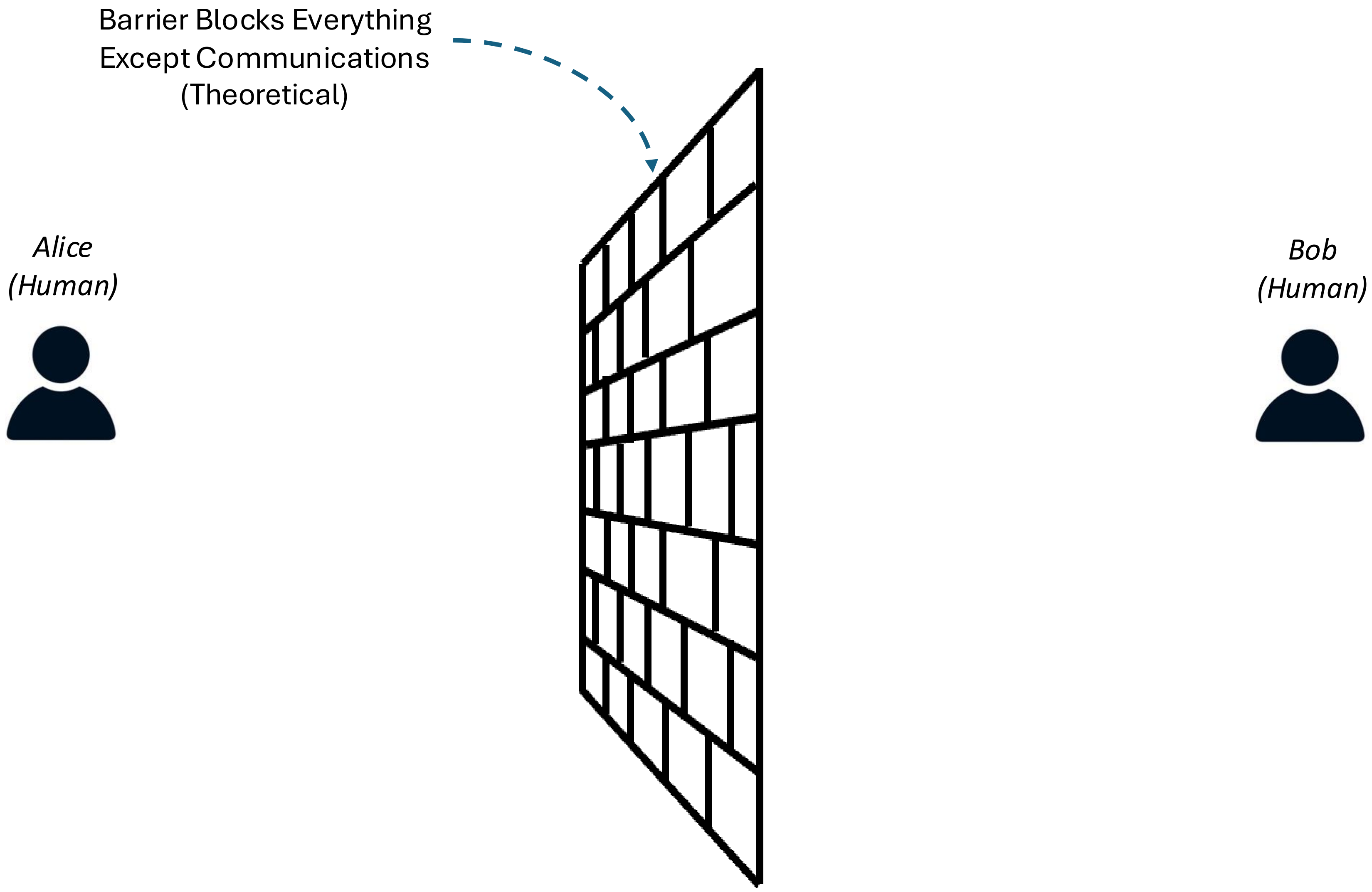
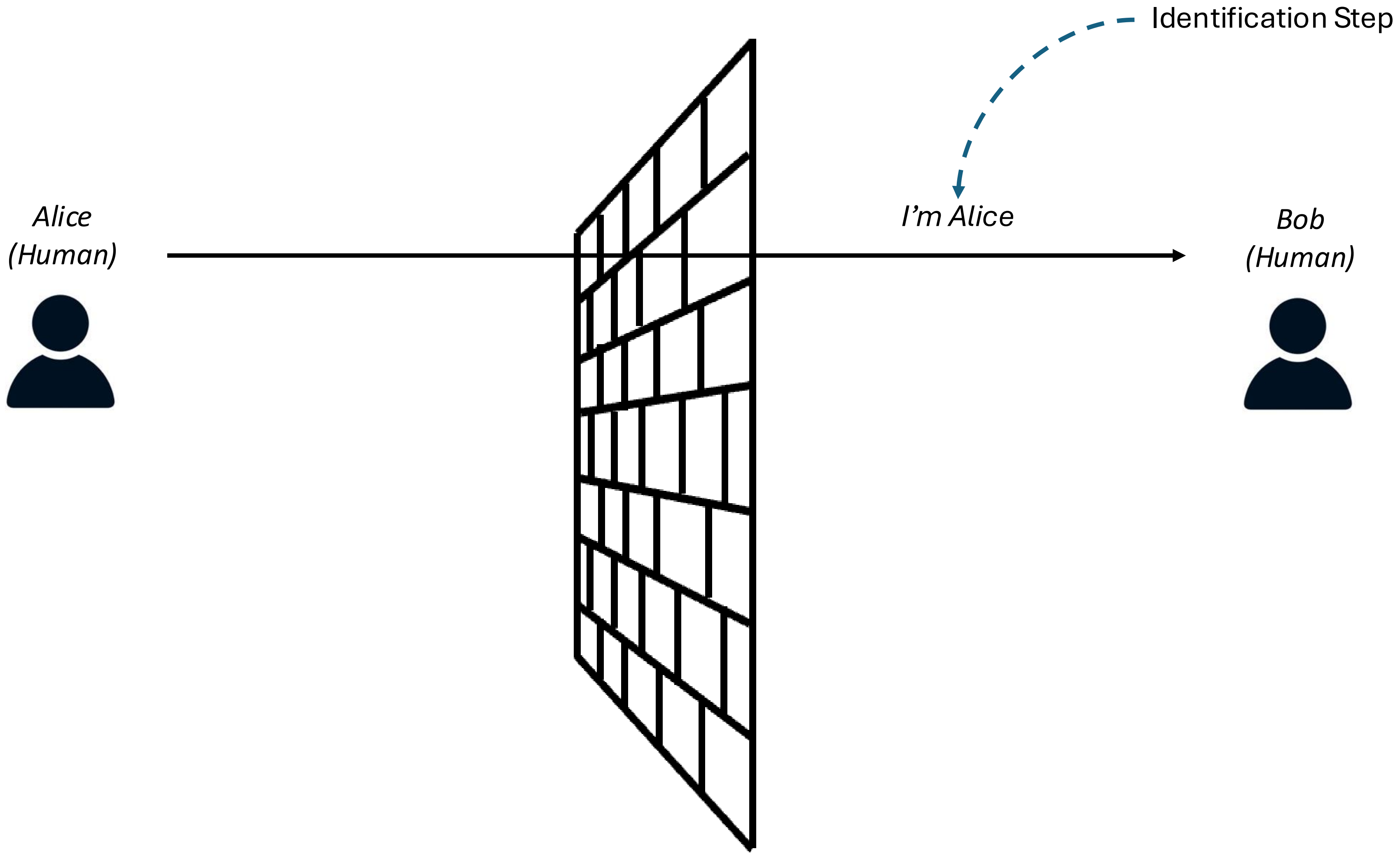
Validating a reported identity is harder than it looks.



Barrier Blocks Everything
Except Communications
(Theoretical)

Alice
(Human)

Bob
(Human)

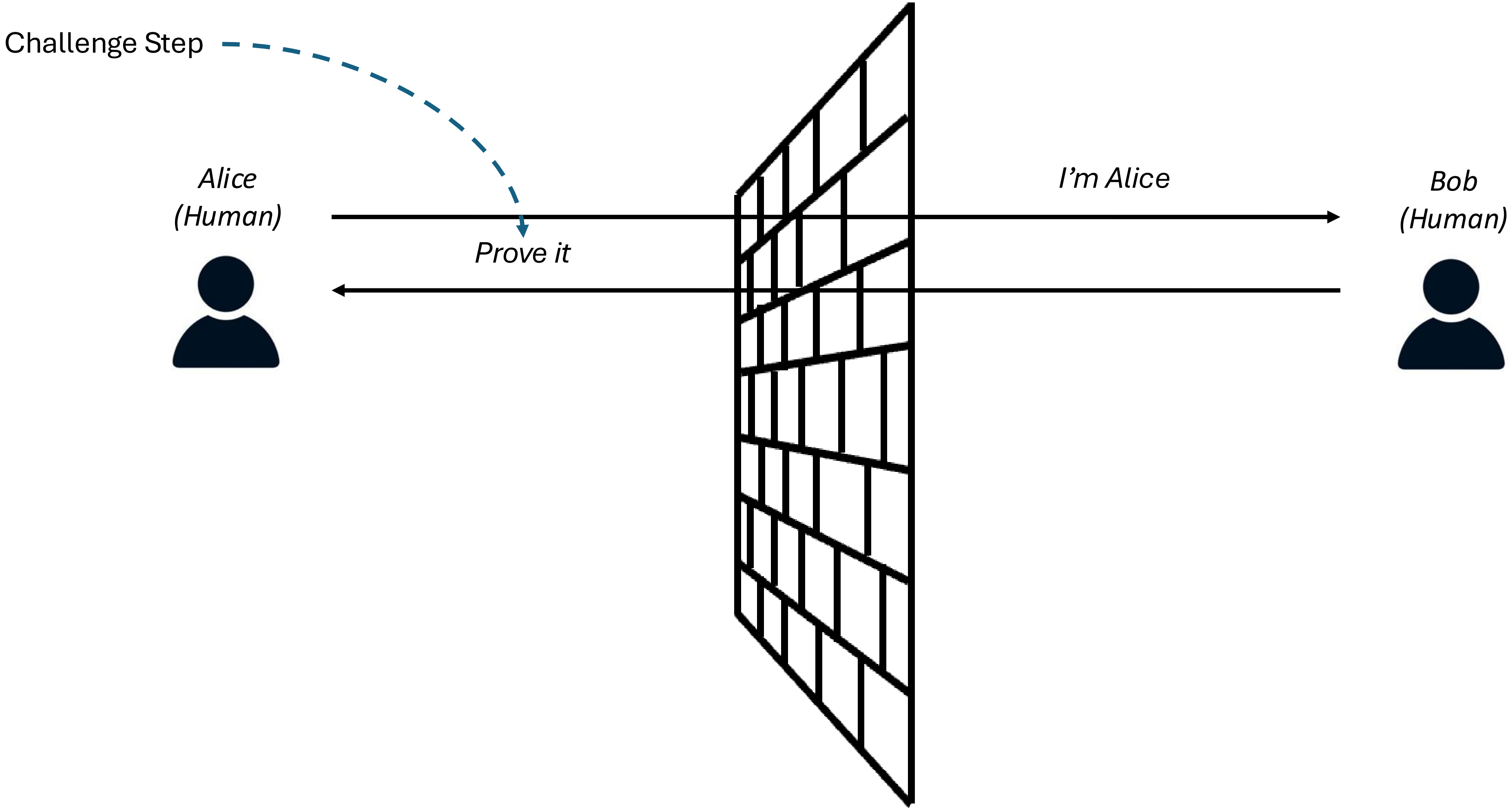


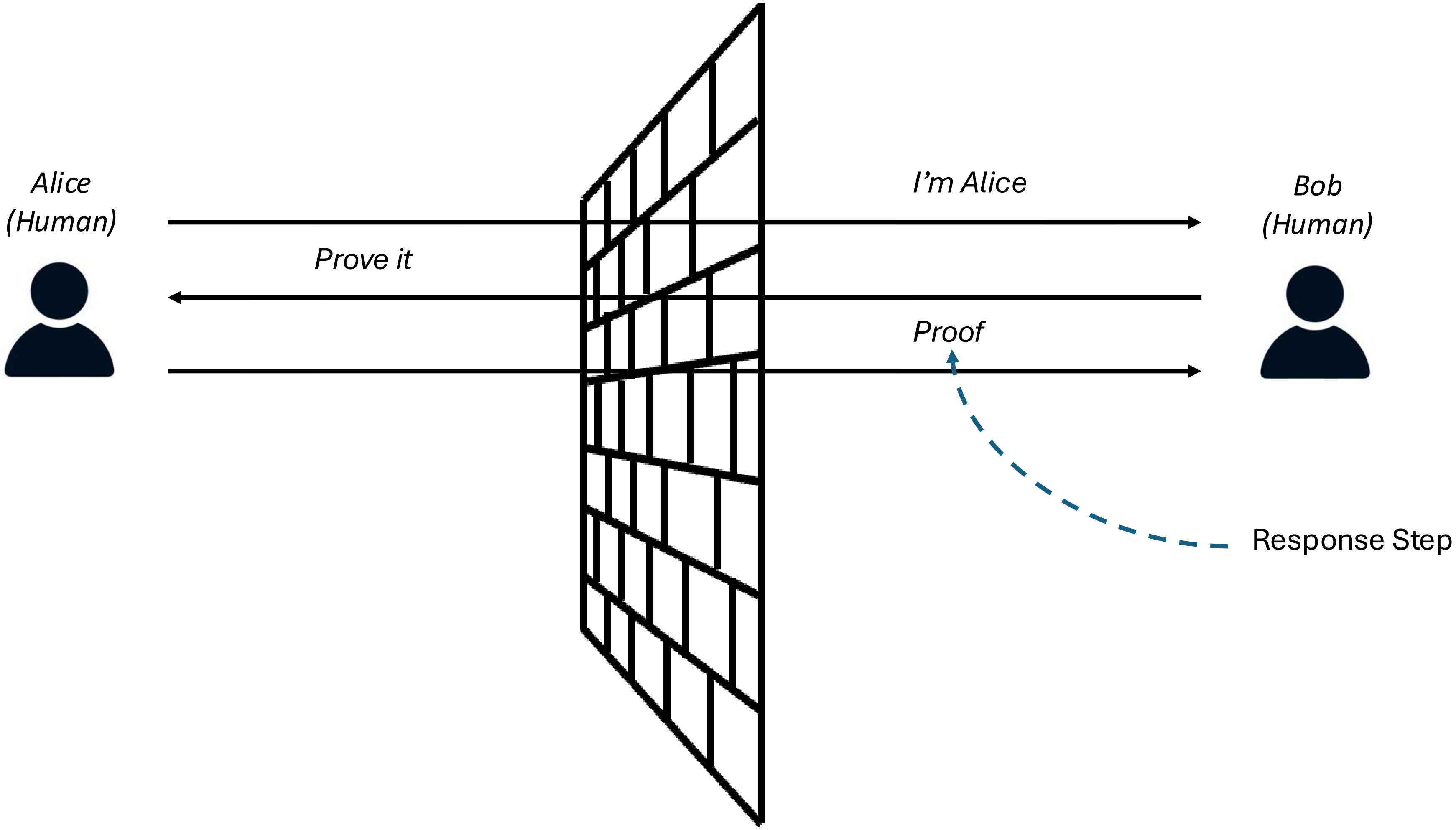
Identification Step

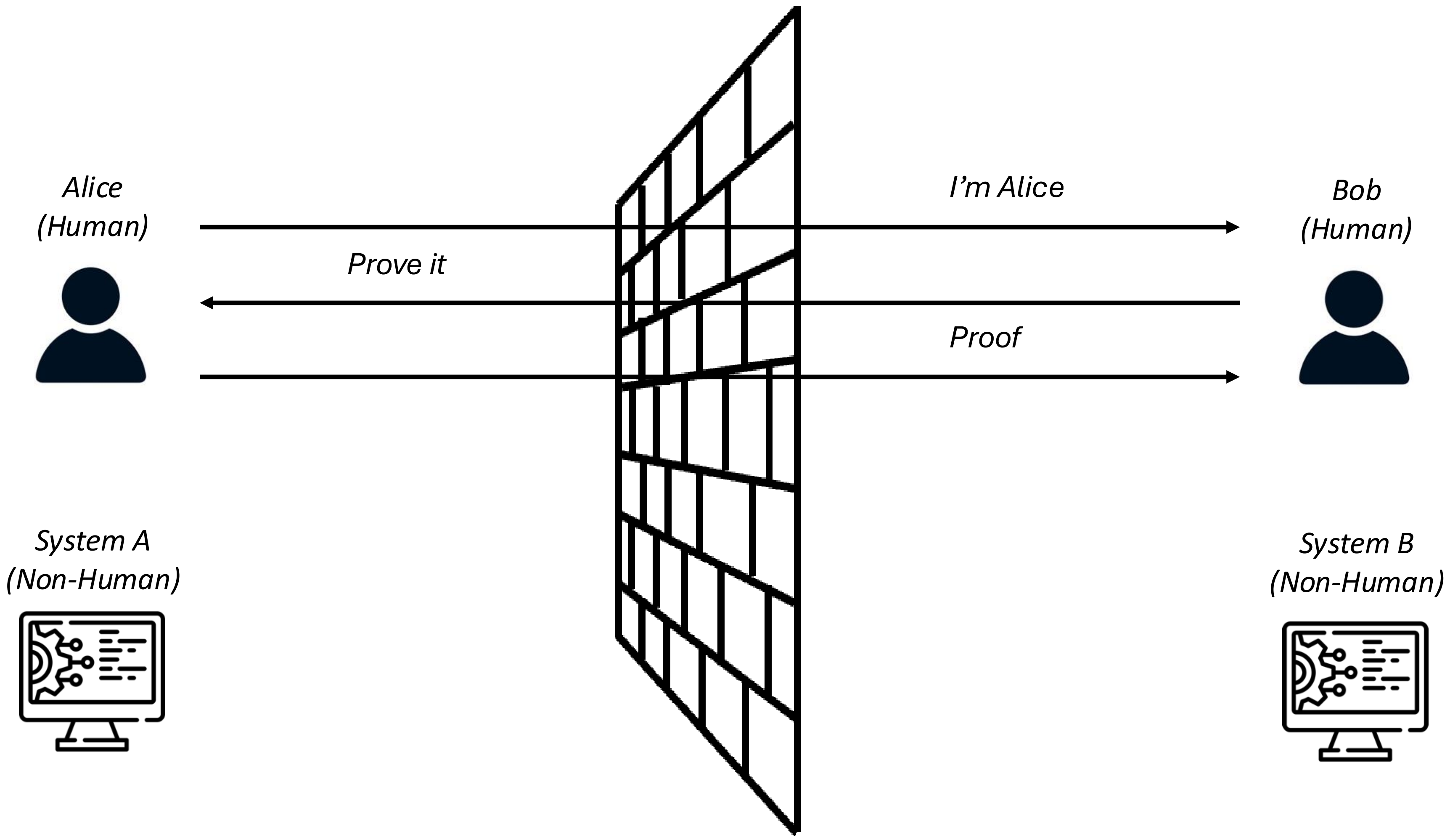
Alice
(Human)

I'm Alice

Bob
(Human)


Challenge Step
Alice
(Human)
I'm Alice
Bob
(Human)
Prove it


Alice
(Human)
I'm Alice
Bob
(Human)
Prove it
Proof
Response Step



Alice
(Human)

