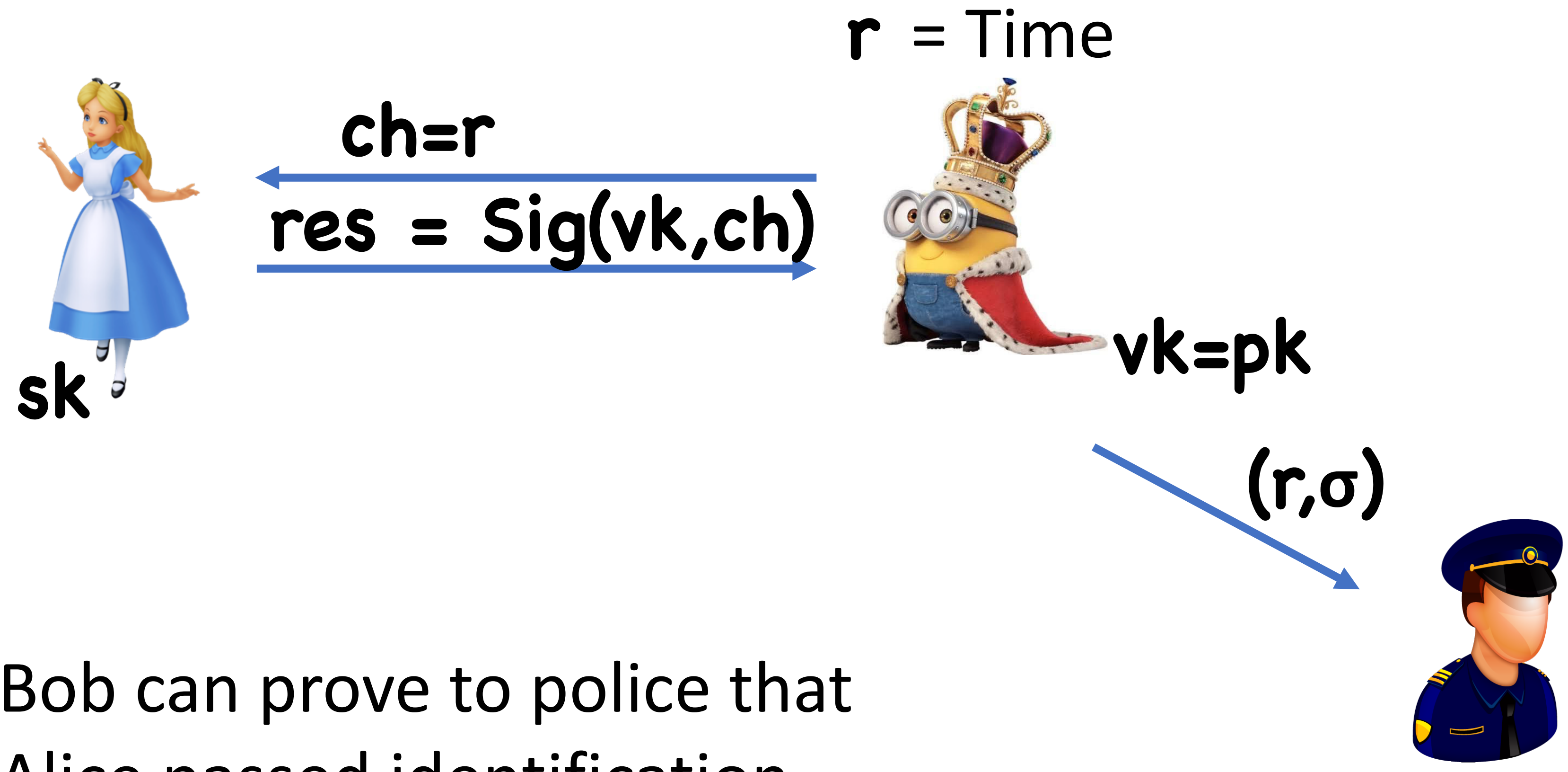
For enc-based C-R, CPA-secure is insufficient

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

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

# Non-Repudiation

Consider signature-based C-R



r = Time

ch=r

res = Sig(vk,ch)

sk

vk=pk

(r,σ)

