CHALMERS
UNIVERSITY OF TECHNOLOGY

Two can keep a secret
If one of them uses Haskell

Papers we love, Göteborg

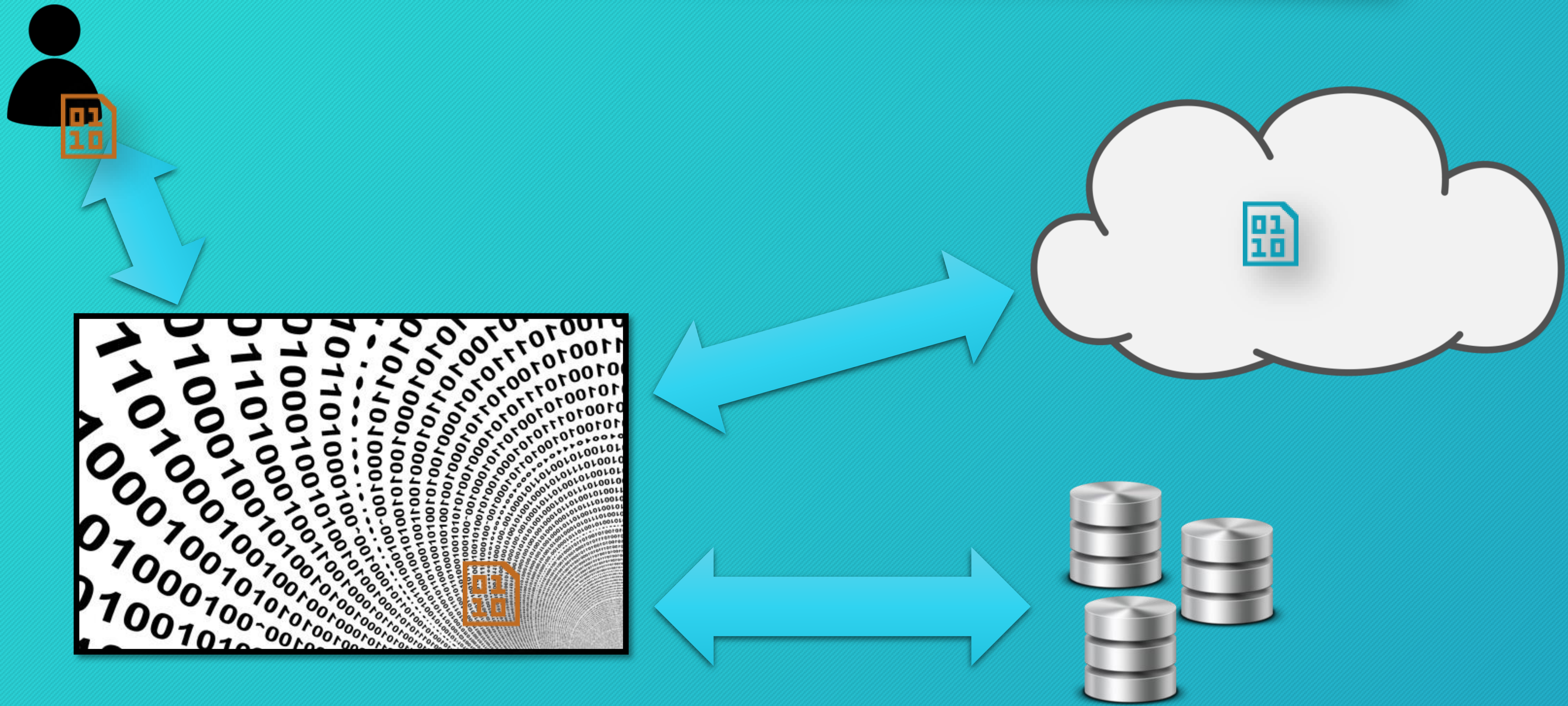
[Alejandro Russo]