I'm Alice

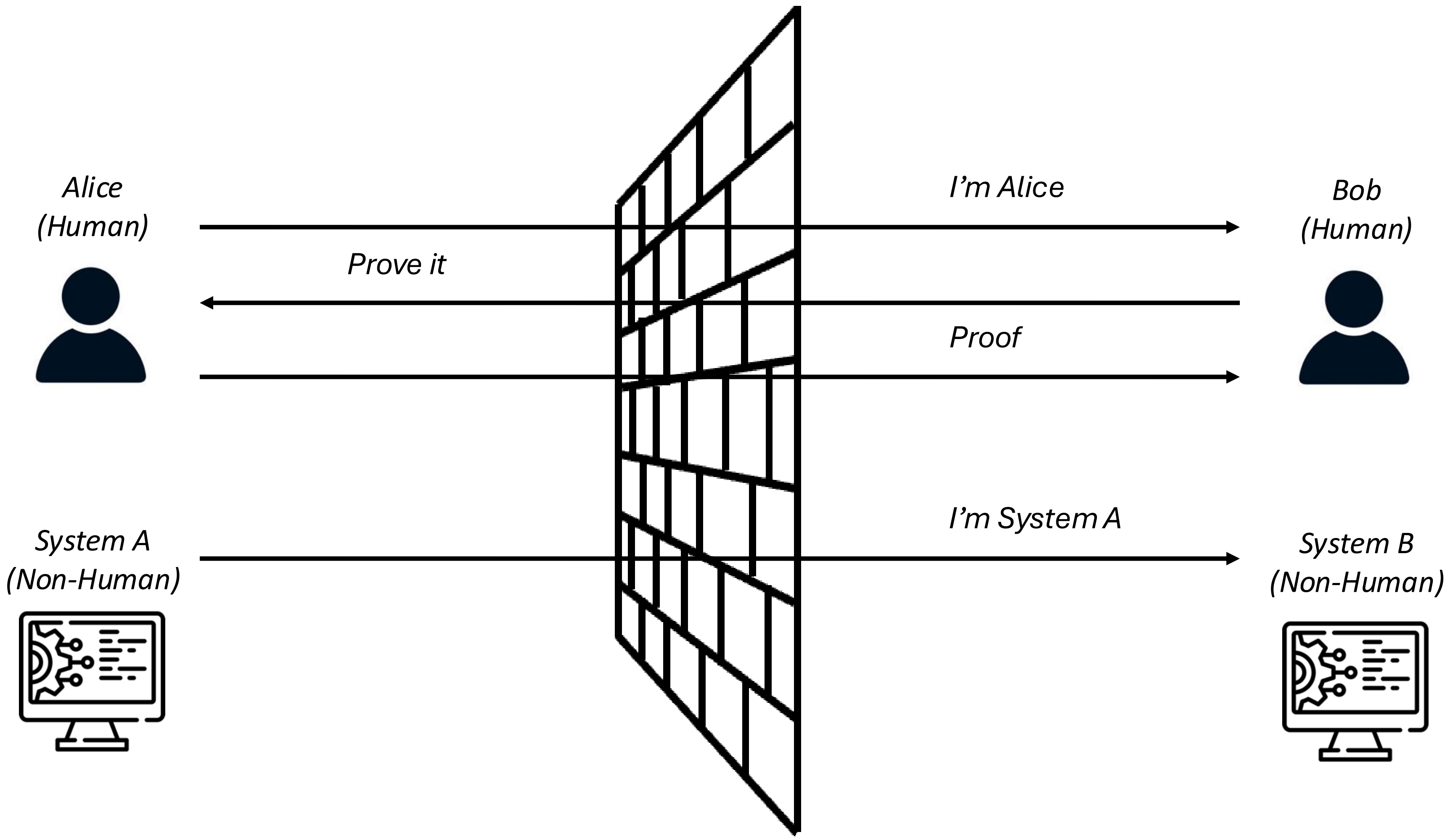
Bob
(Human)

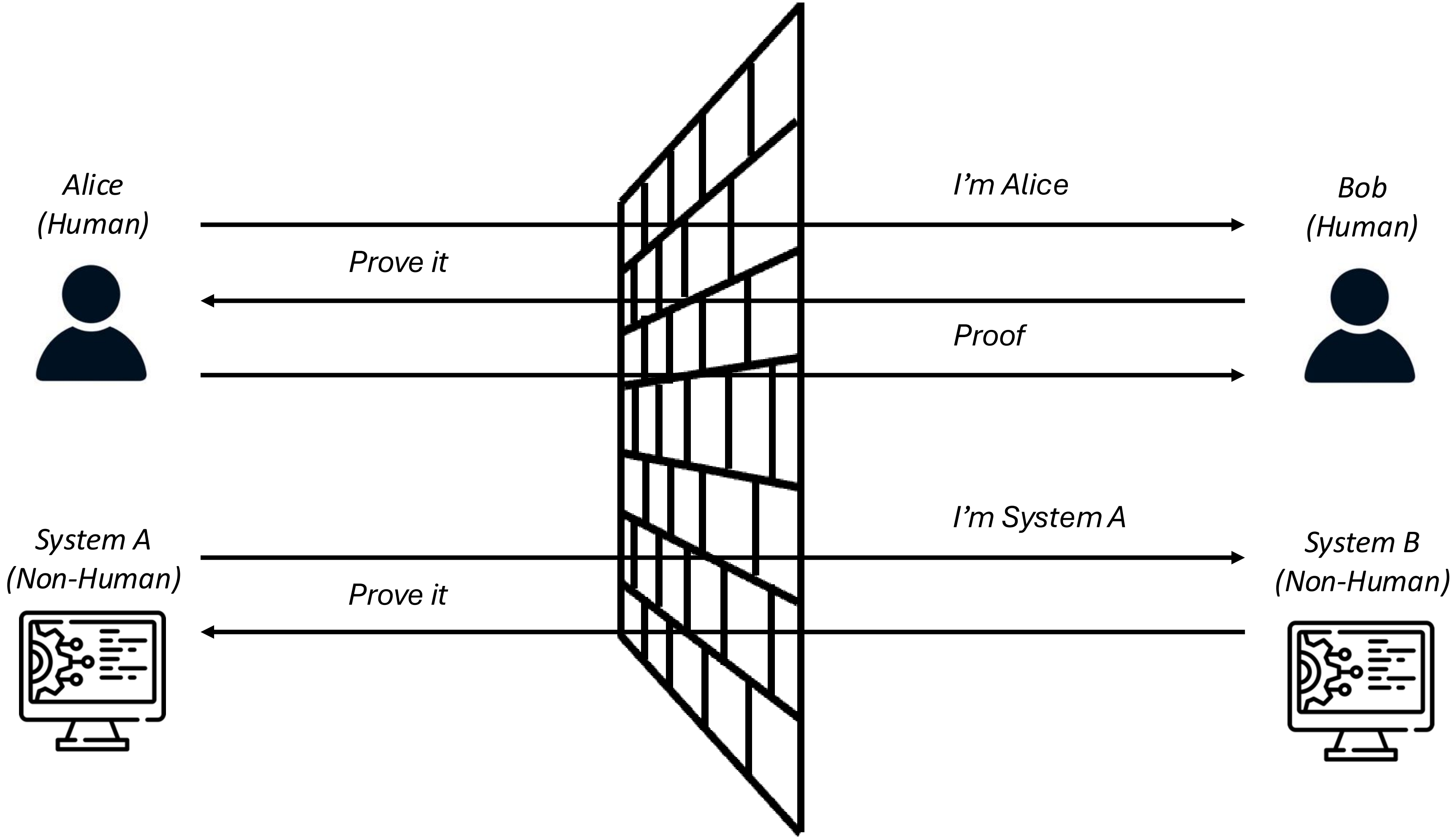
Prove it

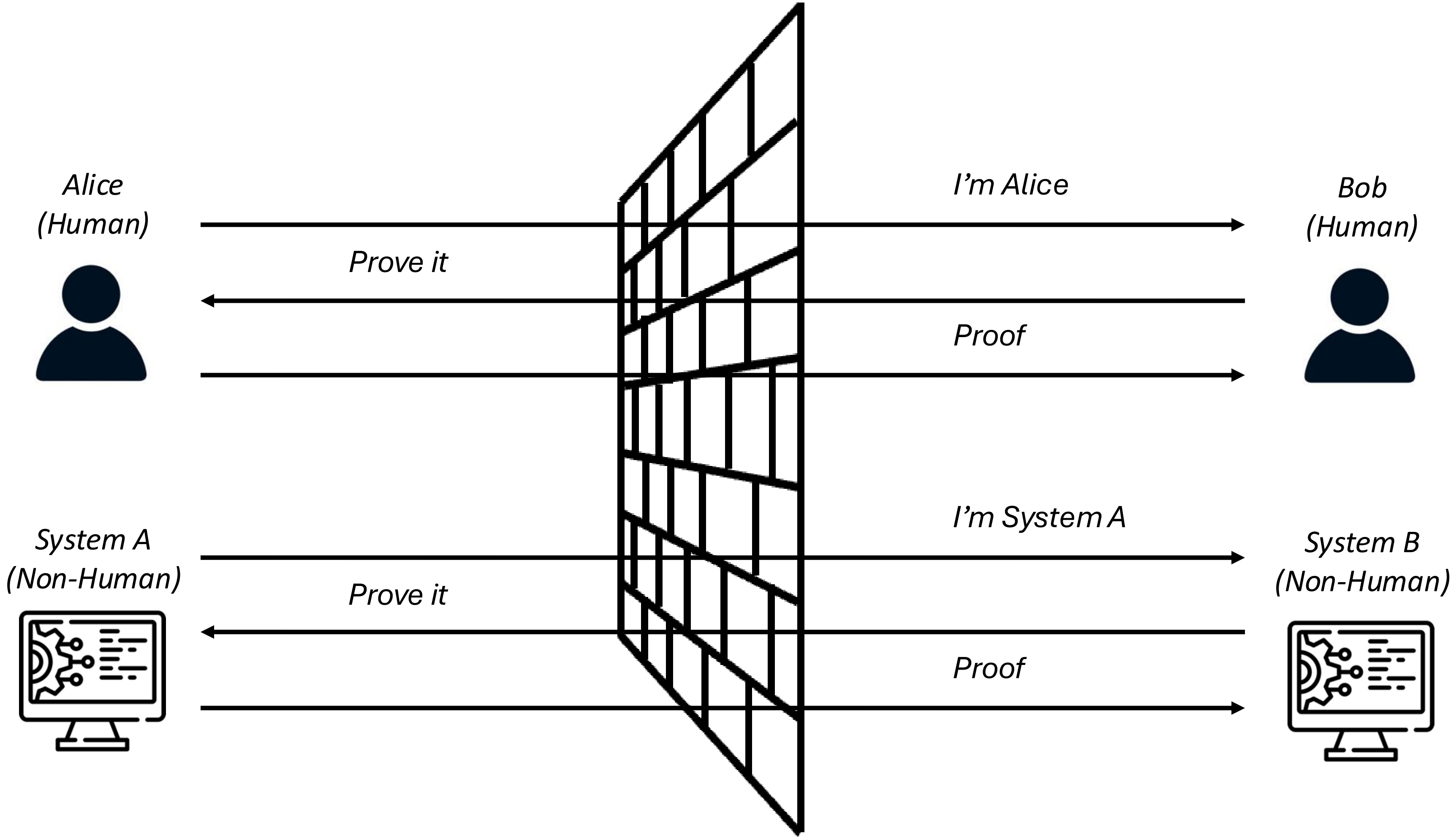
Proof

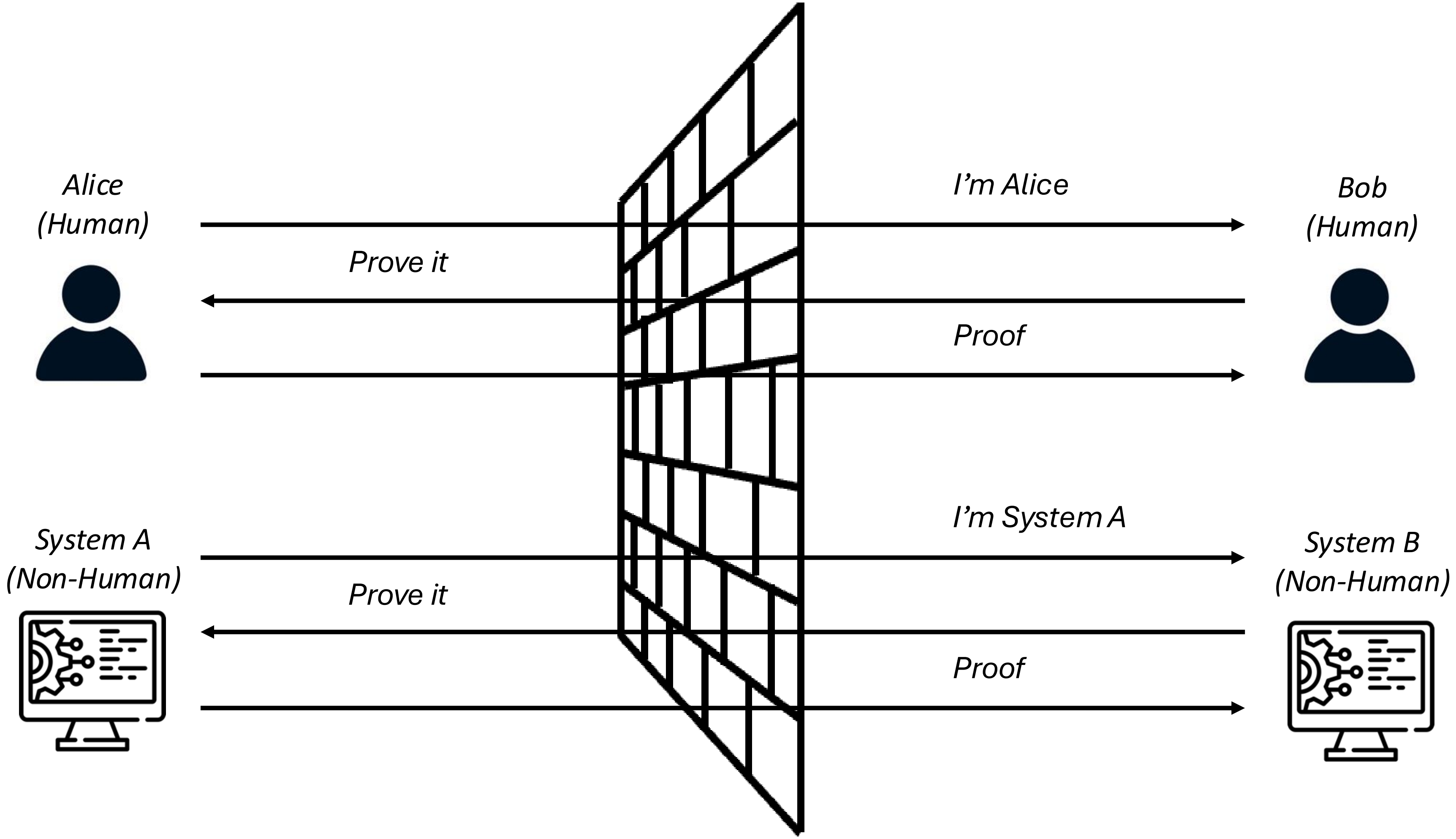
System A
(Non-Human)

System B
(Non-Human)


Alice
(Human)
I'm Alice
Bob
(Human)
Prove it
Proof
System A
(Non-Human)
I'm System A
System B
(Non-Human)

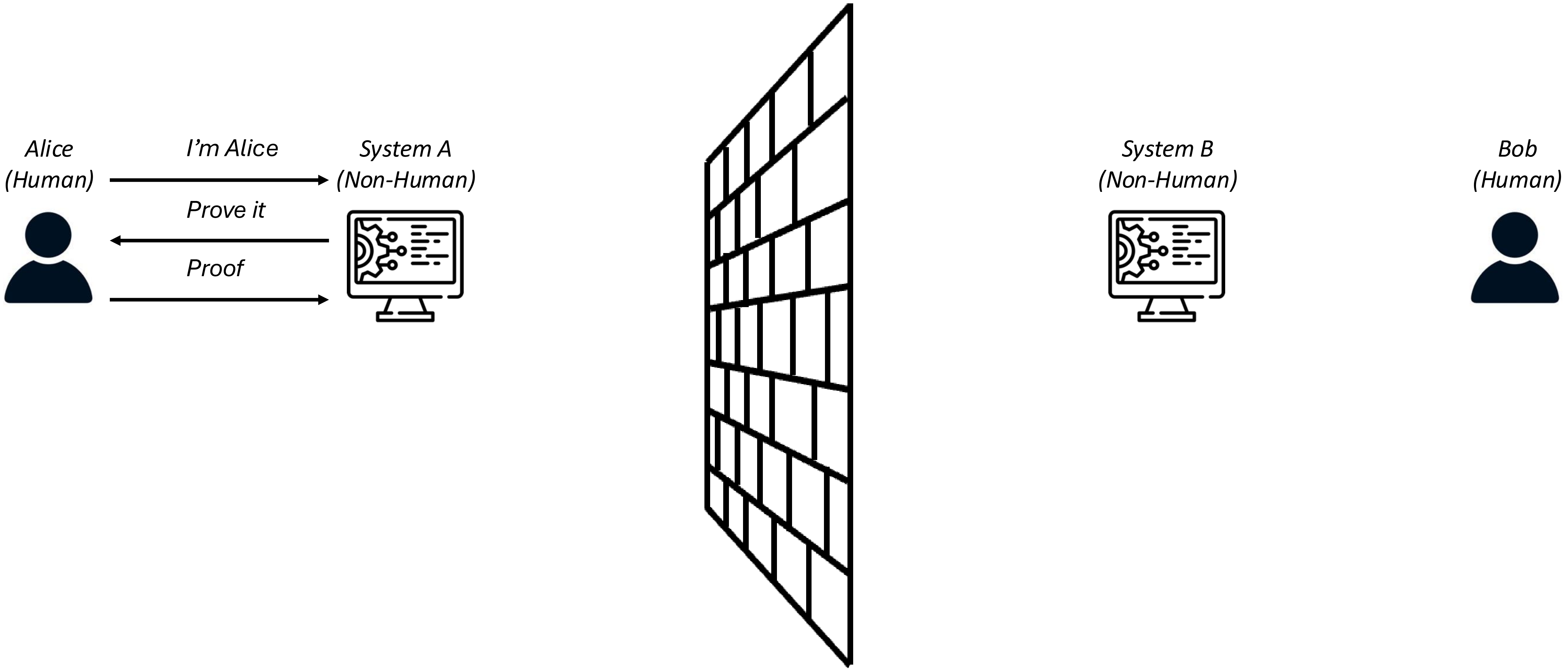

Alice
(Human)
I'm Alice
Bob
(Human)
Prove it
Proof
System A
(Non-Human)
I'm System A
System B
(Non-Human)
Prove it


Alice
(Human)
I'm Alice
Bob
(Human)
Prove it
Proof
System A
(Non-Human)
I'm System A
System B
(Non-Human)
Prove it
Proof



This is the Basis for All Authentication Protocols

Humans use computer systems (e.g., iPhones) to communicate.



