Information Flow Control for Distributed Trusted Execution Environments

Anitha Gollamudi
Stephen Chong
HARVARD UNIVERSITY

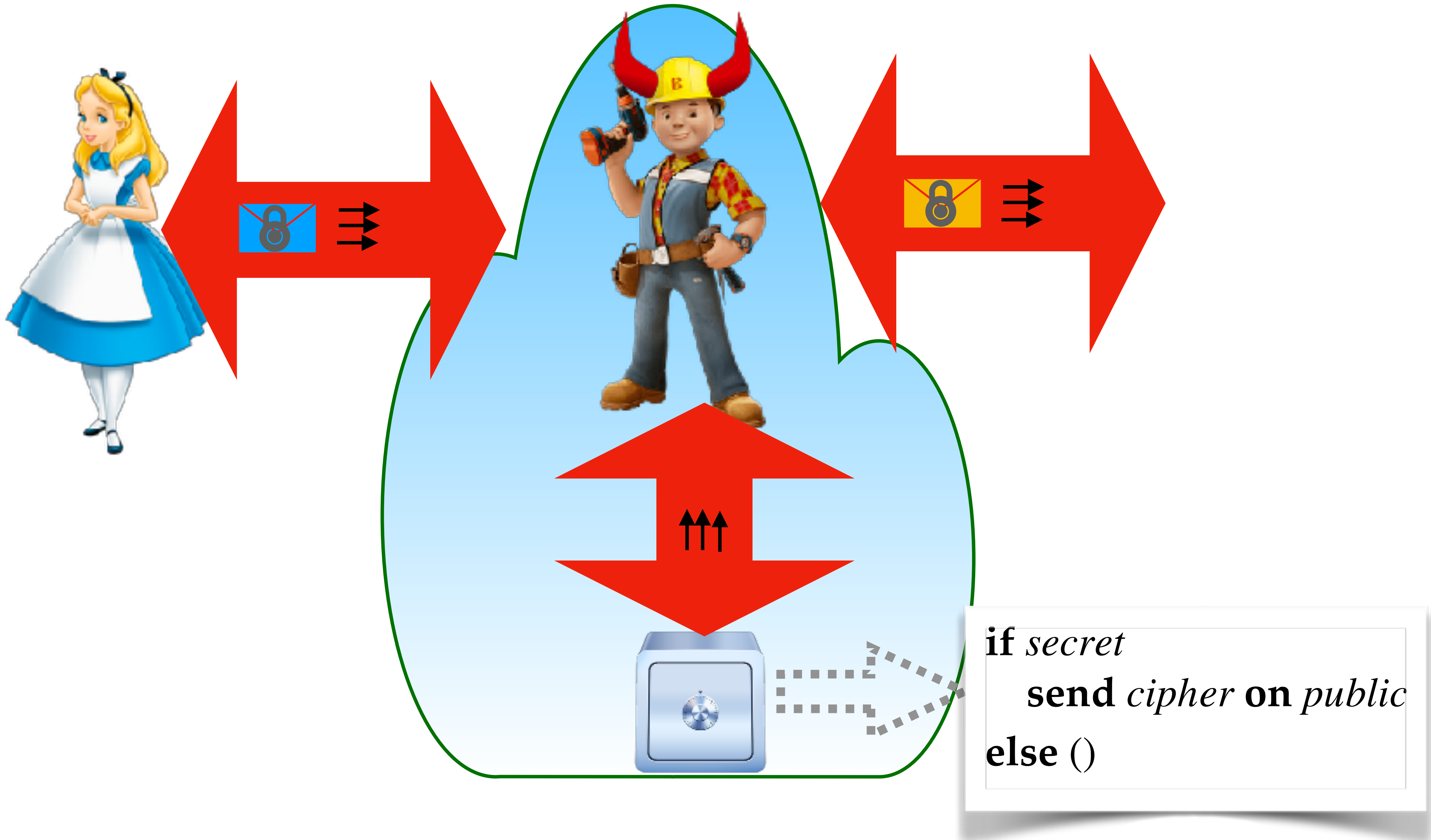
Owen Arden
UNIVERSITY OF CALIFORNIA SANTA CRUZ

Trusted Execution Environment (TEE)

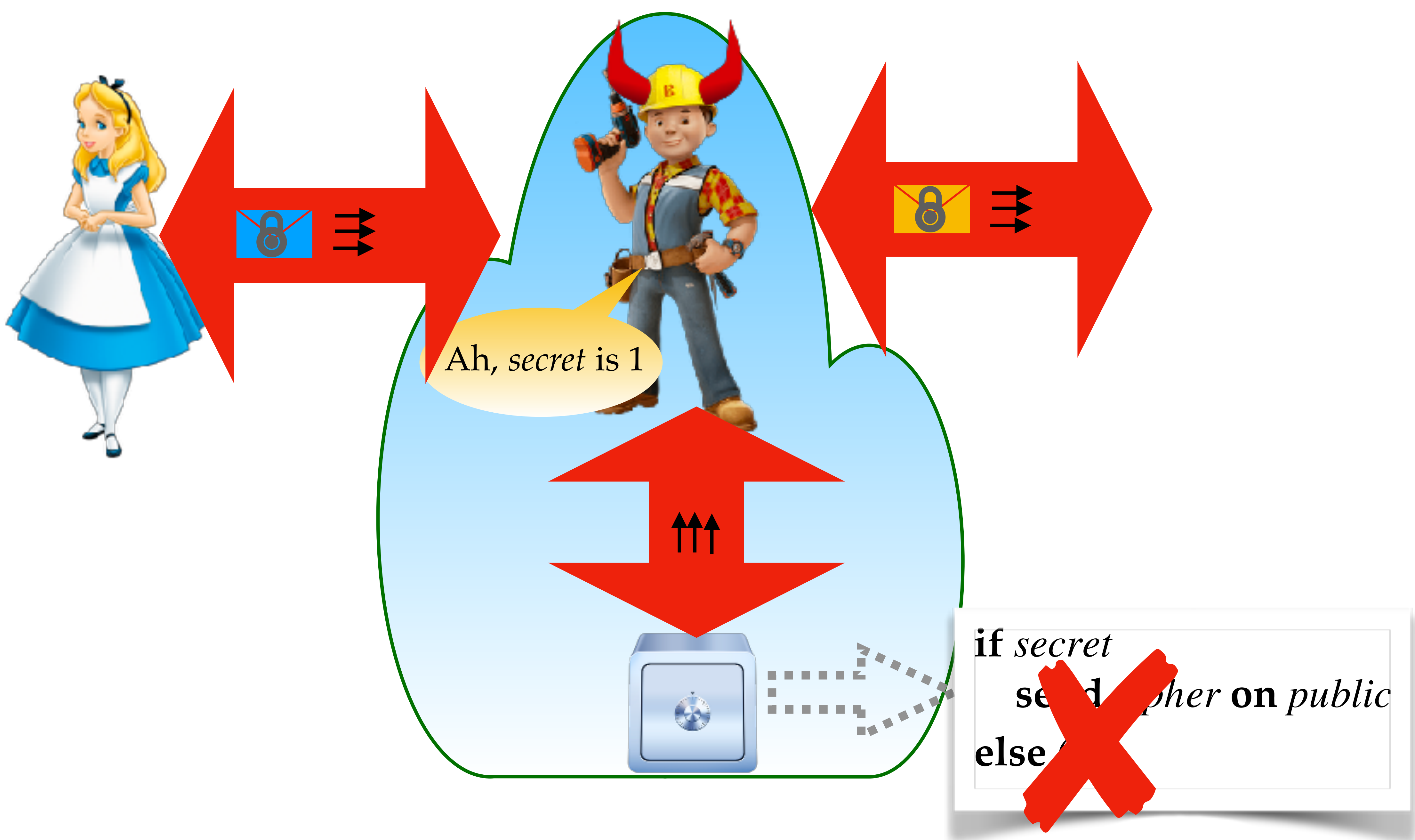
- Protected memory region for code and data
- Offers isolated execution and remote attestation
- Only host can communicate with TEE
- Hardware feature
  - e.g. Intel SGX, ARM TrustZone
- Good fit for offering security in distributed settings

TEEs $\nRightarrow$ Security Guarantees

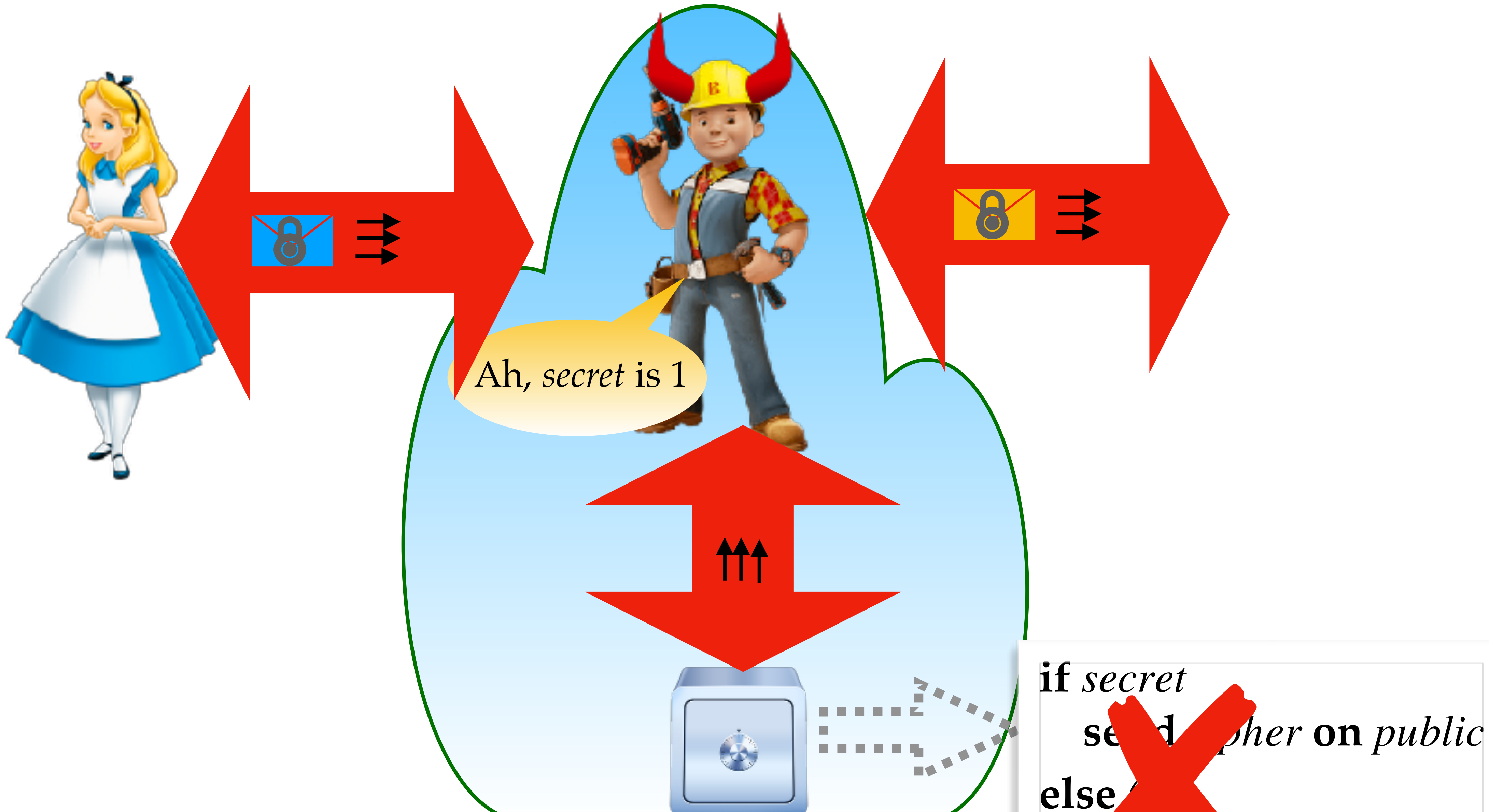
TEEs ⇏ Security Guarantees



TEEs $\not\Rightarrow$ Security Guarantees



TEEs ⇏ Security Guarantees



Information Flow Control (IFC) techniques!

IFC for Distributed TEEs: Challenges

IFC for Distributed TEEs: Challenges

1. Choose ***right*** abstractions for crypto and TEEs

- Focus on application-level security
- ***Reflect*** the capabilities and limitations of TEEs
  - e.g. TEE can communicate only with the host
- Implementable!

IFC for Distributed TEEs: Challenges

1. Choose ***right*** abstractions for crypto and TEEs
    - Focus on application-level security
    - ***Reflect*** the capabilities and limitations of TEEs
        - e.g. TEE can communicate only with the host
    - Implementable!
2. Enforce security

Contributions

Contributions

1. Distributed Flow-limited Authorization calculus for TEEs (DFLATE)

- Supports distributed TEEs
- Design mapping to real system

Contributions

1. Distributed Flow-limited Authorization calculus for TEEs (DFLATE)
   - Supports distributed TEEs
   - Design mapping to real system
2. A permissive security type system
   - Enforces security (noninterference) for confidentiality and integrity

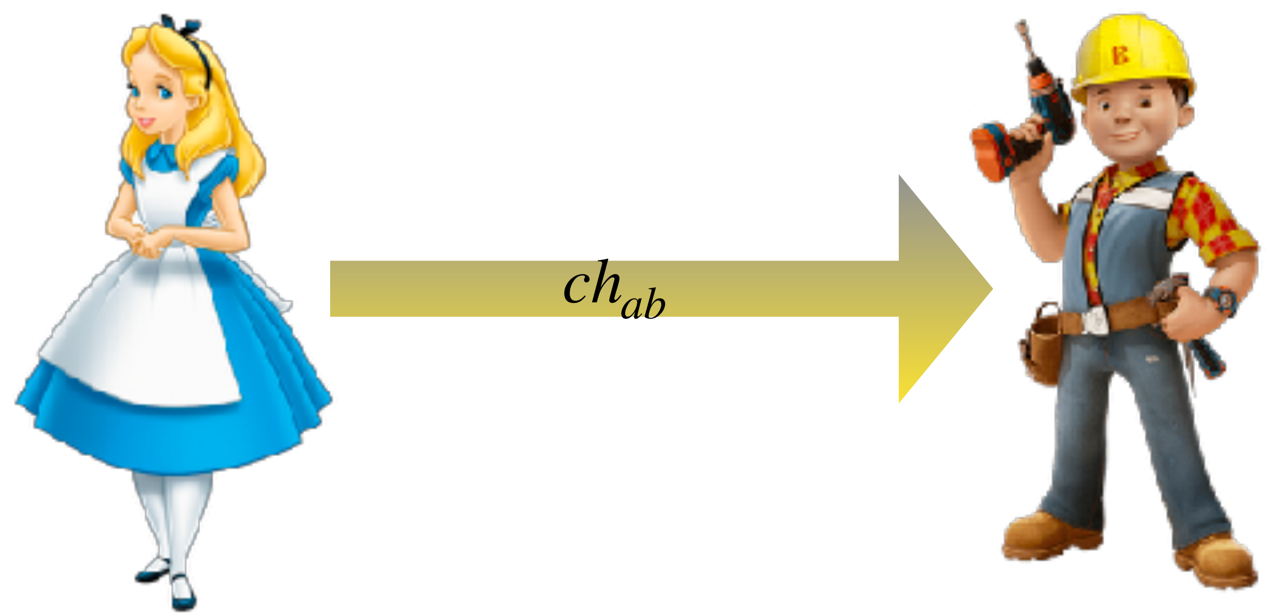
DFLATE

- Simply typed lambda calculus extended with
  - Communication primitives (send/receive/spawn)
  - Abstractions for crypto and TEE
  - Security types

Address Challenge #1

1. Choose ***right*** abstractions for crypto and TEEs
    - Abstractions ***should reflect*** the capabilities and limitations of TEEs
        - e.g. TEE can only communicate with host
    - Focus on application-level security
    - Implementable!
2. Enforce security

Communication

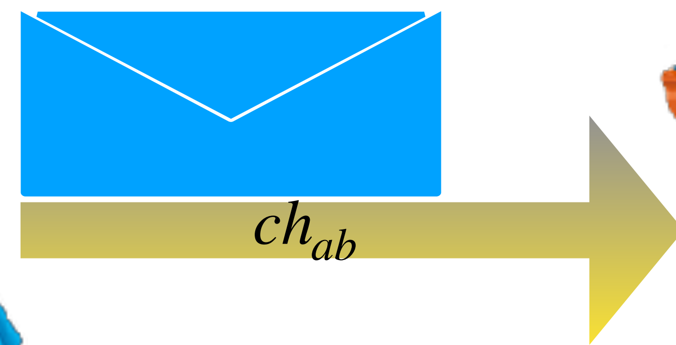


Alice (a) Bob (b)

$\textbf{send}$ $\textit{blue}$ $\textbf{on}$ ch_{ab}

$\textbf{recv}$ ch_{ab} $\textbf{as}$ x $\textbf{in}$
$\ldots$

Communication



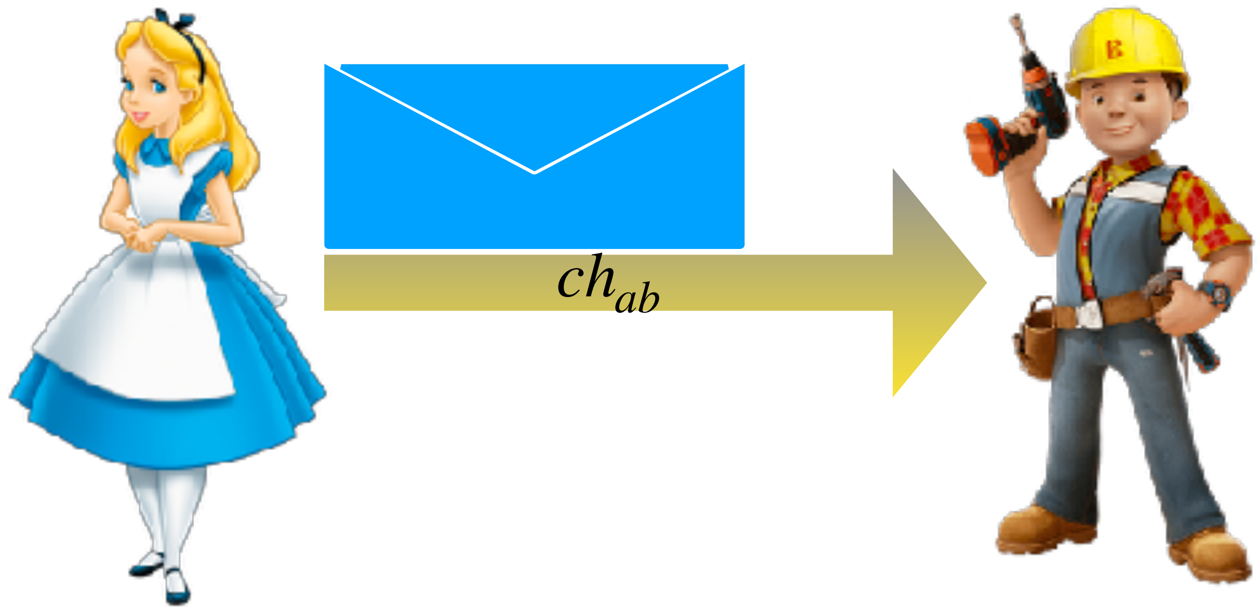
Alice (a)

Bob (b)

send *blue* **on** ch_{ab}

recv ch_{ab} **as** x **in**
$\dots$

Communication

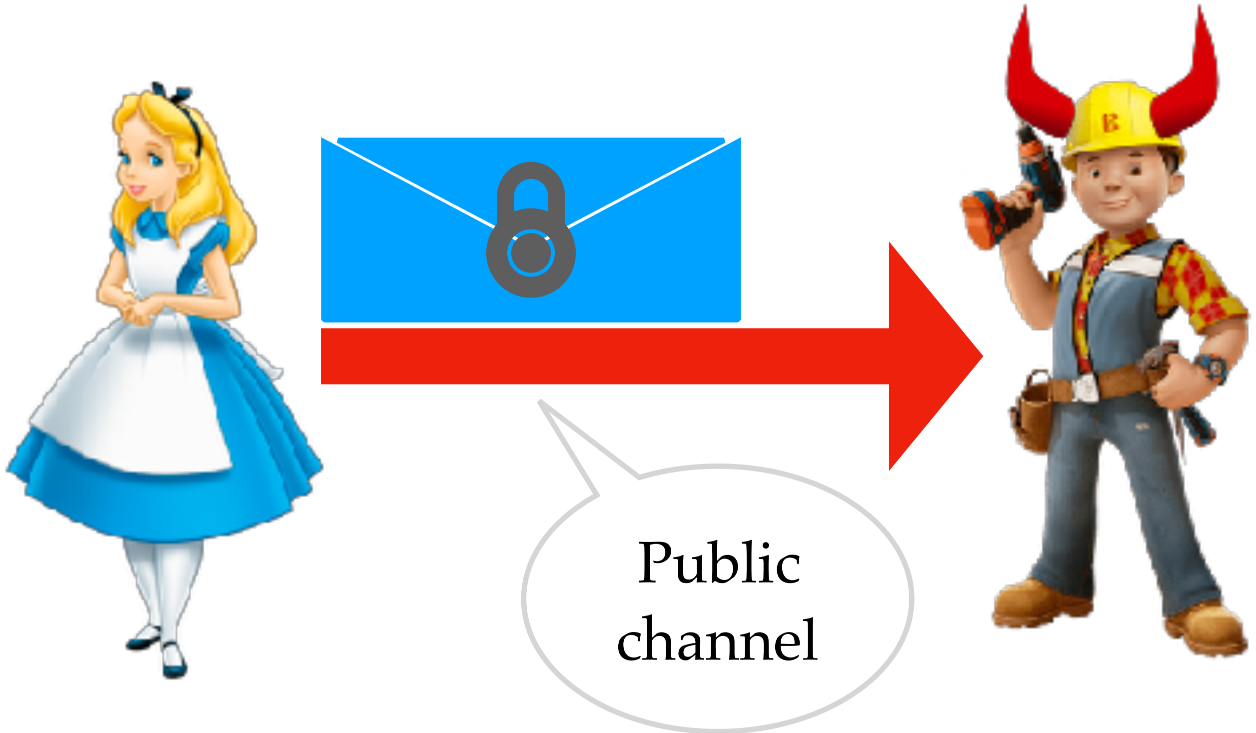


Alice (a) Bob (b)

send $blue$ **on** ch_{ab}

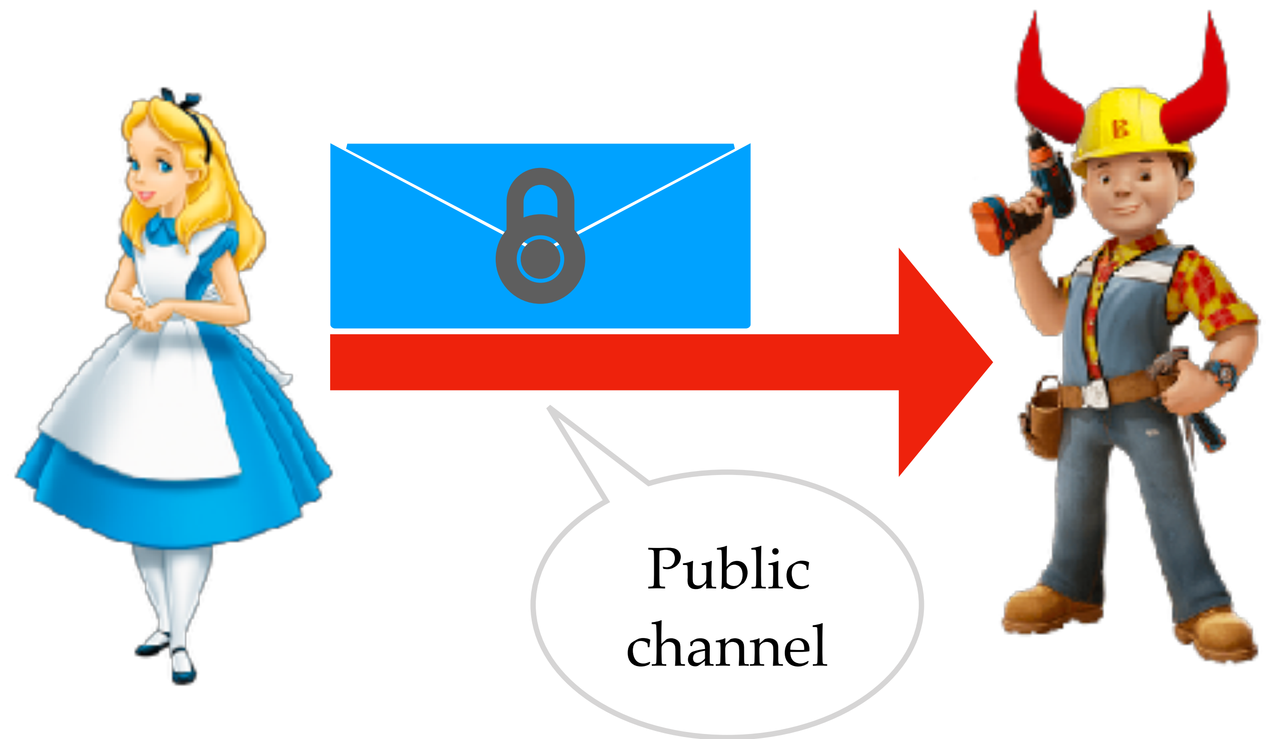
recv ch_{ab} **as** x **in**
$\ldots$

Securing Communication



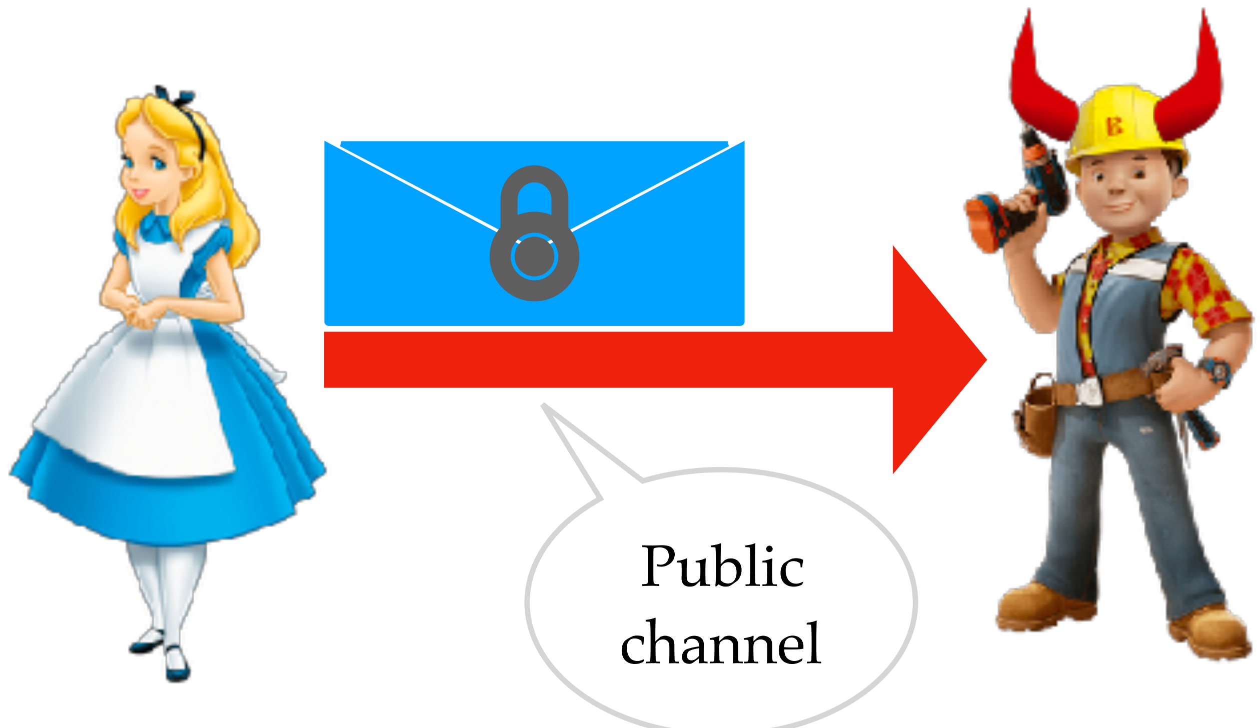
if *secret*
  send *blue* **on** *public*
else ()