Computer Science and Engineering Department

Chalmers University of Technology, Göteborg, Sweden

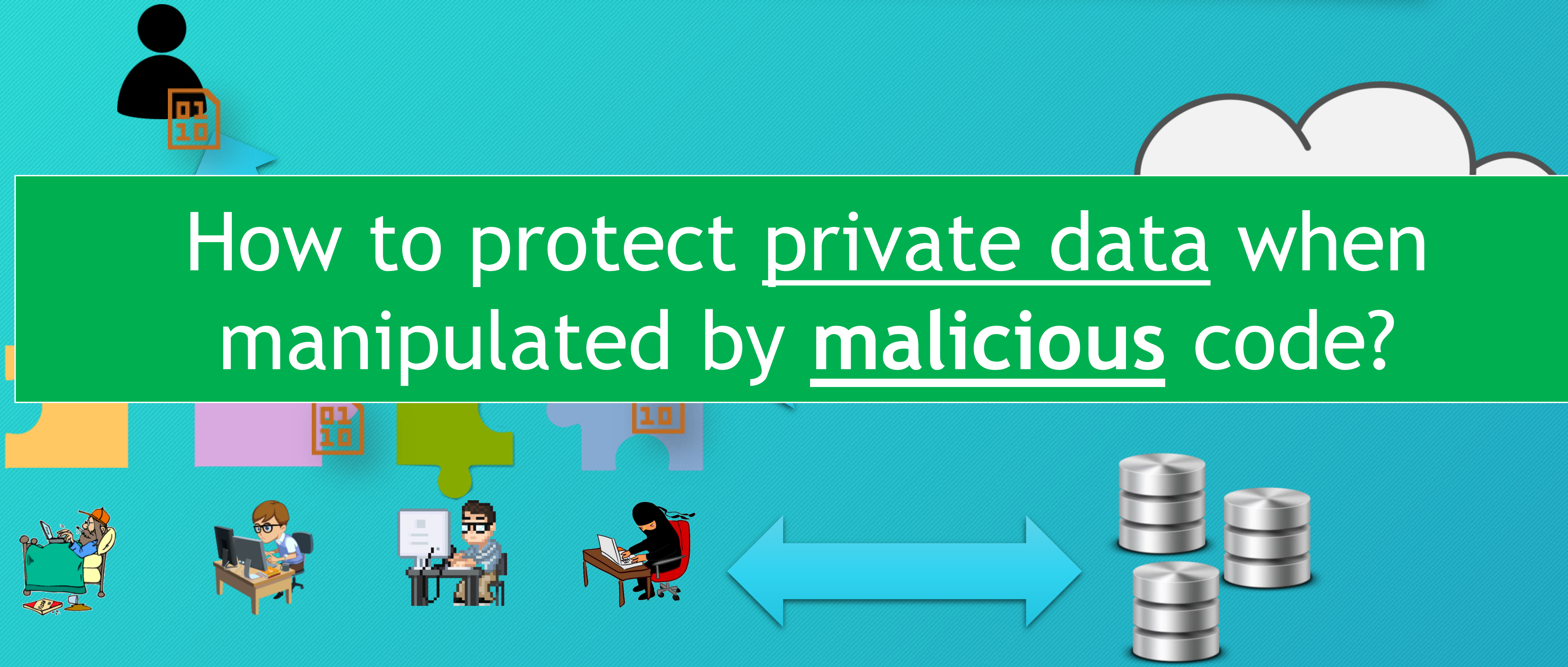
Motivation

Modern software and privacy



Modern software and privacy

Modern software and privacy



How to protect private data when manipulated by **malicious** code?