Alice
(Human)

I’m Alice

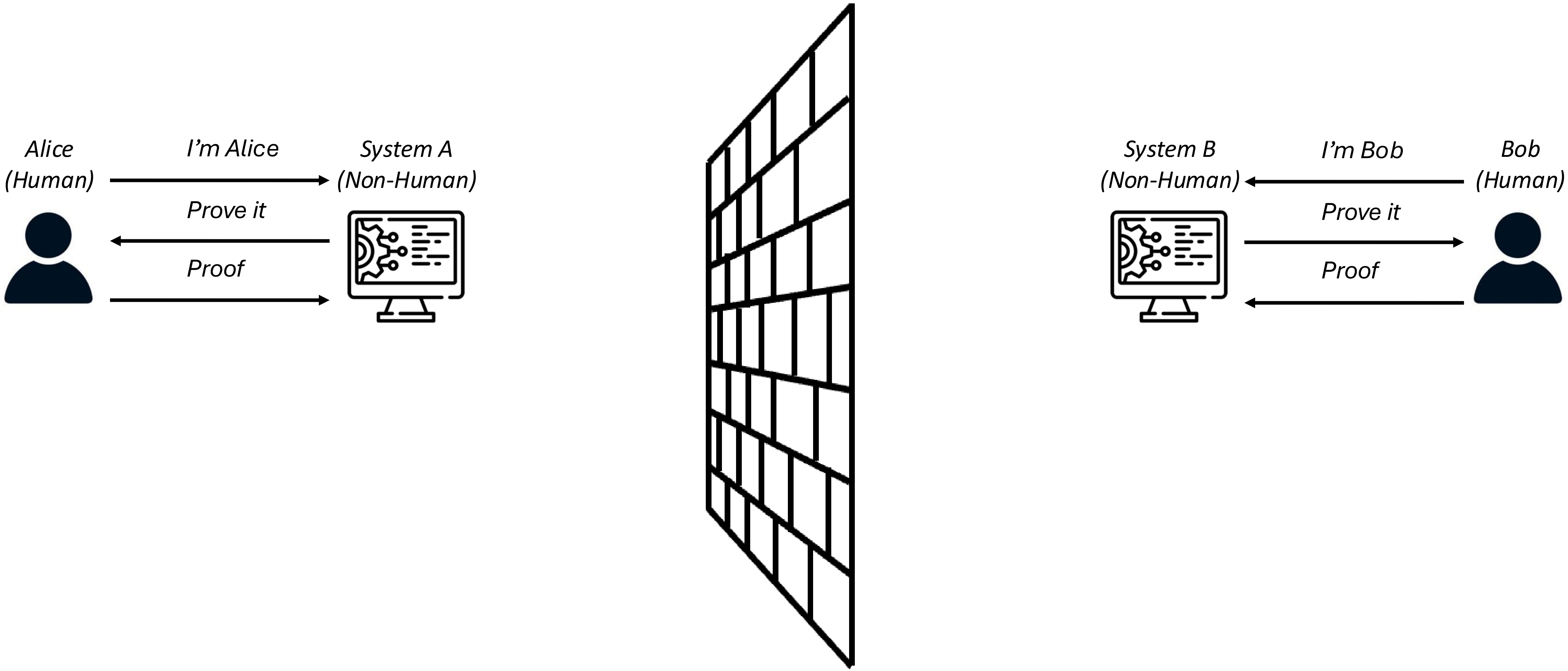
Prove it

Proof

System A
(Non-Human)

System B
(Non-Human)

Bob
(Human)



Alice (Human)

I’m Alice

System A (Non-Human)

Prove it

Proof

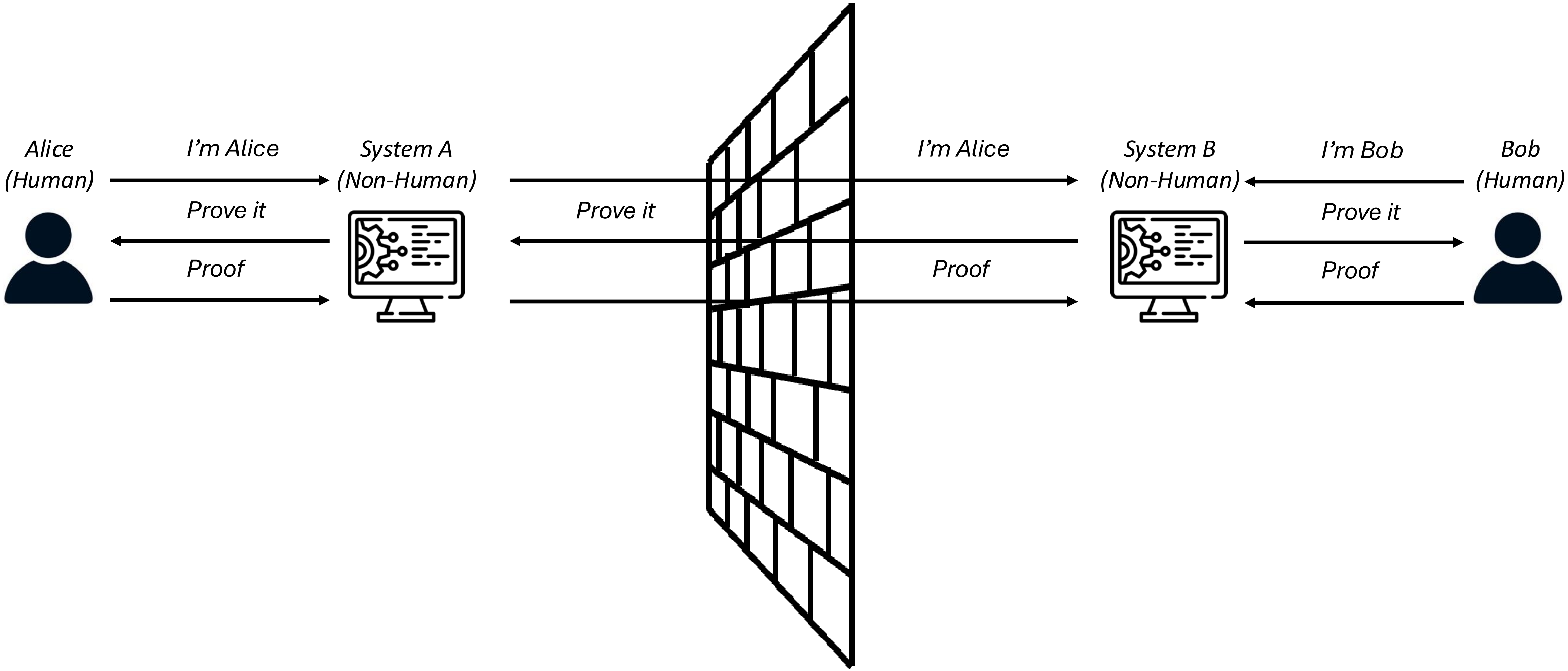
System B (Non-Human)

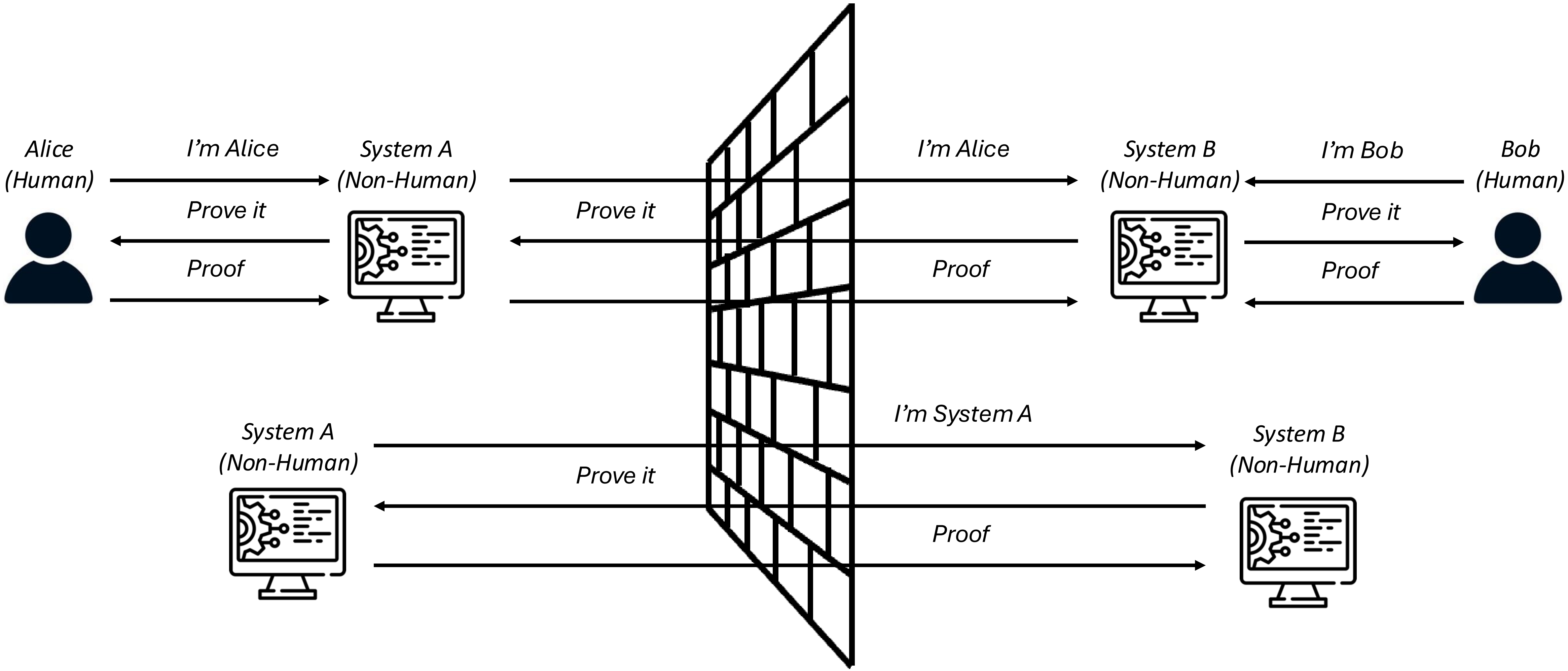
I’m Bob

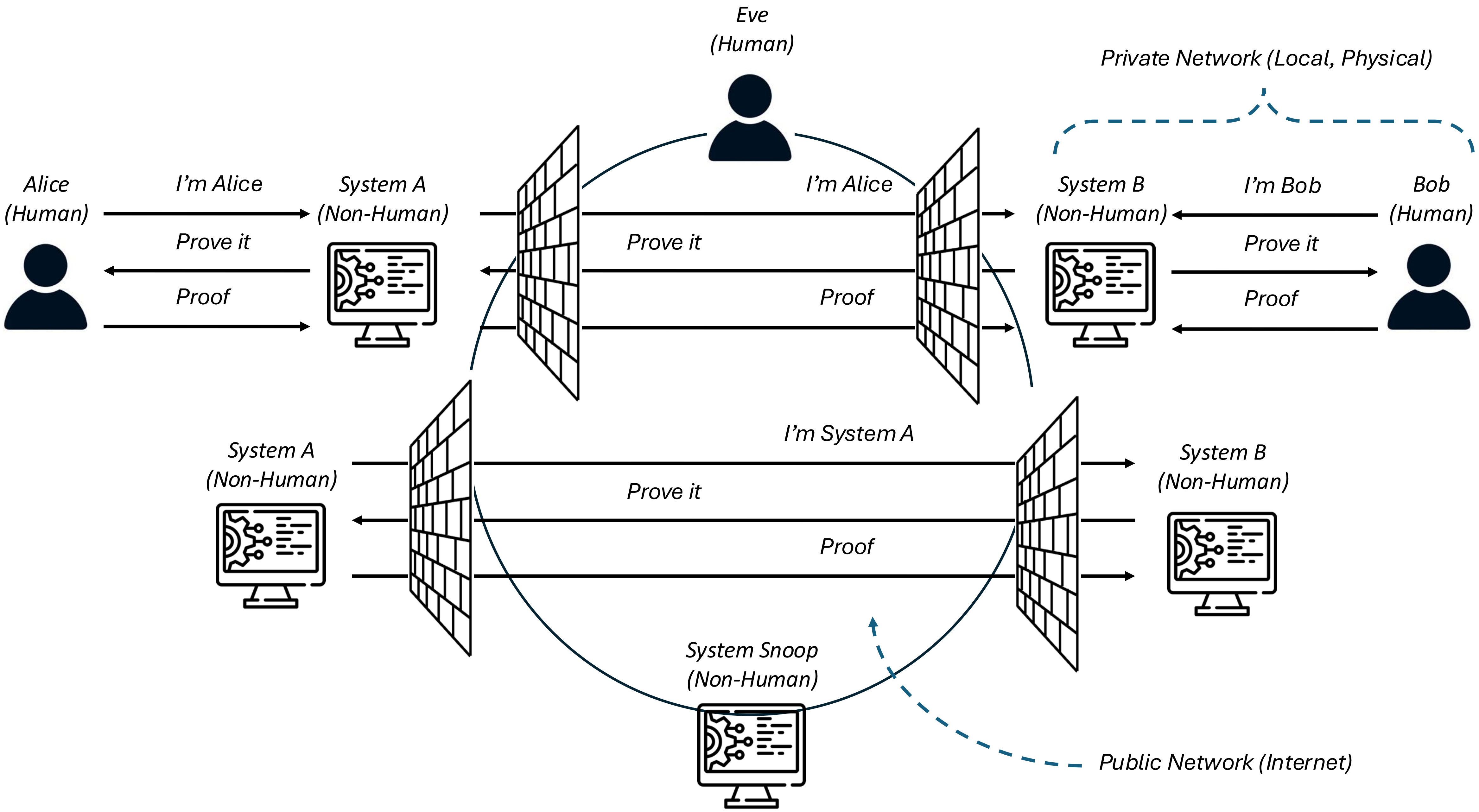
Bob (Human)

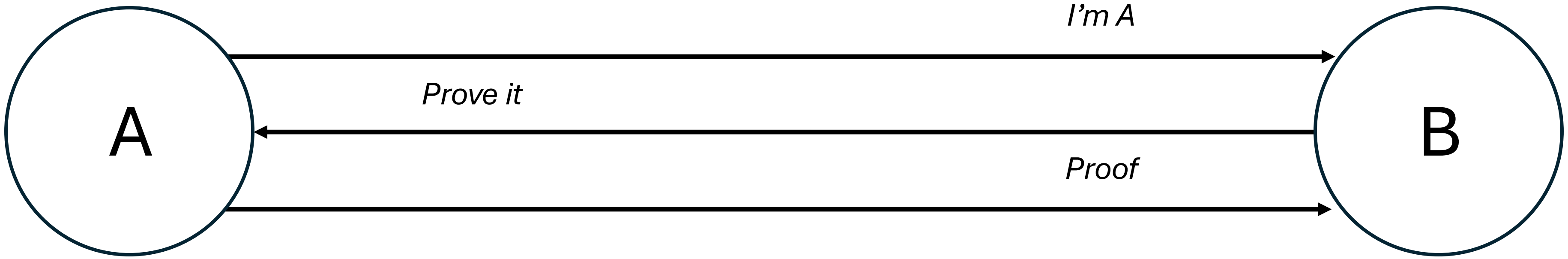
Prove it

Proof


Alice
(Human)
I’m Alice
Prove it
Proof
System A
(Non-Human)
Prove it
I’m Alice
Proof
System B
(Non-Human)
I’m Bob
Prove it
Proof
Bob
(Human)


Alice
(Human)
I'm Alice
Prove it
Proof
System A
(Non-Human)
Prove it
I'm Alice
Proof
System B
(Non-Human)
I'm Bob
Prove it
Proof
Bob
(Human)
System A
(Non-Human)
I'm System A
Prove it
Proof
System B
(Non-Human)


Eve
(Human)
Private Network (Local, Physical)
Alice
(Human)
I'm Alice
Prove it
Proof
System A
(Non-Human)
I'm Alice
Prove it
Proof
System B
(Non-Human)
I'm Bob
Prove it
Proof
Bob
(Human)
System A
(Non-Human)
I'm System A
Prove it
Proof
System B
(Non-Human)
System Snoop
(Non-Human)
Public Network (Internet)

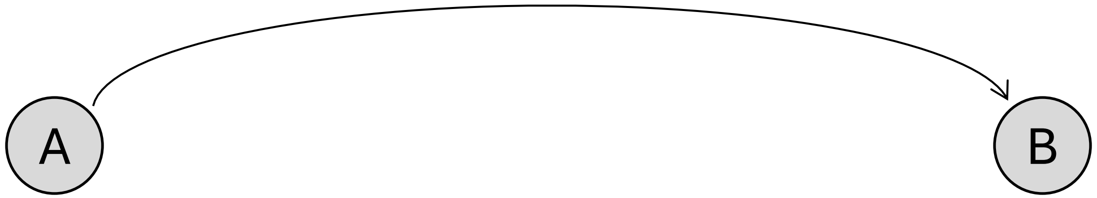


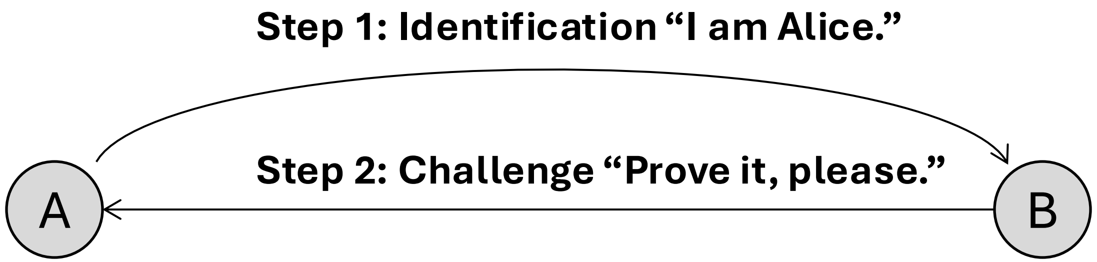
Generalizing Authentication Protocols to *Entities* A and B

Authentication protocols range from simple to complex.

What is the most general protocol schema for authentication?

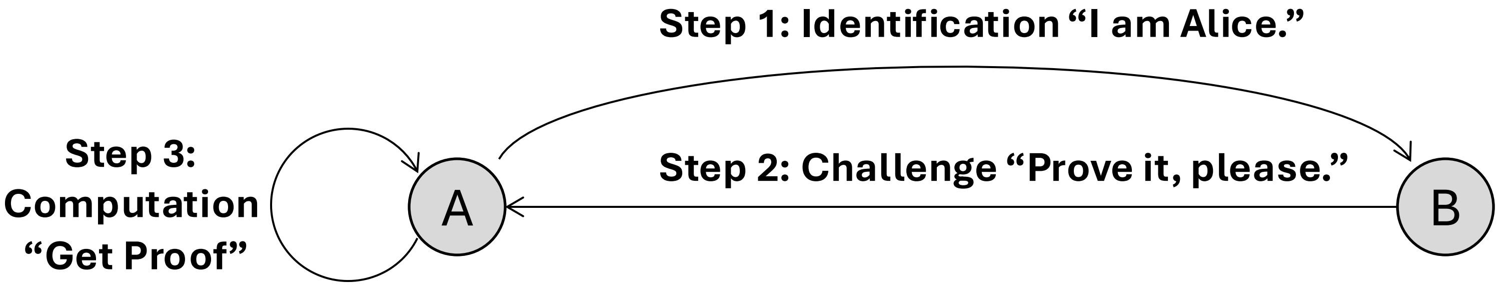
Step 1: Identification “I am Alice.”

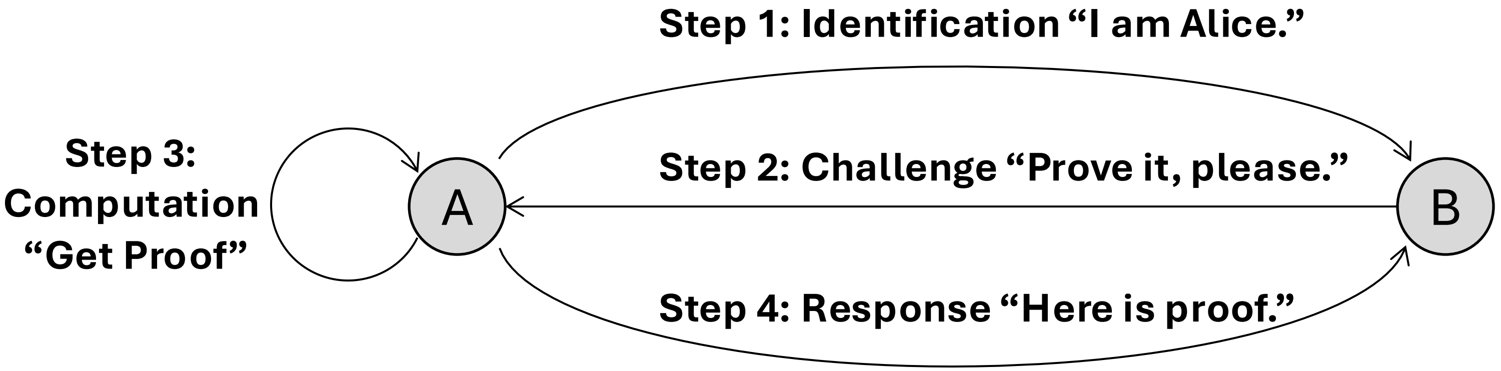




Step 1: Identification “I am Alice.”