Securing Communication



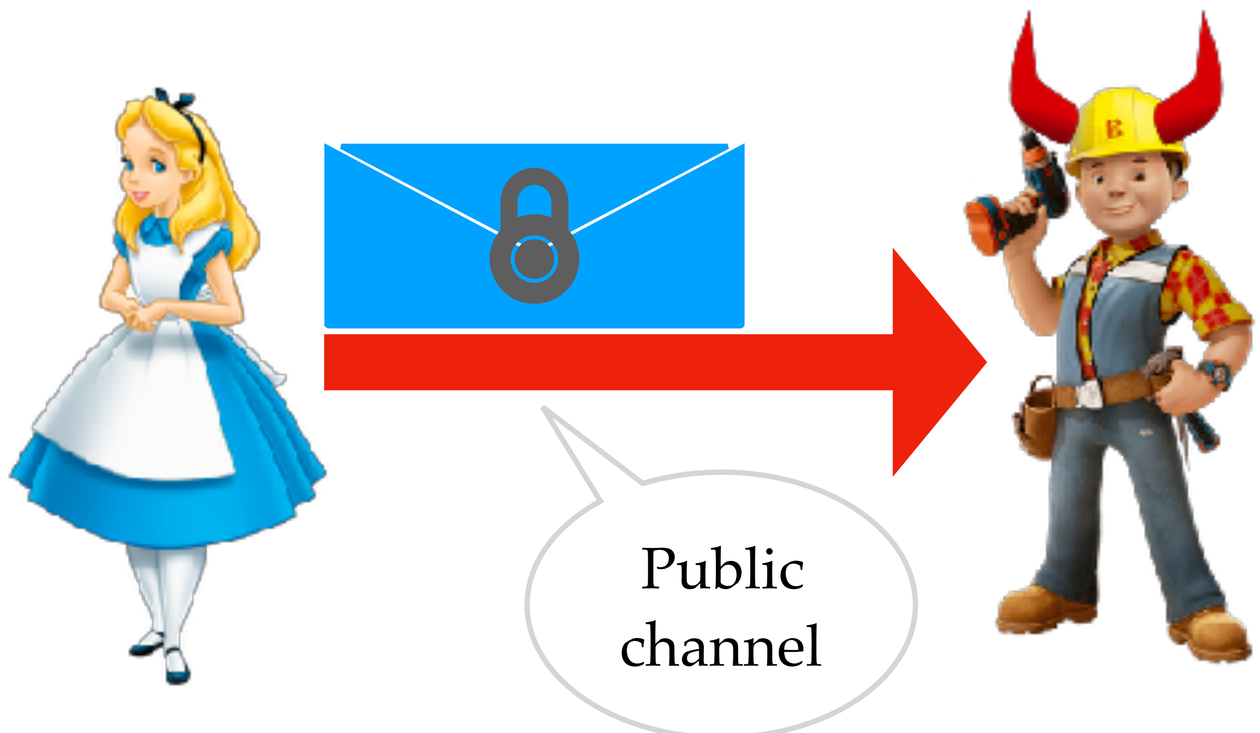
if *secret*
  send *blue* **on** *public*
else ()

Securing Communication



if *secret*
  send *blue* **on** *public*
else ()

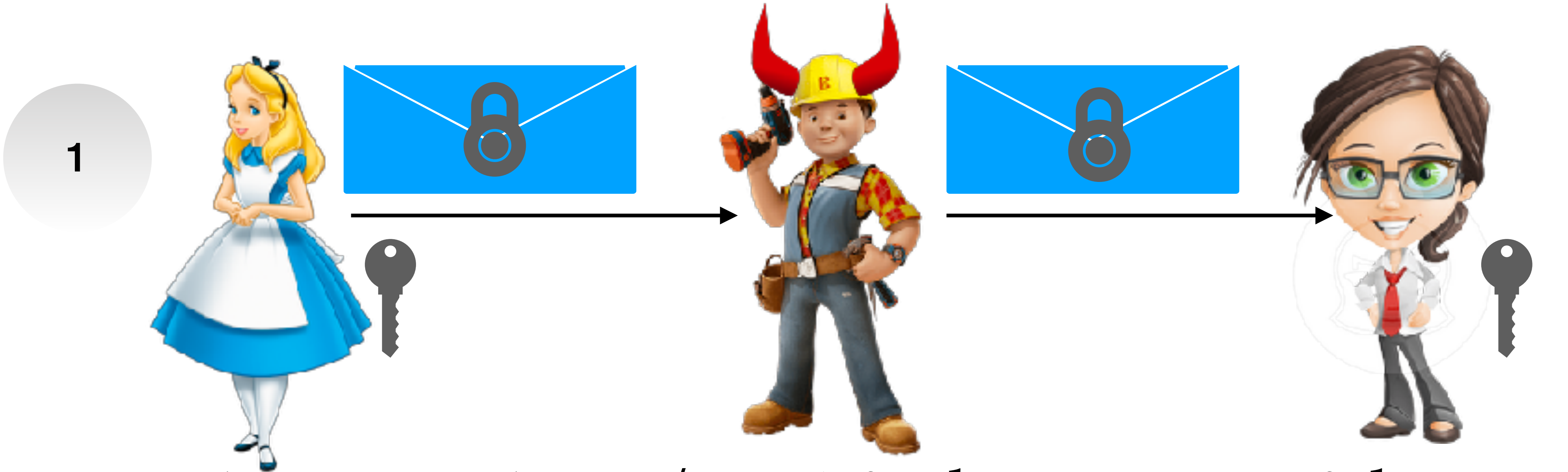
Securing Communication



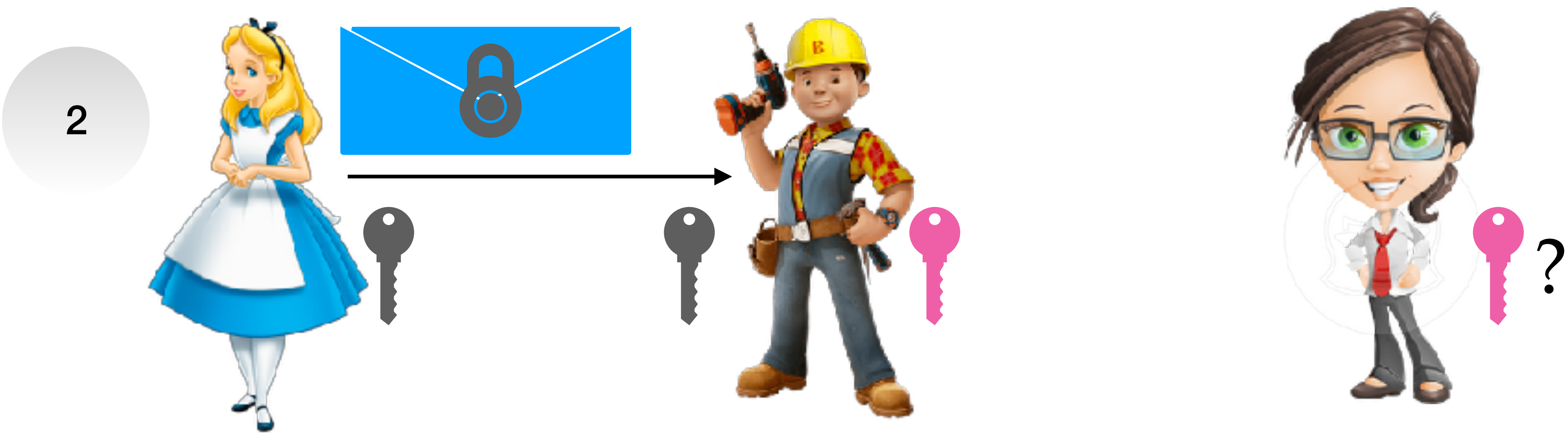

```
if secret
  send blue on public
else ()
```

Security labels on channels prevent leaks due to communication

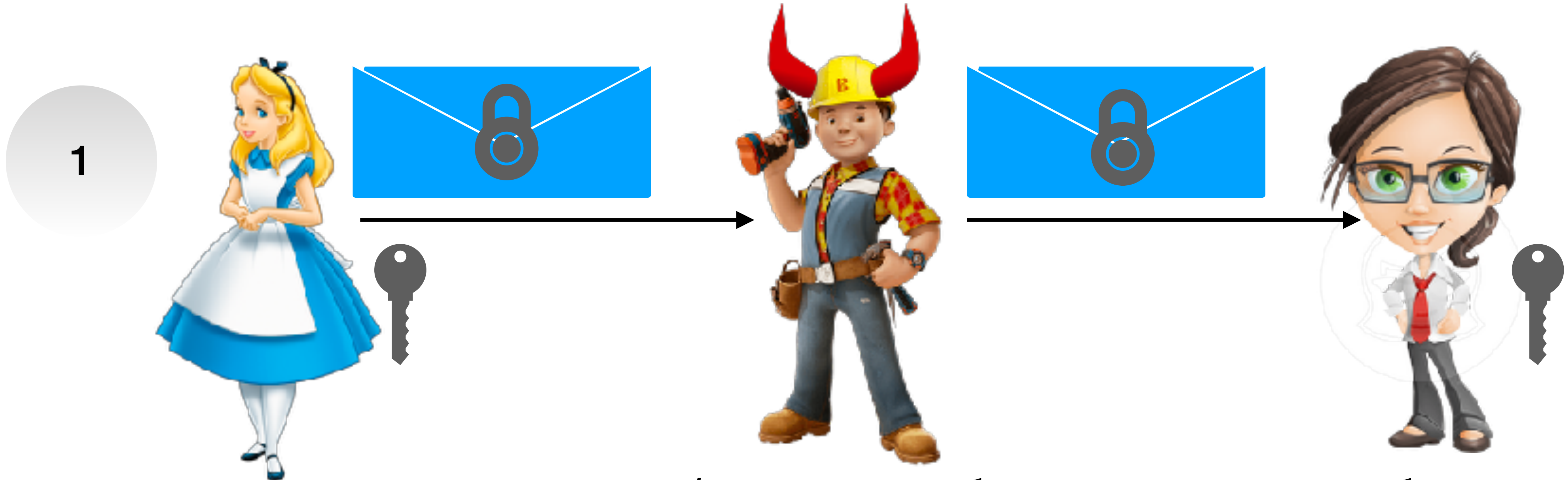
Communication through trusted / untrusted nodes



- Bob can not learn / modify the content of the message
- Bob may learn the existence of the message



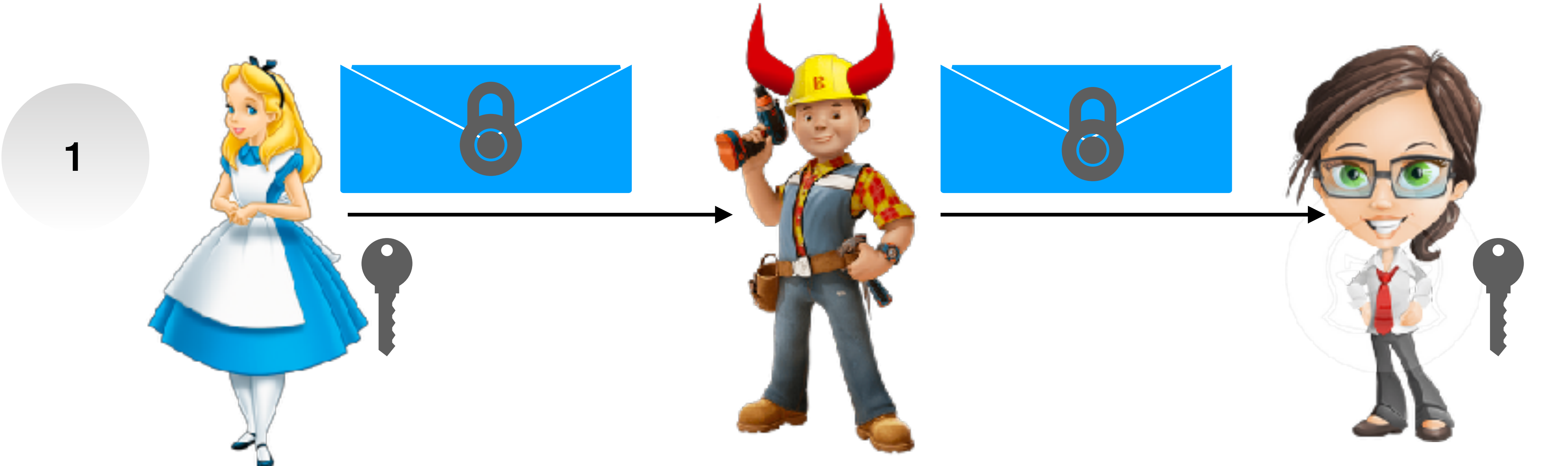
Communication through trusted / untrusted nodes



- Bob can not learn / modify the content of the message
- Bob may learn the existence of the message

Bob can learn the message received from Alice

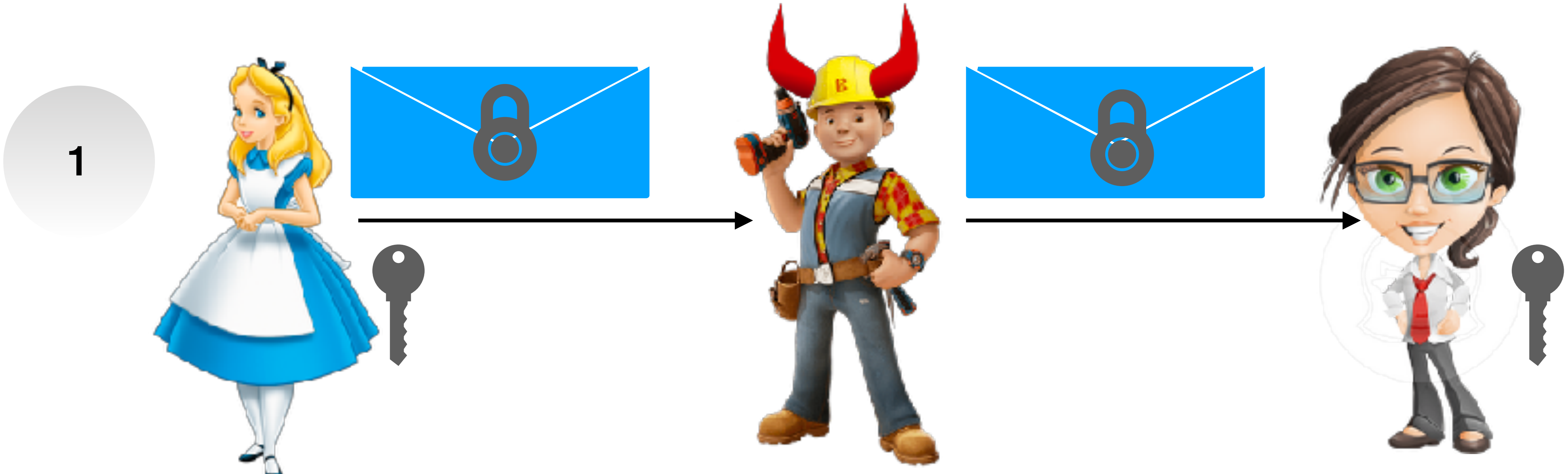
Communication through trusted / untrusted nodes



- Bob can not learn / modify the content of the message
- Bob may learn the existence of the message

Bob can learn the message received from Alice

Communication through trusted / untrusted nodes

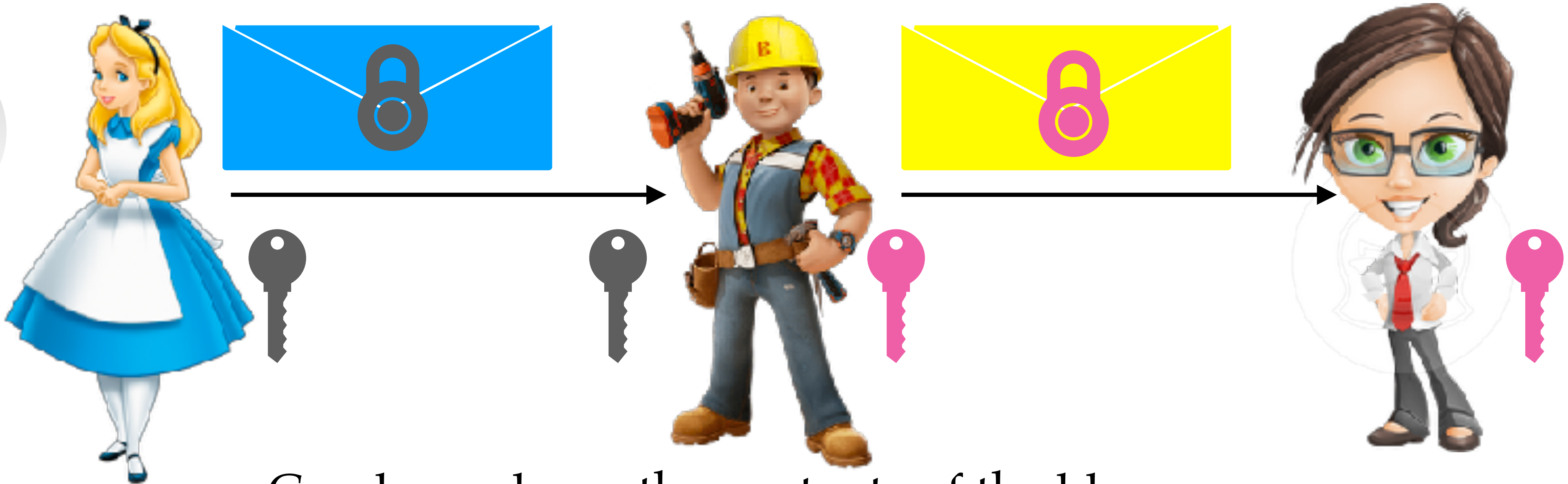


- Bob can not learn/modify the content of the message
- Bob may learn the existence of the message