Modern software and privacy

Data breaches could cripple the growth of mobile wallets

Another Set of Malicious Android Apps Try to Fool Pokemon GO Players

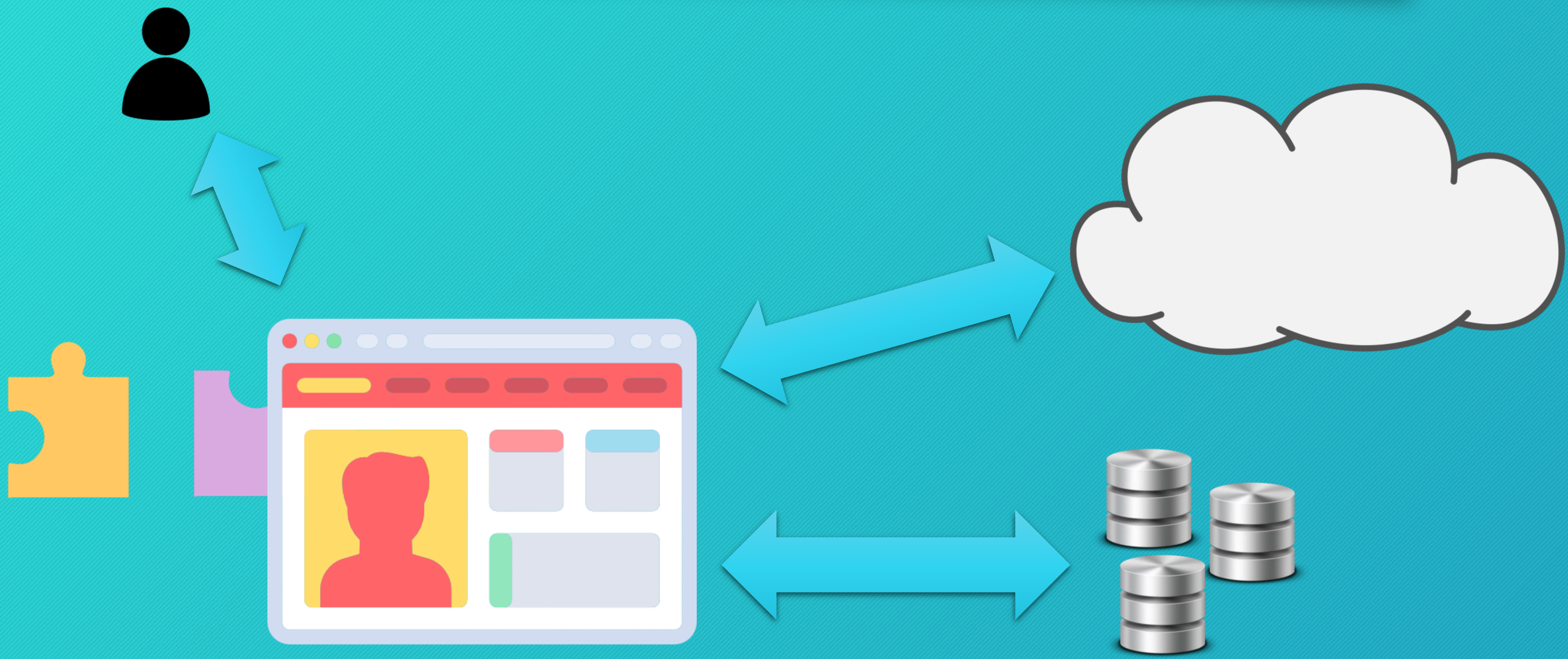
Dell discovers seven new fake Pokemon GO apps

2016 News!

YOUR ANDROID MIGHT BE CONTROLLED BY A MALICIOUS TWITTER ACCOUNT

900 Million Android Devices 'Vulnerable To Attack'

Modern software and privacy



Wordpress, Joomla domains under attack through jQuery JavaScript library

Abuse of the JavaScript library has led to over 4.5 million recent exposures to infection.

2016 News!

Exploit can attack secure websites through ads

HEIST only needs a little JavaScript to do a lot of damage.