Step 2: Challenge “Prove it, please.”


Step 1: Identification "I am Alice."
Step 2: Challenge "Prove it, please."
Step 3:
Computation
"Get Proof"
A
B

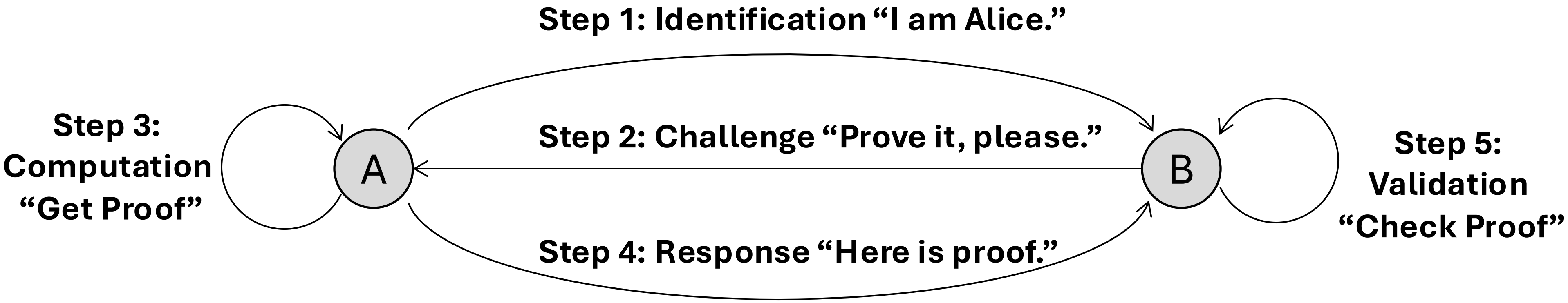


Step 1: Identification “I am Alice.”

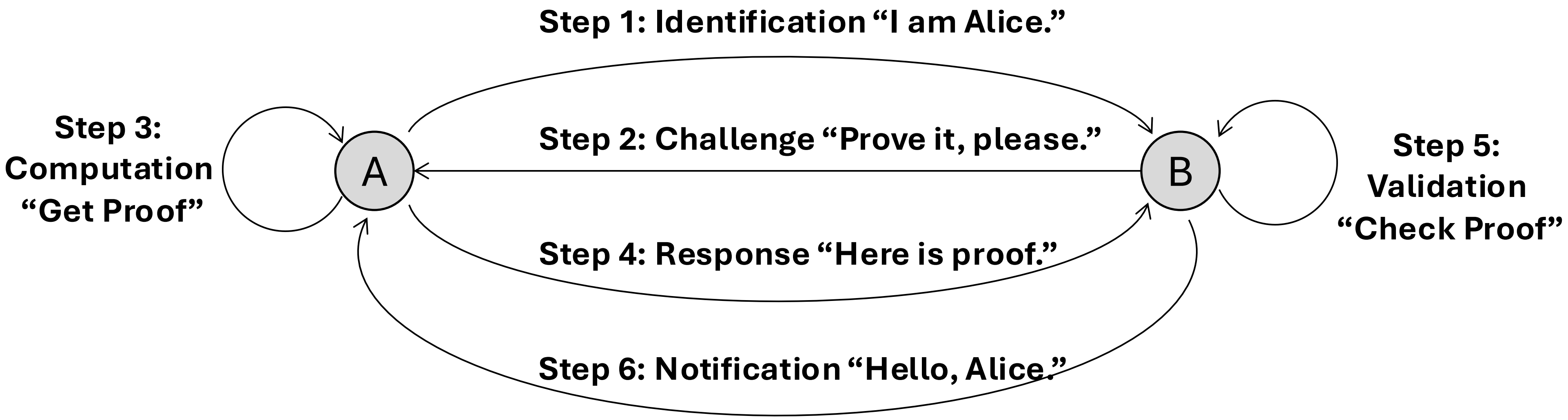
Step 2: Challenge “Prove it, please.”

Step 3: Computation “Get Proof”

Step 4: Response “Here is proof.”



Step 1: Identification "I am Alice."
Step 2: Challenge "Prove it, please."
Step 3: Computation "Get Proof"
Step 4: Response "Here is proof."
Step 5: Validation "Check Proof"
A
B



Step 1: Identification “I am Alice.”

Step 2: Challenge “Prove it, please.”

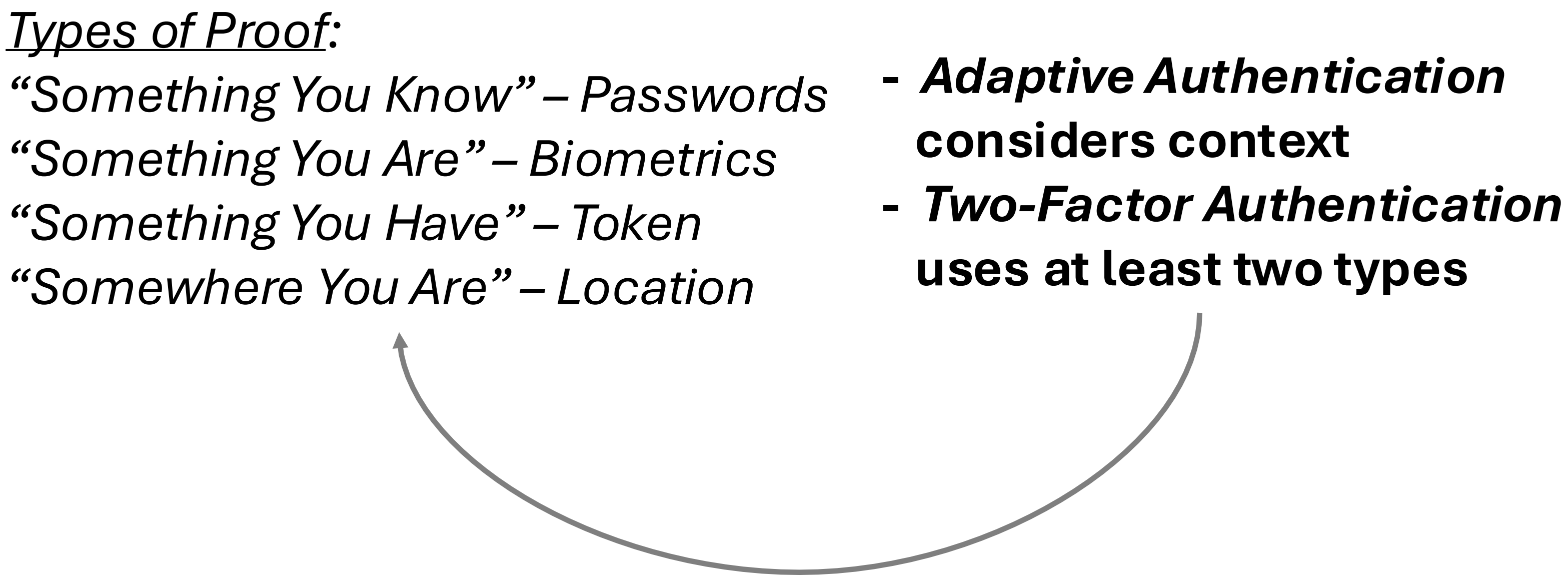
Step 3: Computation “Get Proof”

Step 4: Response “Here is proof.”

Step 5: Validation “Check Proof”

Step 6: Notification “Hello, Alice.”

What proof options exist for authentication?

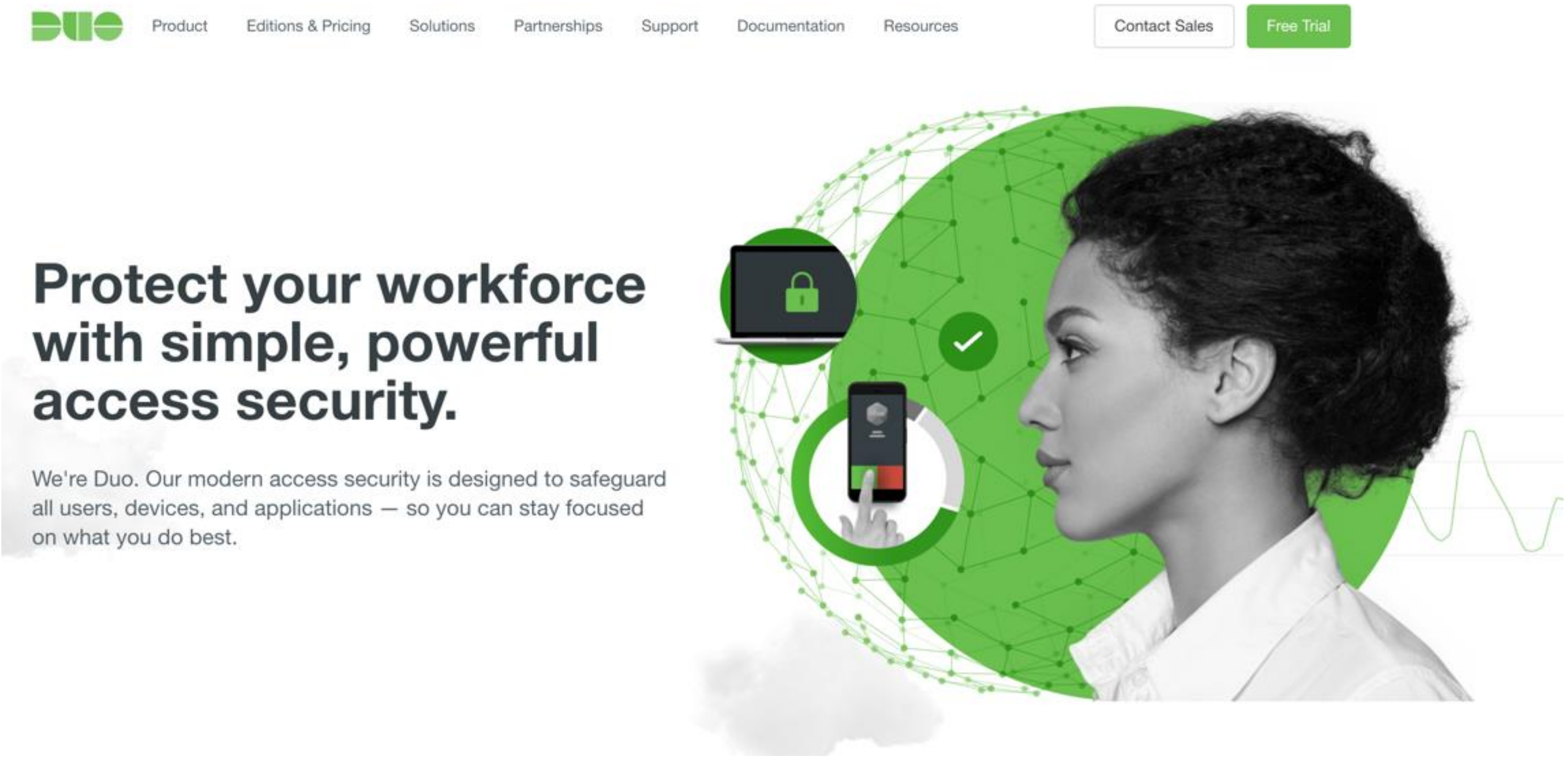


Types of Proof:

“Something You Know” – Passwords
“Something You Are” – Biometrics
“Something You Have” – Token
“Somewhere You Are” – Location

- ***Adaptive Authentication*** **considers context**
- ***Two-Factor Authentication*** **uses at least two types**

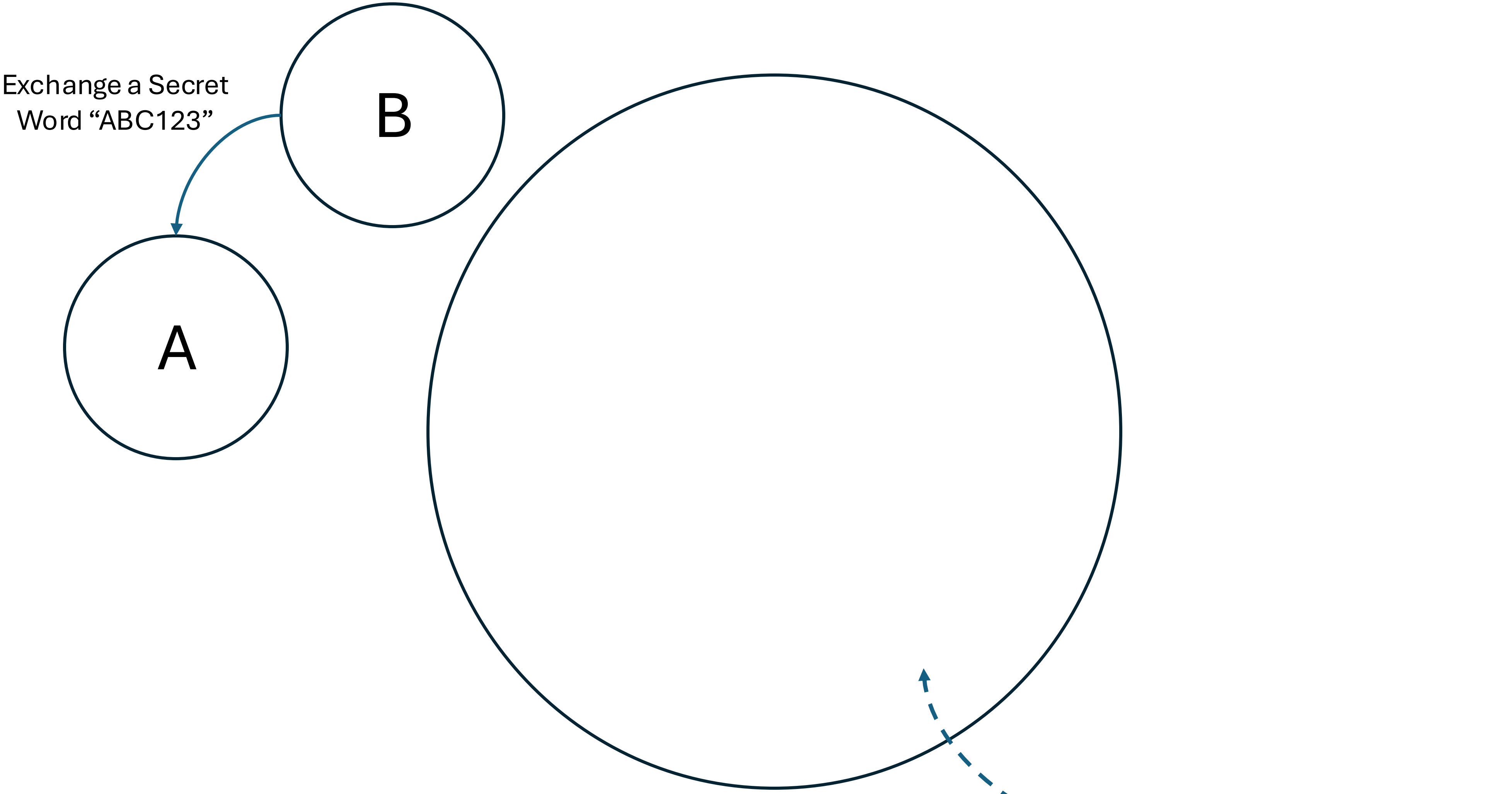
Multifactor Authentication (MFA)
Requires Multiple, Diverse Proof Options



Protect your workforce with simple, powerful access security.

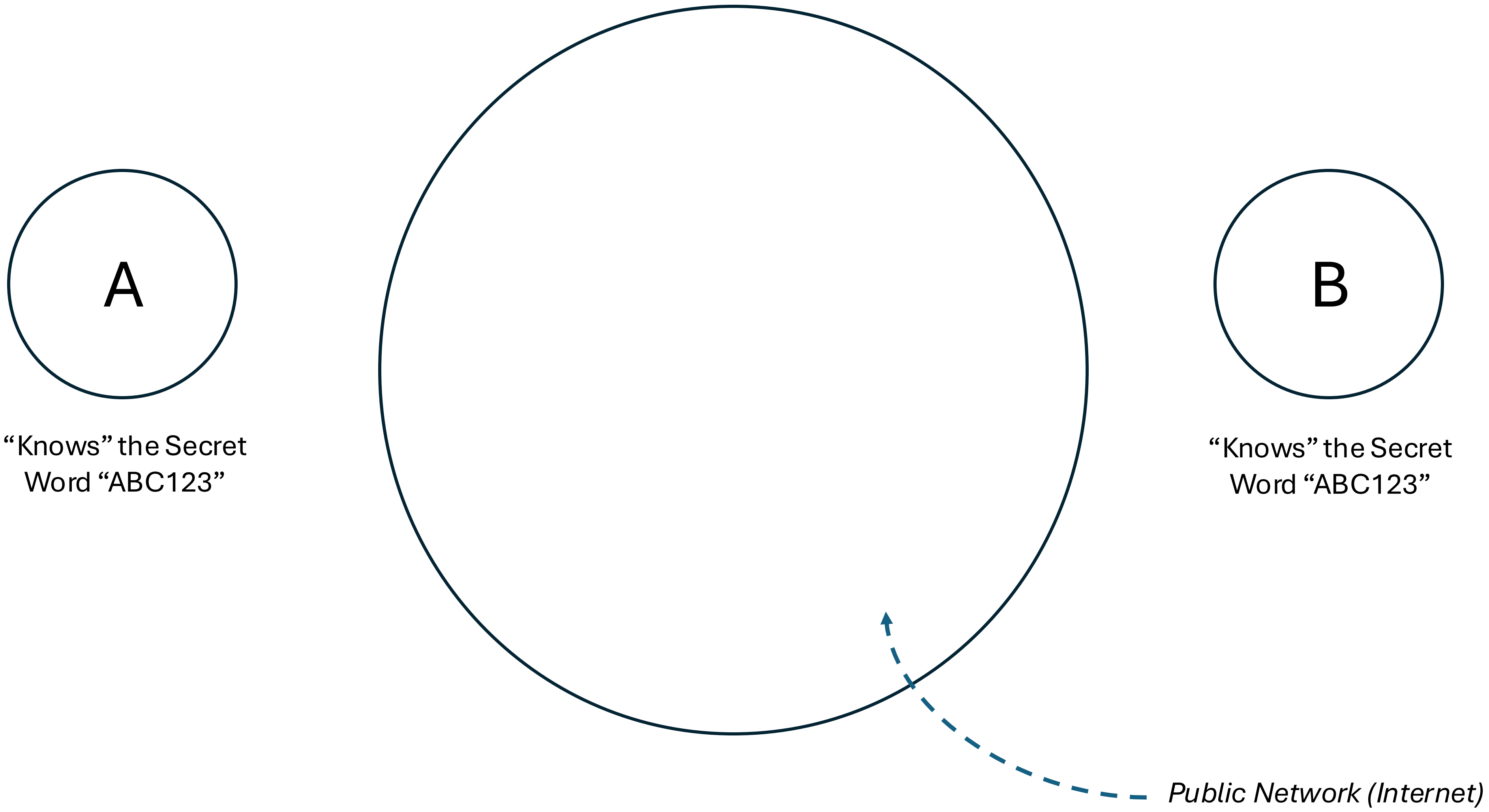
We're Duo. Our modern access security is designed to safeguard all users, devices, and applications — so you can stay focused on what you do best.

How do password protocols work?