Bob can learn the message received from Alice

Communication through trusted/untrusted nodes

2a



Carol **may** learn the contents of the blue message

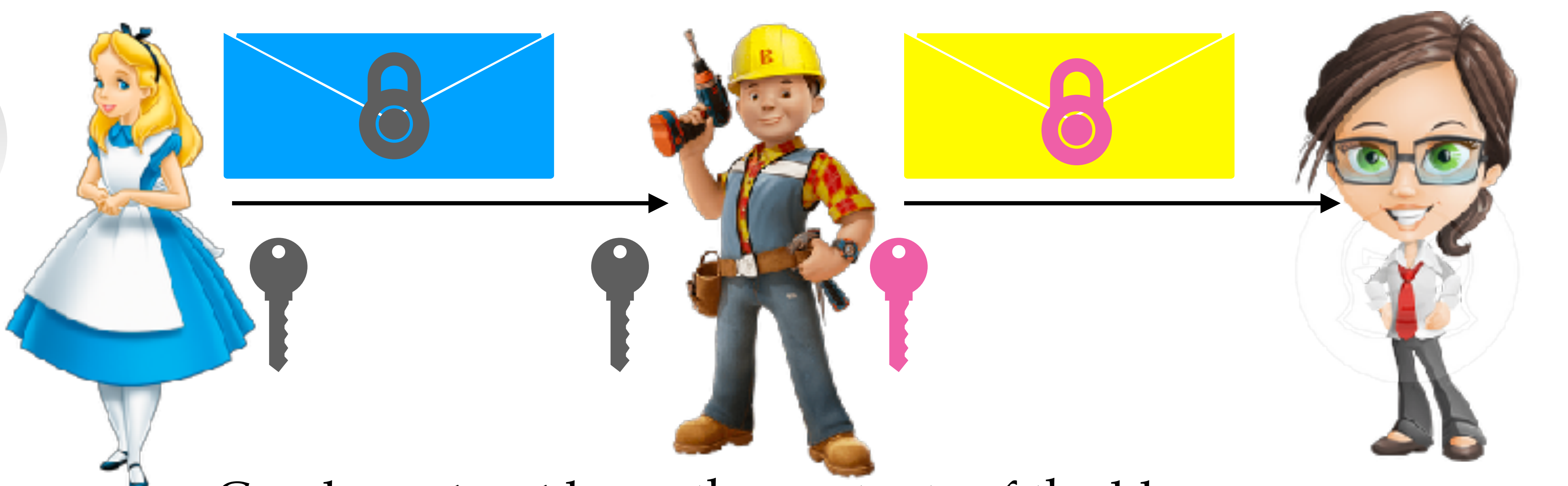
2b

Carol **must not** learn the contents of the blue message

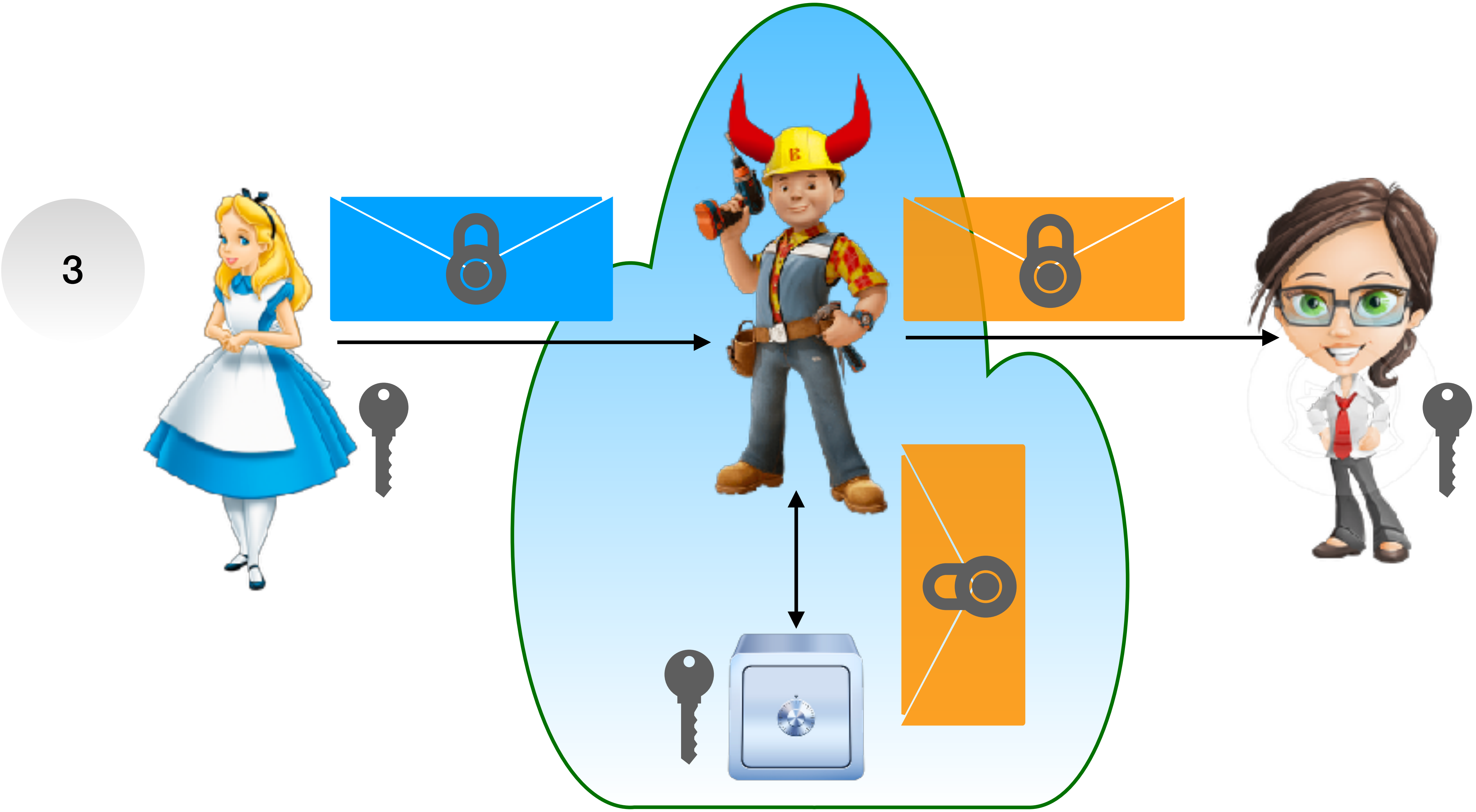
2a

Carol **may** learn the contents of the blue message

2b



Carol **must not** learn the contents of the blue message

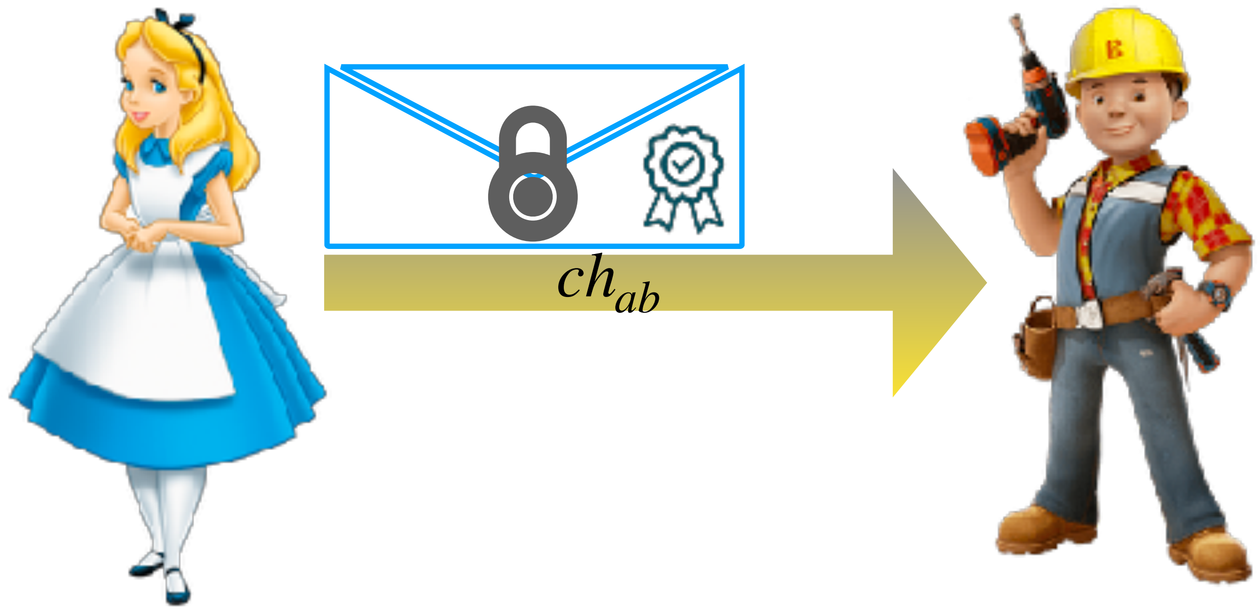


Bob cannot learn / modify the orange message

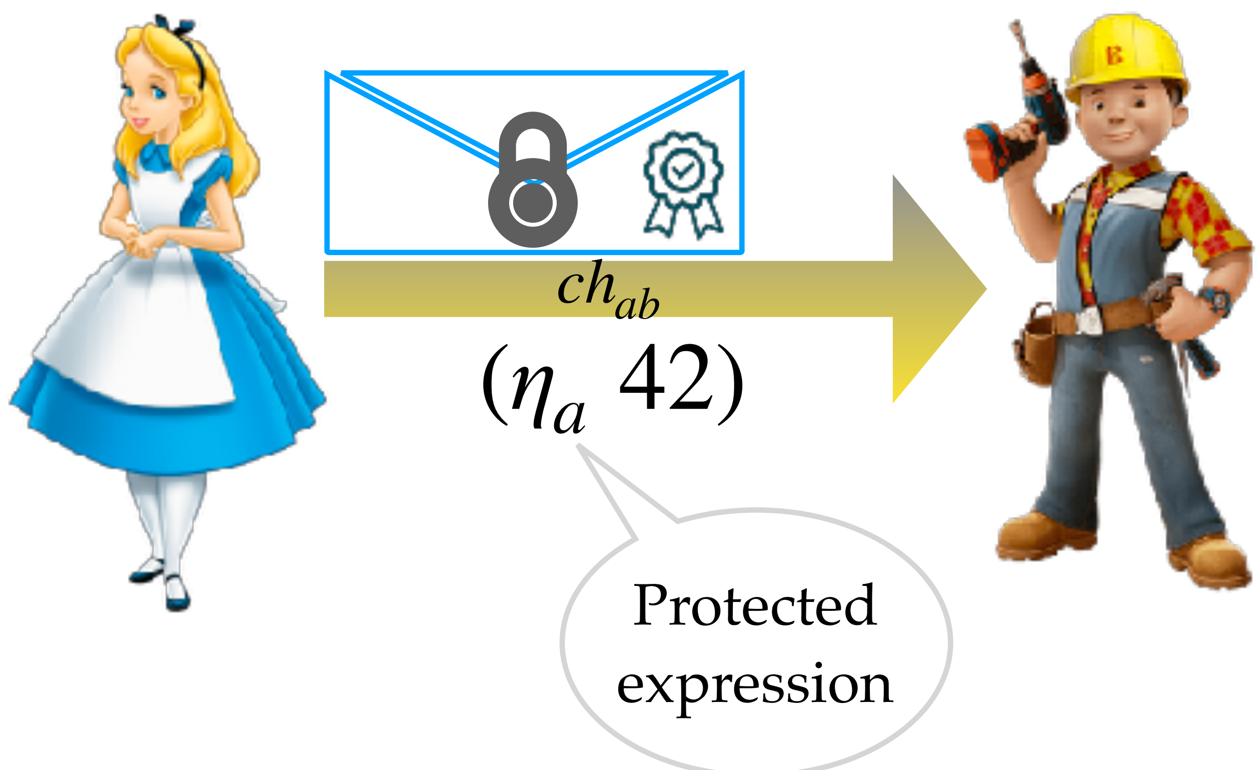
Support for communication with enclaves

DFLATE abstracts crypto mechanisms using ***protected expressions***

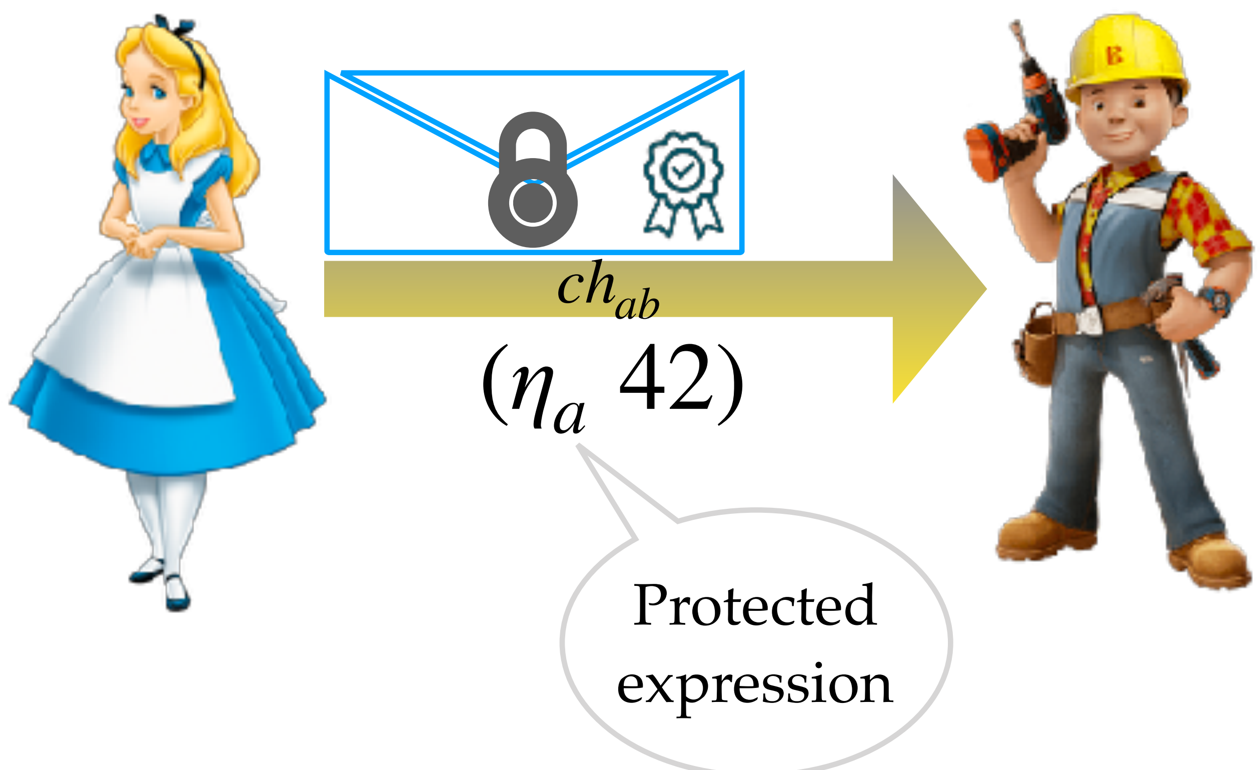
Protected Expression



Protected Expression

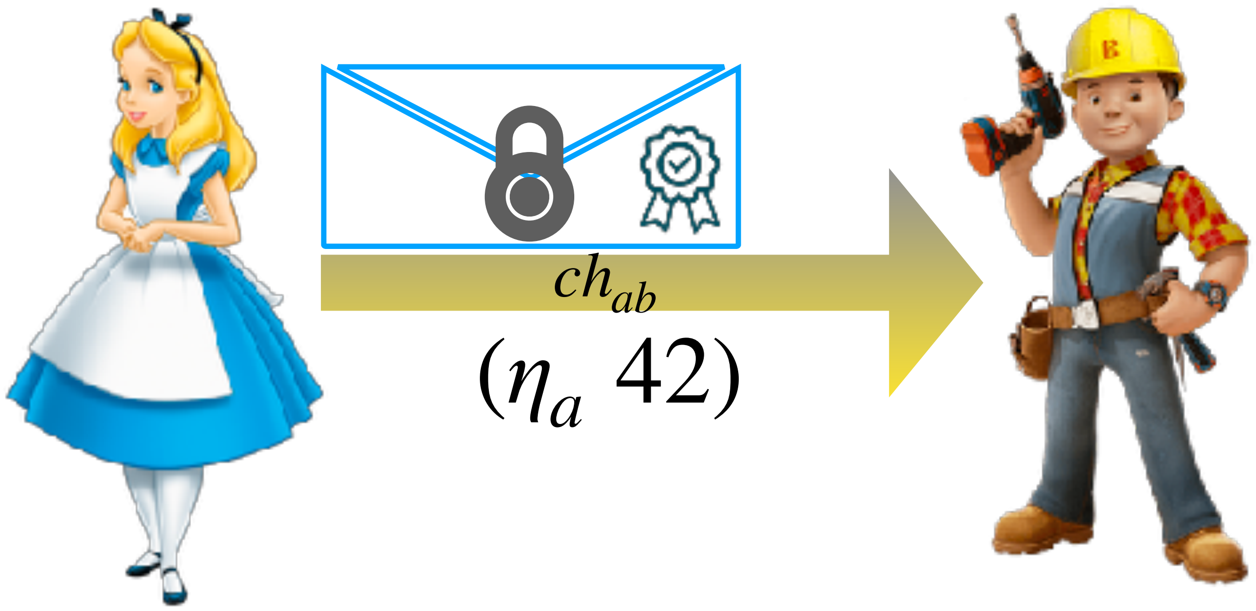


Protected Expression



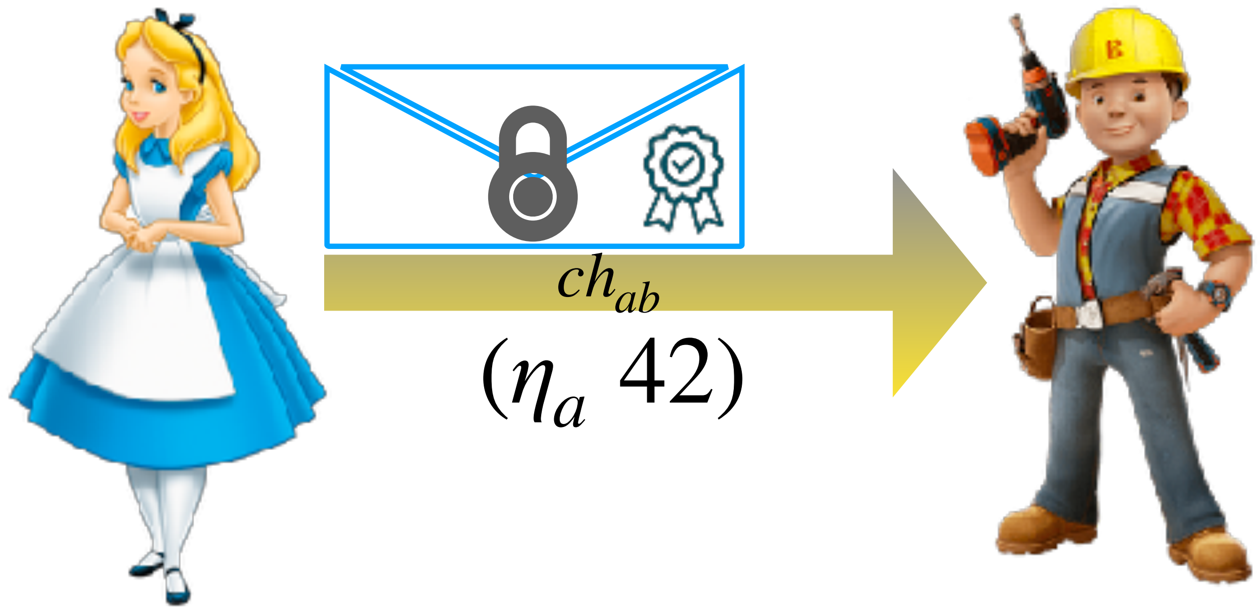
Protected expressions abstract encryption and signing

Protected Expression



$(\eta_a\ 42)$ has type a **says int**

Operating on Protected Expressions

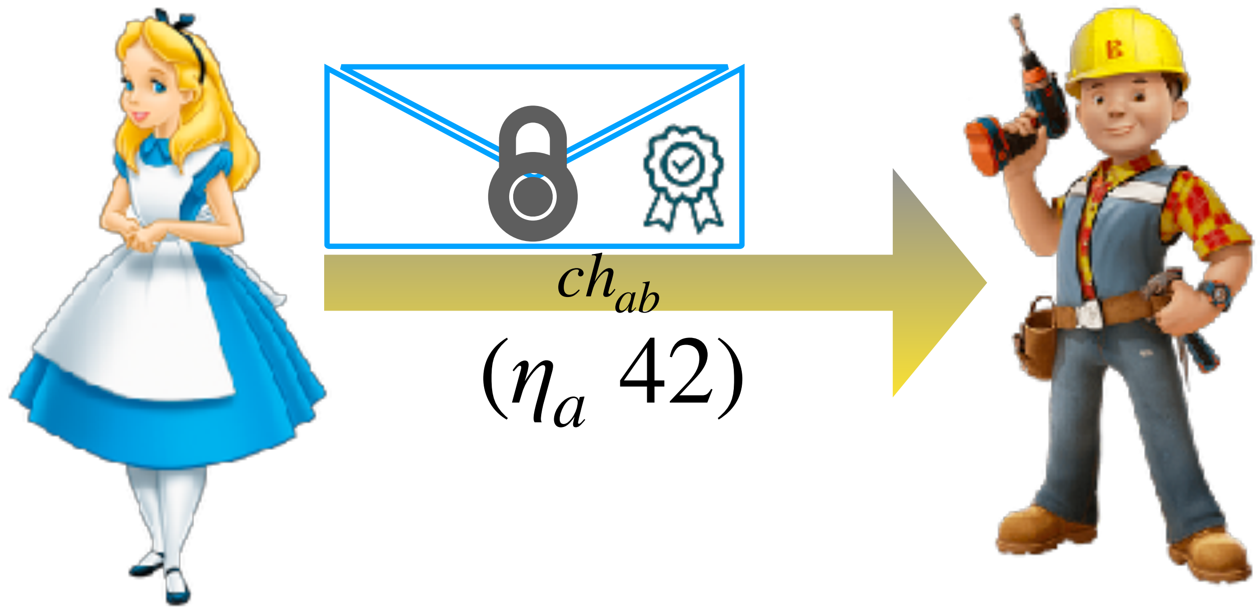


send $(\eta_a\ 42)$ **on** ch_{ab}

recv ch_{ab} **as** *enc* **in**
bind $x = enc$ **in**
$(f\ x)$

Bind abstracts
decryption and signature verification

Operating on Protected Expressions

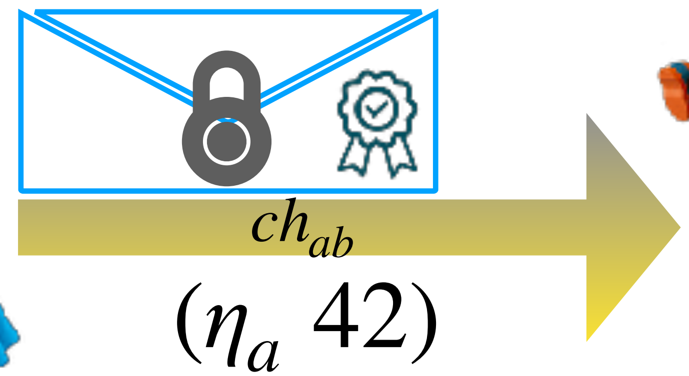


send $(\eta_a\ 42)$ **on** ch_{ab}

recv ch_{ab} **as** enc **in**
bind $x = enc$ **in**
$(f\ x)$

Bind abstracts
decryption and signature verification

Secure Bind



send $(\eta_a\ 42)$ **on** ch_{ab}

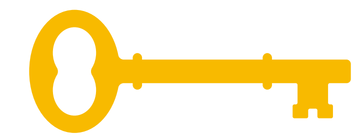
recv ch_{ab} **as** enc **in**
bind $x = enc$ **in**
$(f\ x)$

To successfully decrypt, Alice must authorize Bob

Bob ≽ Alice

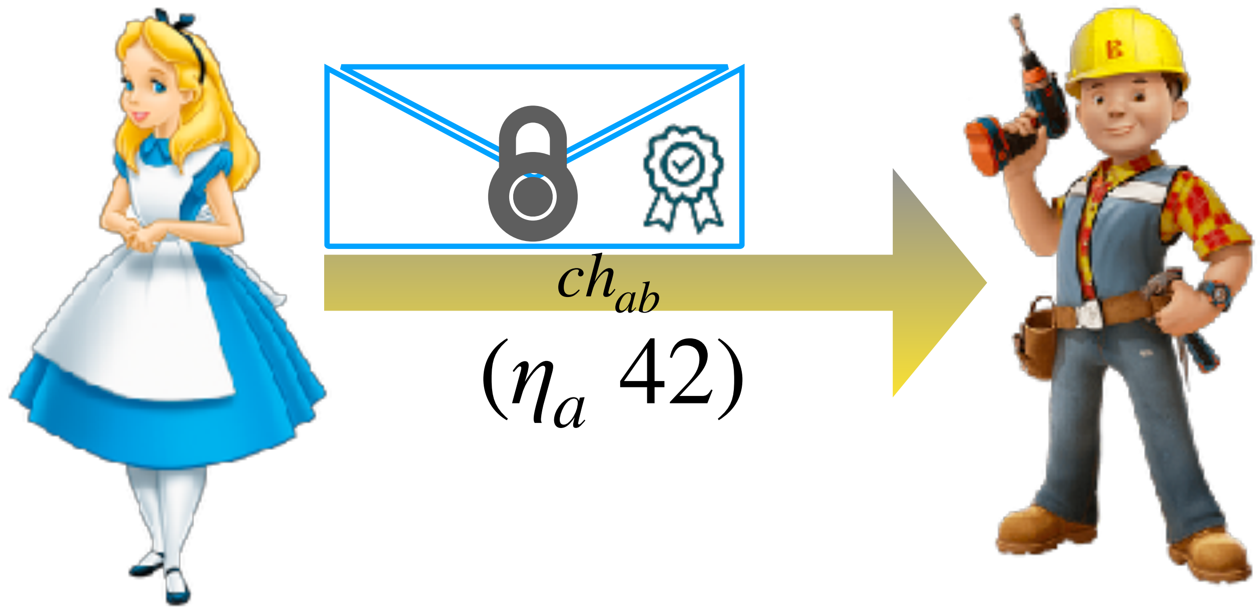
Principals delegate authority using acts-for (≽)

Bob ≽ Alice



Principals delegate authority using acts-for (≽)

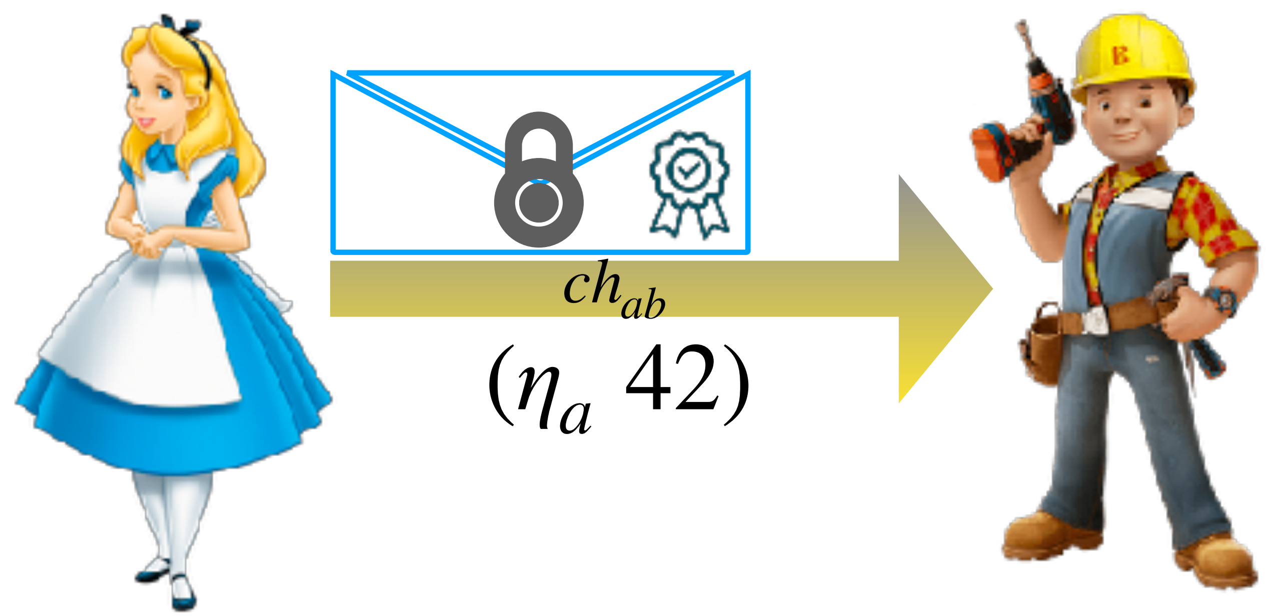
Delegation of Authority



assume $b \succcurlyeq a$ **in**
send $(\eta_a\ 42)$ **on** ch_{ab}

recv ch_{ab} **as** enc **in**
bind $x = enc$ **in**
$(f\ x)$

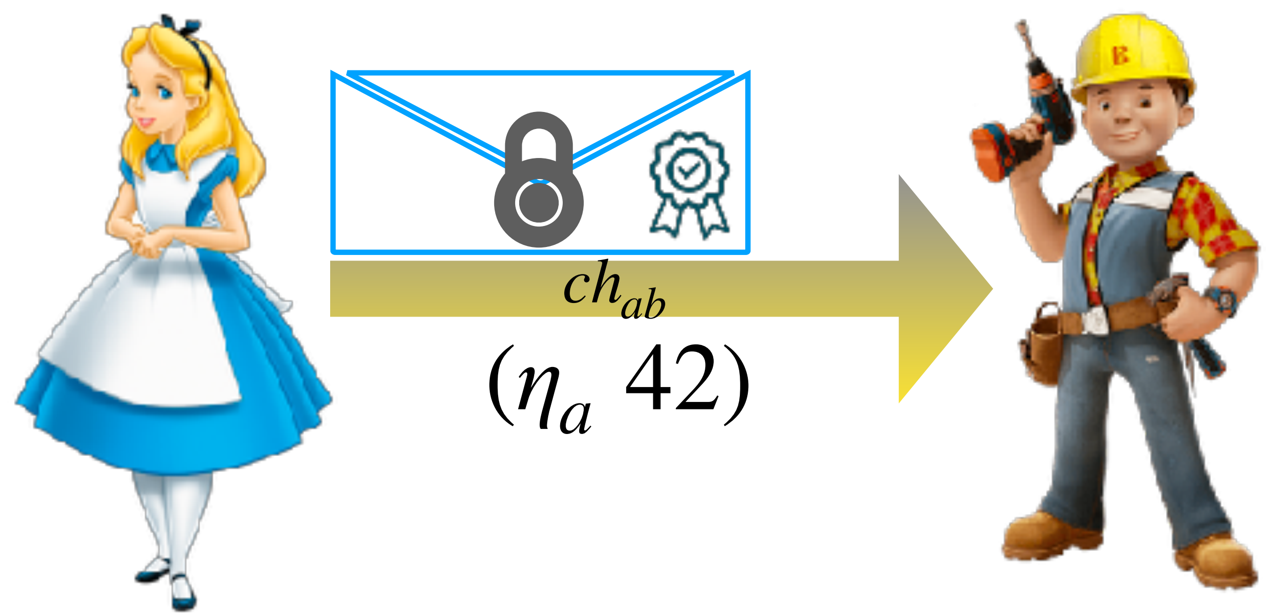
Delegation of Authority



assume $b \succcurlyeq a$ **in**
send $(\eta_a\ 42)$ **on** ch_{ab}

recv ch_{ab} **as** enc **in**
bind $x = enc$ **in**
$(f\ x)$

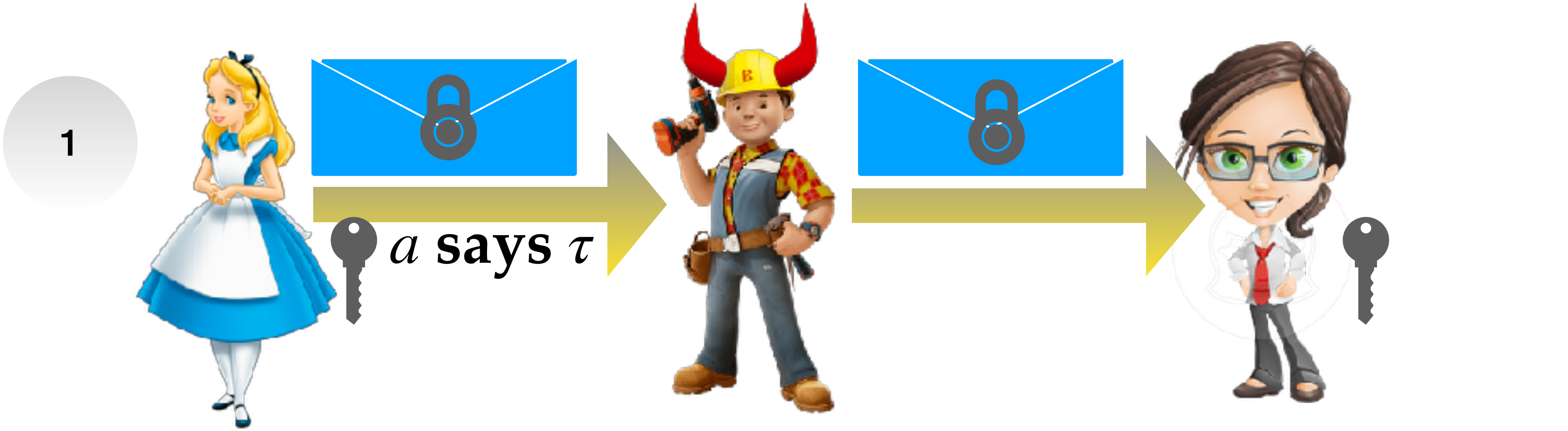
Delegation of Authority



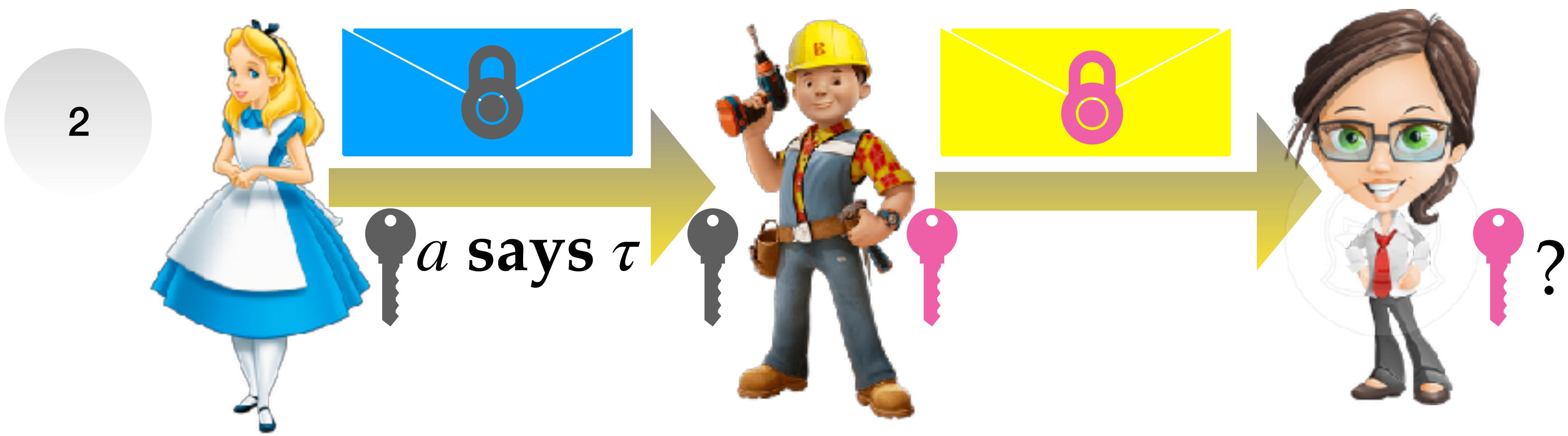
assume $b \geqslant a$ **in**
  send $(\eta_a\ 42)$ **on** ch_{ab}