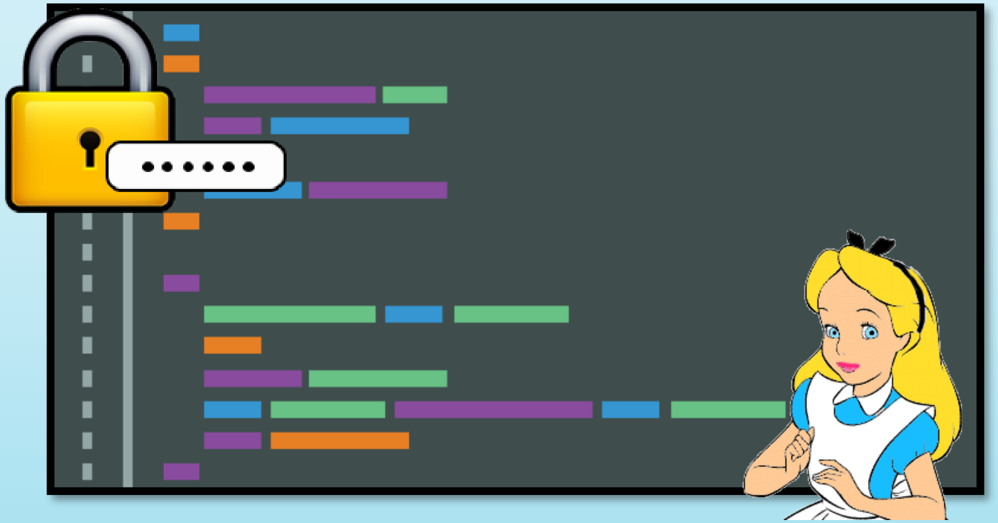
Ransomware created using only JavaScript discovered

Modern software and privacy

How to protect <u>private data</u> when manipulated by <u>buggy-non-malicious</u> code?

Running example

A password manager



```
common :: String -> Bool
```

Feching updates for the dictionaries?

```
common :: String -> IO Bool
```

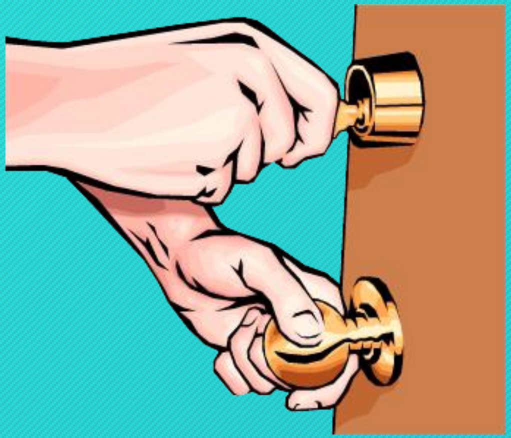
The power to do anything!

```
common :: String -> SafeIO Bool
```

Passwords can be still **sent** to the dictionaries' servers

Restrict IO-actions based on the data being observed

Reflection on enforcing privacy



It is not about *who has access to the information* (DAC)

It is about *how the information is handled* (IFC/MAC)

Jif

JSFlow

COWL

Paragon

Laminar

BFlow

FlowFox

Flume

Asbestos

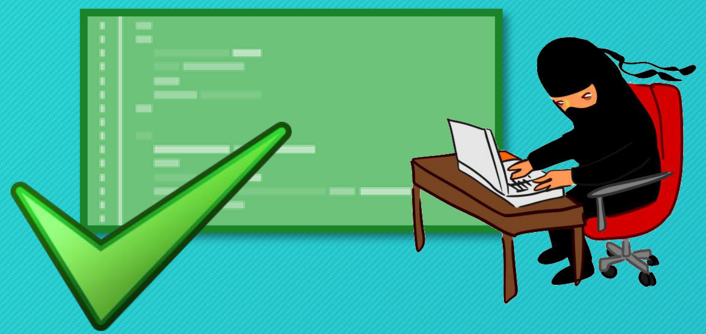
HiStar

⋮

Privacy via Haskell libraries

MAC - a security library

[Russo - ICFP 2015]



References

Exceptions

MVars

Concurrency

: : Types to the rescue!

~200 SLOC

Haskell 98

+ MultiParamTypeClasses

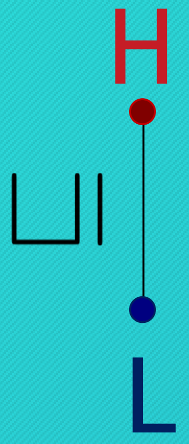
+ Safe Haskell

Security labels

Information flows in this direction

$l \sqsubseteq l'$

- How secret is a piece of data?
- Labels (organized in a lattice)
- Order: `can-flow-to` relationship




```
data L
data H
```

```
class l ⊑ l' where
instance L ⊑ L where
instance H ⊑ H where
instance L ⊑ H where
```