Exchange a Secret
Word “ABC123”

Public Network (Internet)

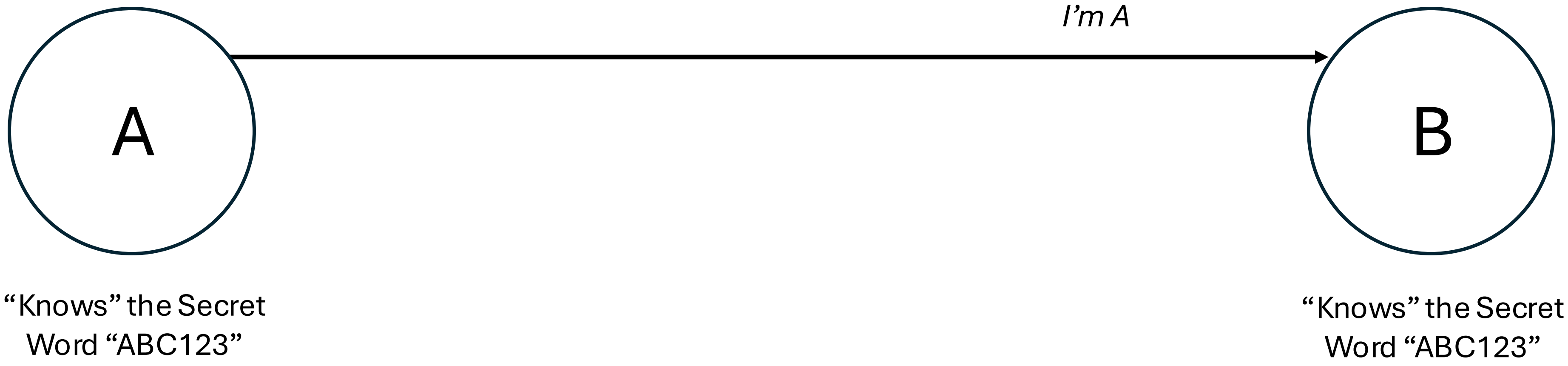


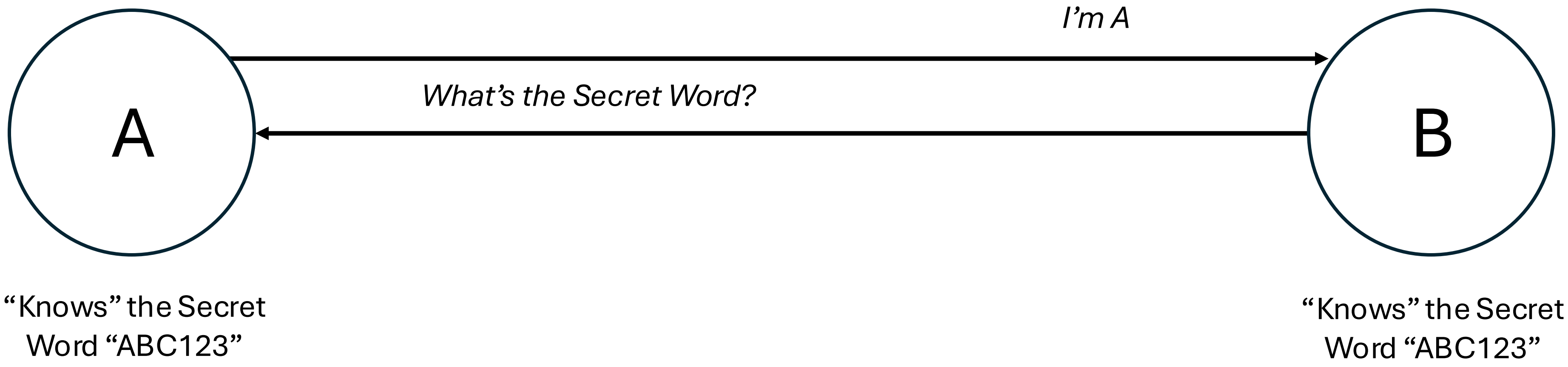
“Knows” the Secret Word “ABC123”

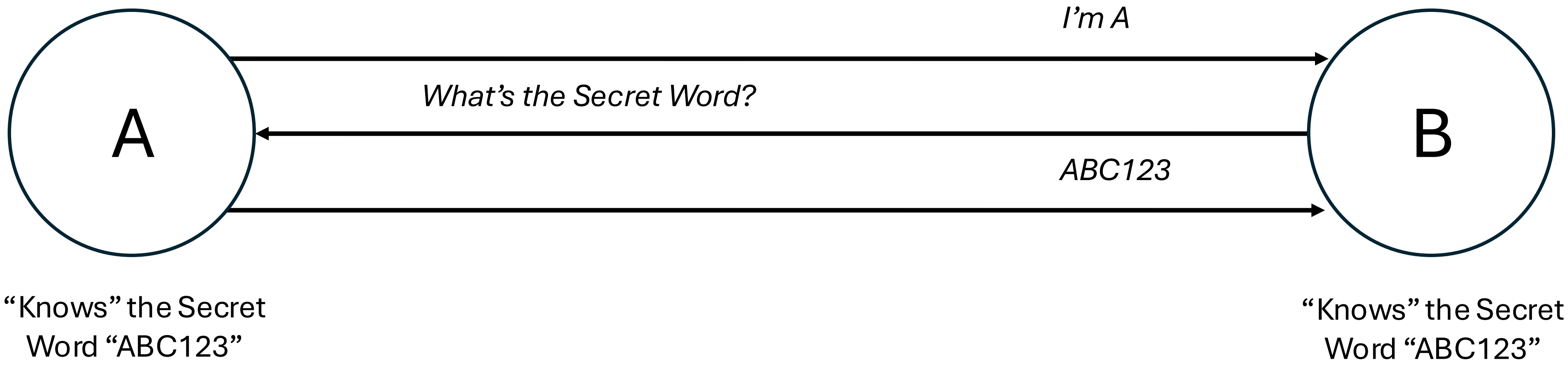
“Knows” the Secret Word “ABC123”

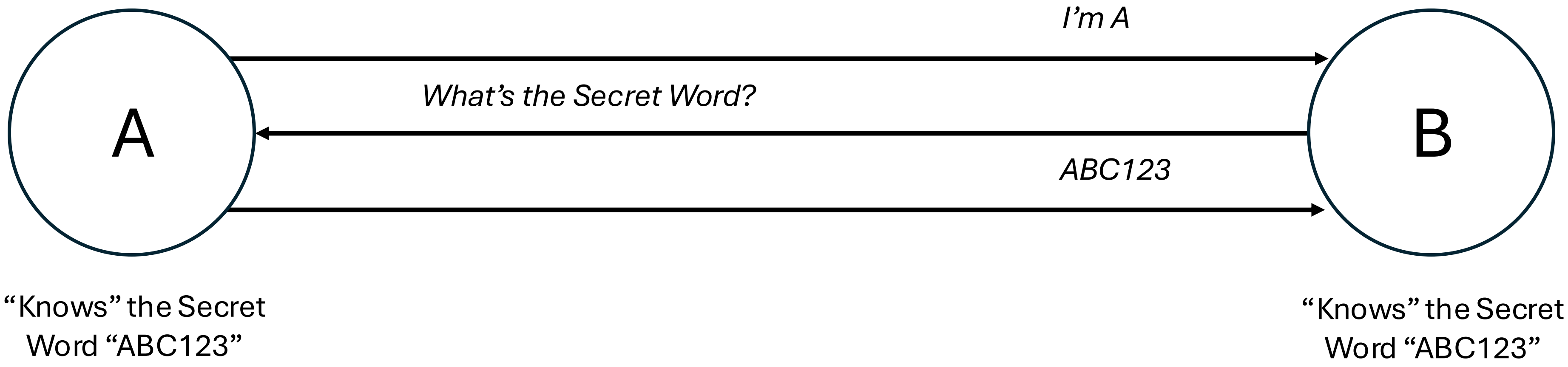
“Knows” the Secret Word “ABC123”

“Knows” the Secret Word “ABC123”

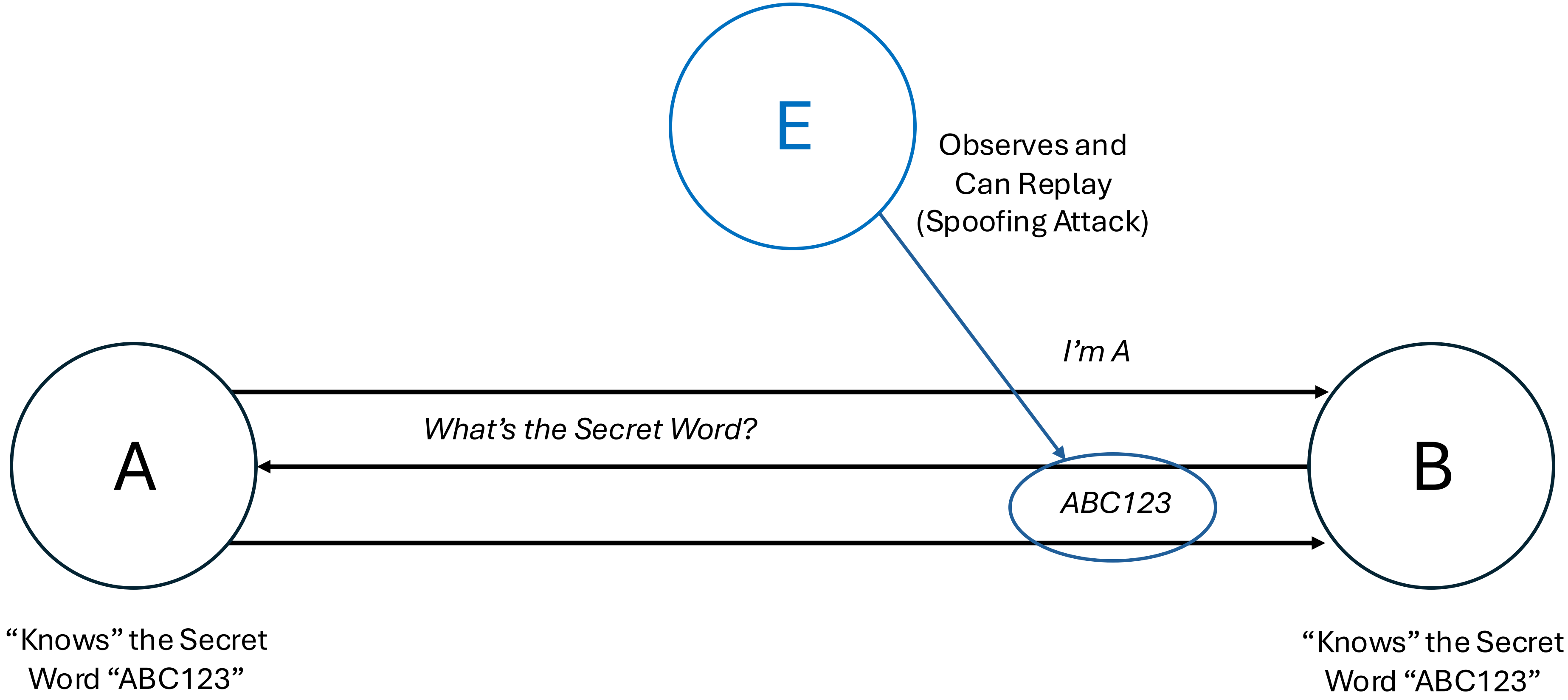

I’m A
A
B
“Knows” the Secret Word “ABC123”
“Knows” the Secret Word “ABC123”


I'm A
What's the Secret Word?
A
B
"Knows" the Secret Word "ABC123"
"Knows" the Secret Word "ABC123"


A
B
I'm A
What's the Secret Word?
ABC123
"Knows" the Secret Word "ABC123"
"Knows" the Secret Word "ABC123"

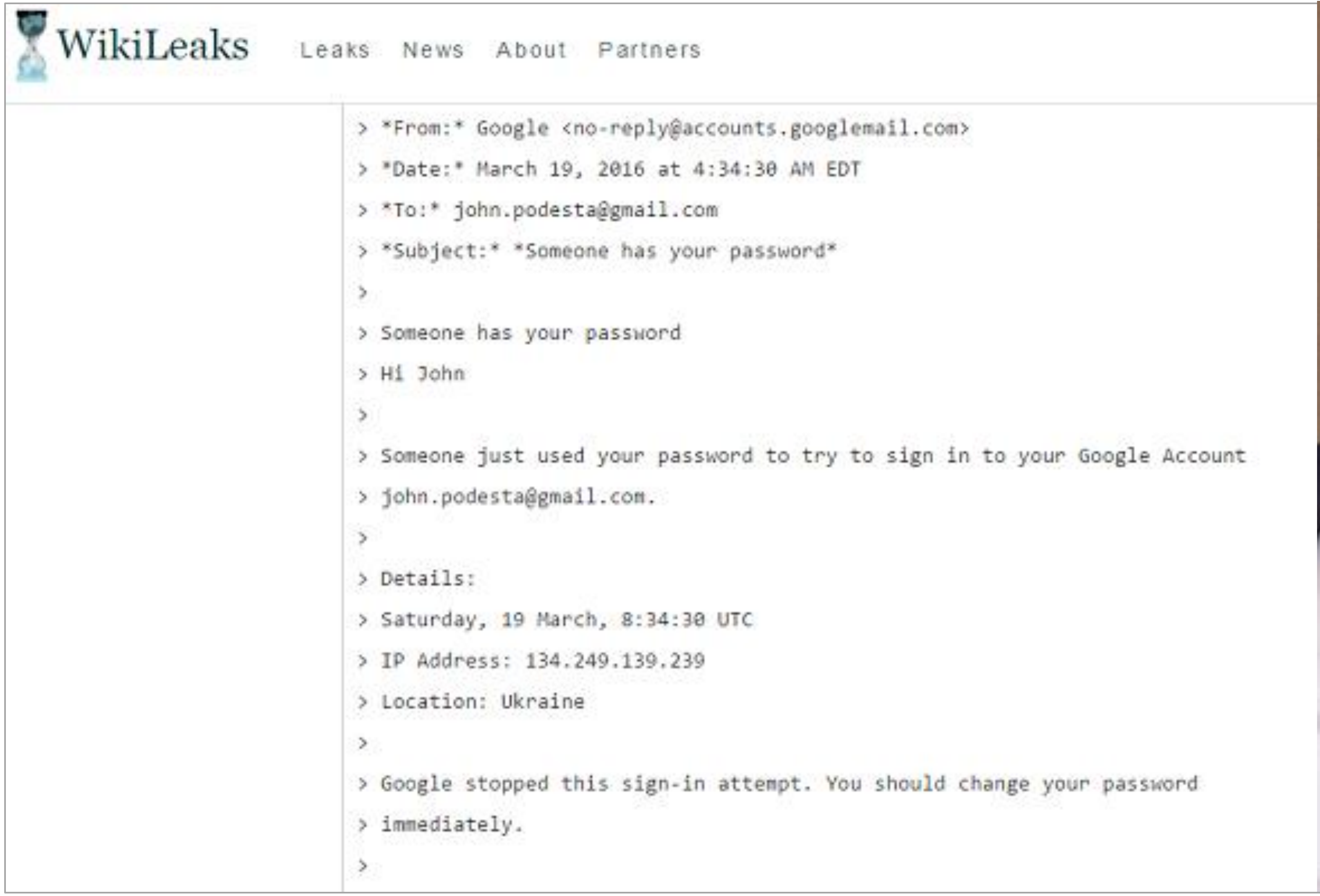

A
B
I'm A
What's the Secret Word?
ABC123
"Knows" the Secret Word "ABC123"
"Knows" the Secret Word "ABC123"

This is the Simplest Password Protocol


E
Observes and
Can Replay
(Spoofing Attack)
I'm A
What's the Secret Word?
A
B
ABC123
"Knows" the Secret
Word "ABC123"
"Knows" the Secret
Word "ABC123"

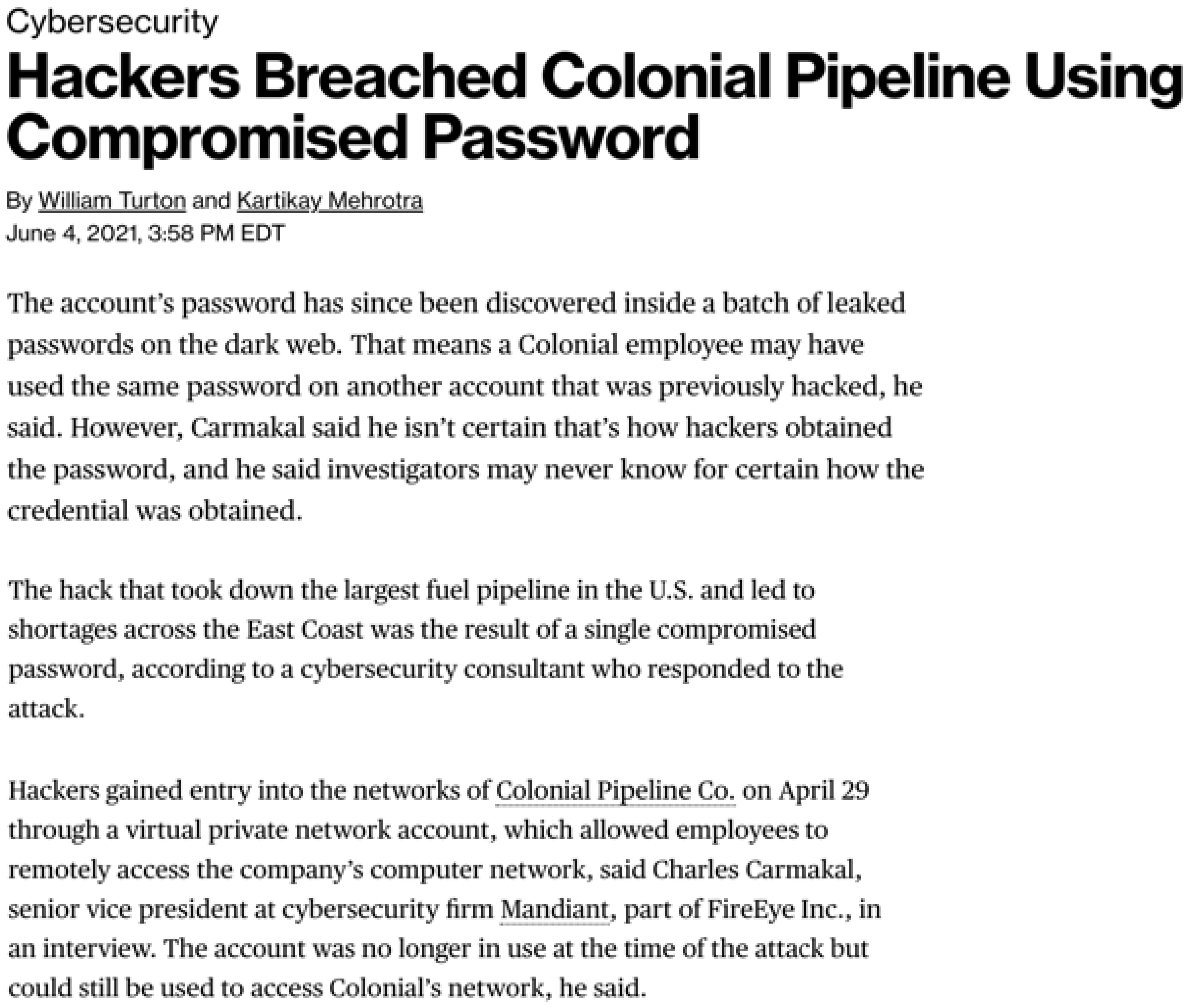
What challenges exist for passwords?