recv ch_{ab} **as** enc **in**
 bind $x = enc$ **in**
  $(f\ x)$

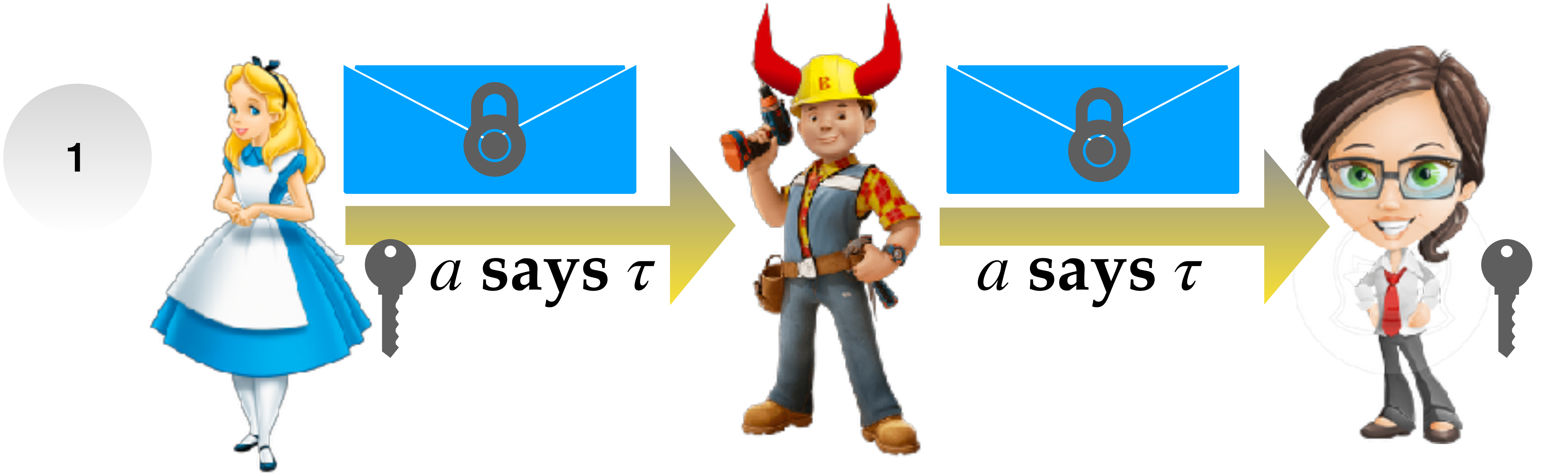
Assume abstracts key sharing among principals



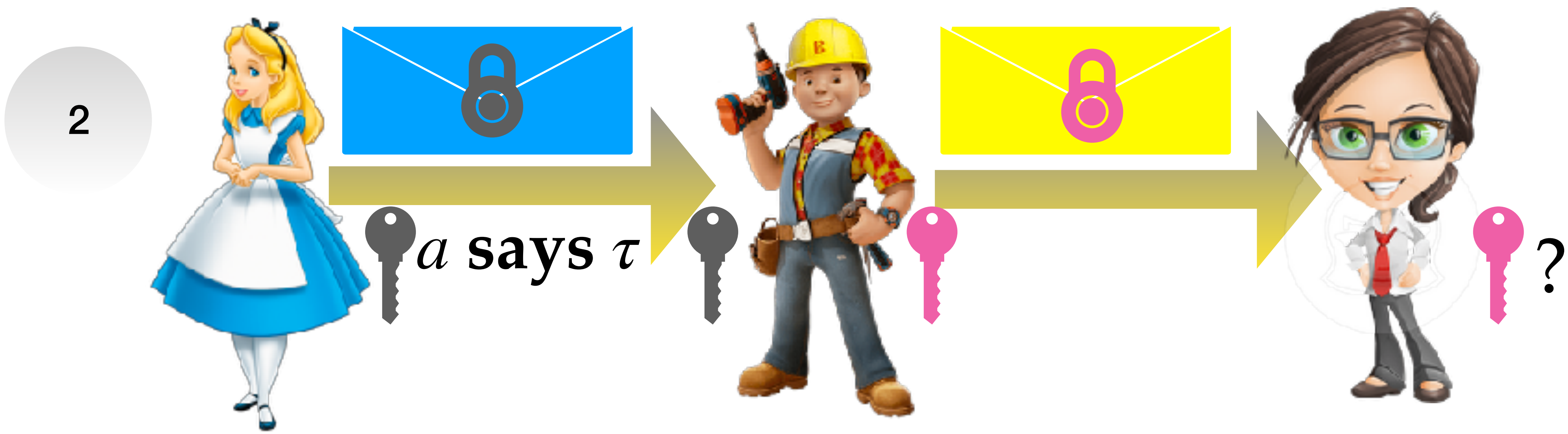
- Bob must not learn/modify the content of the message
- Bob may learn the existence of the message



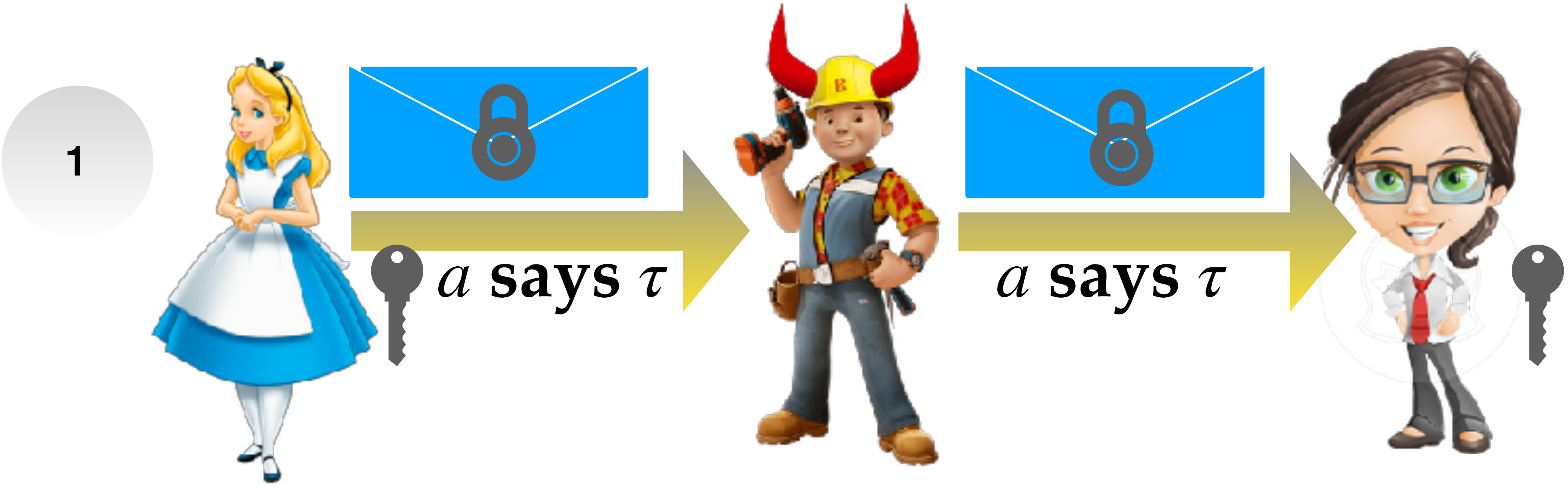
Bob can learn the message received from Alice



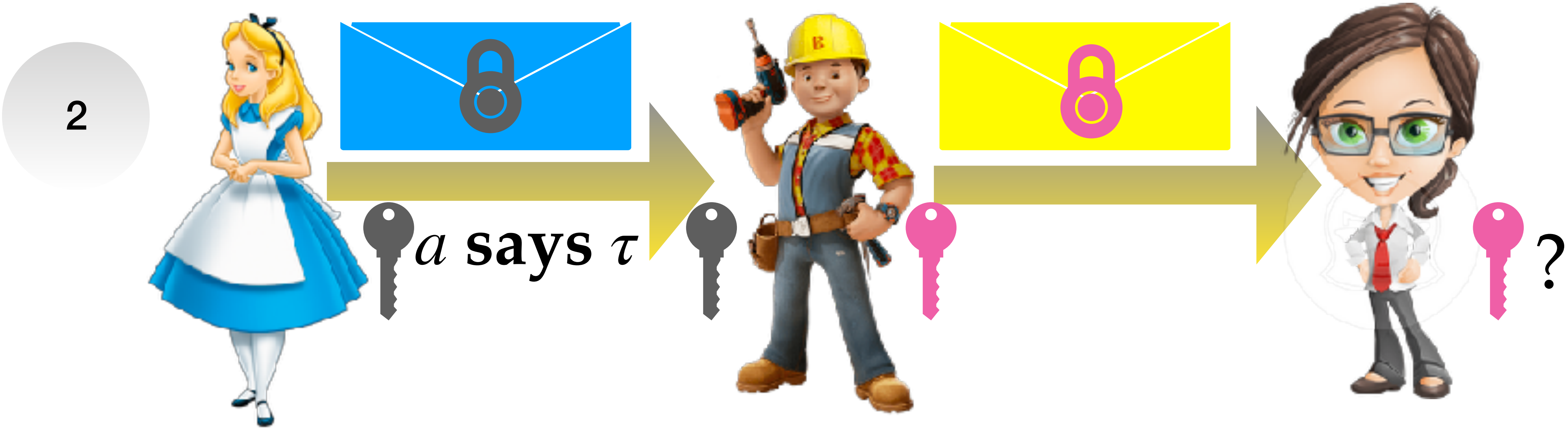
- Bob must not learn / modify the content of the message
- Bob may learn the existence of the message



Bob can learn the message received from Alice



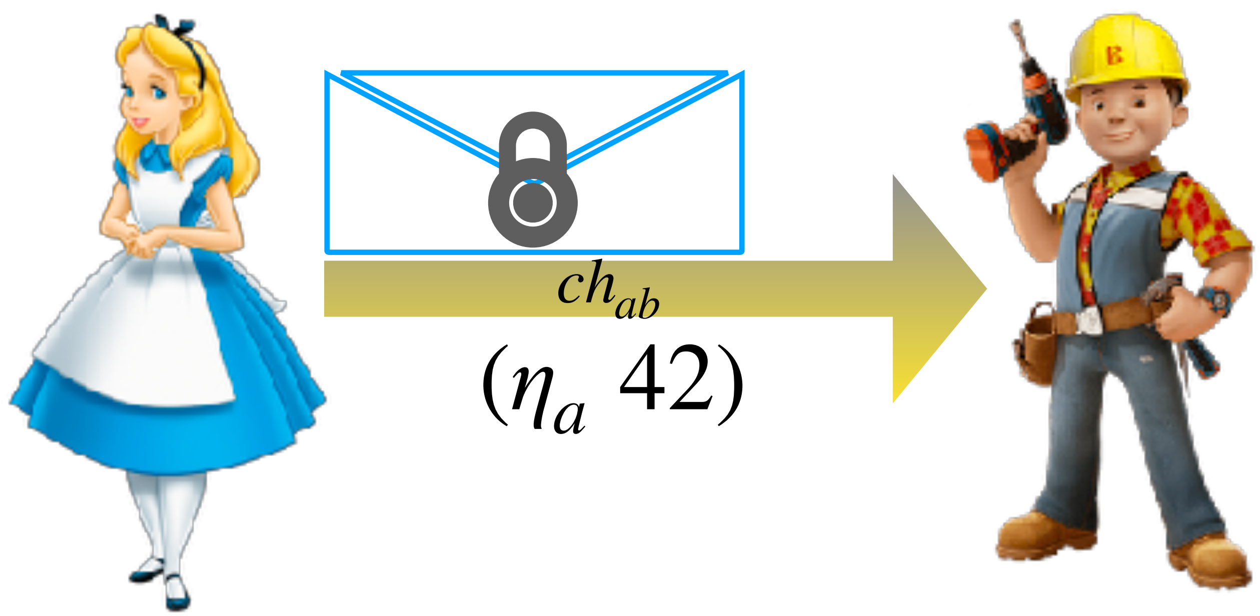
- Bob must not learn / modify the content of the message
- Bob may learn the existence of the message



Bob can learn the message received from Alice

Types enable reasoning about Bob's power

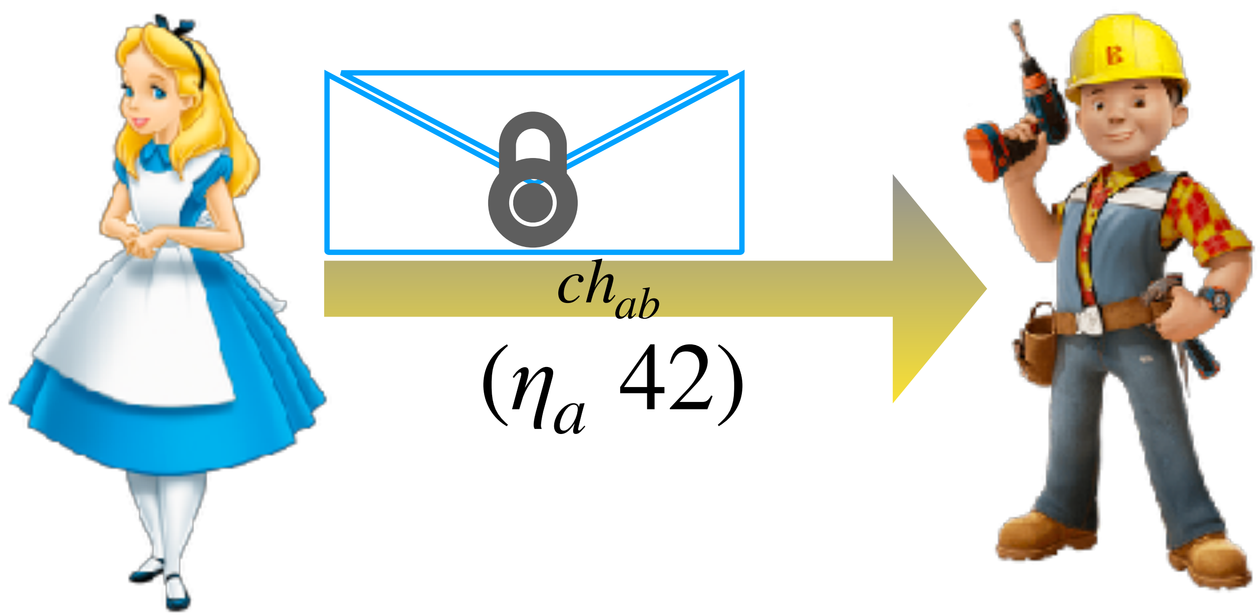
Secure Bind (2)



assume $b \succcurlyeq a$ **in**
  send $(\eta_a\ 42)$ **on** ch_{ab}

recv ch_{ab} **as** enc **in**
 bind $x = enc$ **in**
  $(f\ x)$

Secure Bind (2)

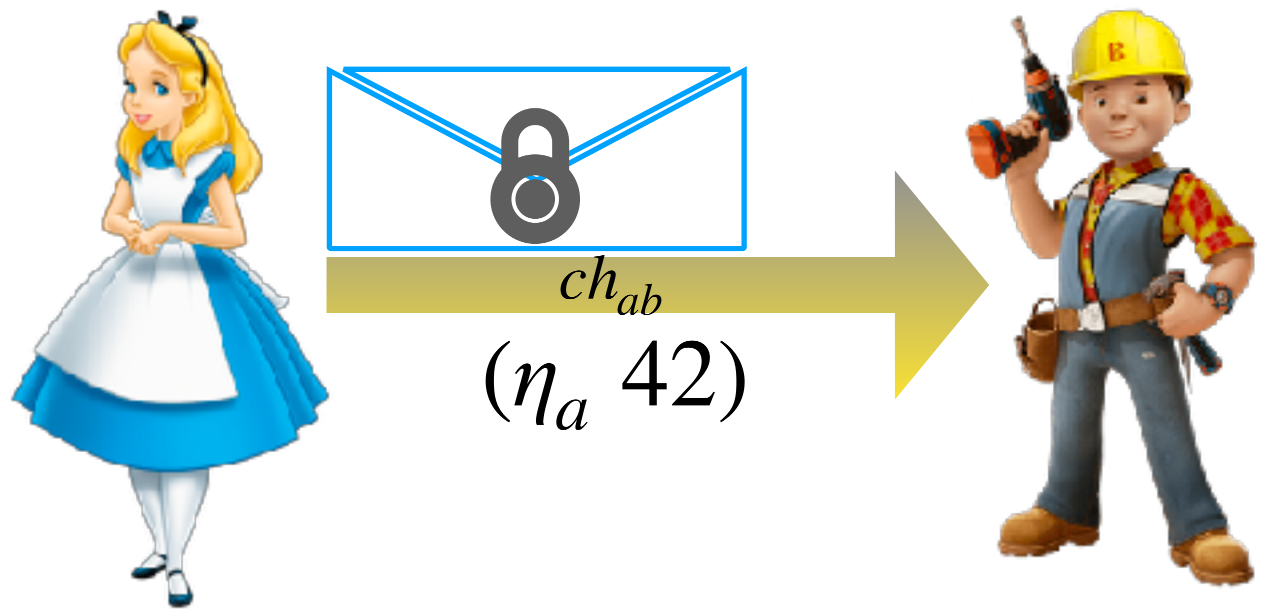


assume $b \succcurlyeq a$ **in**
send $(\eta_a\ 42)$ **on** ch_{ab}

recv ch_{ab} **as** enc **in**
bind $x = enc$ **in**
$(f\ x)$

Bob must not leak the decrypted value

Secure Bind (2)



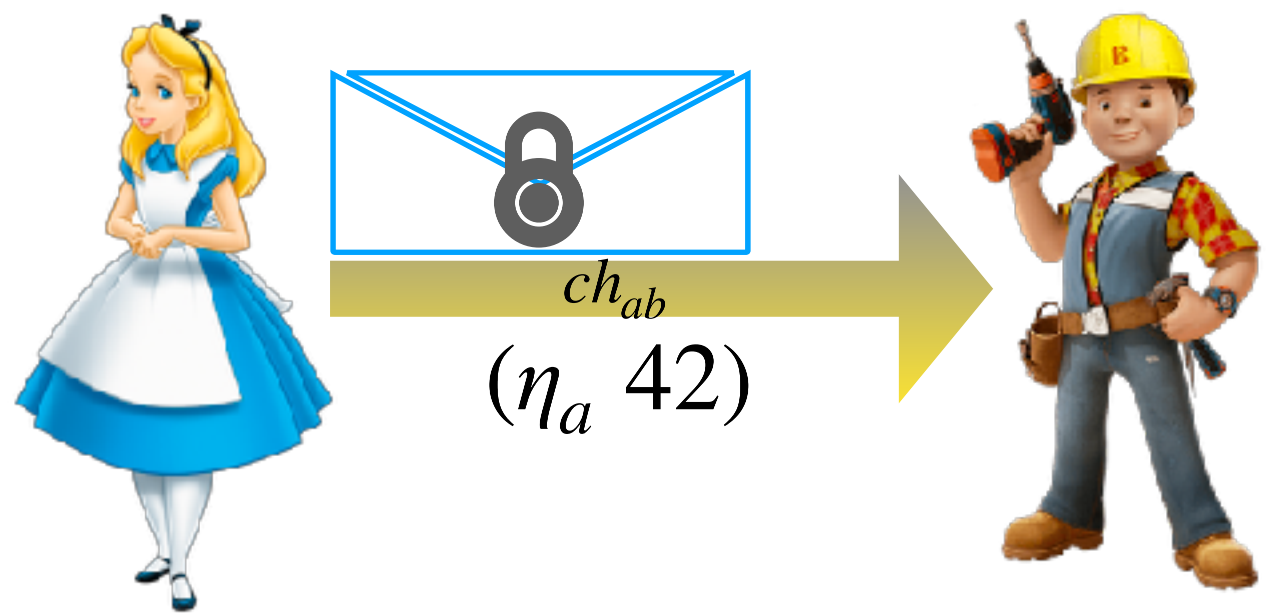
ass
s

$a \sqsubseteq \tau$

recv ch_{ab} **as** *enc* **in**
bind $x = enc$ **in**
$(f\ x)$

The output type of bind must protect Alice

Secure Bind (2)



$$a \sqsubseteq \tau$$

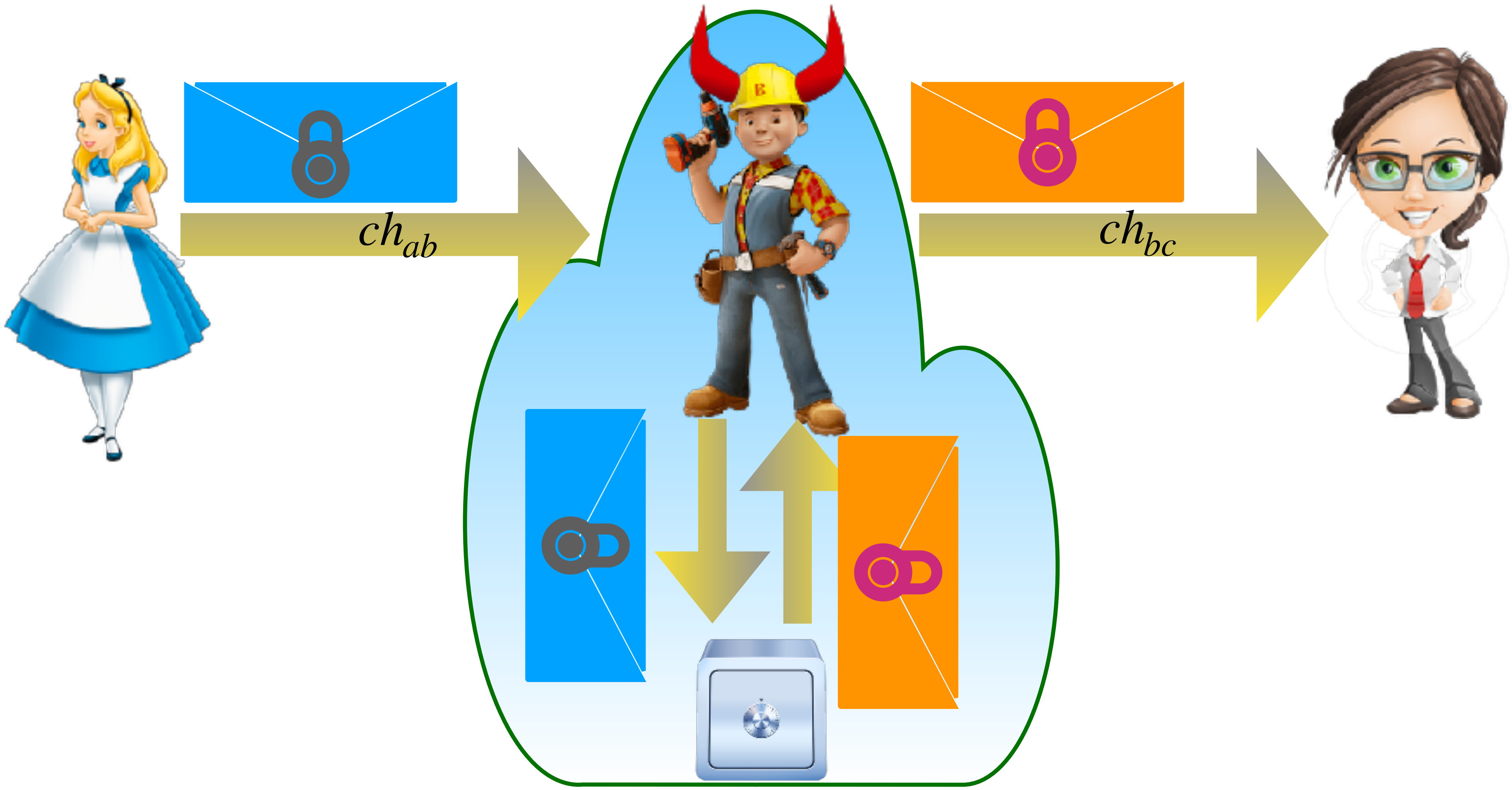
recv ch_{ab} **as** *enc* **in**
bind x = *enc* **in**
$(f\ x)$