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*

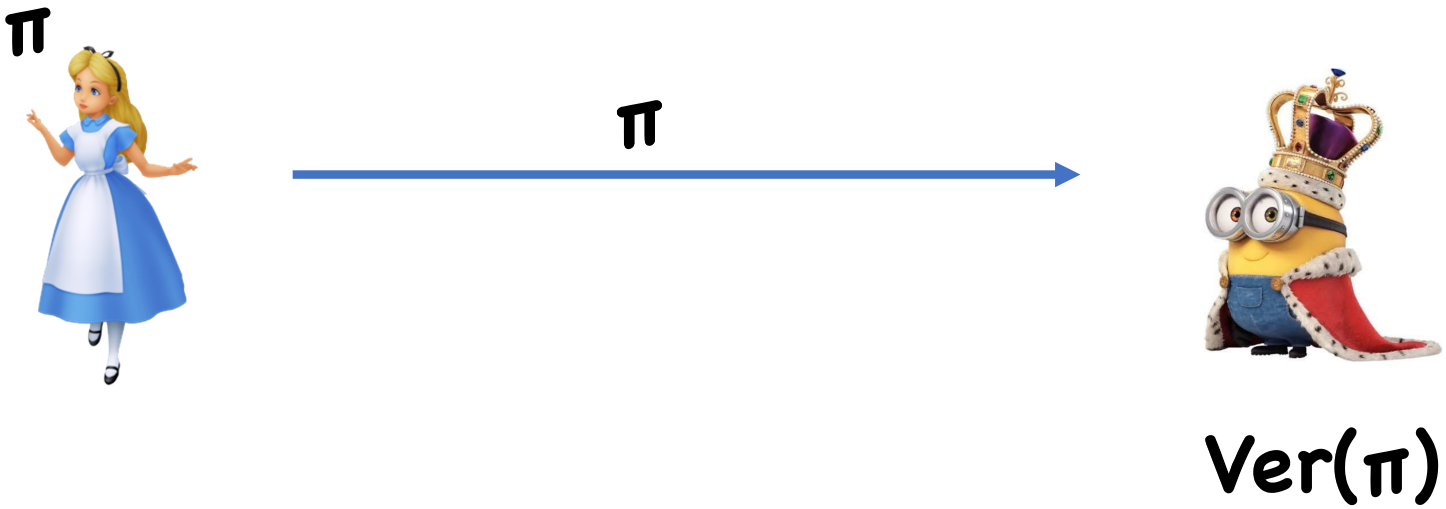
Ex: public key Enc-based C-R

- 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 of messages matters

# Zero Knowledge Proofs

# Mathematical Proof



π

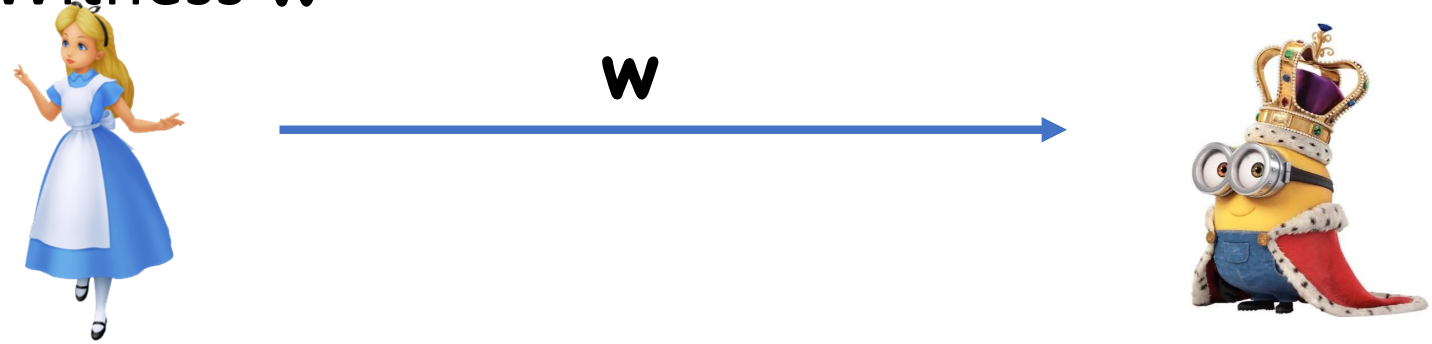
π

Ver(π)