Democratic National Committee Hack 2016



```
> *From:* Google <no-reply@accounts.googlemail.com>
> *Date:* March 19, 2016 at 4:34:30 AM EDT
> *To:* john.podesta@gmail.com
> *Subject:* *Someone has your password*
>
> Someone has your password
> Hi John
>
> Someone just used your password to try to sign in to your Google Account
> john.podesta@gmail.com.
>
> Details:
> Saturday, 19 March, 8:34:30 UTC
> IP Address: 134.249.139.239
> Location: Ukraine
>
> Google stopped this sign-in attempt. You should change your password
> immediately.
>
```

Colonial Pipeline Ransomware Hack 2021



Cybersecurity

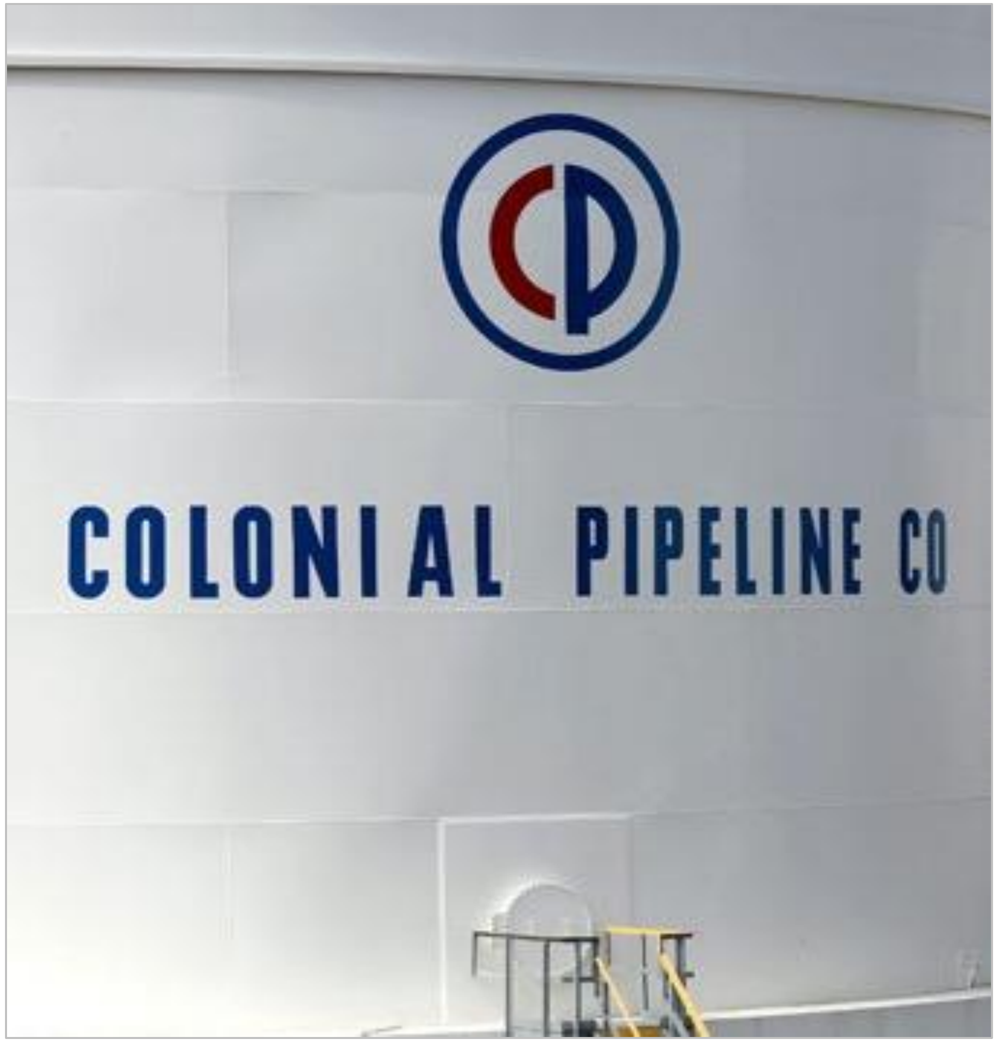
Hackers Breached Colonial Pipeline Using Compromised Password

By William Turton and Kartikay Mehrotra
June 4, 2021, 3:58 PM EDT

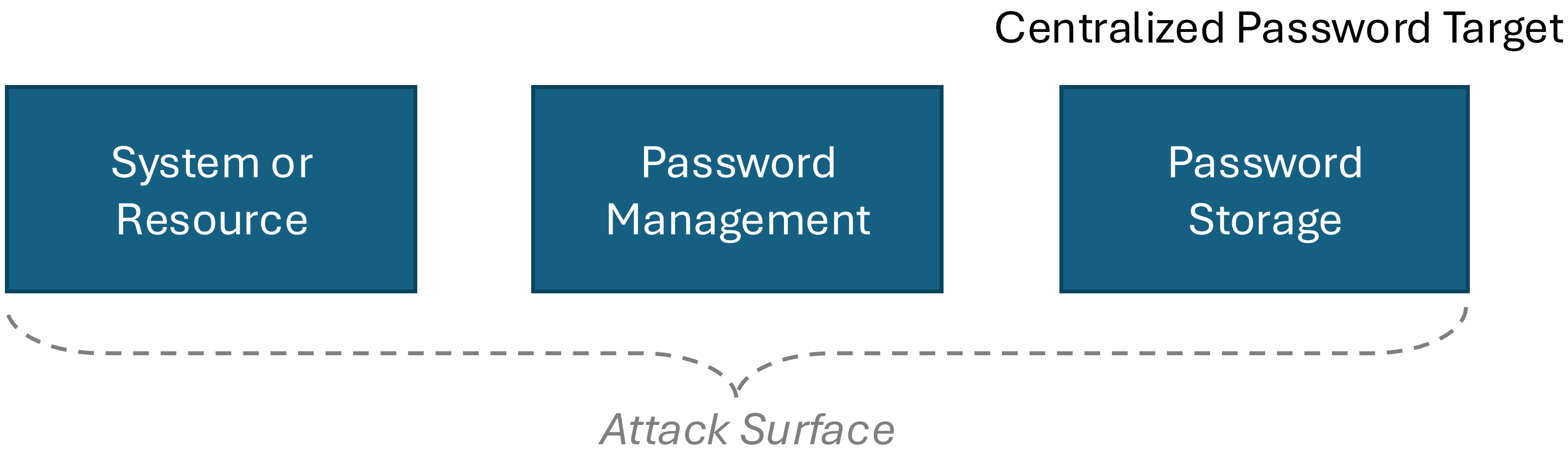
The account's password has since been discovered inside a batch of leaked passwords on the dark web. That means a Colonial employee may have used the same password on another account that was previously hacked, he said. However, Carmakal said he isn't certain that's how hackers obtained the password, and he said investigators may never know for certain how the credential was obtained.

The hack that took down the largest fuel pipeline in the U.S. and led to shortages across the East Coast was the result of a single compromised password, according to a cybersecurity consultant who responded to the attack.

Hackers gained entry into the networks of Colonial Pipeline Co. on April 29 through a virtual private network account, which allowed employees to remotely access the company's computer network, said Charles Carmakal, senior vice president at cybersecurity firm Mandiant, part of FireEye Inc., in an interview. The account was no longer in use at the time of the attack but could still be used to access Colonial's network, he said.



Inherent Threat of Password Repositories



Centralized Password Target

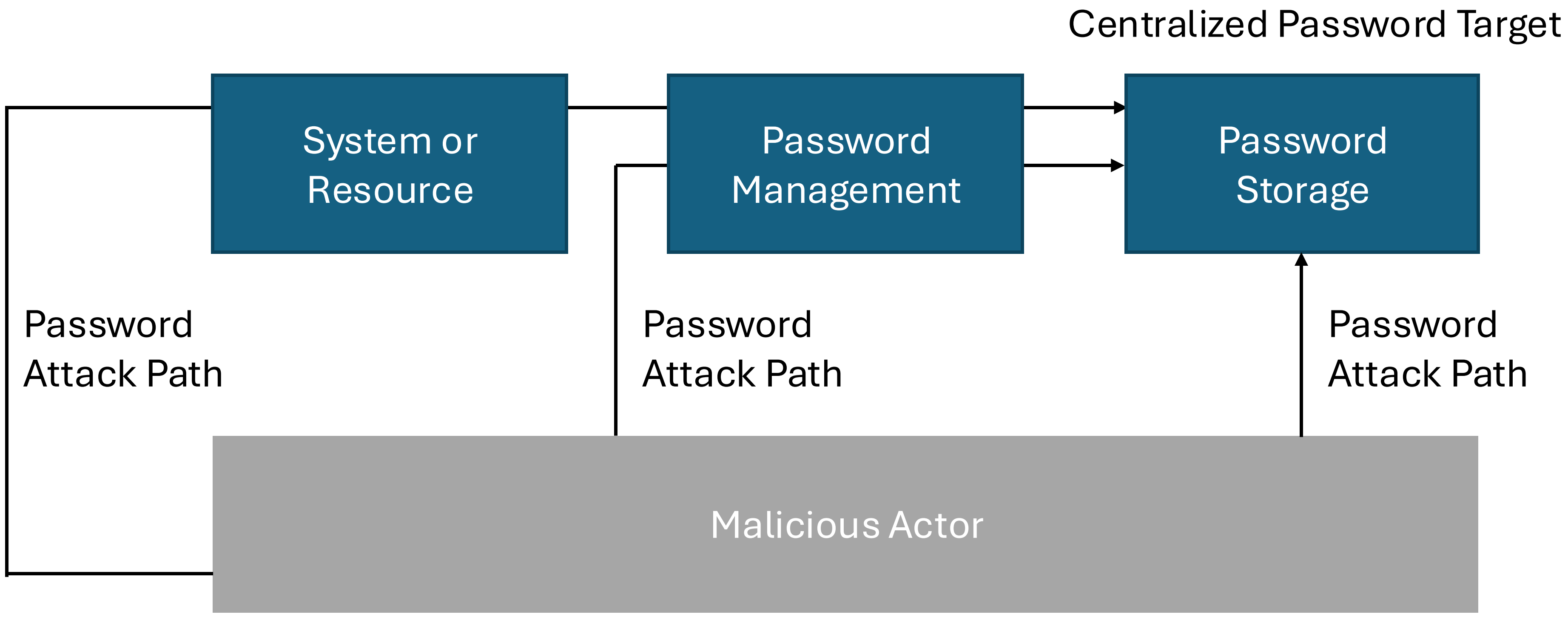
System or Resource

Password Management

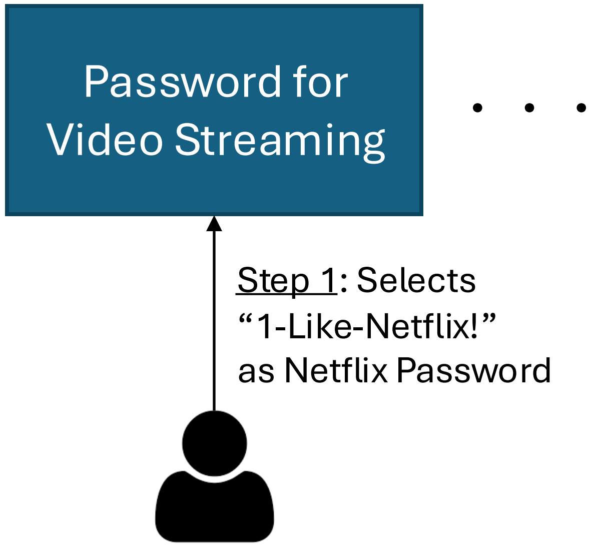
Password Storage

Attack Surface

Inherent Threat of Password Repositories



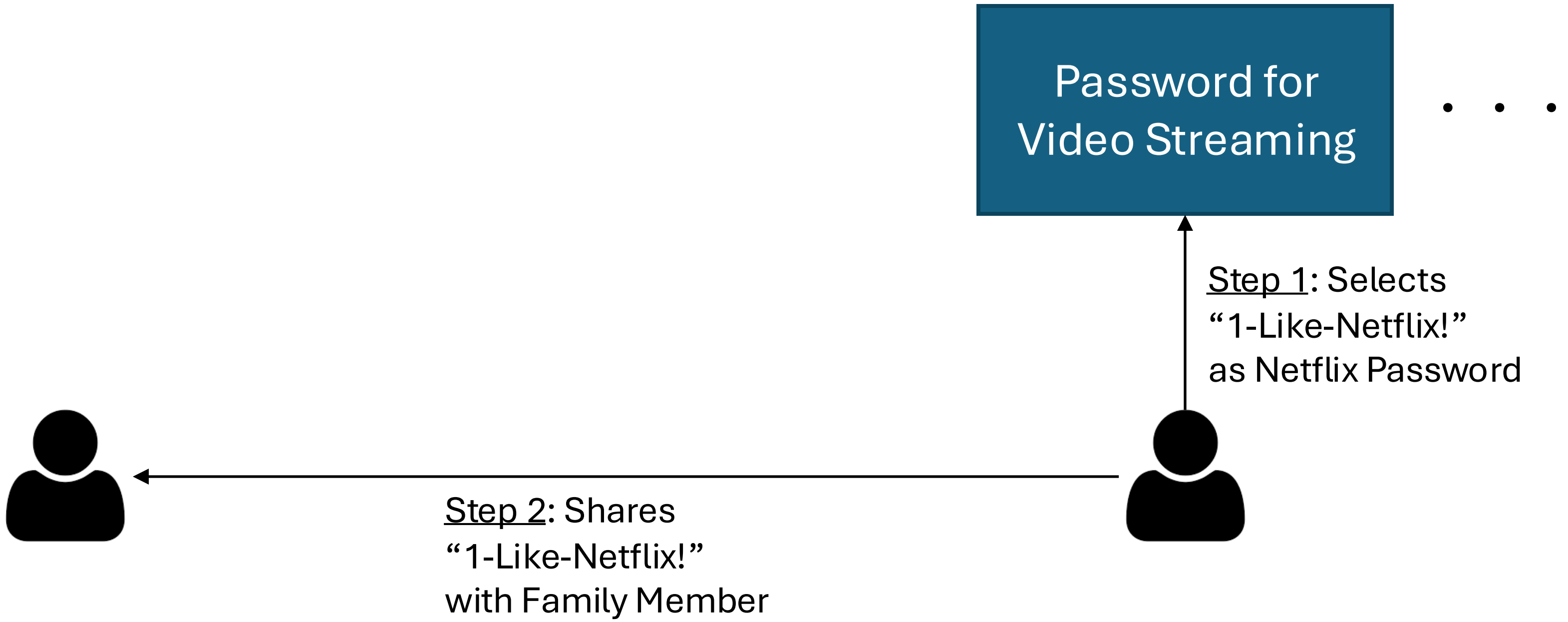
Inherent Threat of Password Reuse



Password for Video Streaming

Step 1: Selects “1-Like-Netflix!” as Netflix Password

Inherent Threat of Password Reuse



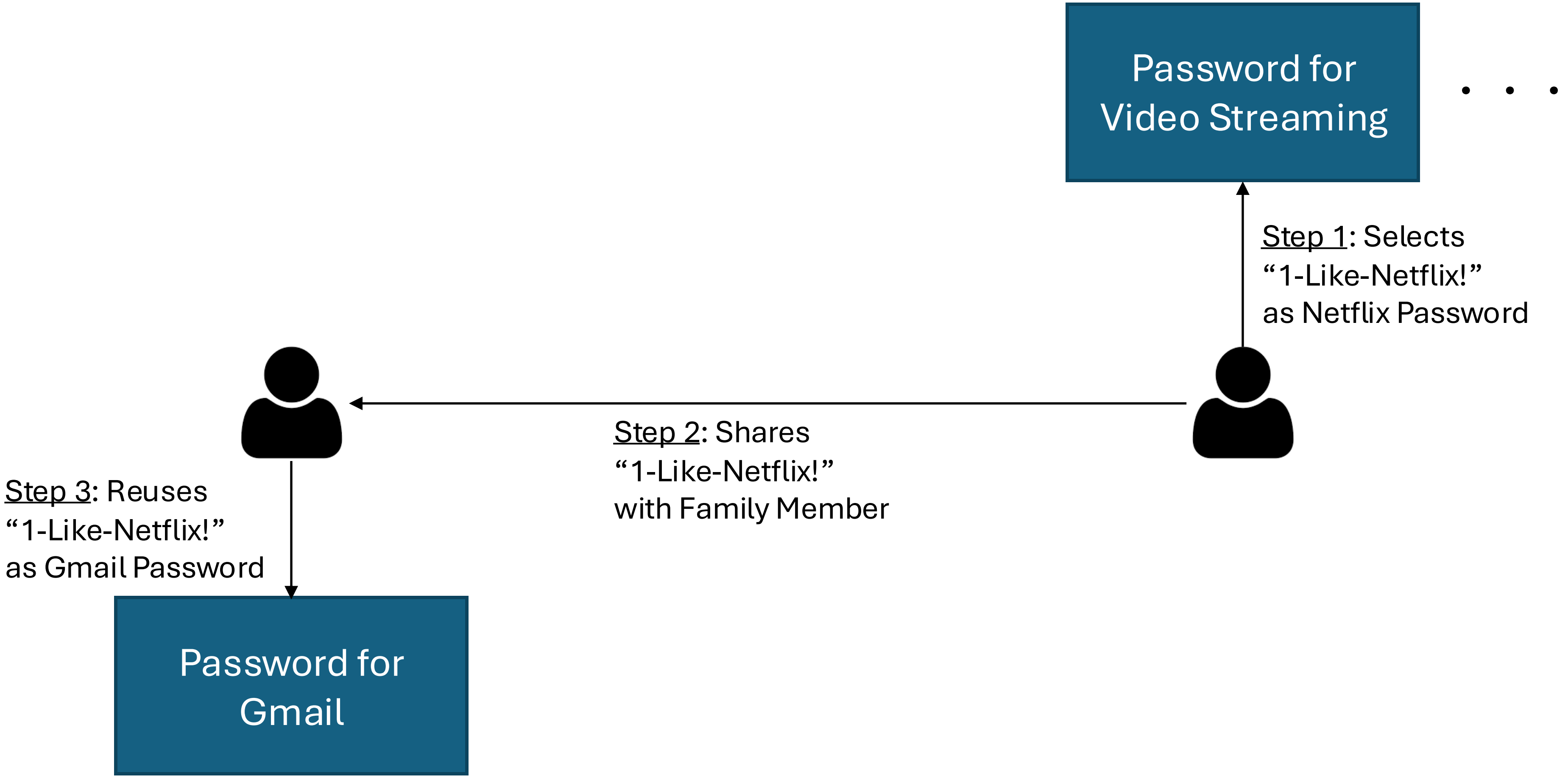
Password for Video Streaming

. . .

Step 1: Selects "1-Like-Netflix!" as Netflix Password

Step 2: Shares "1-Like-Netflix!" with Family Member

Inherent Threat of Password Reuse



Password for Video Streaming

. . .