The output type of bind must protect Alice

Carol **may** learn the contents of the blue message

Carol **must not** learn the contents of the blue message

Type system ensures that Bob uses the decrypted value securely



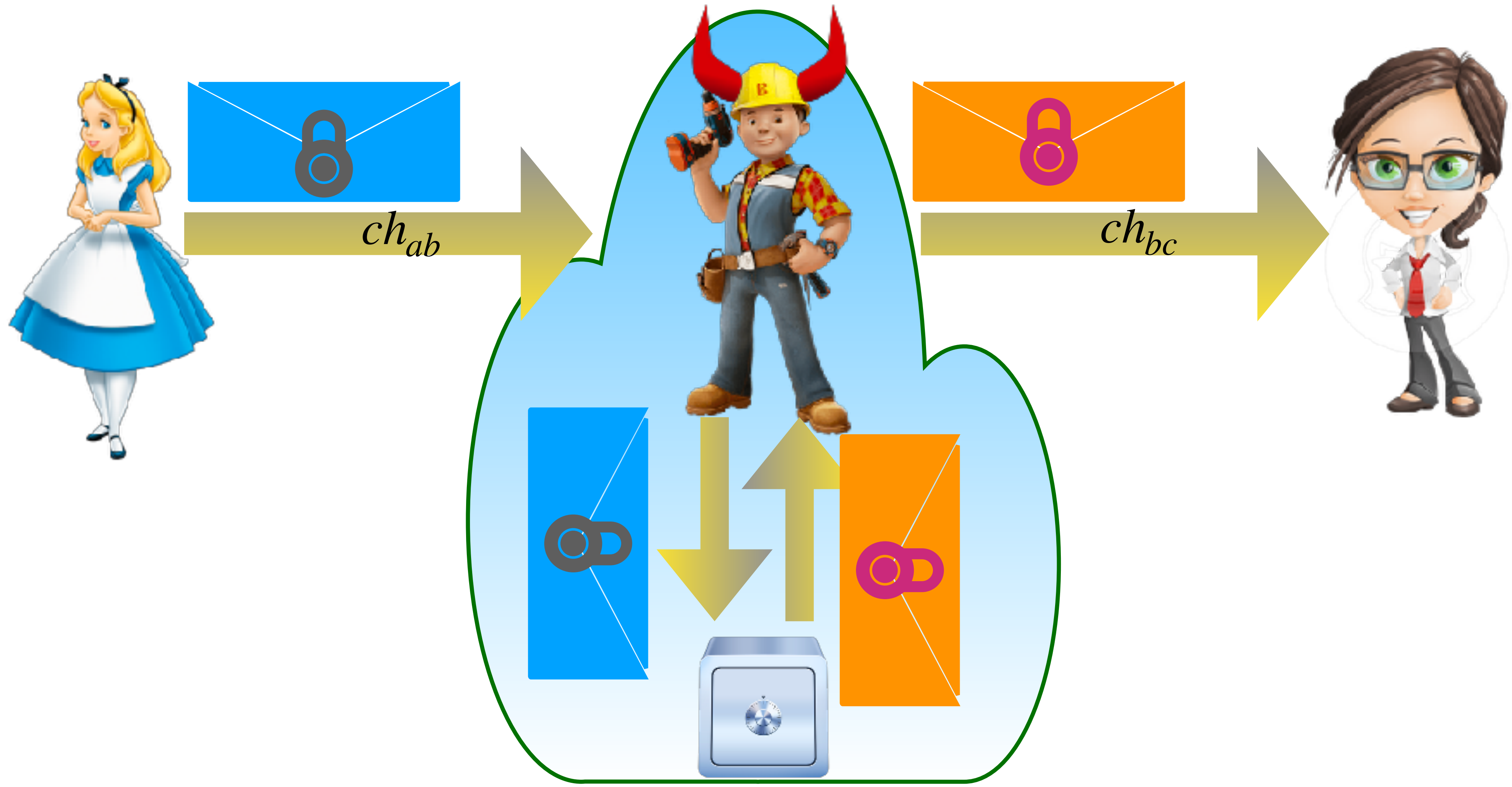
assume $t \geqslant a$ **in**
  send $(\eta_a \; blue)$ **on** ch_{ab}

recv ch_{ab} **as** x
  send x **on** ch_{bt}
   recv ch_{bt} **as** x' **in**
   …

TEE$^{\mathbf{t}}$ {
 …
  send v **on** ch_{bt}
}

assume $t \geqslant c$ **in**
  recv ch_{bc} **as** x **in**
  …

Abstracting TEE



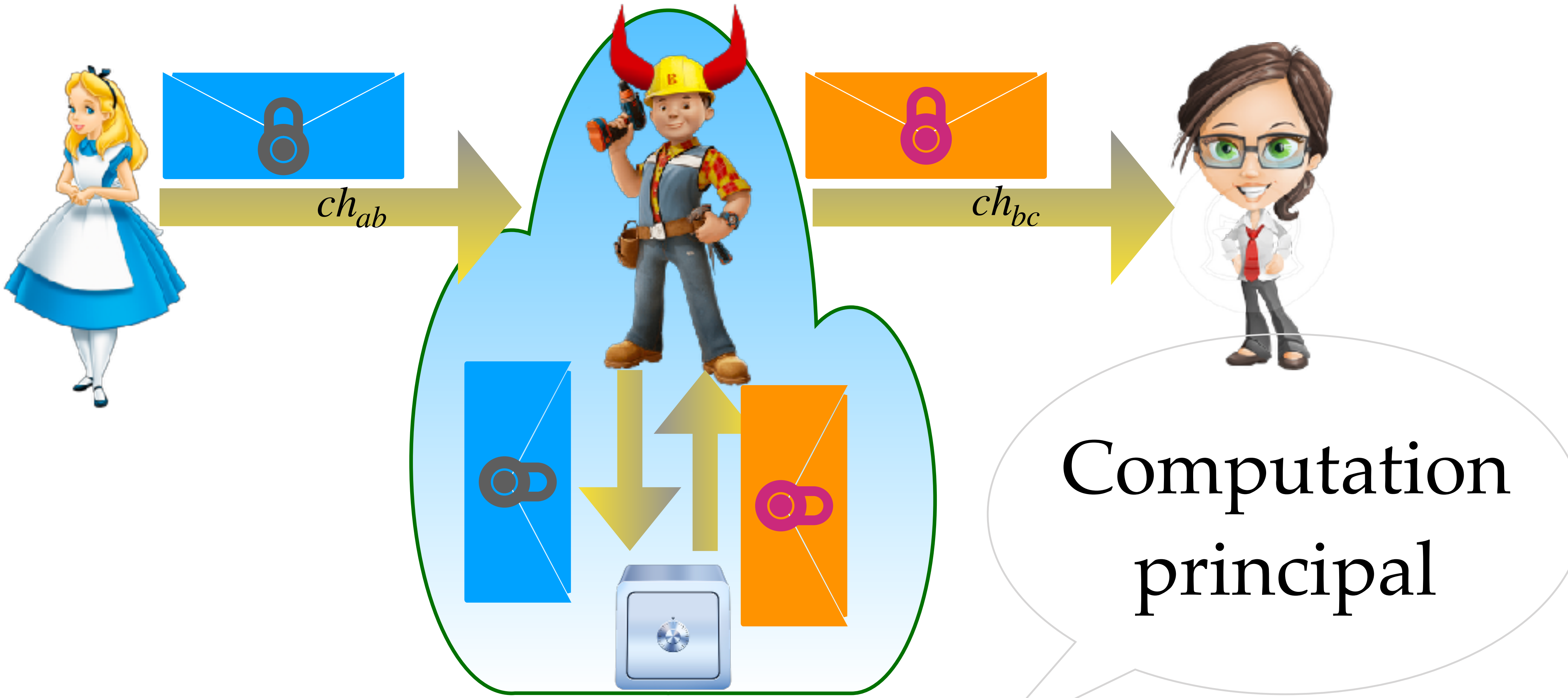
assume $t \geqslant a$ in
send $(\eta_a\ blue)$ on ch_{ab}

recv ch_{ab} **as** x
send x **on** ch_{bt}
recv ch_{bt} **as** x' **in**
...

TEE$^{\mathbf{t}}$ {
...
send v **on** ch_{bt}
}

assume $t \geqslant c$ in
recv ch_{bc} as x in
...

Abstracting TEE



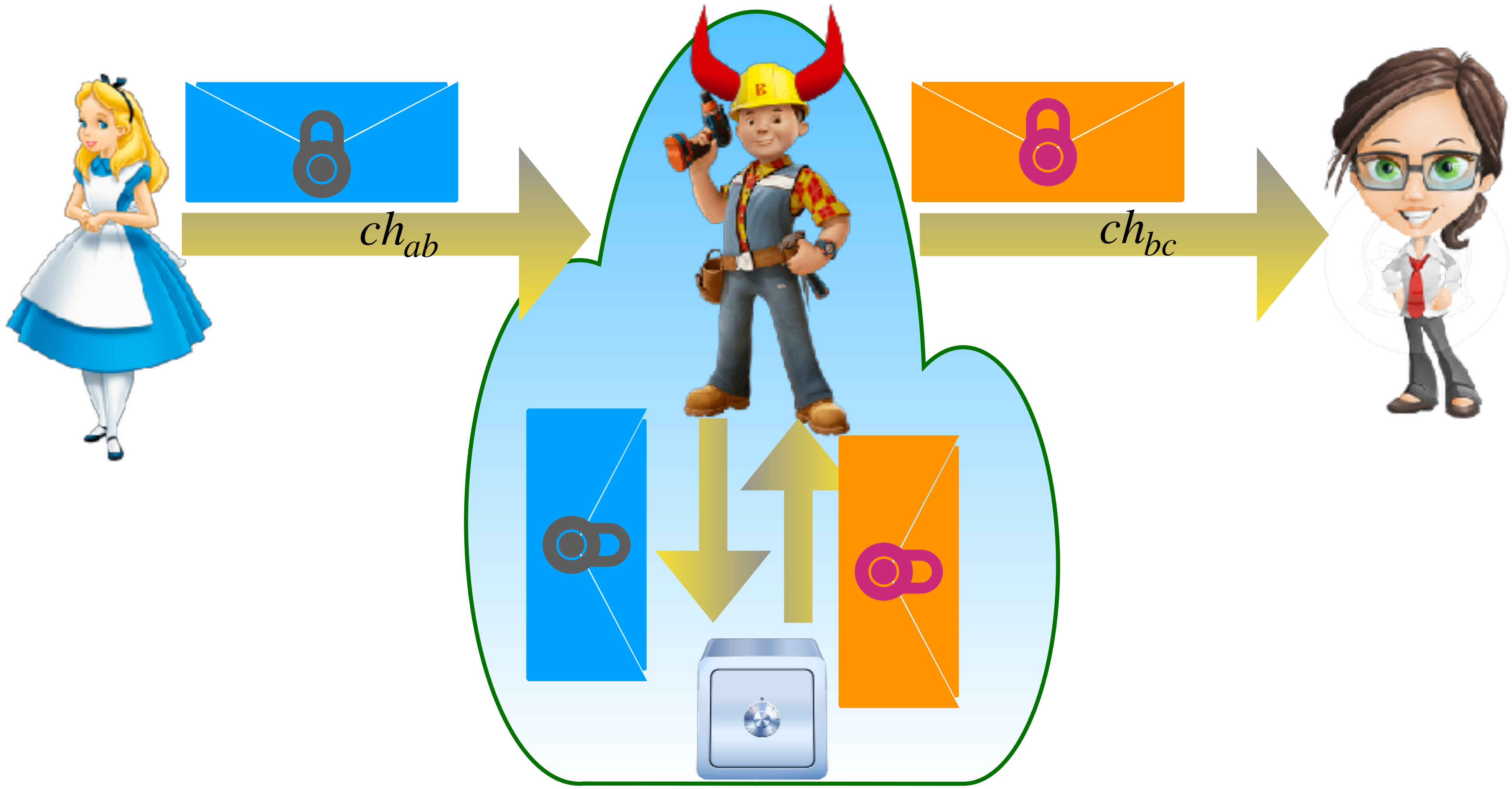
assume $t \geqslant a$ in
send $(\eta_a\ blue)$ on ch_{ab}

recv ch_{ab} **as** x
send x **on** ch_{bt}
recv ch_{bt} **as** x' **in**
…

TEE^{t} {
…
send v **on** ch_{bt}
}

assume $t \geqslant c$ in
recv ch_{bc} as x in
…

Abstracting TEE



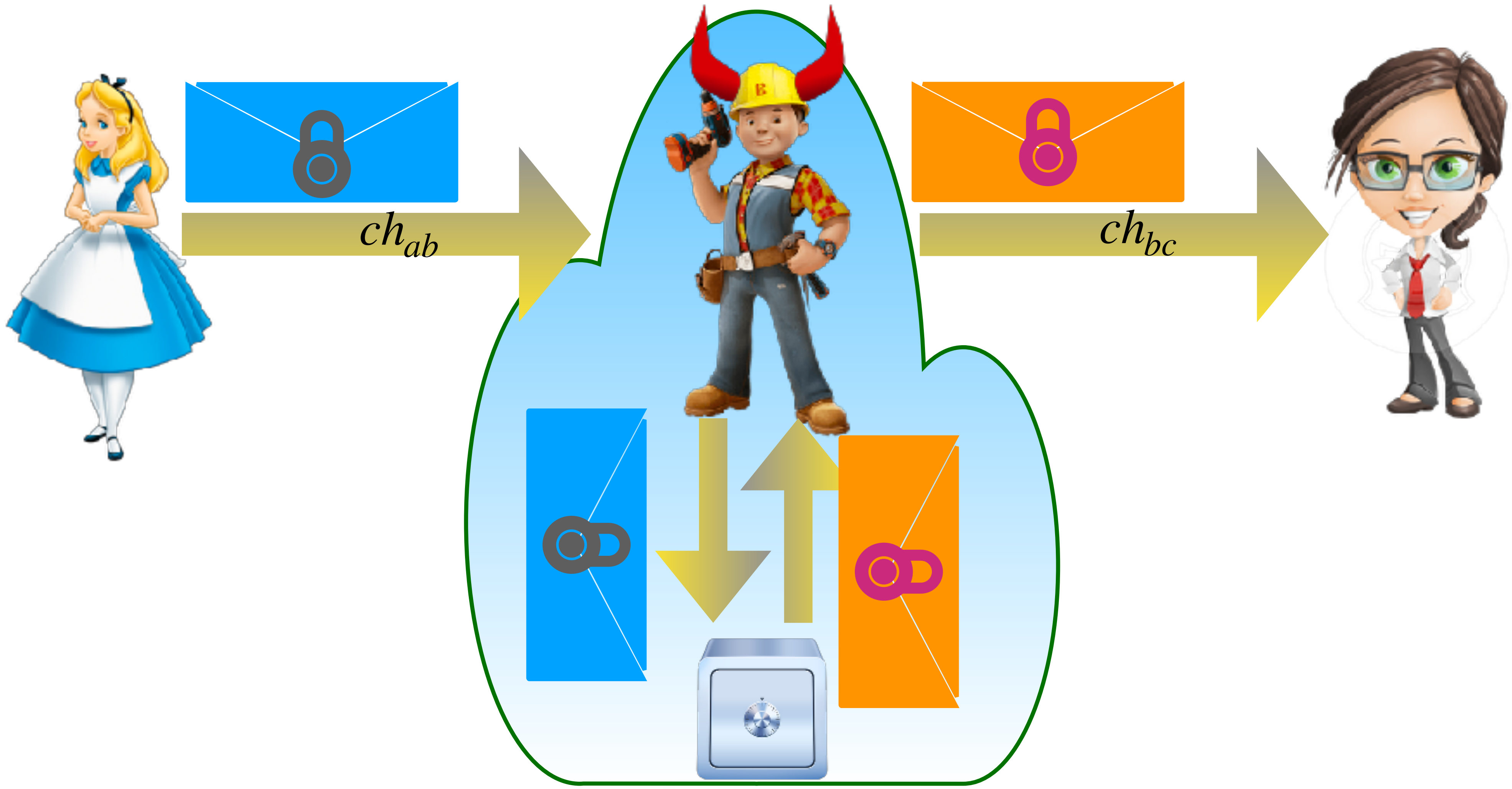
assume $t \succcurlyeq a$ **in**
send $(\eta_a\ blue)$ **on** ch_{ab}

recv ch_{ab} **as** x
send ch_{bt} x **then**
recv ch_{bt} **as** x' **in**
…

TEE$^{\mathbf{t}}$ {
…
send v **on** ch_{bt}
}

assume $t \succcurlyeq c$ **in**
recv ch_{bc} **as** x **in**
…

Abstracting TEE



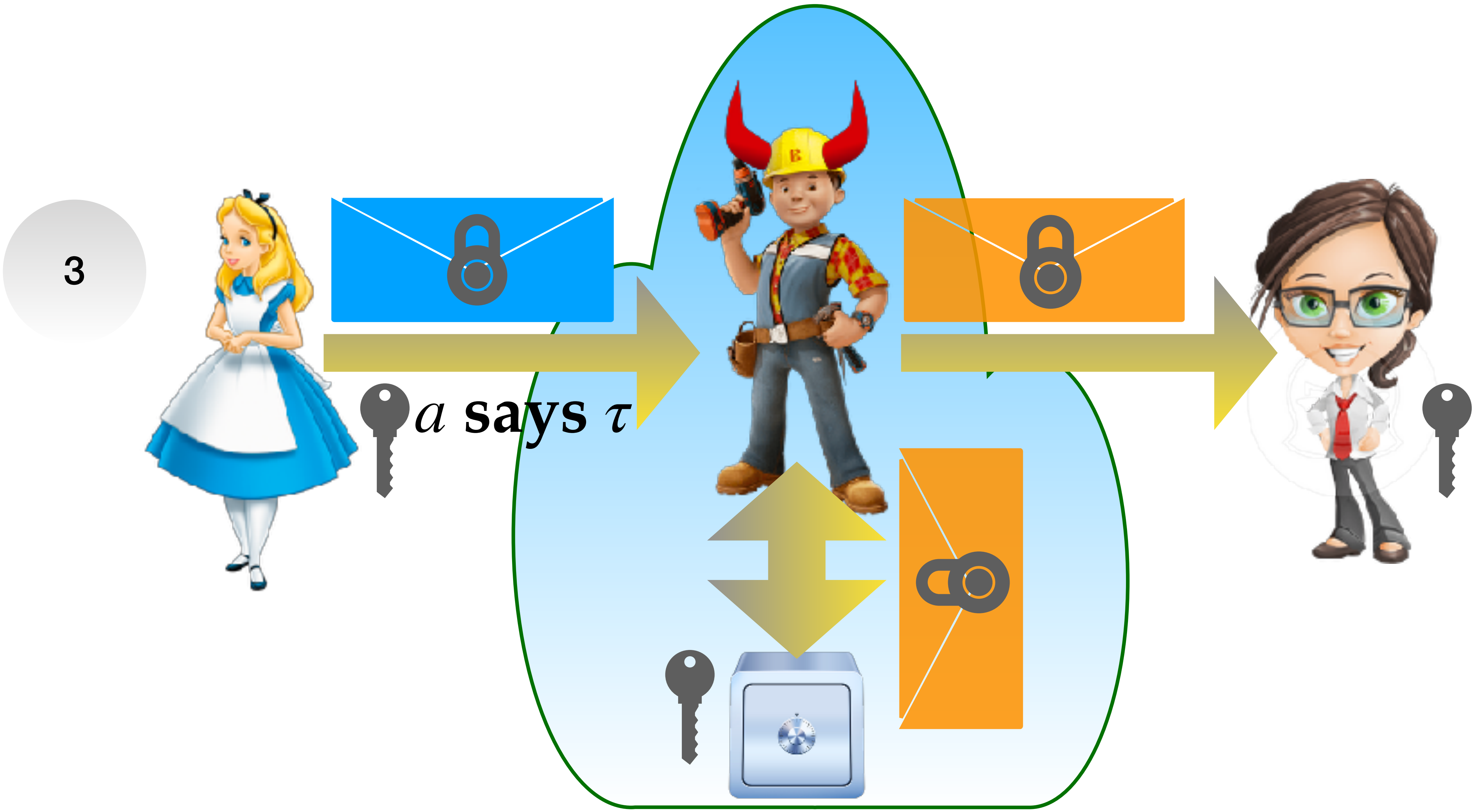
assume $t \succcurlyeq a$ **in**
send $(\eta_a\ blue)$ **on** ch_{ab}

recv ch_{ab} **as** x
send ch_{bt} x **then**
recv ch_{bt} **as** x' **in**
…

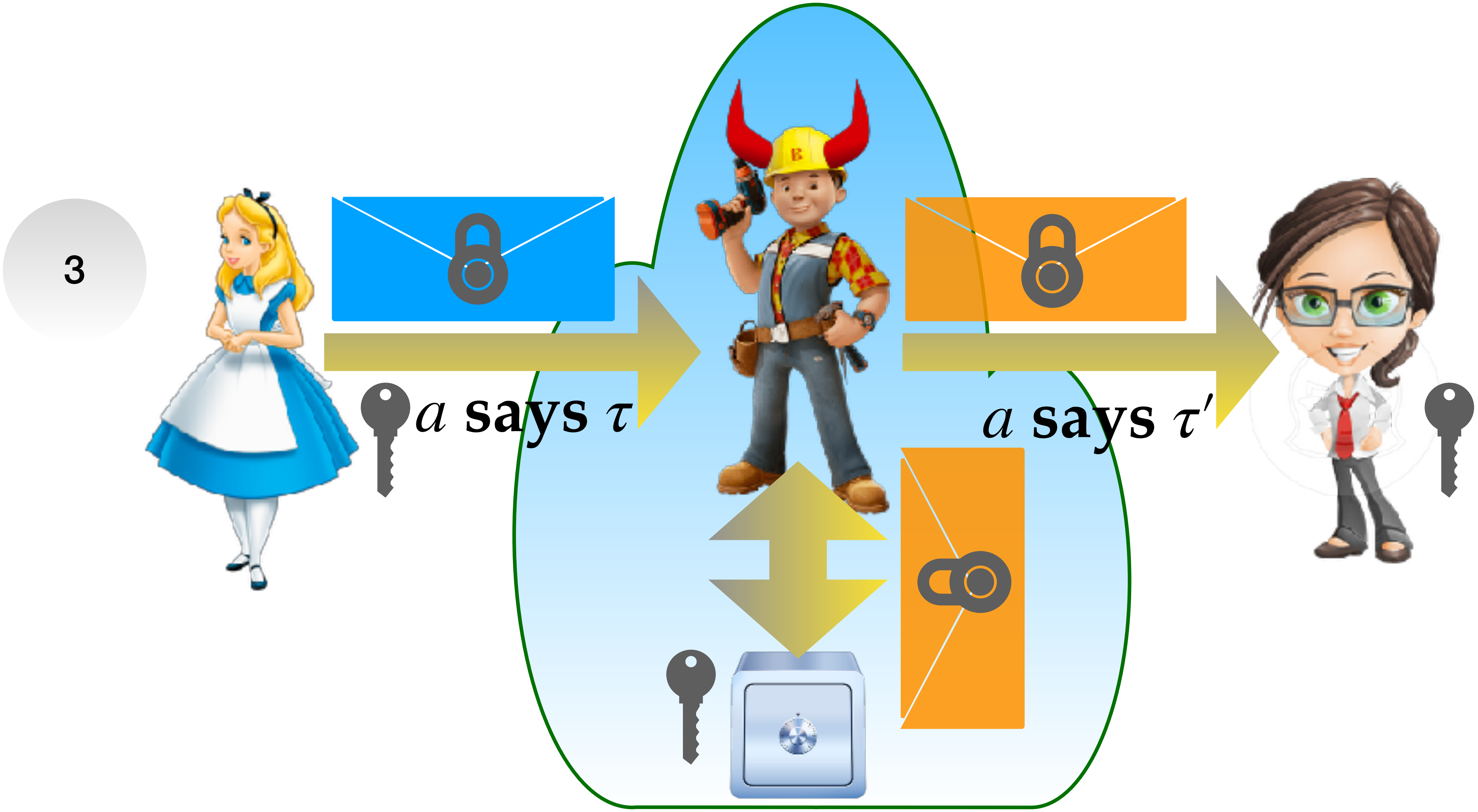
TEE^{t} {
…
send v **on** ch_{bt}
}

assume $t \succcurlyeq c$ **in**
recv ch_{bc} **as** x **in**
…

Abstracting TEE



Bob cannot learn / modify the orange message



Bob cannot learn / modify the orange message

Implementing DFLATE

Design

- DFLATE abstractions are implementable!