# Mathematical Proof

Statement **x**

Witness **w**



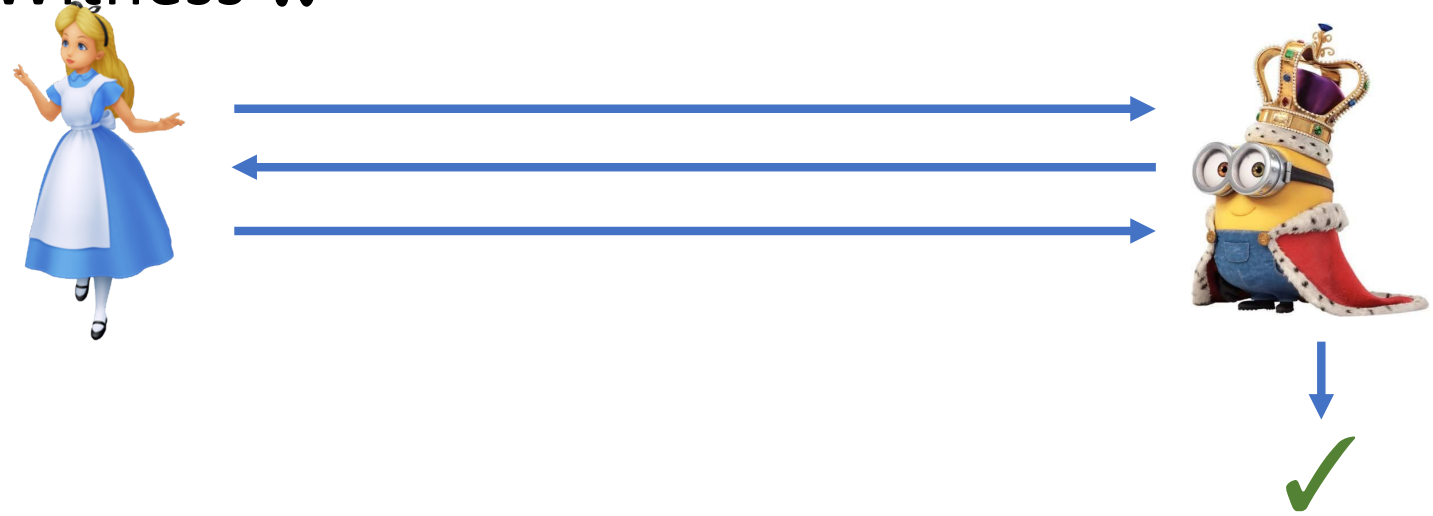
**w**

**Ver(x,w)**

# Interactive Proof

Statement **x**

Witness **w**



# Properties of Interactive Proofs

Let **(P,V)** be a pair of probabilistic interactive algorithms for the proof system

**Completeness:** If **w** is a valid witness for **x**, then **V** should always accept

**Soundness:** If **x** is false, then no cheating prover can cause **V** to accept

- Perfect: accept with probability **0**
- Statistical: accept with negligible probability
- Computational: cheating prover is comp. bounded

# Zero Knowledge

Intuition: prover doesn’t learn anything by engaging in the protocol (other than the truthfulness of $\mathbf{x}$)

How to characterize what adversary “knows”?

- Only outputs a bit
- May “know” witness, but hidden inside the programs state

# Zero Knowledge

First Attempt:

∃ "simulator" 🦁 s.t. for every true statement **x**, valid witness **w**,

🦁(x) $\approx_c$ **P(x,w)** → ← → **V(x)**

# Zero Knowledge

First Attempt:

Assumes Bob obeys protocol

- “Honest Verifier”

But what if Bob deviates from specified prover algorithm to try and learn more about the witness?

# Zero Knowledge

For every malicious verifier **V***, ∃ “simulator”🦁
s.t. for every true statement **x**, valid witness **w**,

$$🦁(x) \quad \approx_c \quad P(x,w) \begin{array}{c}\longrightarrow\\ \longleftarrow\\ \longrightarrow\end{array} V^*(x)$$

# QR Protocol

Statements: $\mathbf{x}$ is a Q.R. mod $\mathbf{N}$

Witness: $\mathbf{w}$ s.t. $\mathbf{w^2 \bmod N = x}$

Protocol:

$\mathbf{w}$

$u \leftarrow Z_N^*$

$y \leftarrow u^2 \bmod N$

$y \rightarrow$

$\leftarrow b$

$z = w^b u \bmod N \rightarrow$

$b \leftarrow \{0,1\}$

$\downarrow$

$z^2 == x^b y \bmod N?$

# QR Protocol

Zero Knowledge:

What does Bob see?

- A random QR $\mathbf{y}$,
- A random bit $\mathbf{b}$,
- A random root of $\mathbf{x^b y}$

Idea: simulator chooses $\mathbf{b}$, then $\mathbf{y}$,

- Can choose $\mathbf{y}$ s.t. it always knows a square root of $\mathbf{x^b y}$

# QR Protocol

Honest Verifier Zero Knowledge:

🦁 **(x):**

- Choose a random bit **b**
- Choose a random string **z**
- Let $y = x^{-b}z^2$
- Output **(y,b,z)**