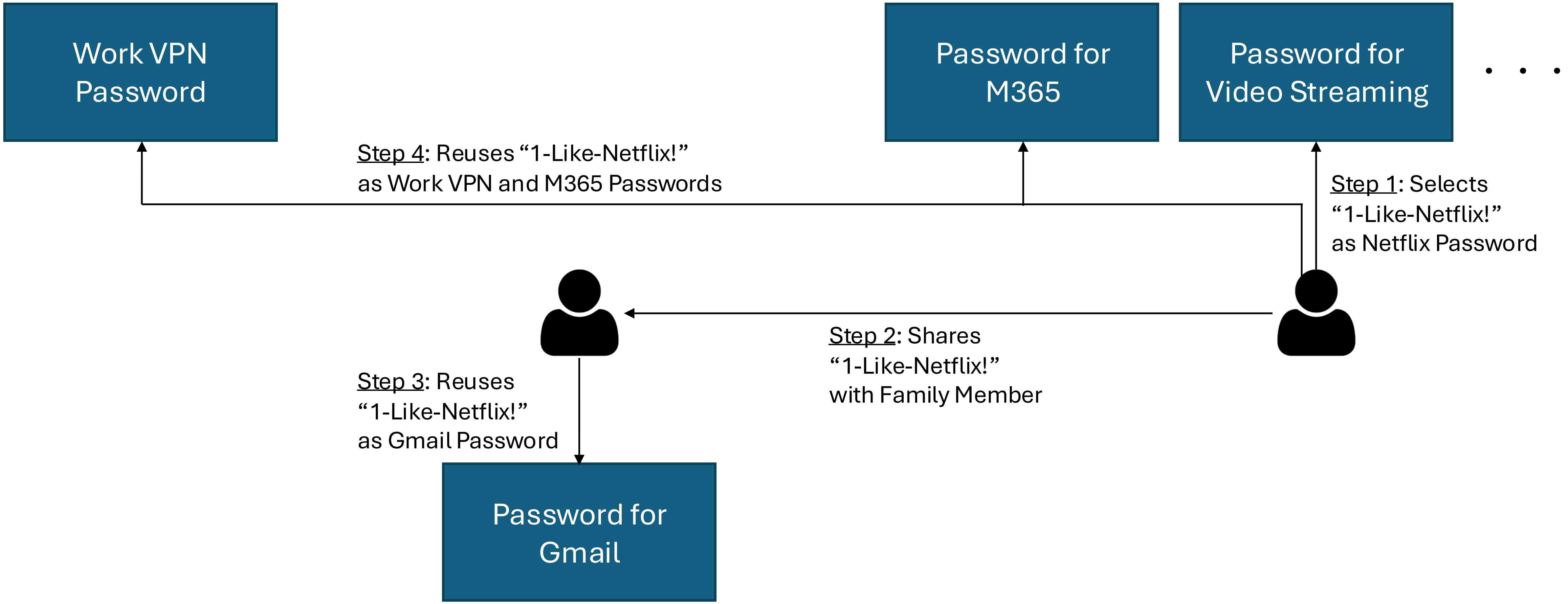
Step 1: Selects "1-Like-Netflix!" as Netflix Password

Step 2: Shares "1-Like-Netflix!" with Family Member

Step 3: Reuses "1-Like-Netflix!" as Gmail Password

Password for Gmail

Inherent Threat of Password Reuse



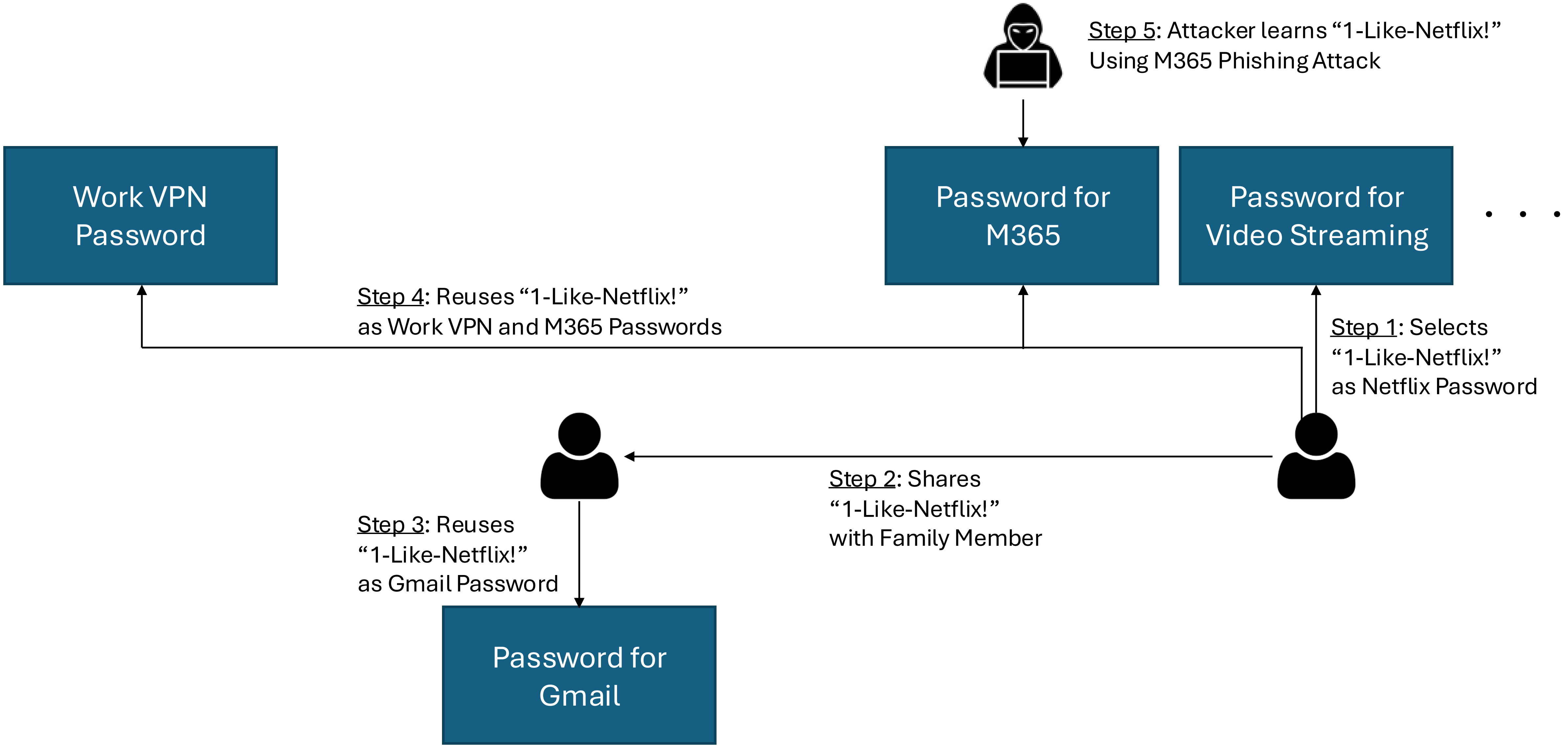
Step 1: Selects "1-Like-Netflix!" as Netflix Password

Step 2: Shares "1-Like-Netflix!" with Family Member

Step 3: Reuses "1-Like-Netflix!" as Gmail Password

Step 4: Reuses "1-Like-Netflix!" as Work VPN and M365 Passwords

Inherent Threat of Password Reuse



Step 5: Attacker learns “1-Like-Netflix!” Using M365 Phishing Attack

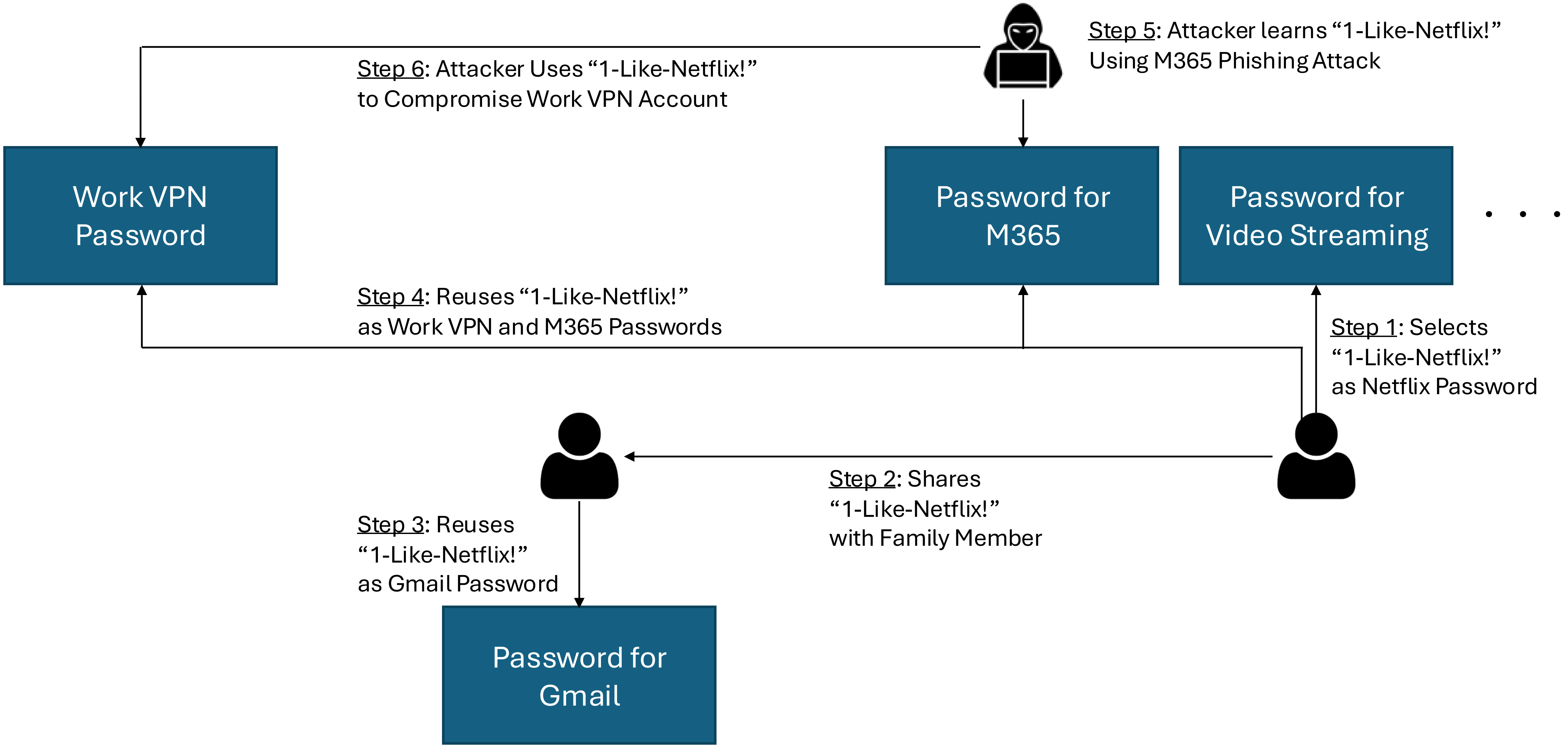
Step 4: Reuses “1-Like-Netflix!” as Work VPN and M365 Passwords

Step 1: Selects “1-Like-Netflix!” as Netflix Password

Step 2: Shares “1-Like-Netflix!” with Family Member

Step 3: Reuses “1-Like-Netflix!” as Gmail Password

Inherent Threat of Password Reuse



Step 5: Attacker learns “1-Like-Netflix!” Using M365 Phishing Attack

Step 6: Attacker Uses “1-Like-Netflix!” to Compromise Work VPN Account

Work VPN Password

Password for M365

Password for Video Streaming

. . .

Step 4: Reuses “1-Like-Netflix!” as Work VPN and M365 Passwords

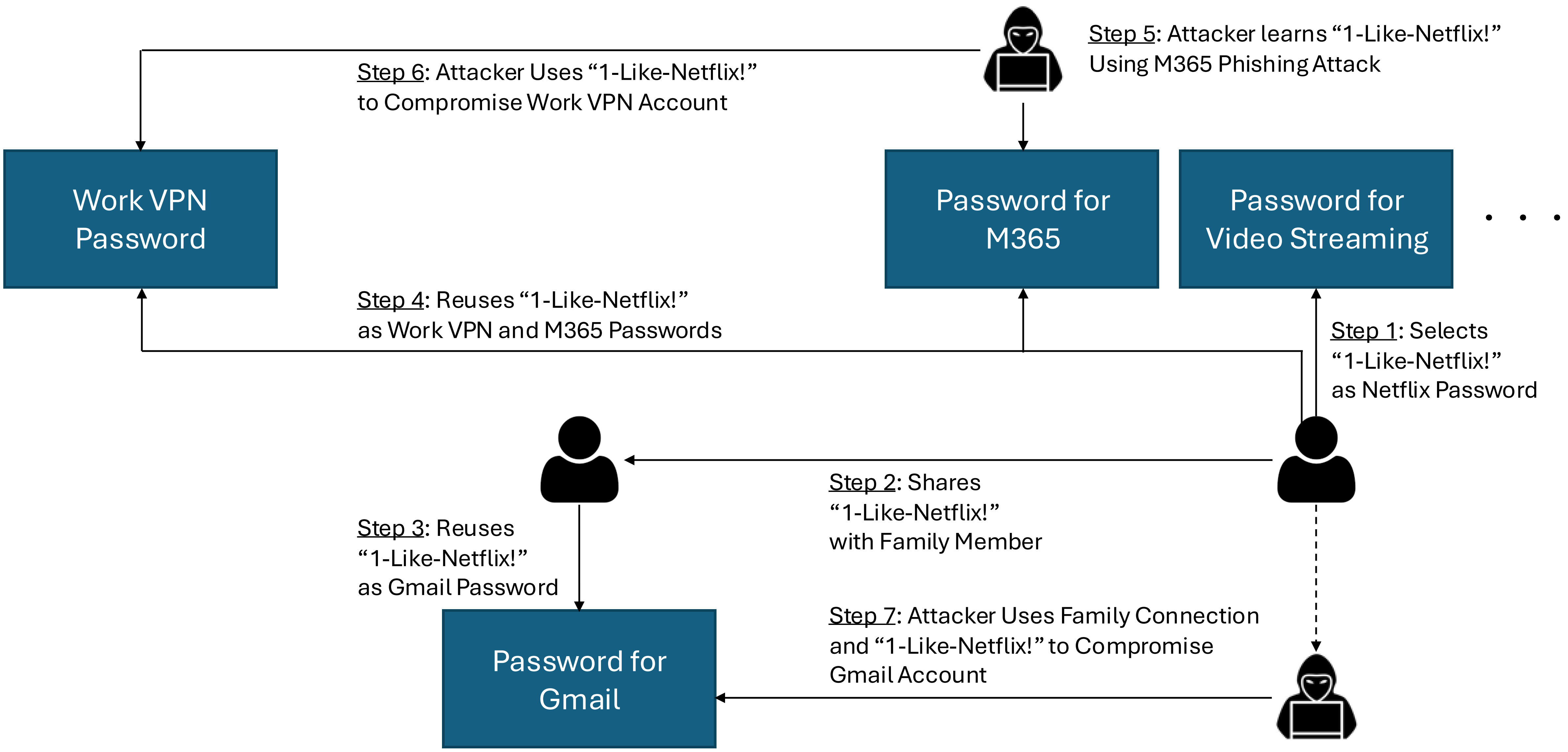
Step 1: Selects “1-Like-Netflix!” as Netflix Password

Step 2: Shares “1-Like-Netflix!” with Family Member

Step 3: Reuses “1-Like-Netflix!” as Gmail Password

Password for Gmail

Inherent Threat of Password Reuse



Step 5: Attacker learns “1-Like-Netflix!” Using M365 Phishing Attack

Step 6: Attacker Uses “1-Like-Netflix!” to Compromise Work VPN Account

Work VPN Password

Password for M365

Password for Video Streaming

. . .

Step 4: Reuses “1-Like-Netflix!” as Work VPN and M365 Passwords

Step 1: Selects “1-Like-Netflix!” as Netflix Password

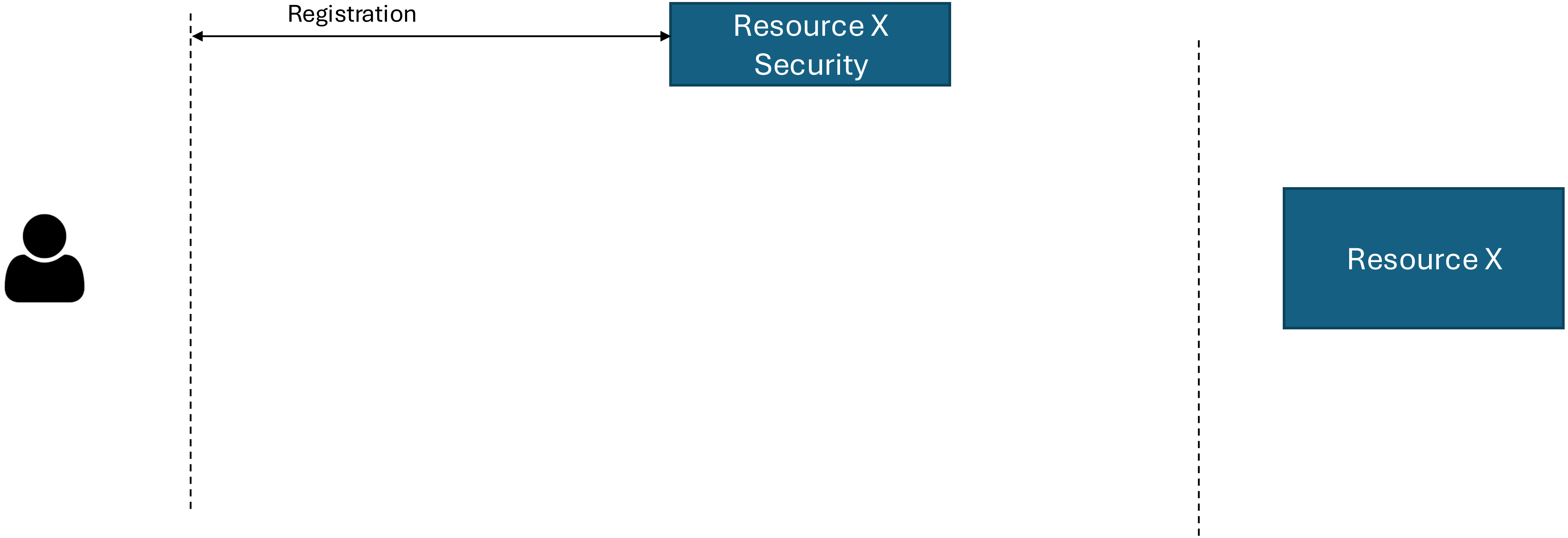
Step 2: Shares “1-Like-Netflix!” with Family Member

Step 3: Reuses “1-Like-Netflix!” as Gmail Password

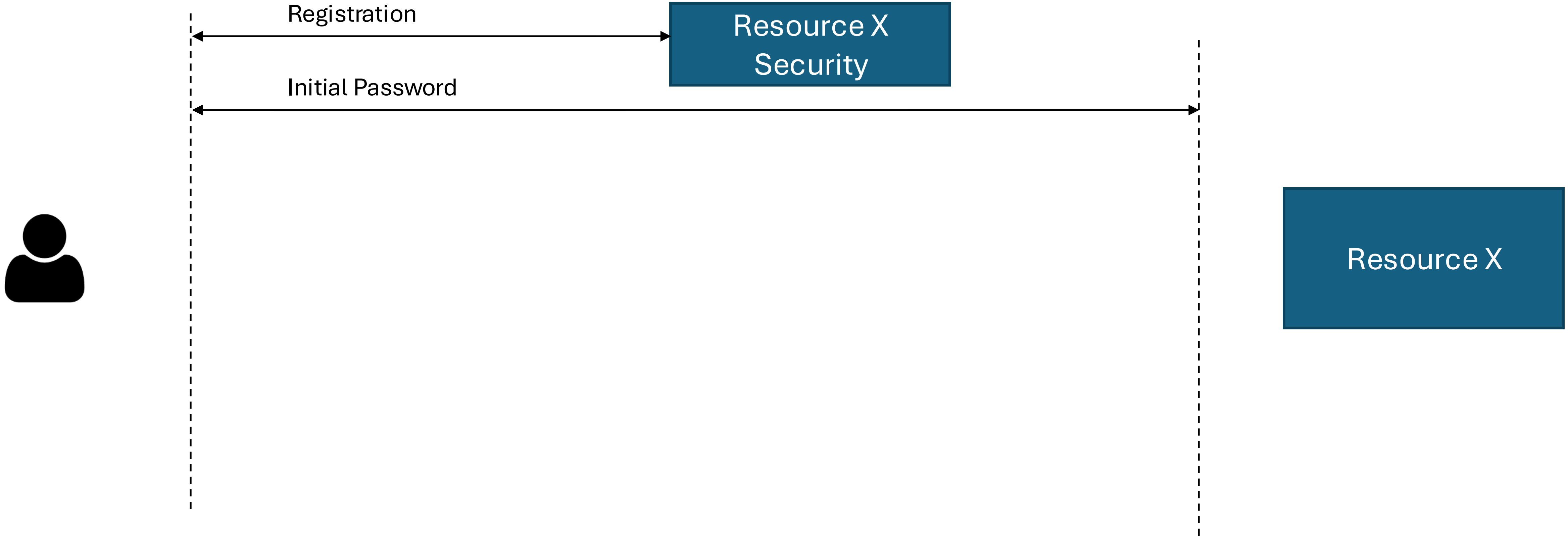
Step 7: Attacker Uses Family Connection and “1-Like-Netflix!” to Compromise Gmail Account