Design

- DFLATE abstractions are implementable!
- TEEs can be implemented by Intel SGX enclaves
    - SGX provides remote attestation
    - SGX enclave communicates through the host

Design

- DFLATE abstractions are implementable!
- TEEs can be implemented by Intel SGX enclaves
  - SGX provides remote attestation
  - SGX enclave communicates through the host
- Protected expressions can be implemented using public key encryption and digital signatures
  - However, this requires access to the corresponding signing/decryption keys
  - Key distribution, especially for enclaves, is non-trivial

Key Distribution

- A global key master has key pairs for all principals
- Key master provisions the nodes and enclaves with necessary private keys

Key Distribution

Key Master

To obtain keys, a node proves its identity to the key master

Key Distribution

Key Master

To obtain keys, a node proves its identity to the key master

Key Distribution

Key Master

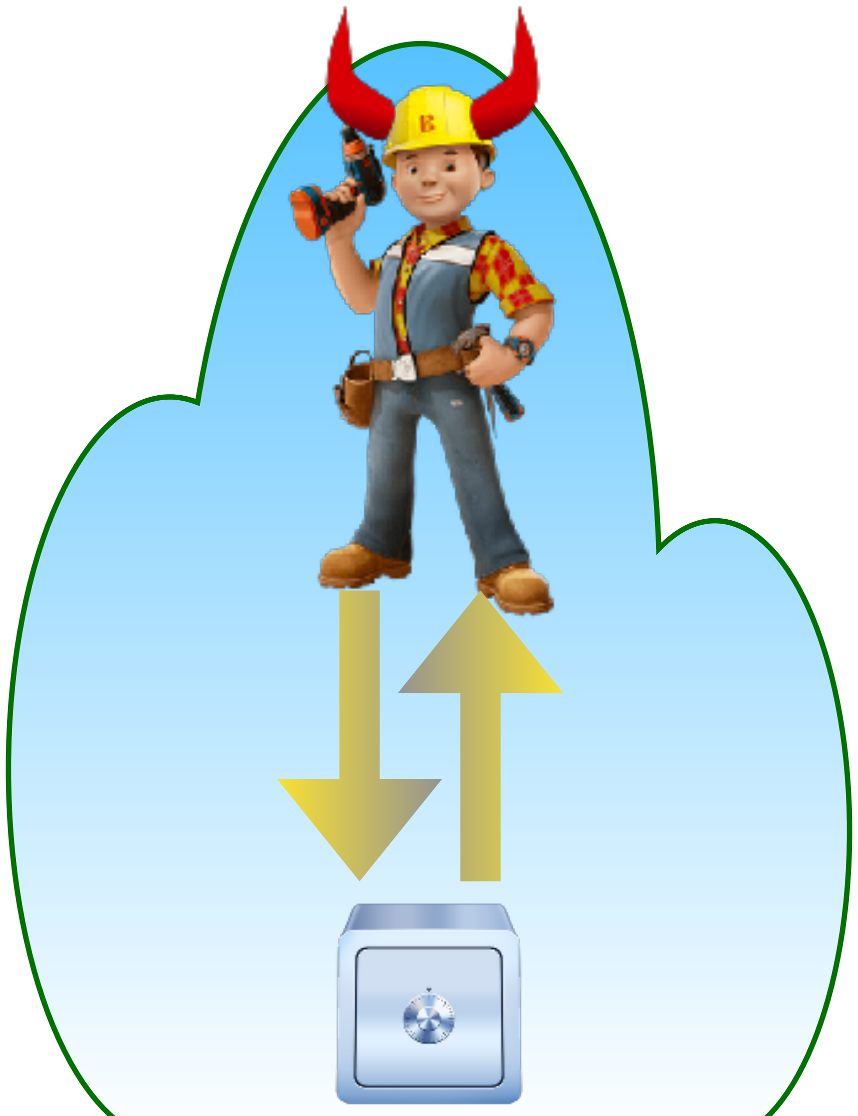
To obtain keys, a node proves its identity to the key master

Key Distribution

Key Master

To obtain keys, a node proves its identity to the key master

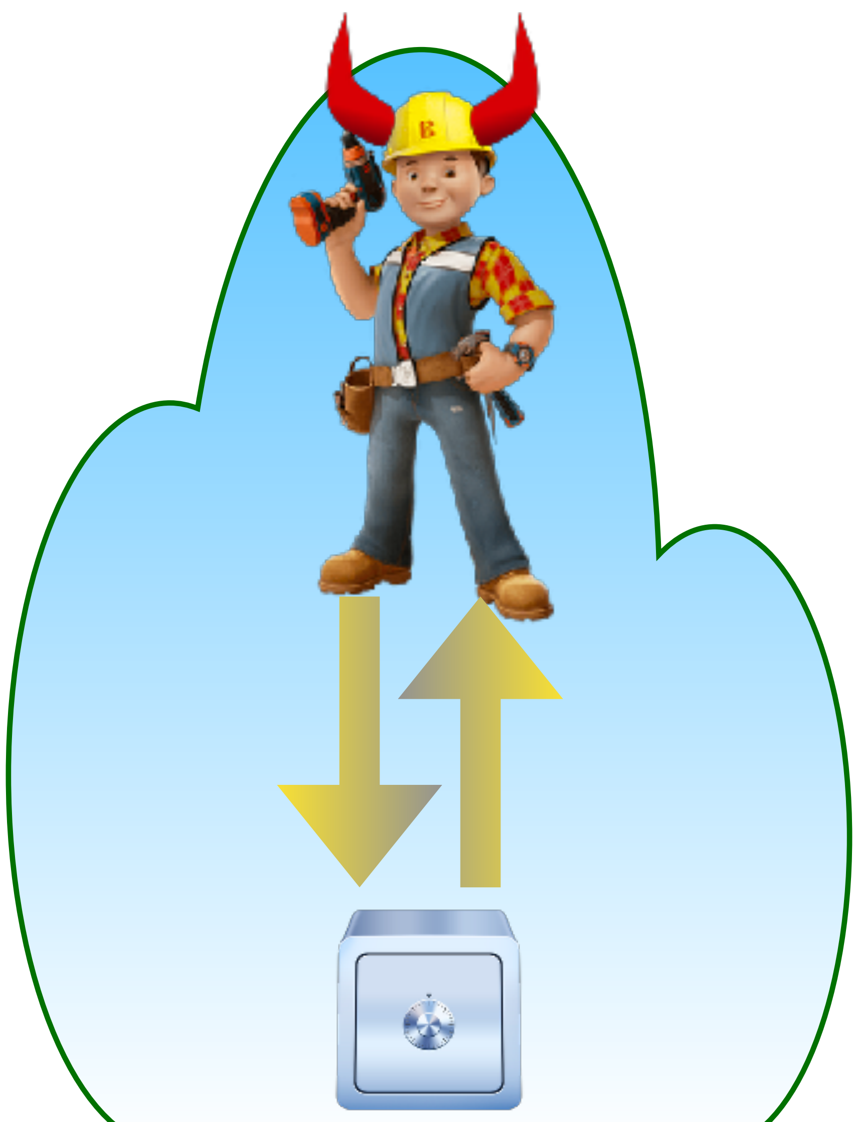
Key Distribution



Key Master

To obtain keys, an enclave attests itself to the key master

Key Distribution



Key Master

To obtain keys, an enclave attests itself to the key master

Key Distribution

Key Master

To obtain keys, an enclave attests itself to the key master

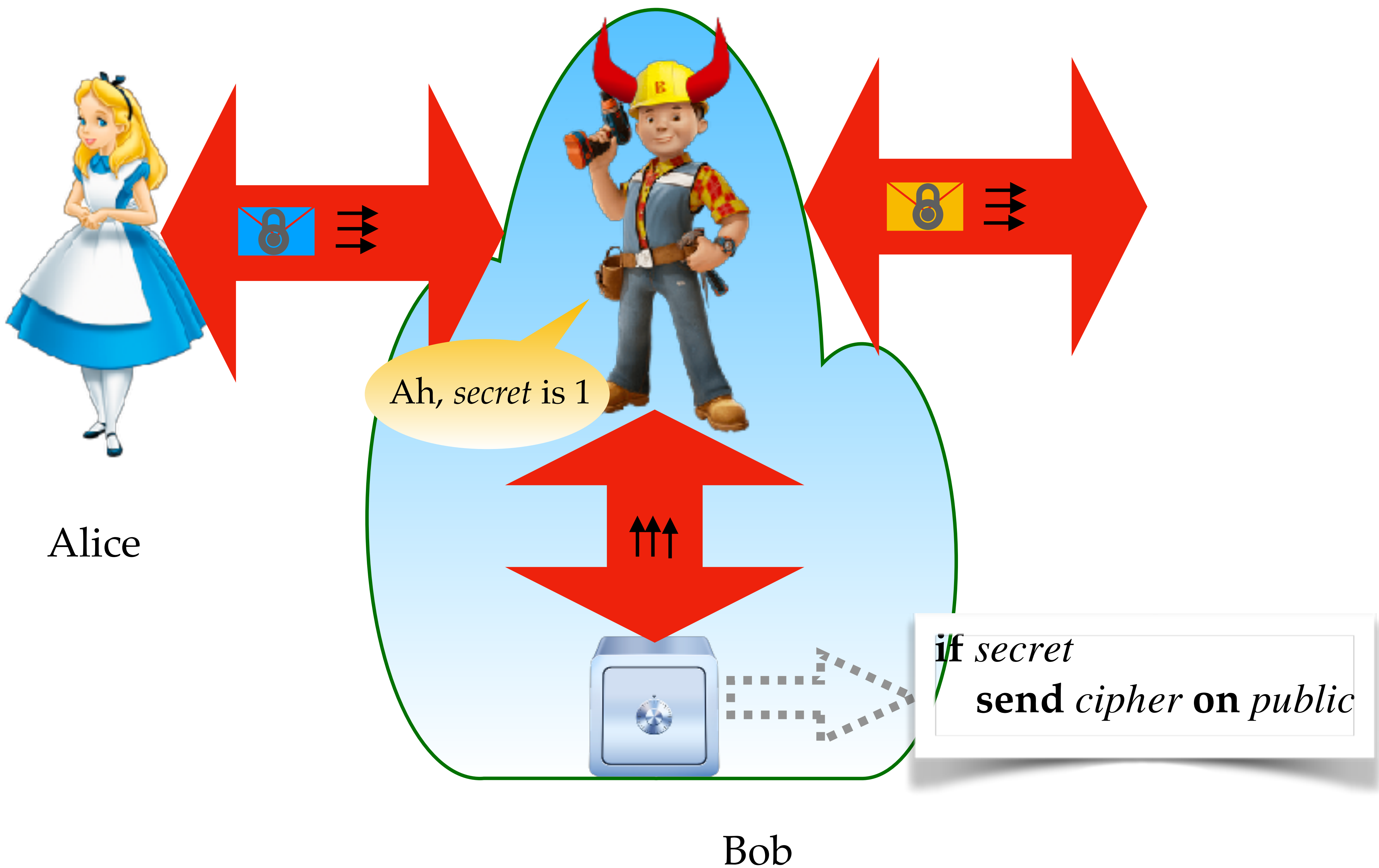
Address Challenge #2

1. Choose *right* abstractions for crypto and TEEs
    - Abstractions ***should reflect*** the capabilities and limitations of TEEs
        - e.g. TEE can only communicate with host
    - Focus on application-level security
    - Implementable!
2. Enforce security

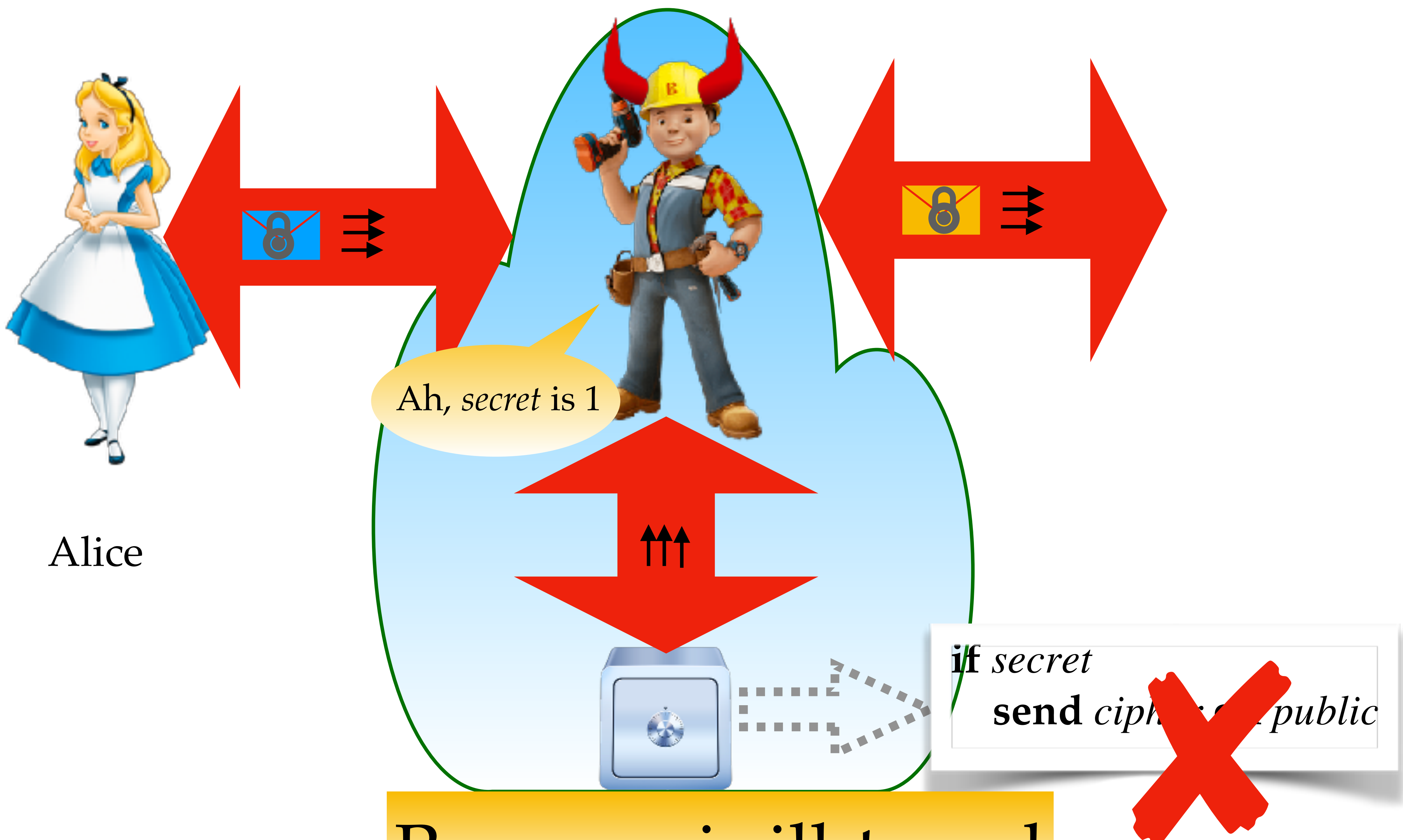
Security

- Formal definition of security is noninterference (NI)
  - Confidentiality NI: private inputs can’t influence public outputs
  - Integrity NI: Low integrity inputs can’t influence high integrity outputs
- Type system enforces security