Labeling data

```
newtype Labeled l a
```

```
credit_card :: Labeled H Int    weather :: Labeled L String
```

Secure computations

It encapsulates privacy preserving IO-actions

```
newtype MAC l a
instance Monad (MAC l)
```

It handles data with sensitivity l

run^{MAC} `:: MAC l a -> IO a`

```
public :: MAC L Int
secret :: MAC H String
```

Secure computations

[Bell and LaPadula 73]

Mandatory Access Control

l_{read}

Read

MAC l a

Write

l_{write}

$l_{read} \sqsubseteq l$

$l \sqsubseteq l_{write}$

No read up

No write down

Secure computations

[Bell and LaPadula 73]

Mandatory Access Control

L

Read

MAC H a

L $\sqsubseteq$ H

No read up

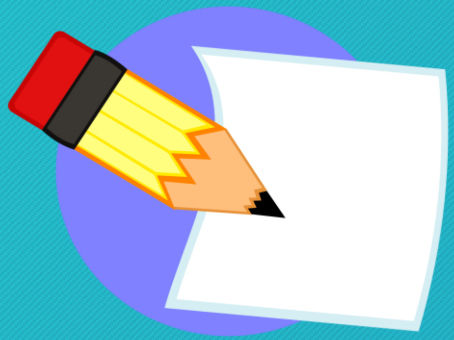
Secure computations

[Bell and LaPadula 73]

Mandatory Access Control

MAC $\color{red}{H}$ a Write

$\color{red}{H}$



$\color{red}{H} \sqsubseteq \color{red}{H}$

No write down

Labeled data and computations

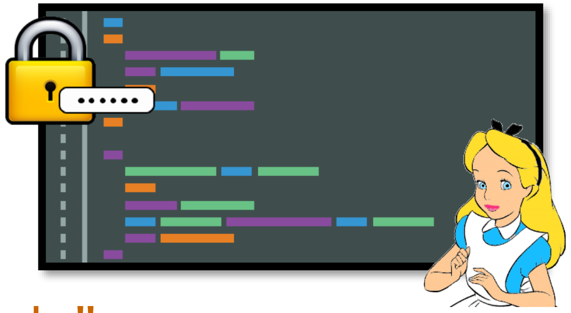
```
type Labeled l a

label^MAC :: l ⊑ h => a -> MAC l (Labeled h a)

unlabel^MAC :: l ⊑ h => Labeled l a -> MAC h a
```

References

MVars

```
Labeled l (IORef a)    Labeled l (IO.MVar a)
```

Password strength checker



```
module Alice where
import qualified Bob

password :: IO String
password = do
  putStr "Please, select your password:"
  pwd <- getLine
  mac_H <- runMAC $ ( do lpwd <- labelMAC pwd :: MAC L (Labeled H String) )
                        Bob.common lpwd )
  bool  <- runMAC $ mac_H
  …
```

Labeled as secret

To fetch dictionary updates

to read the password

```
module Bob where
…
common :: Labeled H String -> MAC L (MAC H Bool)
```

Nested MAC-actions

```
module Bob where
…
common :: Labeled H String -> MAC L (MAC H Bool)
```

It might contain arbitrary long nested MAC l-actions

It converts MAC h actions into MAC l ones

$\texttt{cast}^{\texttt{MAC}} \; \texttt{::} \; \texttt{l} \sqsubseteq \texttt{h} \; \texttt{=>} \; \texttt{MAC h a -> MAC l (Labeled h a)}$