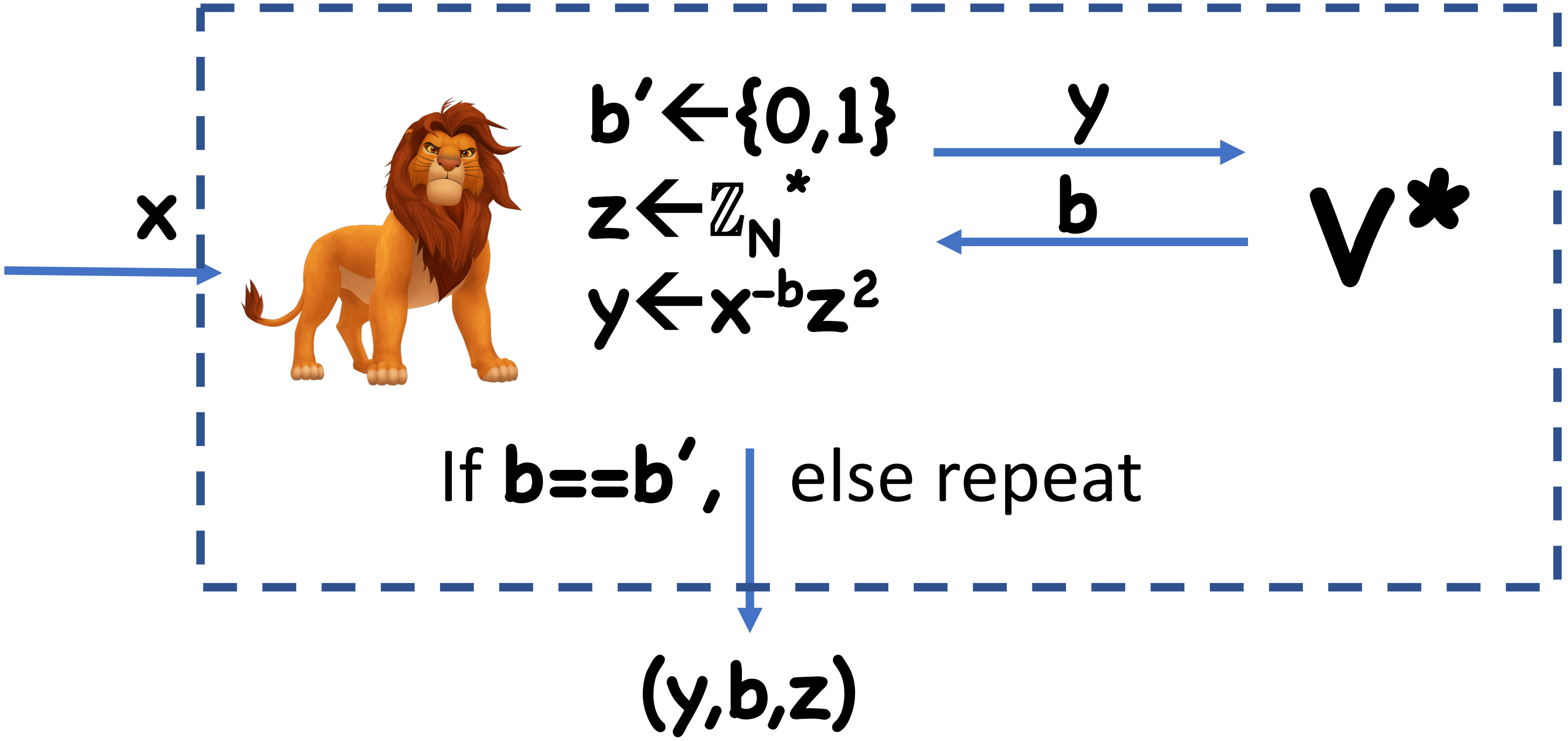
- If **x** is a QR, then **y** is a random QR, no matter what **b** is
- **z** is a square root of $x^b y$

↓

**(y,b,z)** is distributed identically to **(P,V)(x)**

# QR Protocol

(Malicious Verifier) Zero Knowledge:



$x$

$b' \leftarrow \{0,1\}$

$z \leftarrow \mathbb{Z}_N^*$

$y \leftarrow x^{-b} z^2$

$y$

$b$

$V^*$

If **$b==b'$**, else repeat

$(y,b,z)$

# QR Protocol

(Malicious Verifier) Zero Knowledge:

Proof:

- If **x** is a QR, then **y** is a random QR, independent of **b'**
- Conditioned on **b'=b**, then **(y,b,z)** is identical to random transcript seen by **V***
- **b'=b** with probability **1/2**

# Zero Knowledge Proofs

Known:

- Proofs for any NP statement assuming just one-way functions
- Non-interactive ZK proofs for any NP statement using trapdoor permutations

# Applications

Identification protocols
Signatures

Protocol Design:

- E.g. CCA secure PKE
  - To avoid mauling attacks, provide ZK proof that ciphertext is well formed
  - Problem: ZK proof might be malleable
  - With a bit more work, can be made CCA secure
- Example: multiparty computation
  - Prove that everyone behaved correctly

# Reminders

HW6 Due Today

HW7 Due May 1