Example Revisited

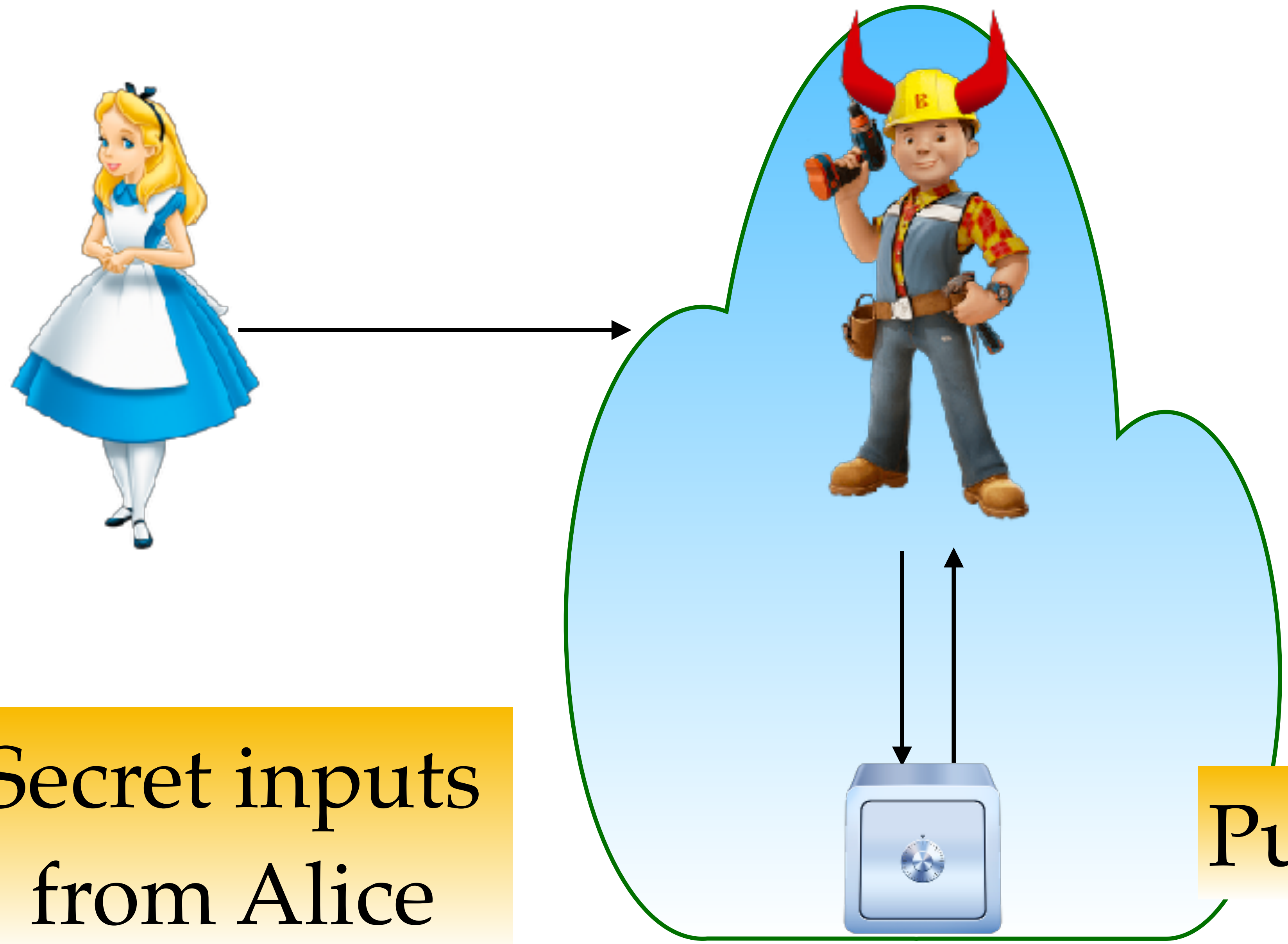


Example Revisited



Program is ill-typed

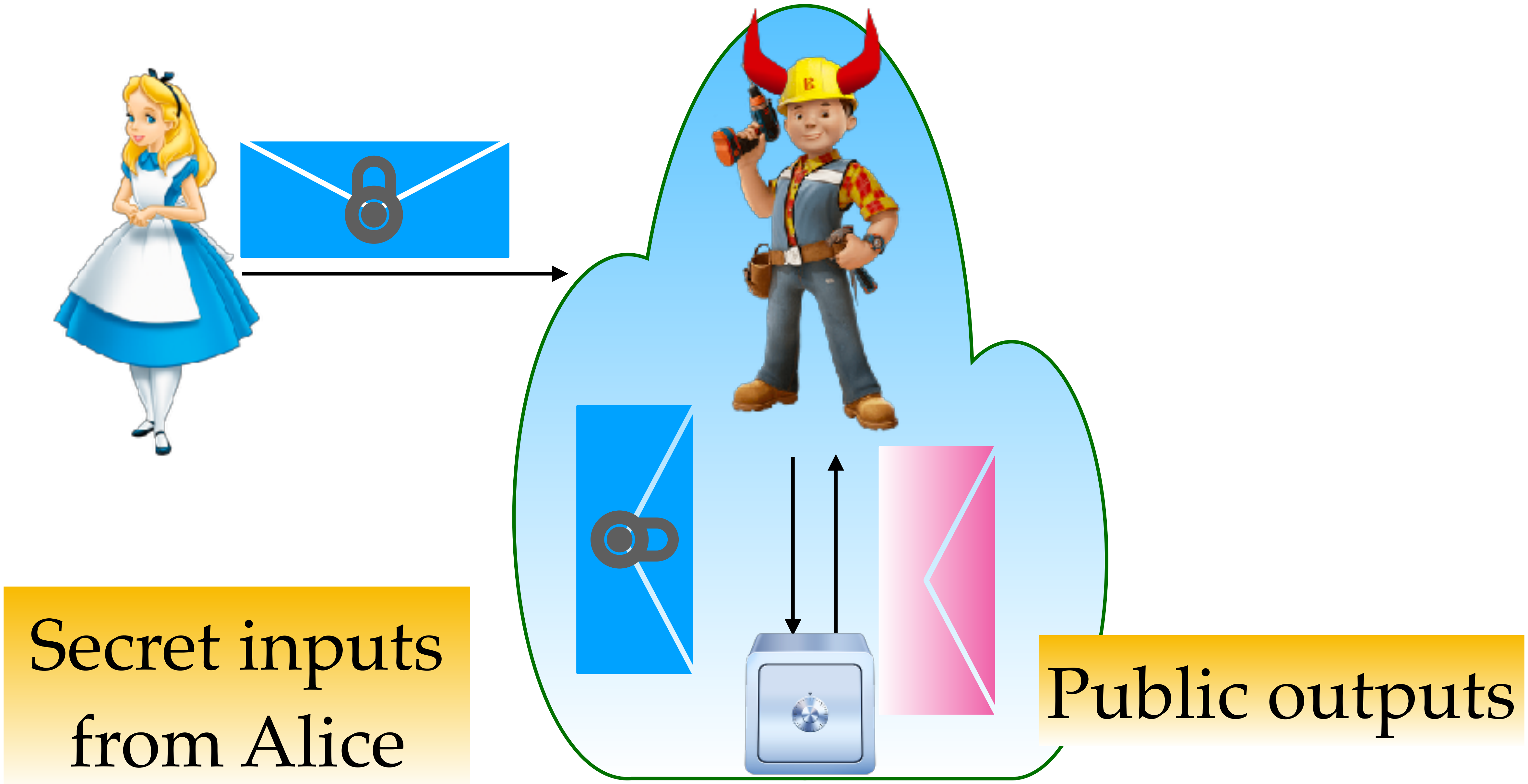
Confidentiality Theorem



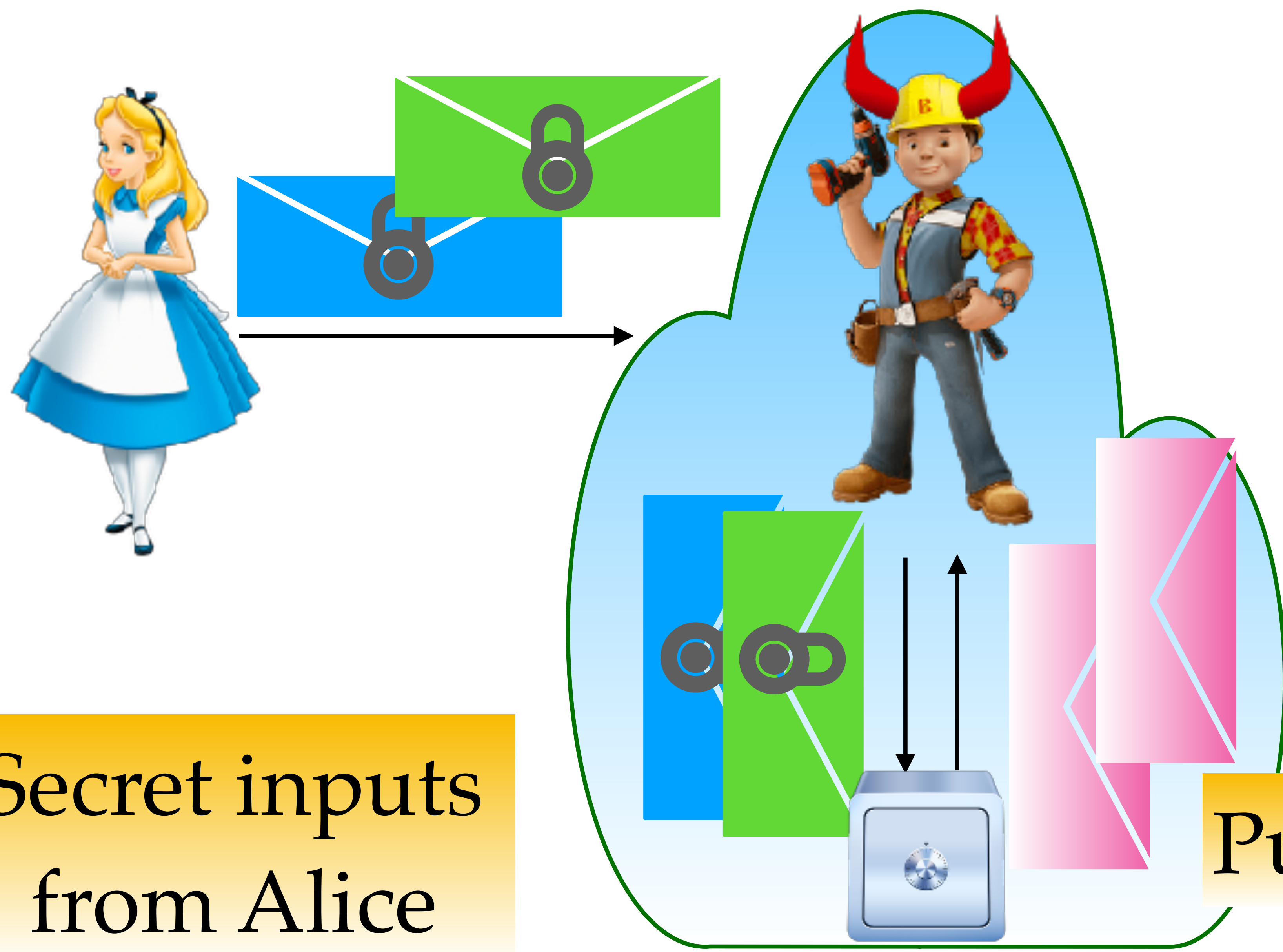
Secret inputs from Alice

Public outputs

Confidentiality Theorem



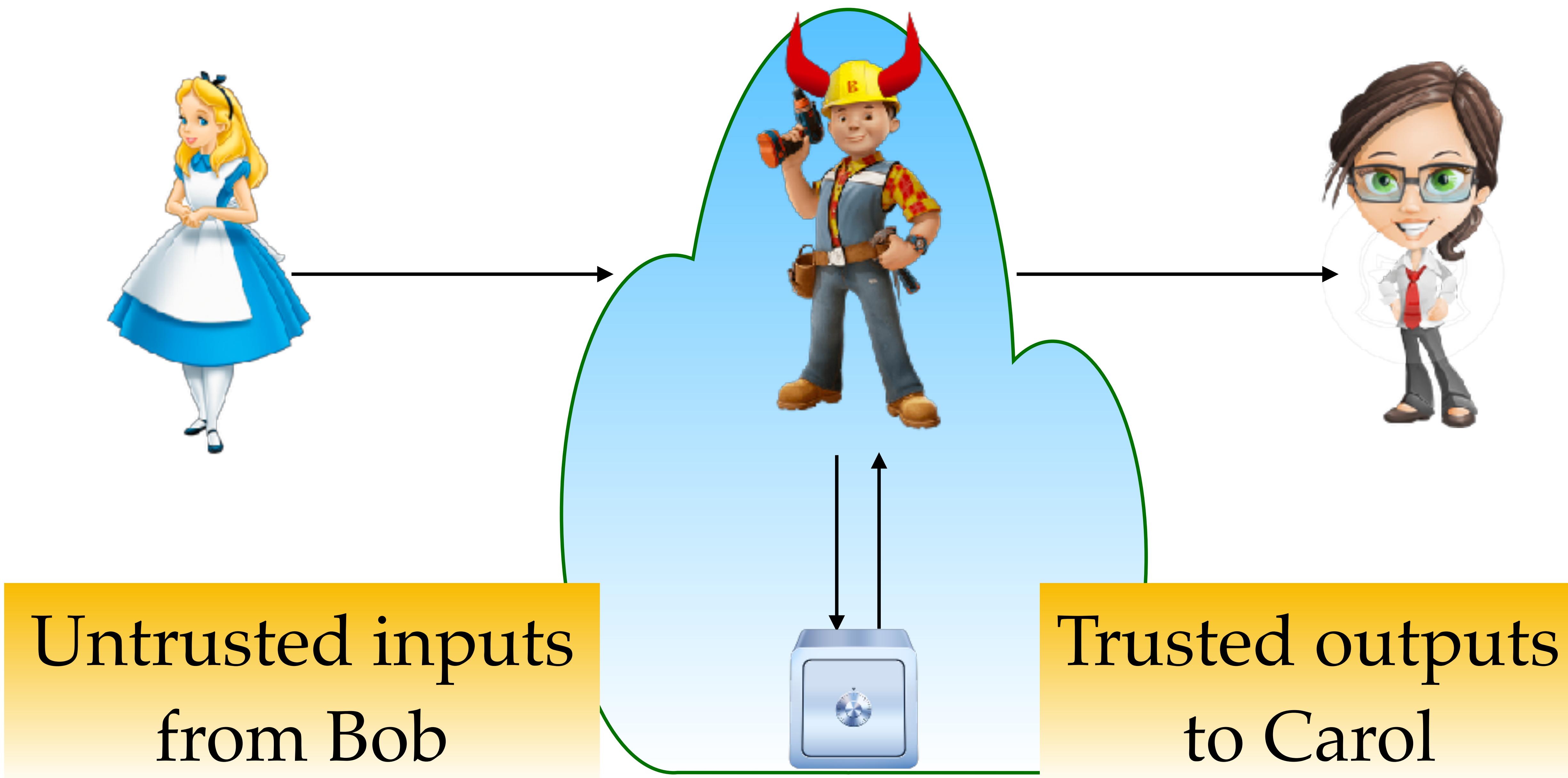
Confidentiality Theorem



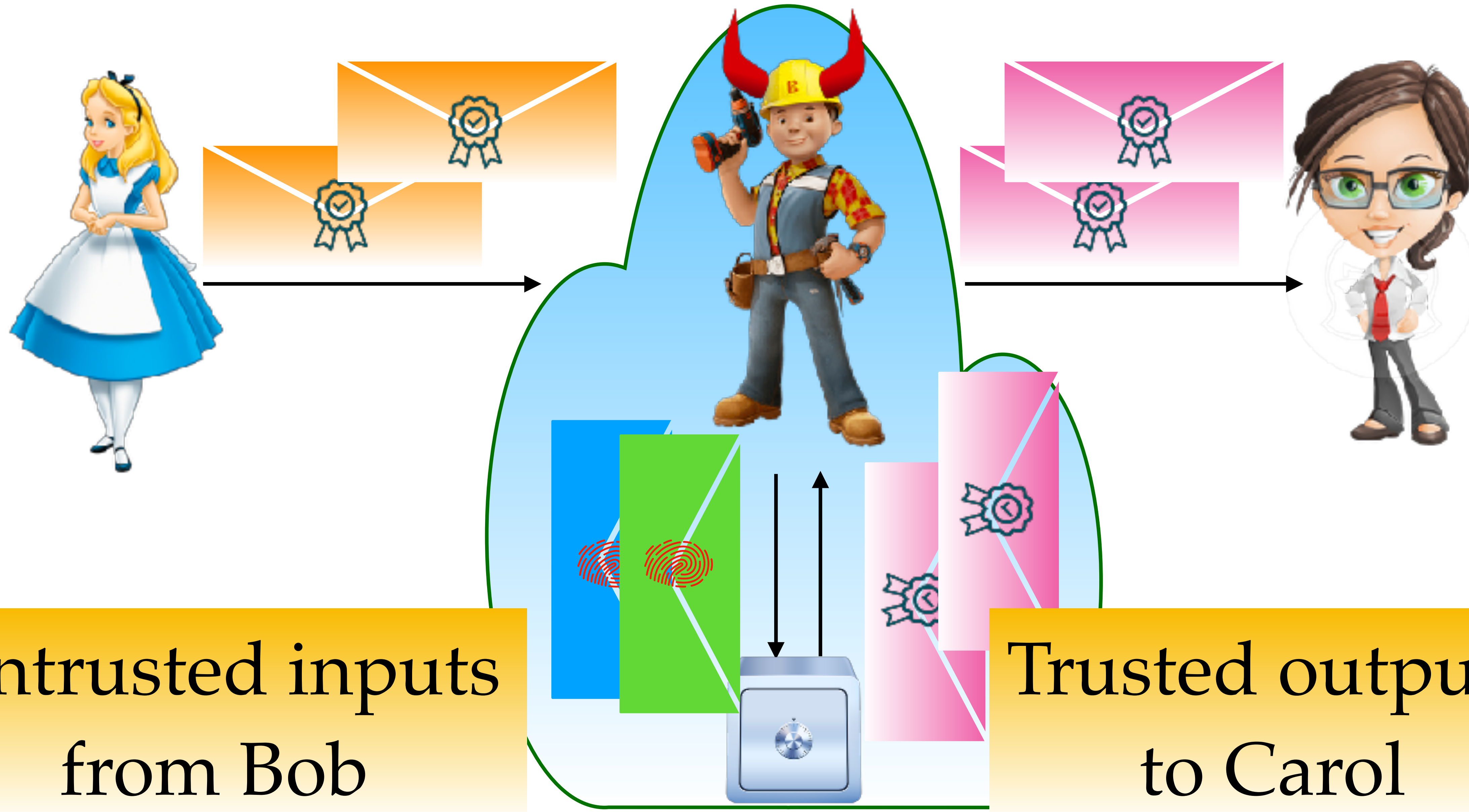
Secret inputs from Alice

Public outputs

Integrity Theorem



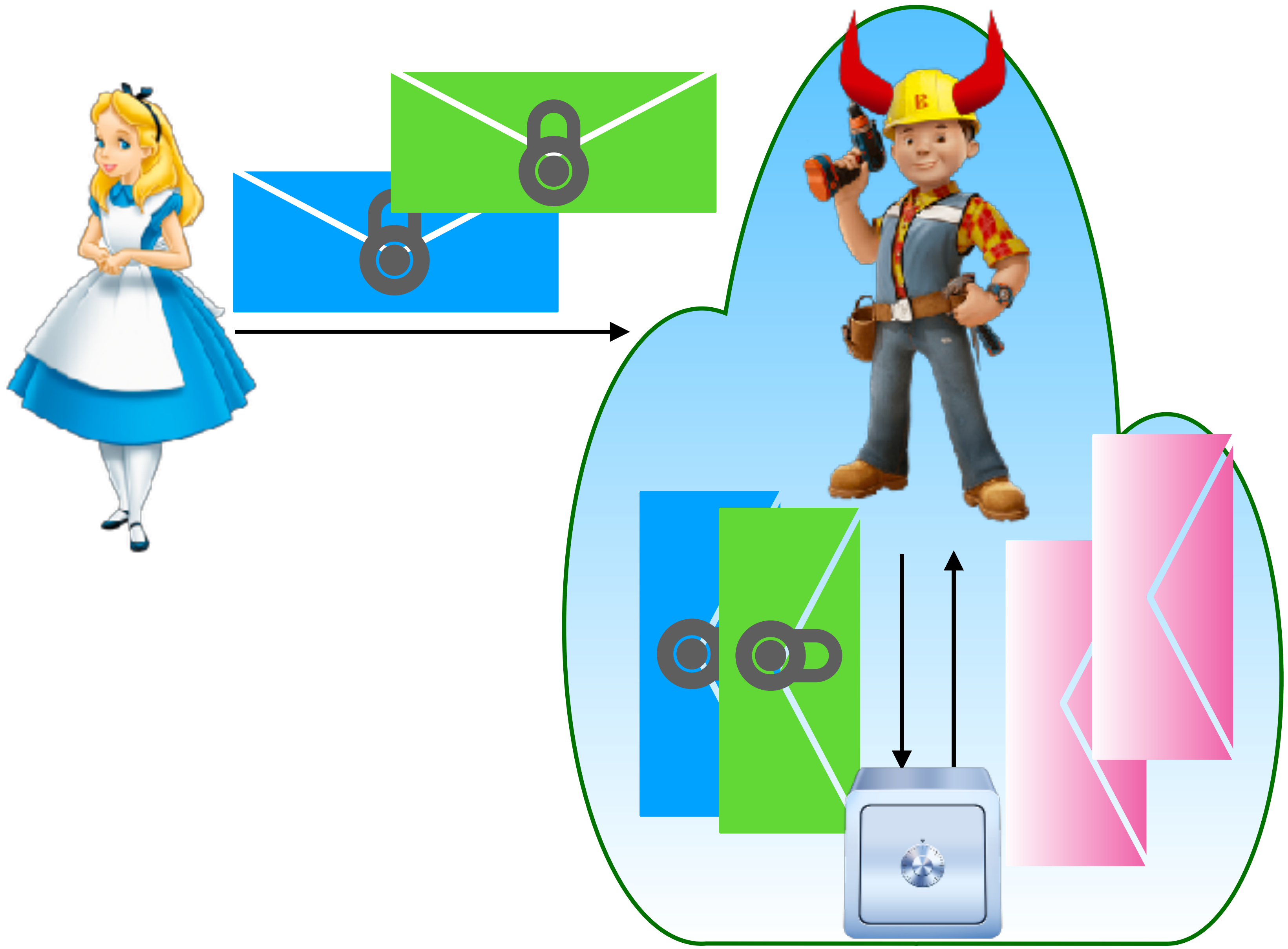
Integrity Theorem



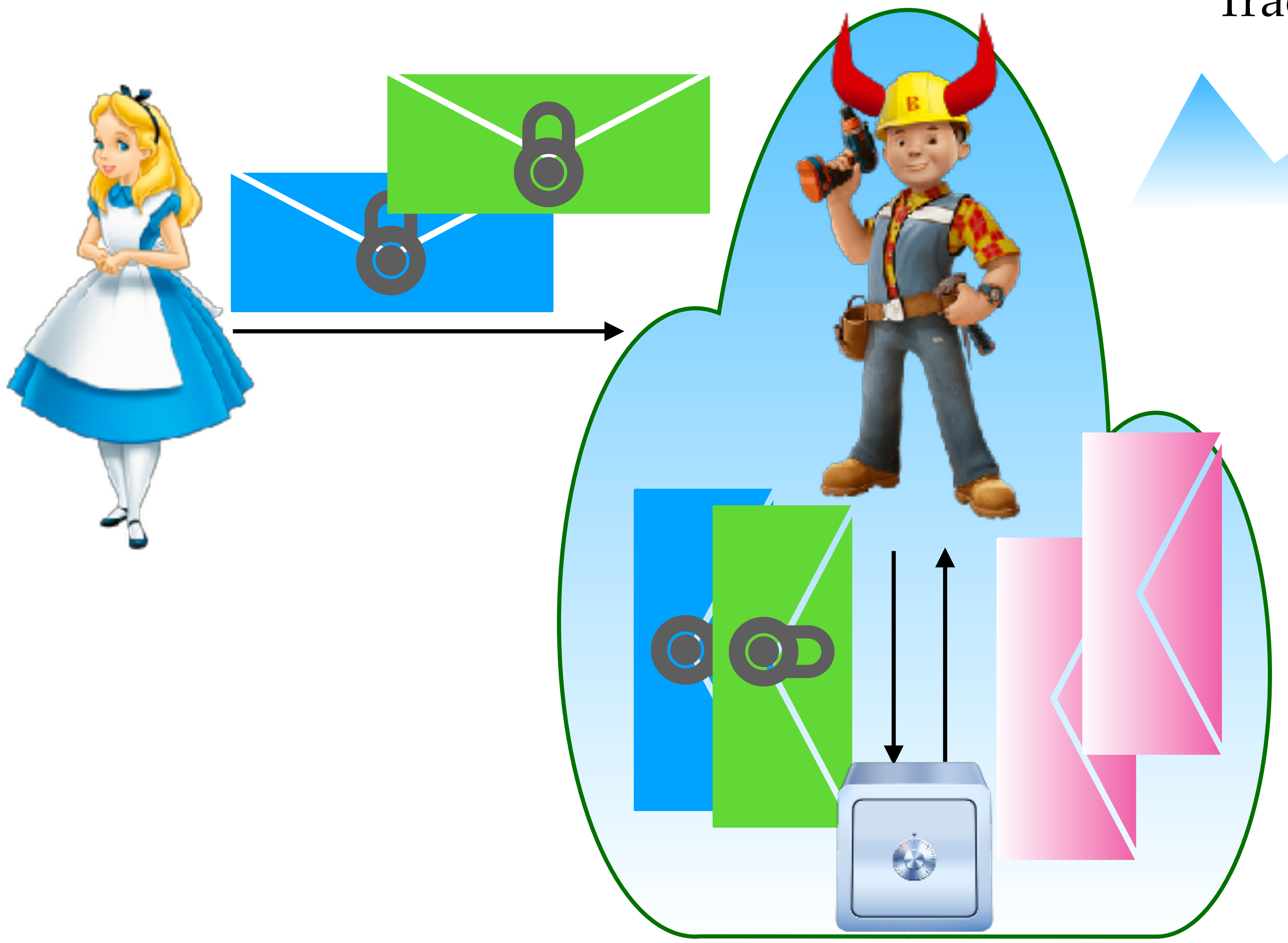
Untrusted inputs from Bob

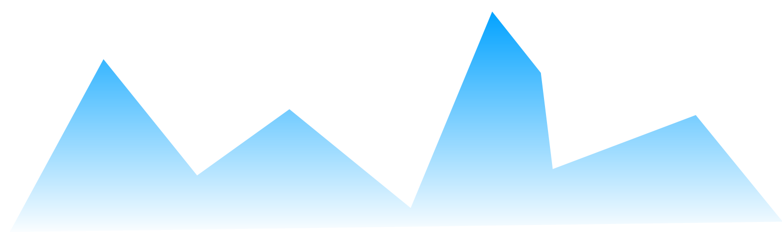
Trusted outputs to Carol

Compromised-node Noninterference

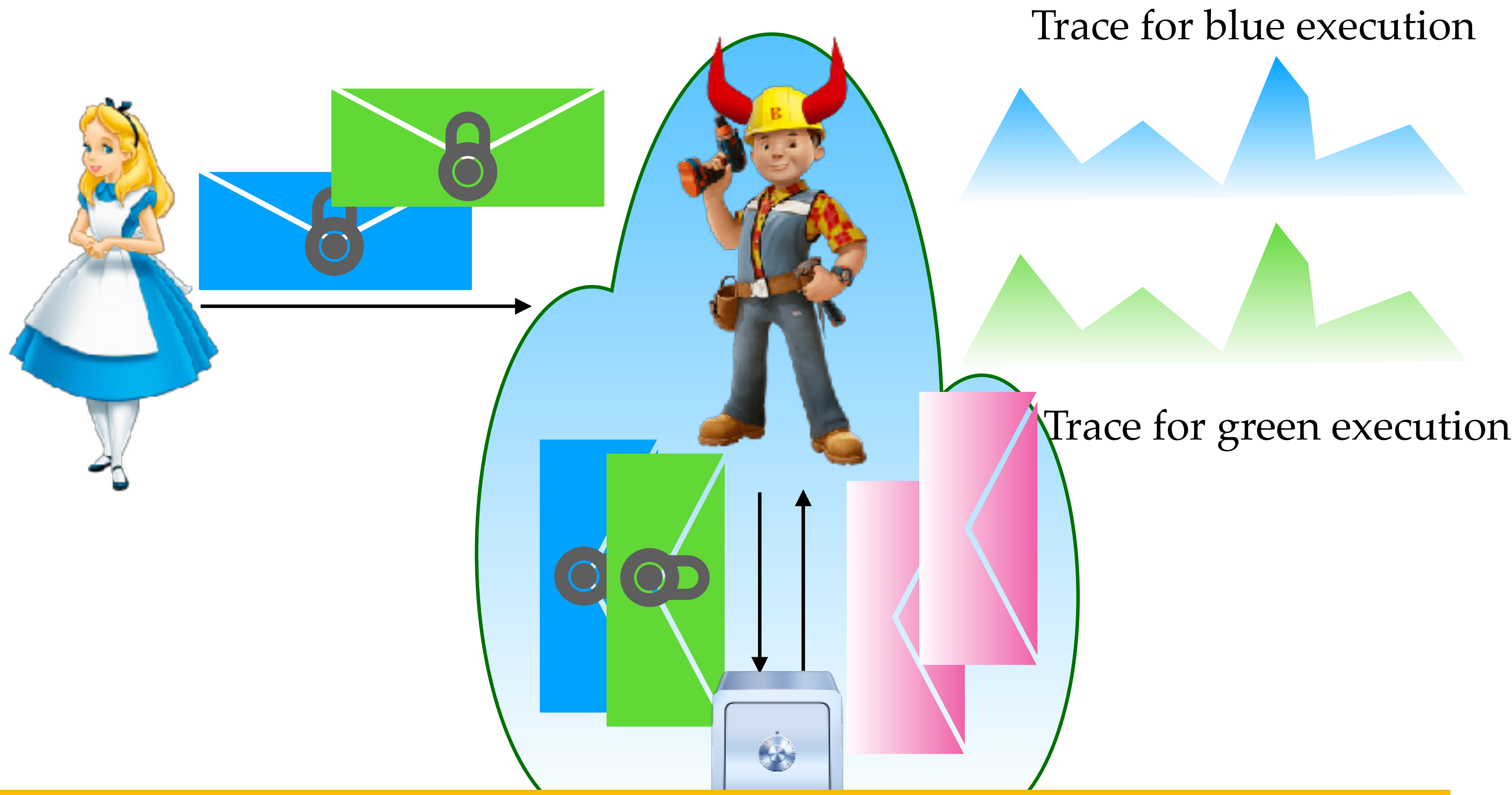


Compromised-node Noninterference



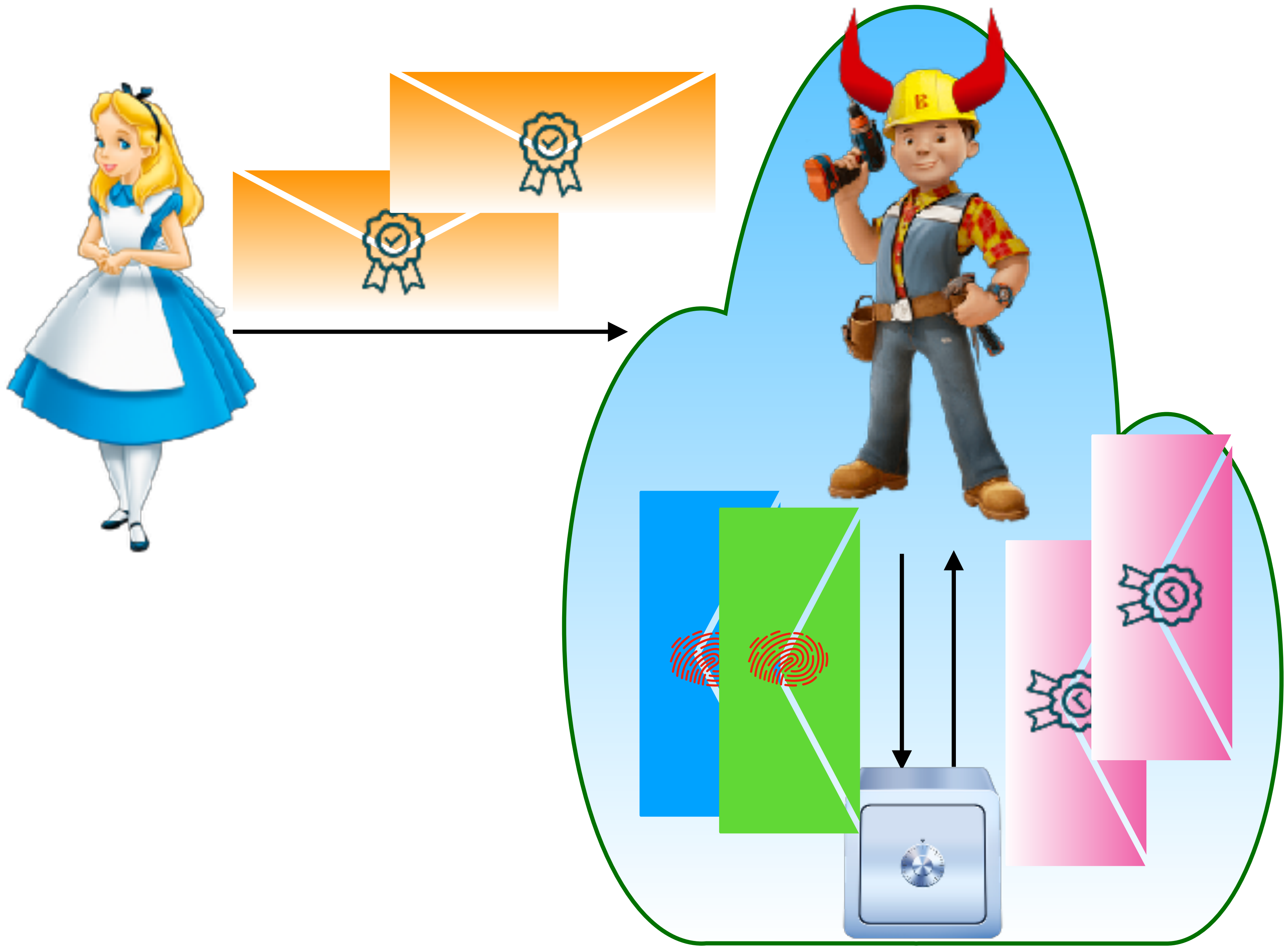


Compromised-node Noninterference

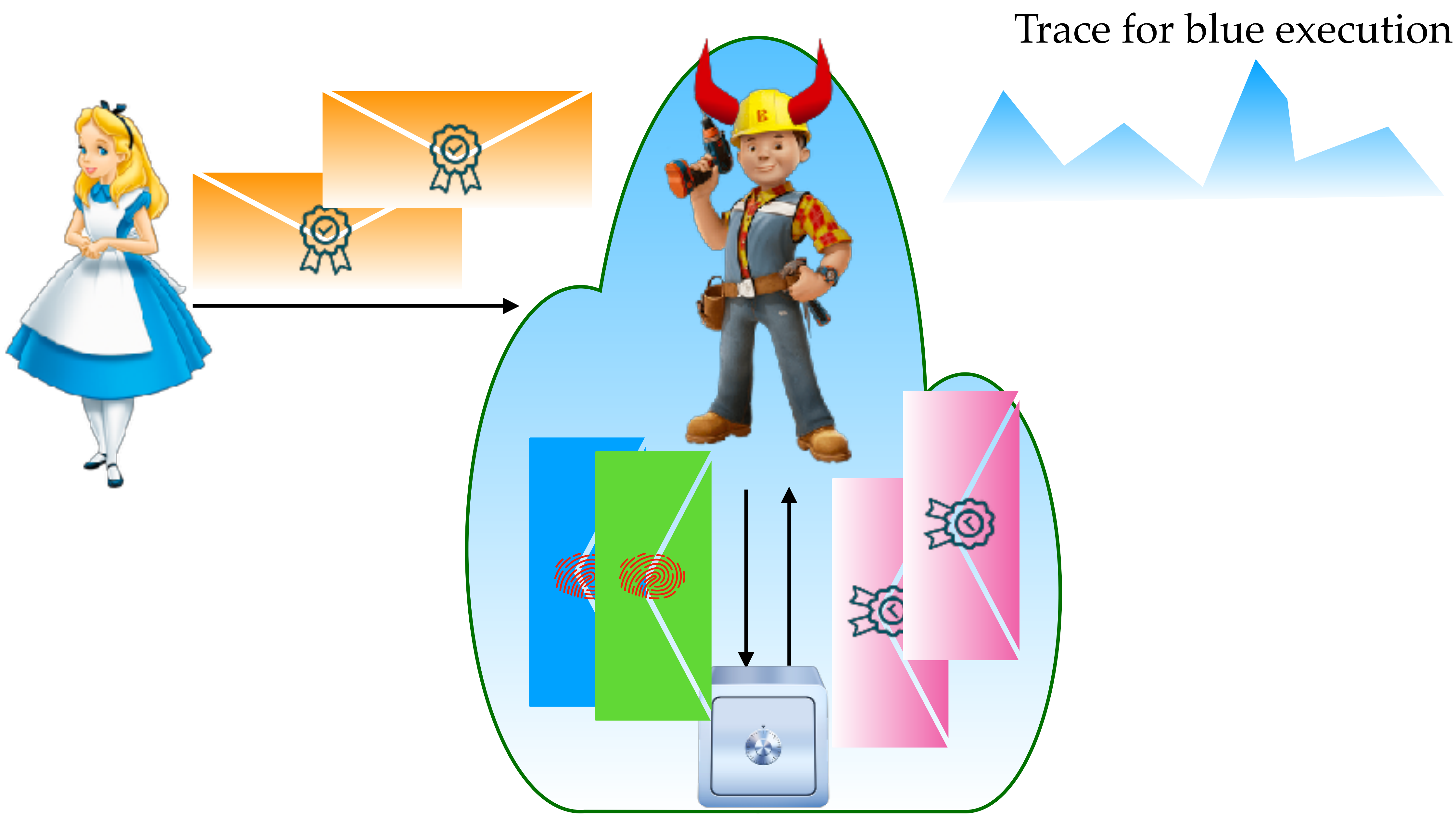


Traces observed (by Bob) for executions with different secret inputs are equal

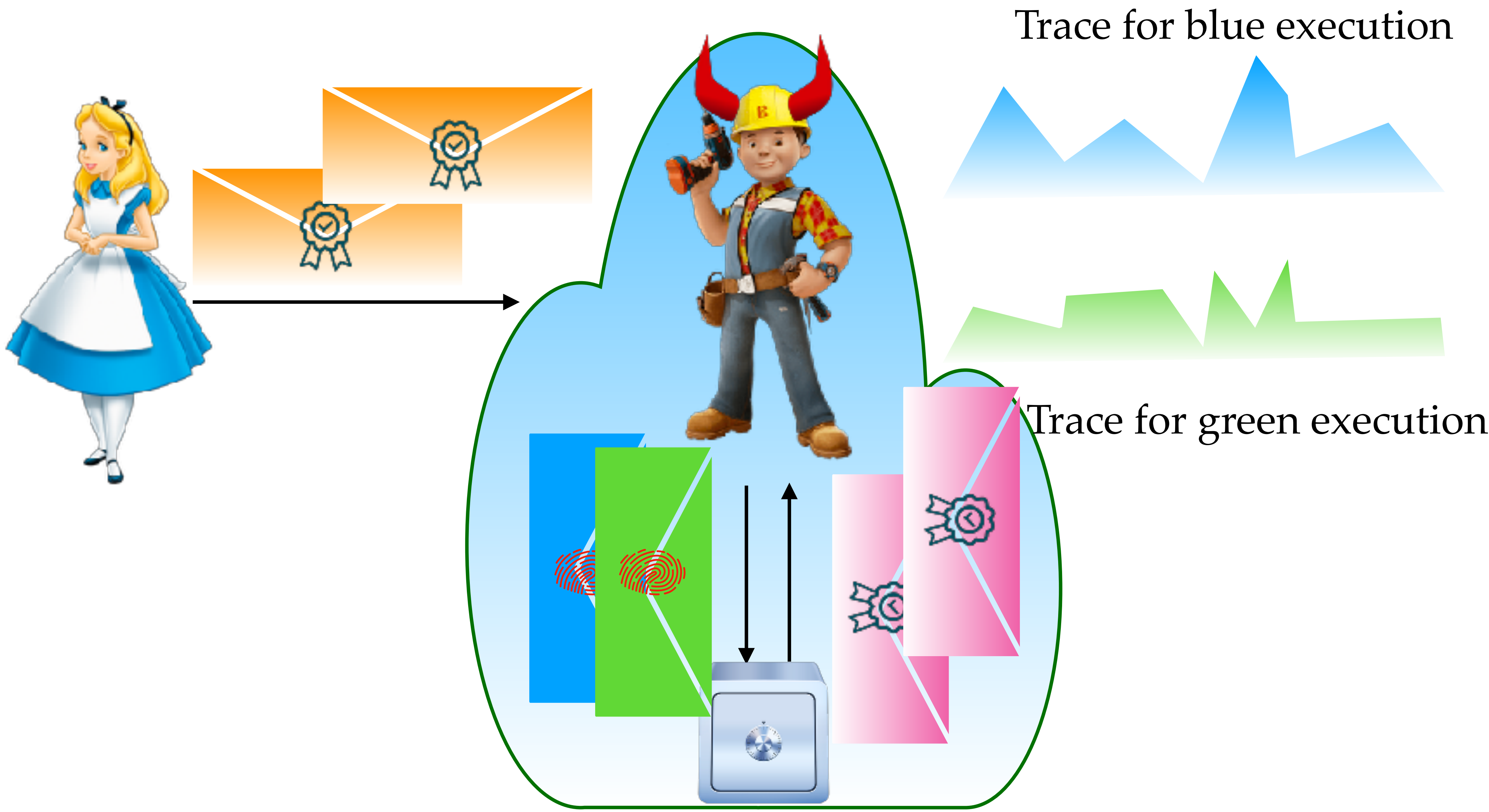
Compromised-node Noninterference



Compromised-node Noninterference



Compromised-node Noninterference



Traces observed can be different

Confidentiality vs Integrity Guarantees

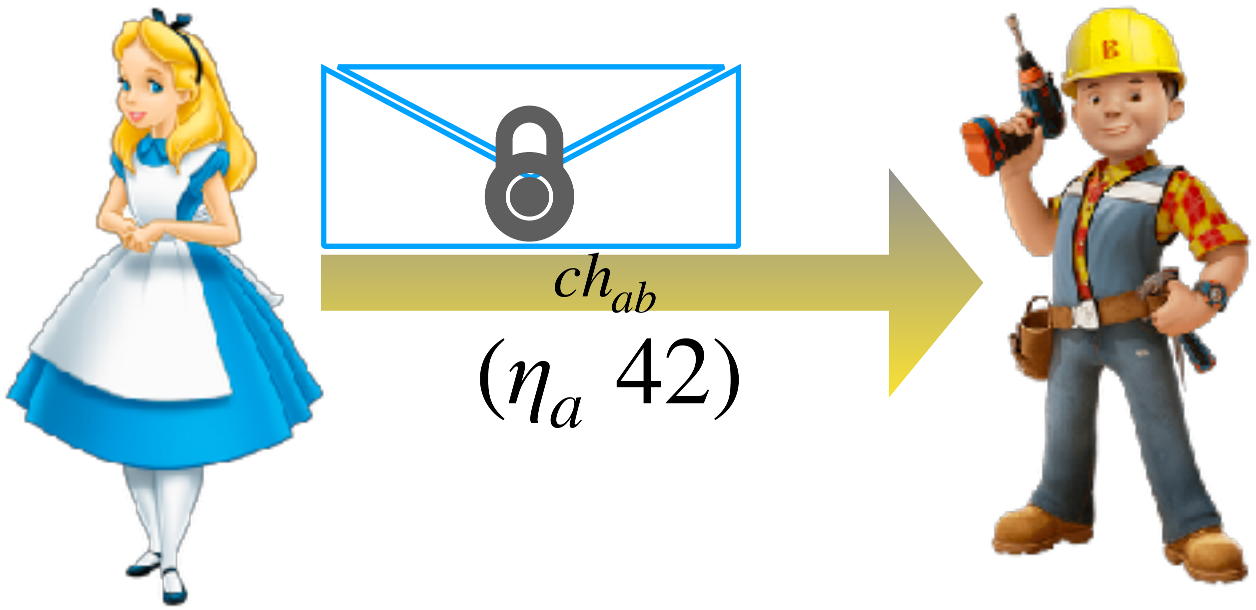
- Asymmetry due to the ability to suppress messages
- Faithfully models the expressive power of the integrity attacker
  - Without undermining the guarantees of cryptography and TEEs