It preserves the sensitivity of the result

$\textbf{do} \; \texttt{r <-} \; \texttt{m}_1$

$\quad \texttt{cast}^{\texttt{MAC}} \; \boxed{\texttt{(f r :: MAC H ())}}$

$\quad \texttt{m}_2$

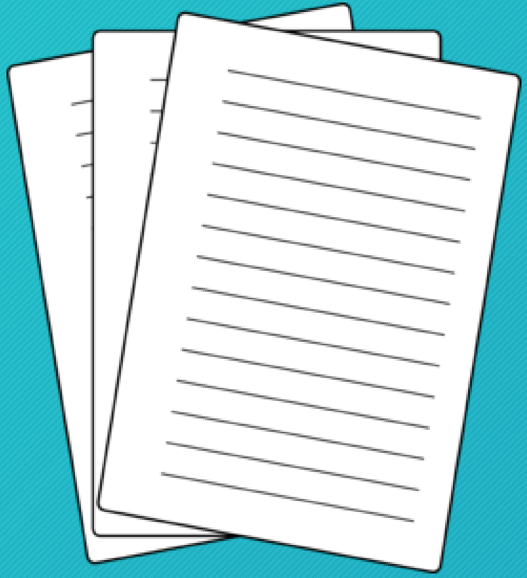
$\texttt{:: MAC L String}$

Core API

```
newtype MAC l a
newtype Labeled l a
instance Monad (MAC l)

labelMAC   :: l ⊑ h => a -> MAC l (Labeled h a)

unlabelMAC :: l ⊑ h => Labeled l a -> MAC h a

castMAC    :: l ⊑ h => MAC h a -> MAC l (Labeled h a)
```

Information-flow libraries

Library	Enforcement
MAC	**Static**
HLIO	**Hybrid**
LIO	**Dynamic**



[HASKELL 2011]
[ICFP 2012]
[OSDI 2012]
[ESORICS 2013]
[CSF 2014]
[ICFP 2015]
[ICFP 2015]

Haskell

Why going dynamic?

Sensitivity of data might depend on the data itself

```
fetchURL :: String -> MAC ? HTML
```

Security policies might change dynamically

```
my_pictures :: MAC L [Picture]
```

HLIO

[Buiras et al. - ICFP 2015]

Dynamic information-flow control

```
newtype LIO lat a
newtype Labeled lat a
instance Monad (LIO lat)

class Lattice lat where
    ⊑ :: lat -> lat -> Bool

label^LIO     :: Lattice lat => lat -> a -> LIO lat (Labeled lat a)

unlabel^LIO   :: Lattice lat => Labeled lat a -> LIO lat a

toLabeled^LIO :: Lattice lat =>
                 LIO lat a -> LIO lat (Labeled lat a)
```

```
newtype MAC l a
newtype Labeled l a
instance Monad (MAC l)

label^MAC     :: l ⊑ h => a -> MAC l (Labeled h a)

unlabel^MAC   :: l ⊑ h => Labeled l a -> MAC h a

cast^MAC      :: l ⊑ h => MAC h a -> MAC l (Labeled h a)
```

LIO

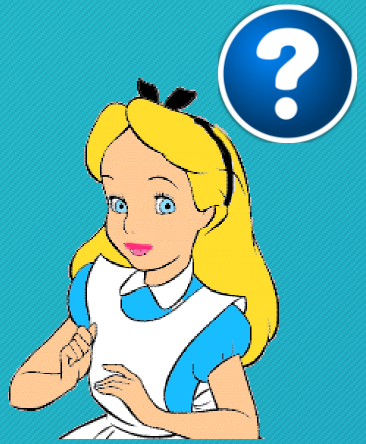
[Stefan et al. – HASKELL 2011]

Exceptions (Control Flow)

Handling errors

Untrusted code

- The password strenght checker crashes when the network is down
  - Exception thrown



$\texttt{throw}^{\text{MAC}}$ `:: Exception e => e -> MAC l a`

$\texttt{catch}^{\text{MAC}}$ `:: Exception e =>`
`MAC l a -> (e -> MAC l a) -> MAC l a`

(Ninja) Bob in action!