Conclusion

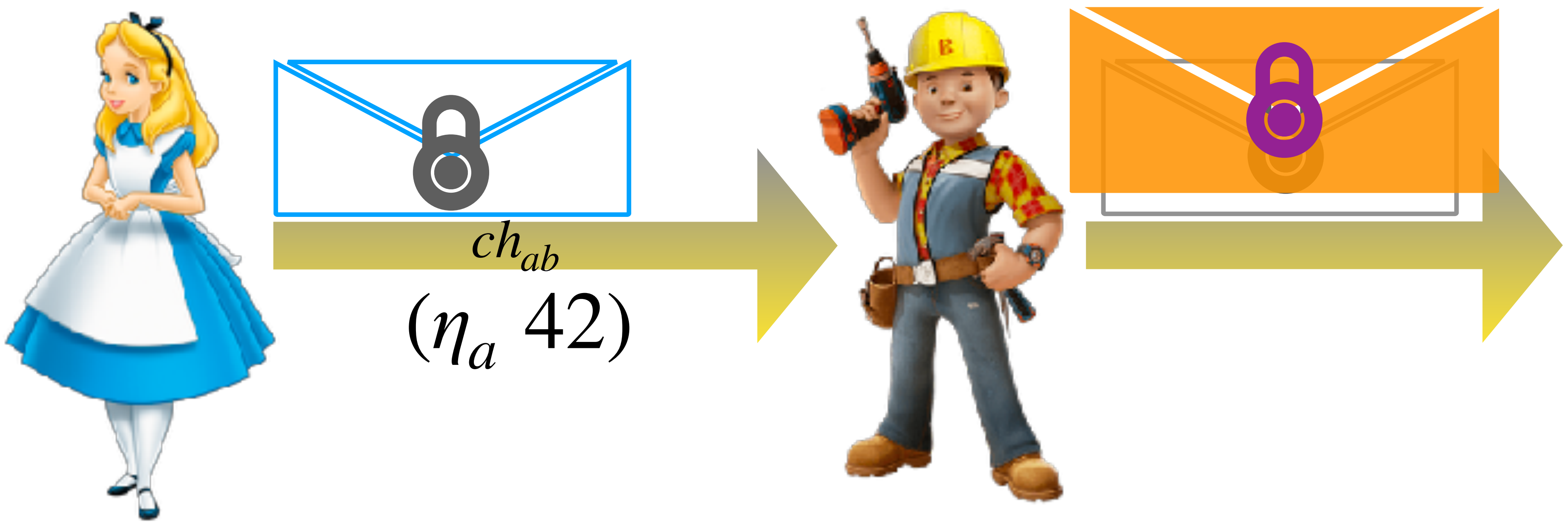
- DFLATE: A programming model for distributed TEEs
- Design for implementing the abstractions in DFLATE
- DFLATE enforces confidentiality and integrity

Backup Slides

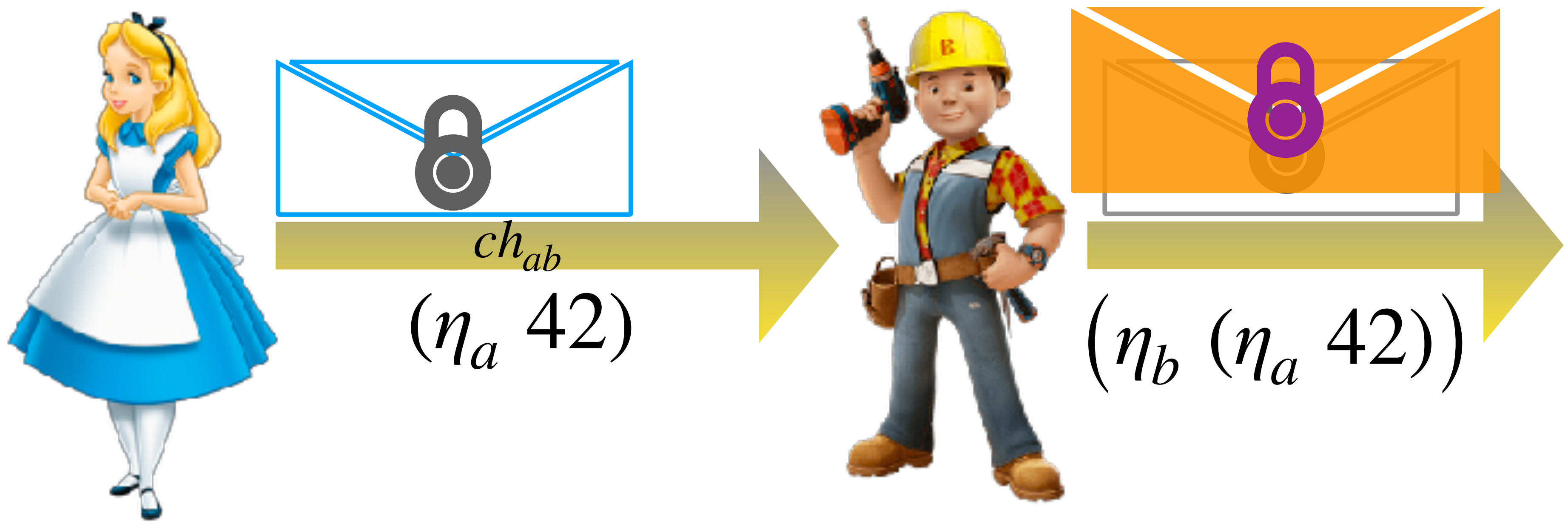
Nested Protection



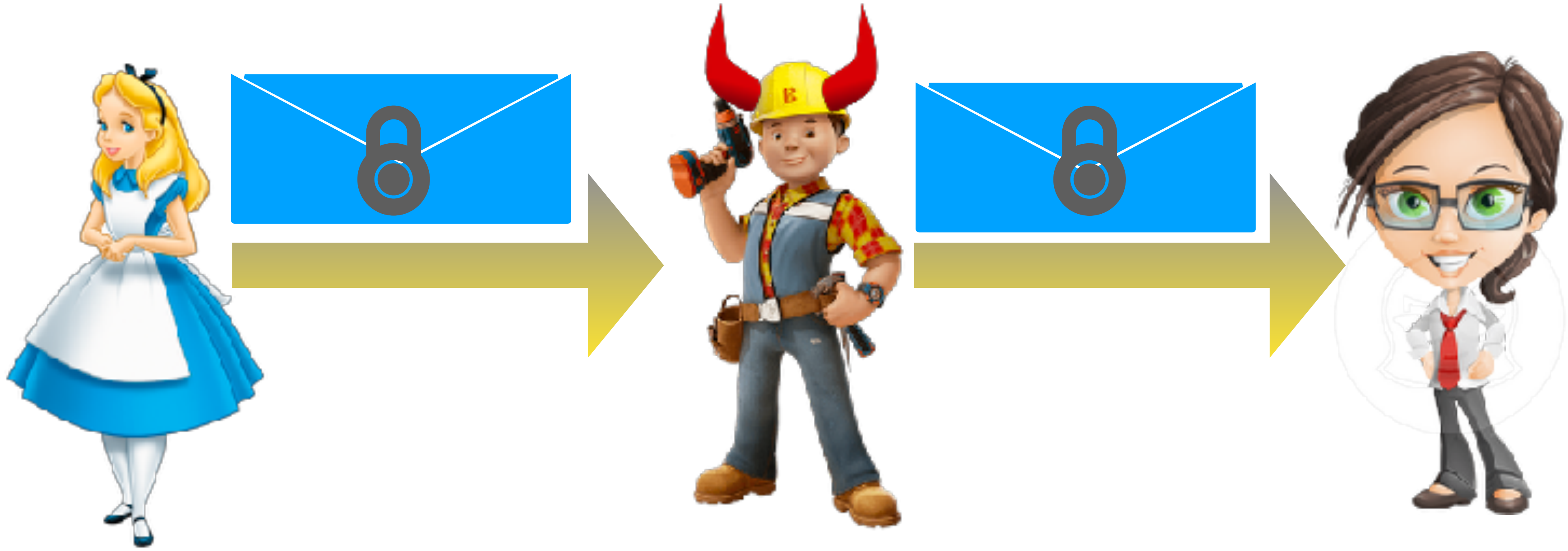
Nested Protection



Nested Protection

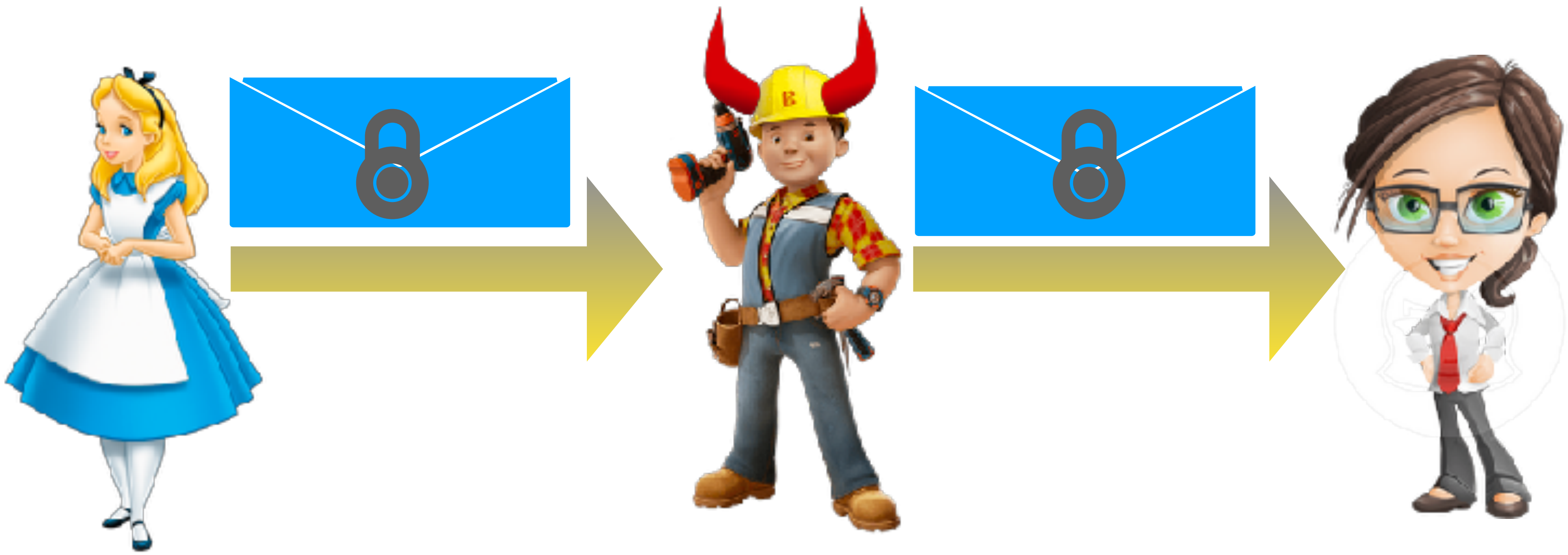


$(\eta_b\ (\eta_a\ 42))$ has type **b says a says int**



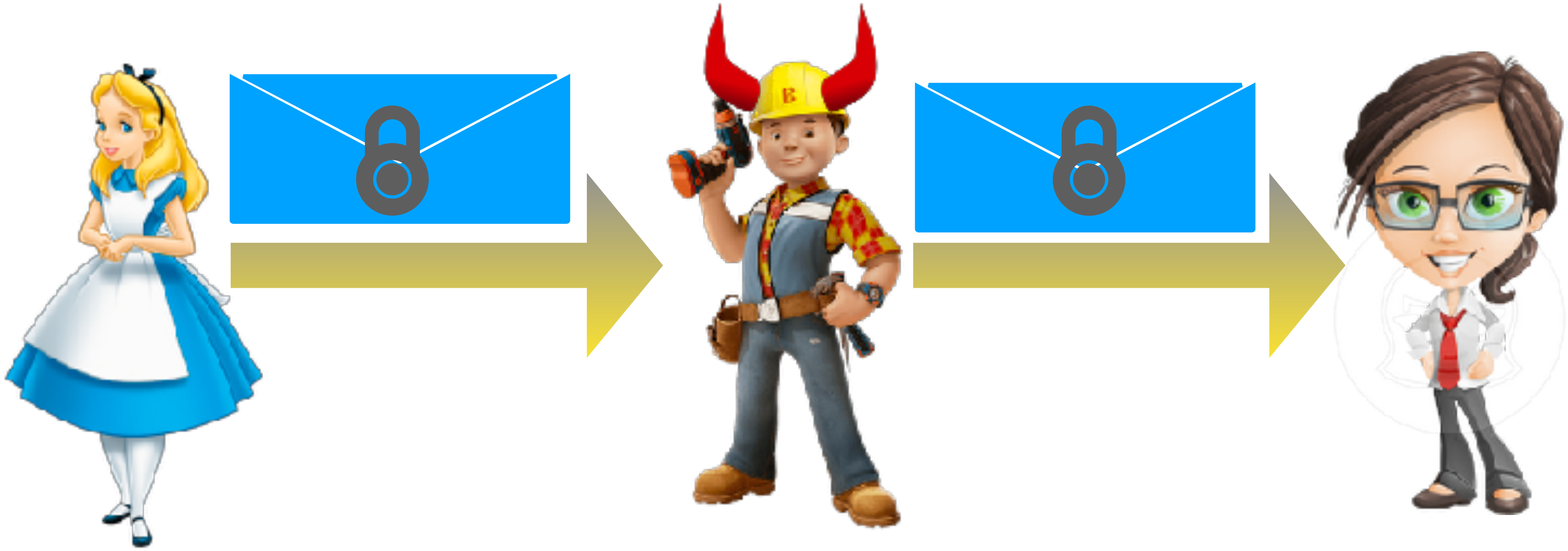
send ch_{ab} $(\eta_a\ m_{blue})$

recv ch_{ab} **as** x **in**
send ch_{bc} x



$\textbf{send}\ ch_{ab}\ (\eta_a\ m_{blue})$

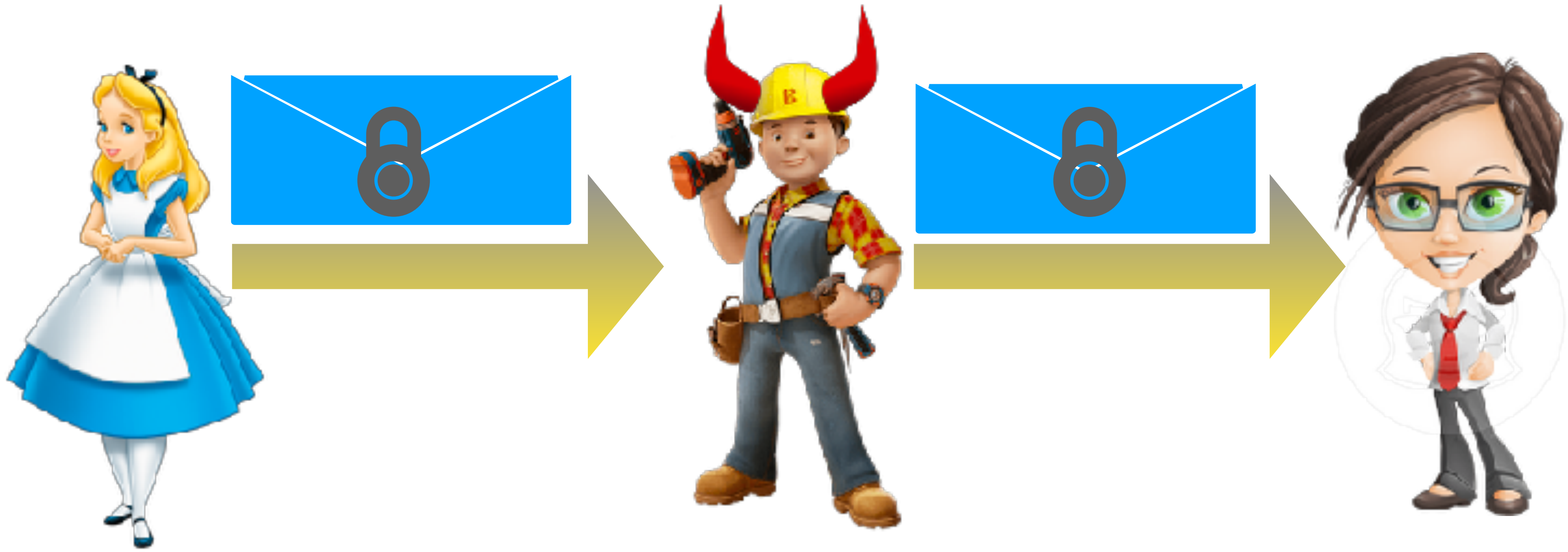
$\textbf{assume}\ b \succcurlyeq a\ \textbf{in}$
$\textbf{recv}\ ch_{ab}\ \textbf{as}\ x\ \textbf{in}$
$\textbf{send}\ ch_{bc}\ x$



send ch_{ab} $(\eta_a\ m_{blue})$

assume $b \geqslant a$ **in**
recv ch_{ab} **as** x **in**
send ch_{bc} x

Malicious declassification

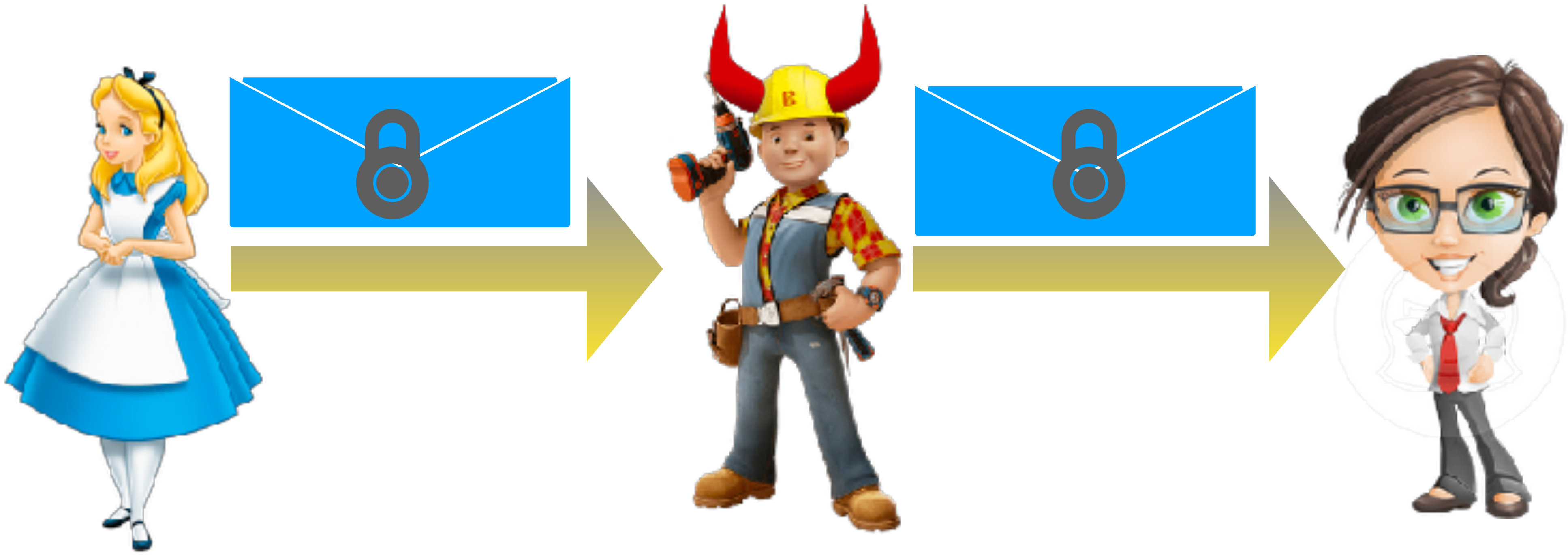


send ch_{ab} $(\eta_a\ m_{blue})$

assume $b \succcurlyeq a$ **in**
recv ch_{ab} **as** x **in**
send ch_{bc} x

Malicious declassification

Type system prevents malicious declassifications and endorsements



send ch_{ab} $(\eta_a\ m_{blue})$

assume $b \geqslant a$ **in**
recv ch_{ab} **as** x **in**
send ch_{bc} x

Insufficient authority to add delegation