Untrusted code

`secret == True`

- Received by Bob:1

`secret == False`

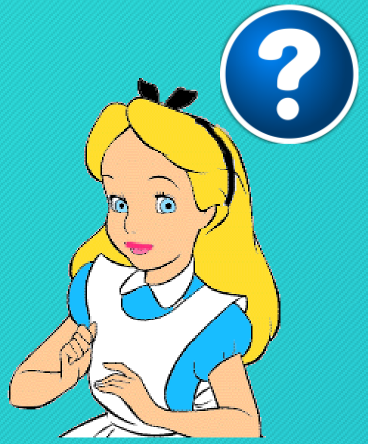
- Received by Bob:1,0

```
catch^MAC(
 do send_1 :: MAC L ()
    cast^MAC $ do
       …
       when (secret) (error “crash!”)
       return ()
    send_0 :: MAC L ()
                           )
 (\e :: SomeException -> return ())
```

The problem

Types do not capture that!

```
do send_1 :: MAC L ()
   cast^MAC $ do
     …
     when (secret) (error “crash!”)
     return ()
   send_0 :: MAC L ()
```



Exceptions raised in a sensitive environment can suppress subsequent less sensitive effects

A simple solution

[Stefan et al. - JFP 2015]

```
do send_1 :: MAC L ()
   cast^MAC $ do
     …
     when (secret) (error "crash!")
     return ()
   send_0 :: MAC L ()
```

secret == **True**	secret == **False**

- Received by Bob: **1,0**

```
cast^MAC m = …
  (catch^MAC (m >>= label)
  (\(e :: SomeException) -> label (throw e)))
```

Core API

```
newtype MAC l a
newtype Labeled l a
instance Monad (MAC l)

label^MAC   :: l ⊑ h => a -> MAC l (Labeled h a)
unlabel^MAC :: l ⊑ h => Labeled l a -> MAC h a
cast^MAC    :: l ⊑ h => MAC h a -> MAC l (Labeled h a)

throw^MAC  :: Exception e => e -> MAC l a
catch^MAC  :: Exception e =>
              MAC l a -> (e -> MAC l a) -> MAC l a
```

Covert Channels

(Ninja) Bob is not giving up!

[Askarov et al. 08]

Types do not capture that!

```
do send_1 :: MAC L ()
   cast^MAC $ do
     …
     when (secret) (error “crash!”)
     return ()
   send_0 :: MAC L ()
```

`secret == True`

- Received by Bob:**1**

`secret == False`

- Received by Bob:1,**0**

```
loop :: a
loop = loop
```

Termination leaks

[Askarov et al. 08]

$$O(2^{|secret|})$$

Sequential programs

Notorious covert channels

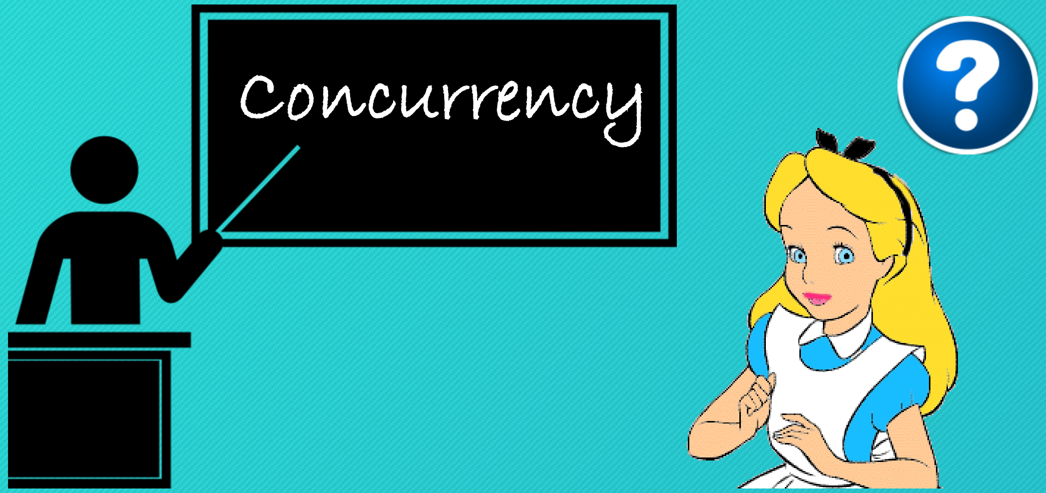
Timing external events

[Stefan et al. – ICFP 2012]

Power consumption

Concurrency

Concurrency



It spawns a computation of the same kind

$\texttt{fork}^{\mathrm{MAC}}$ `:: MAC l () -> MAC l ()`

Exploiting termination and concurrency

a 0
b 1
…

```
do send_a_1_Bob :: MAC L ()
   castMAC $ do
     …
     when (bit_a) (loop)
     return ()
   send_a_0_Bob :: MAC L ()
```

```
do send_b_1_Bob :: MAC L ()
   castMAC $ do
     …
     when (bit_b) (loop)
     return ()
   send_b_0_Bob :: MAC L ()
```

`bit_a == True`

- Sent to Bob: (a,1)

`bit_a == False`

- Sent to Bob: (a,1),(a,0)

`bit_b == True`

- Sent to Bob: (b,1)

`bit_b == False`

- Sent to Bob: (b,1),(b,0)

Termination leaks and concurrency

$$O(|secret|)$$

Concurrent programs

Concurrency and termination

[Stefan et al. – ICFP 2012]

- Dangerous mix

$\texttt{cast}^{MAC}$ `:: Less l h =>`

`MAC h a -> MAC l (Labeled h a)`

$\texttt{fork}^{MAC}$ `:: MAC l () -> MAC l ()`

`do send_0_Bob :: MAC L ()`

$\texttt{cast}^{MAC}$ `$ do`

`…`

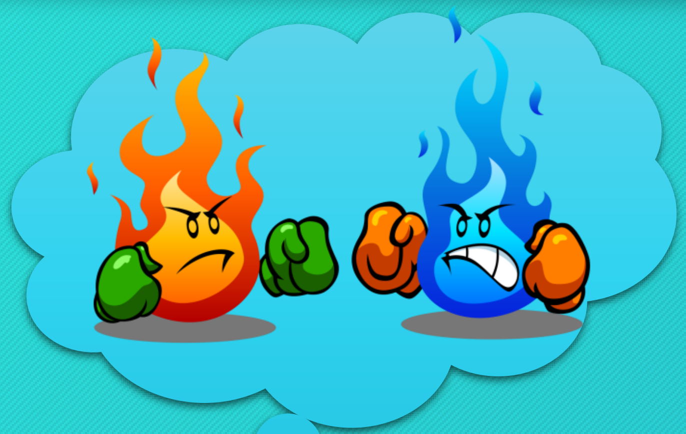
`when (secret) (loop)`

`return ()`

`send_1_Bob :: MAC L ()`

Concurrency and termination

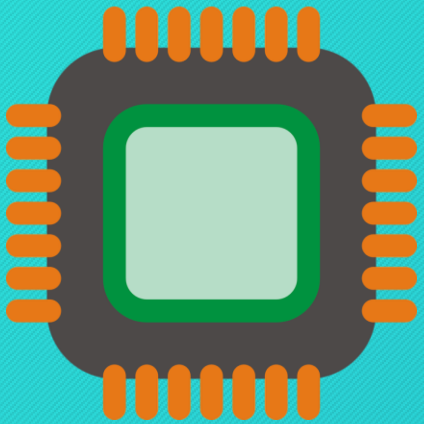
[Stefan et al. – ICFP 2012]



```
fork^MAC :: l ⊑ h =>
              MAC h () -> MAC l ()
```

```
do send_0_Bob :: MAC L ()
   fork^MAC $ do
       …
       when (secret) (loop)
       return ()
   send_1_Bob :: MAC L ()
```

Other covert channels for security libraries



Caches

[Stefan et al. – ESORICS 2013]

[Buiras et al. – TGC 2013]

λ

Lazy evaluation

[Buiras and Russo – NORDSEC 2013]

Summary

- Privacy is a pressing demand
  - Untrusted (third-party) code
  - Buggy-non-malicious code
- Access Control is not enough!
  - Track how sensitive data propagates
- Haskell plays a unique privileged role
  - Security via libraries

Summary

- Covert channels
  - Bandwidth
  - Attacker power
- Adding features
  - Control-flow leaks (Exceptions)
  - Bandwidth magnification (concurrency)
  - New covert channels

Summary

References

Exceptions

MVars

Concurrency

`cabal install mac`

~200 SLOC

No read up

No write down

Haskell 98

\+ MultiParamTypeClasses

\+ Safe Haskell