Password for Gmail

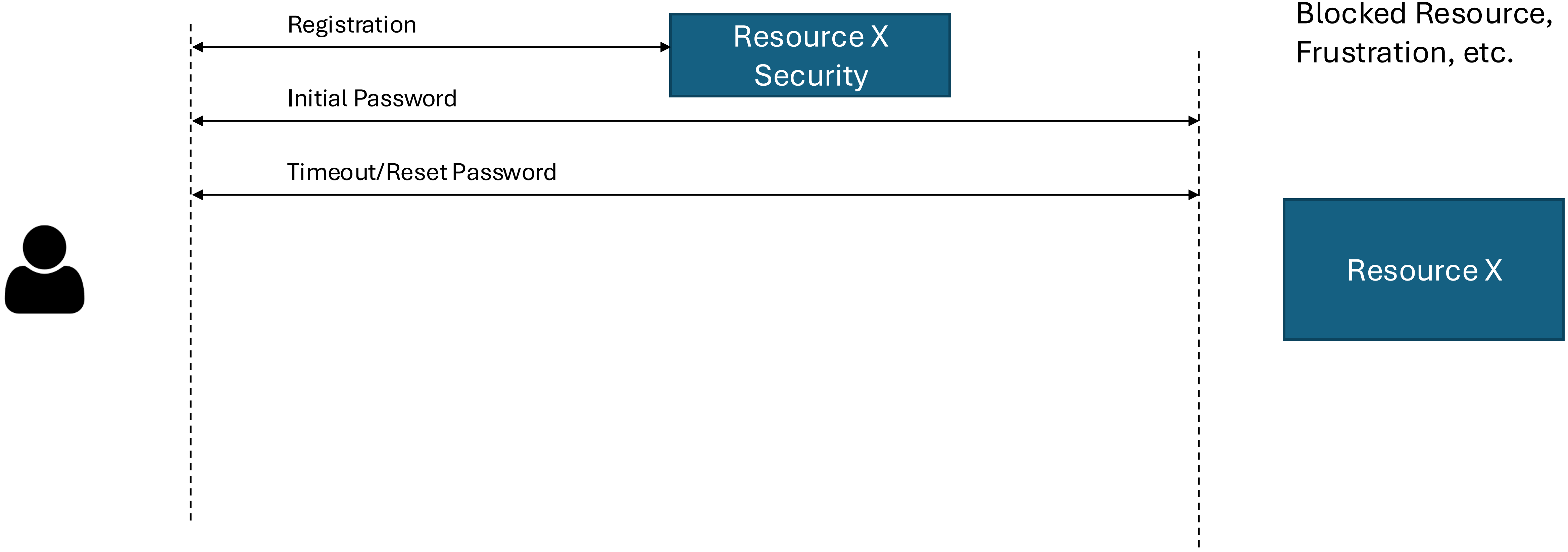
Inherent Friction from Password Usage



Inherent Friction from Password Usage

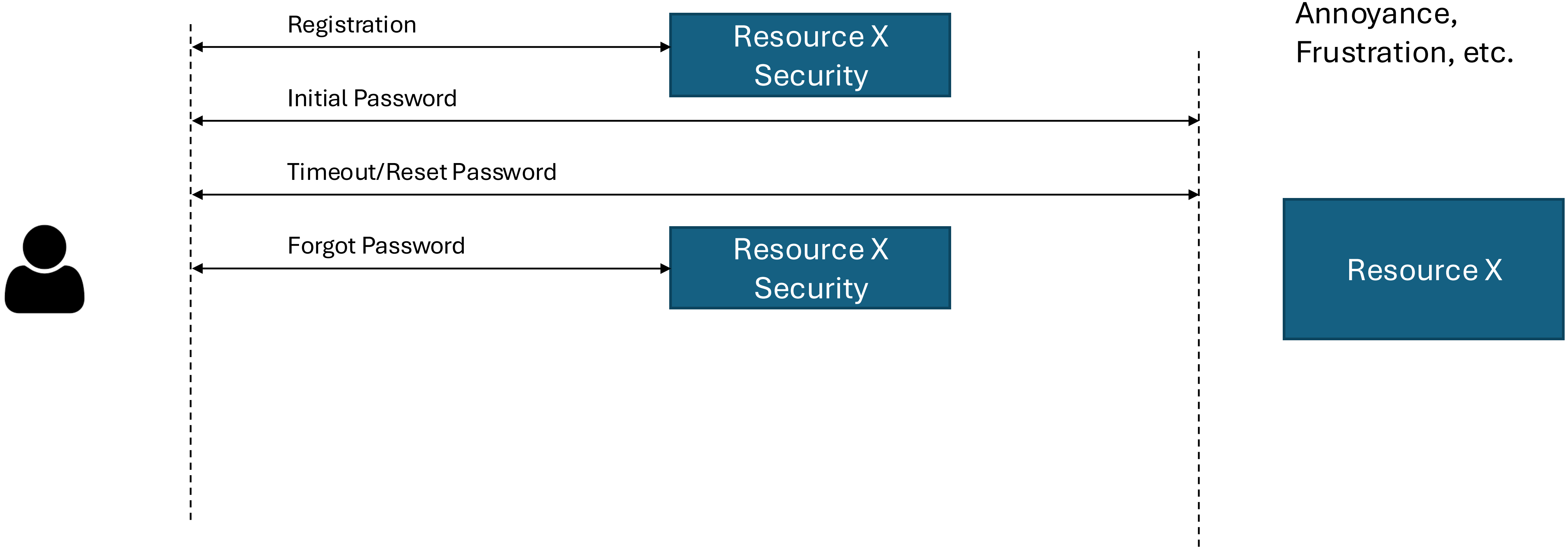


Inherent Friction from Password Usage



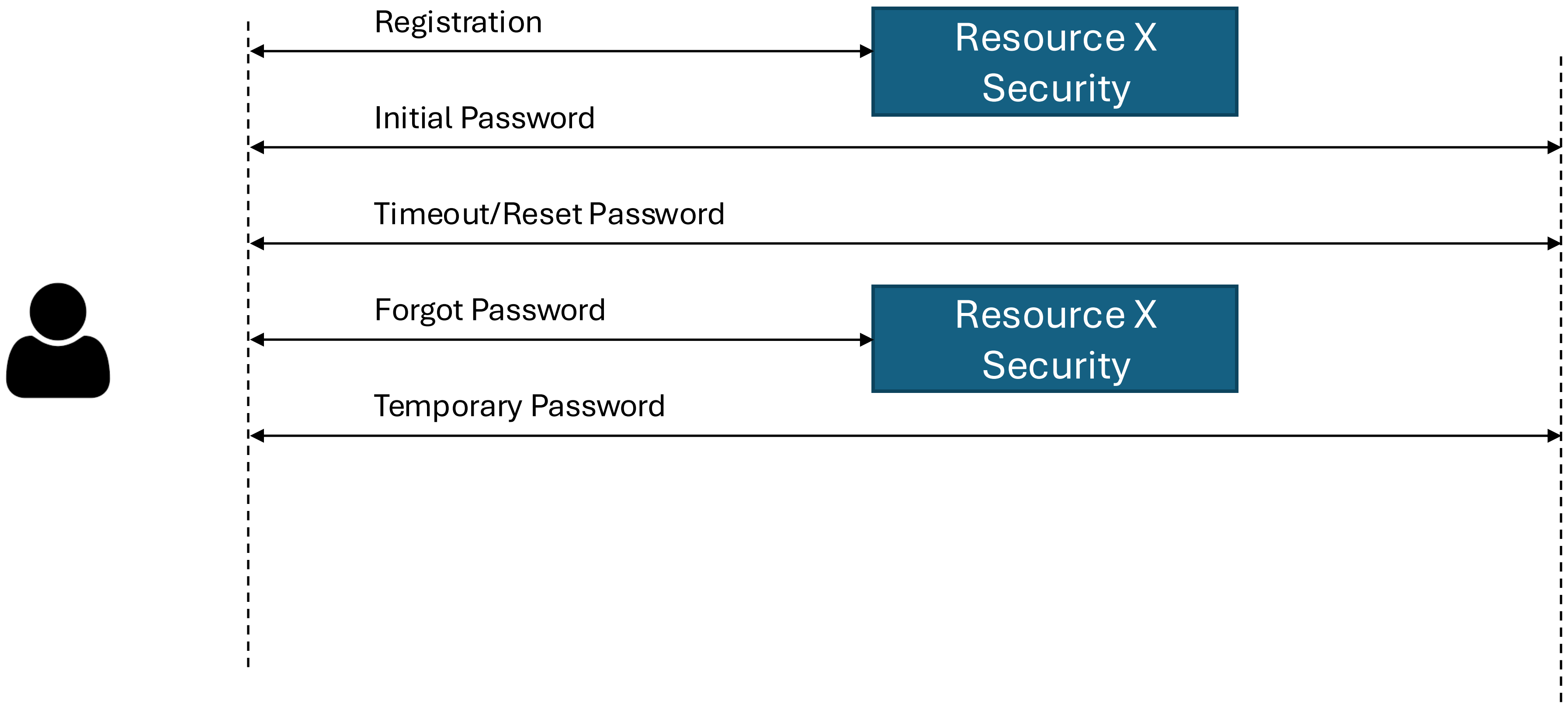
Friction:
Blocked Resource, Frustration, etc.

Inherent Friction from Password Usage



Friction:
Annoyance,
Frustration, etc.

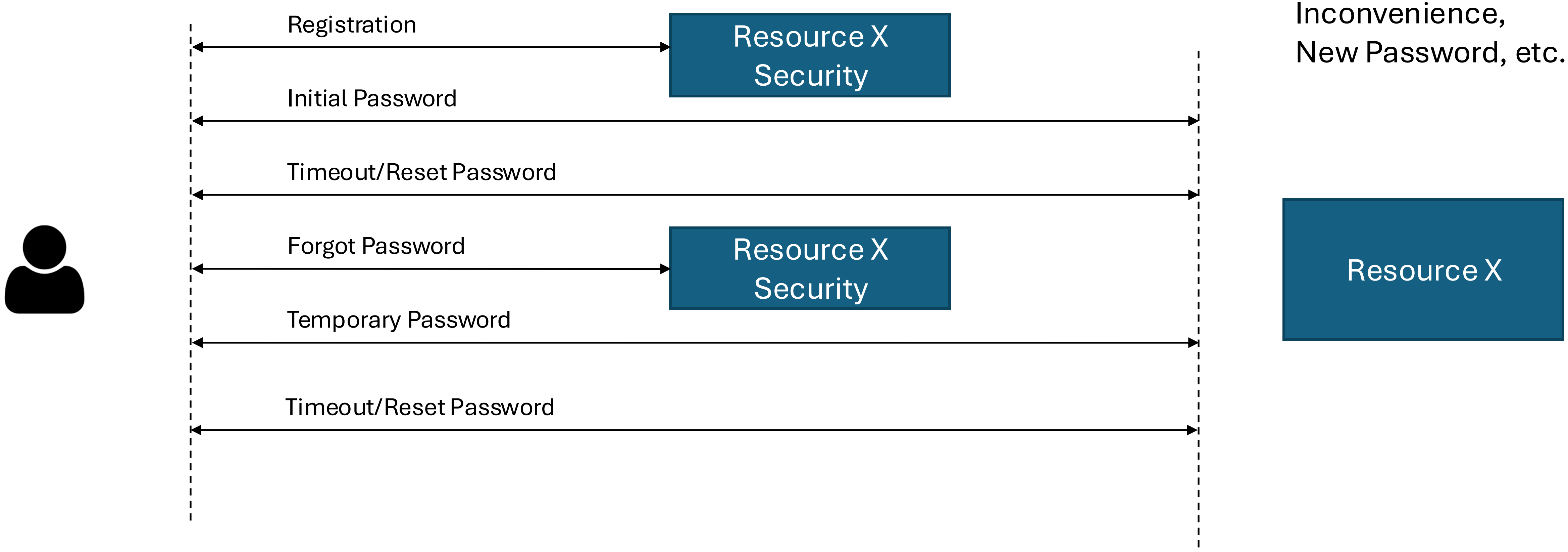
Inherent Friction from Password Usage



Friction:
Inconvenience,
New Password, etc.

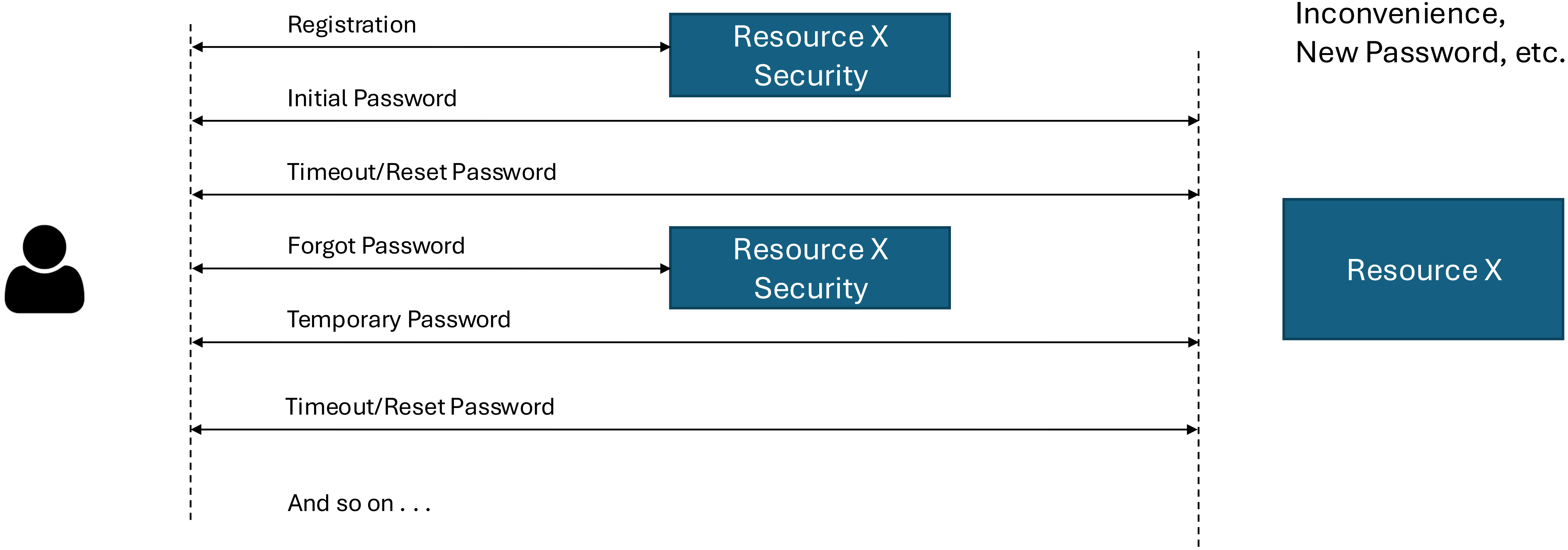
Inherent Friction from Password Usage

Friction:
Inconvenience,
New Password, etc.



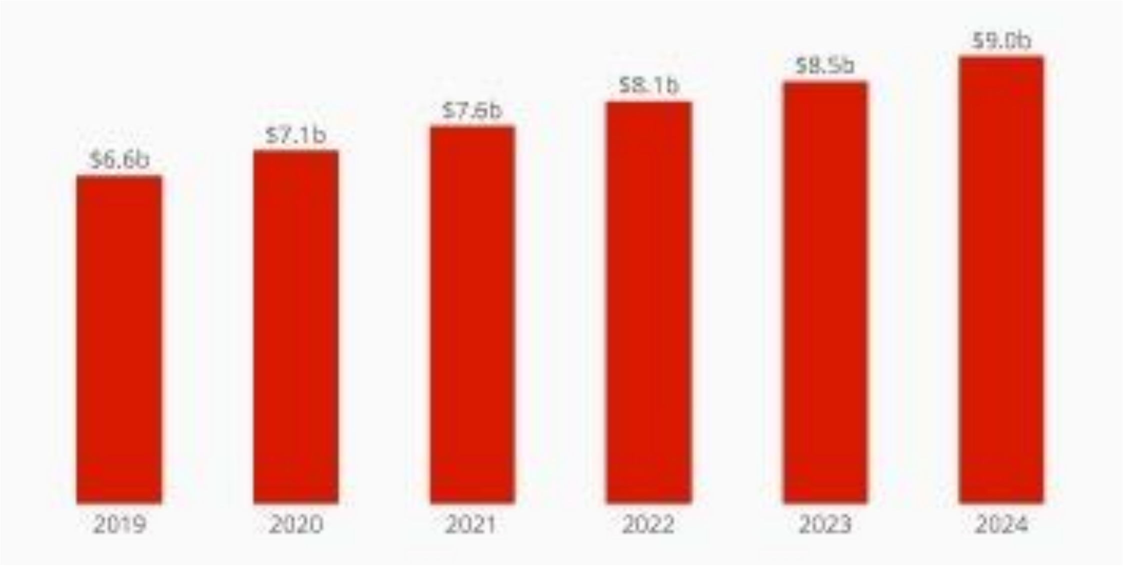
Inherent Friction from Password Usage

Friction:
Inconvenience,
New Password, etc.



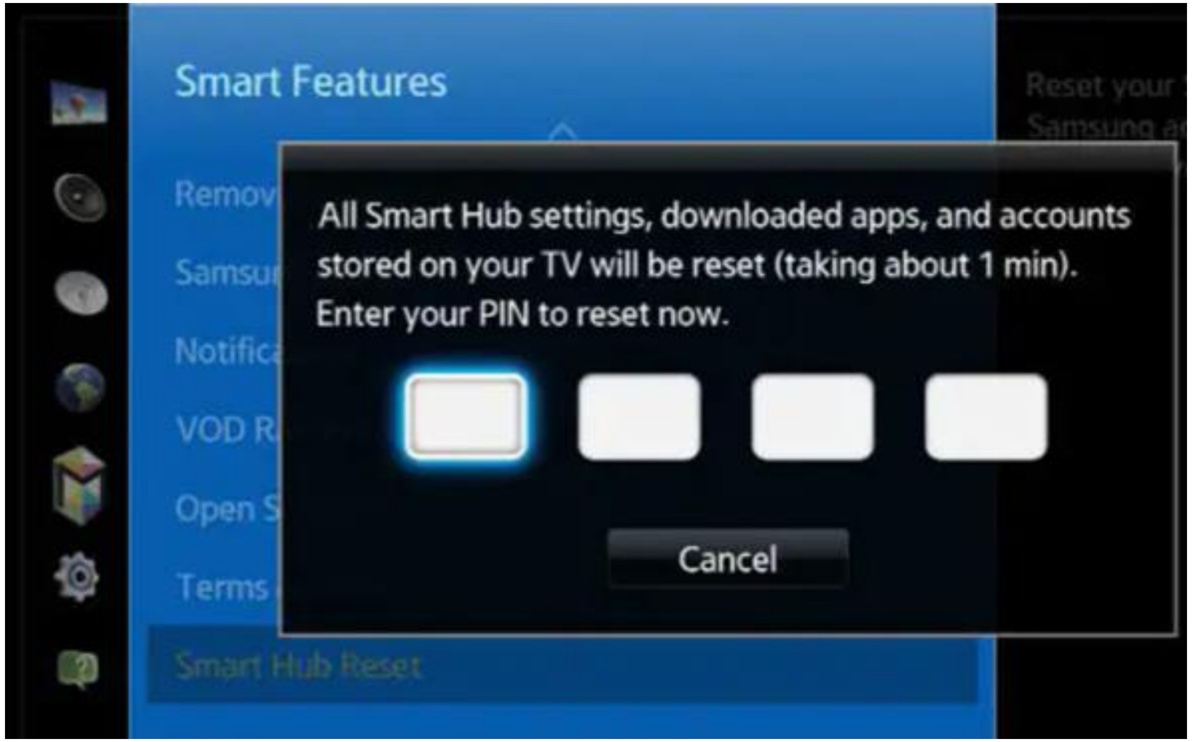
Password Issues with Smart TV/Streaming Channels

Estimated Revenue Losses for US Pay TV Industry from Piracy and Account Sharing



Source: Statistica

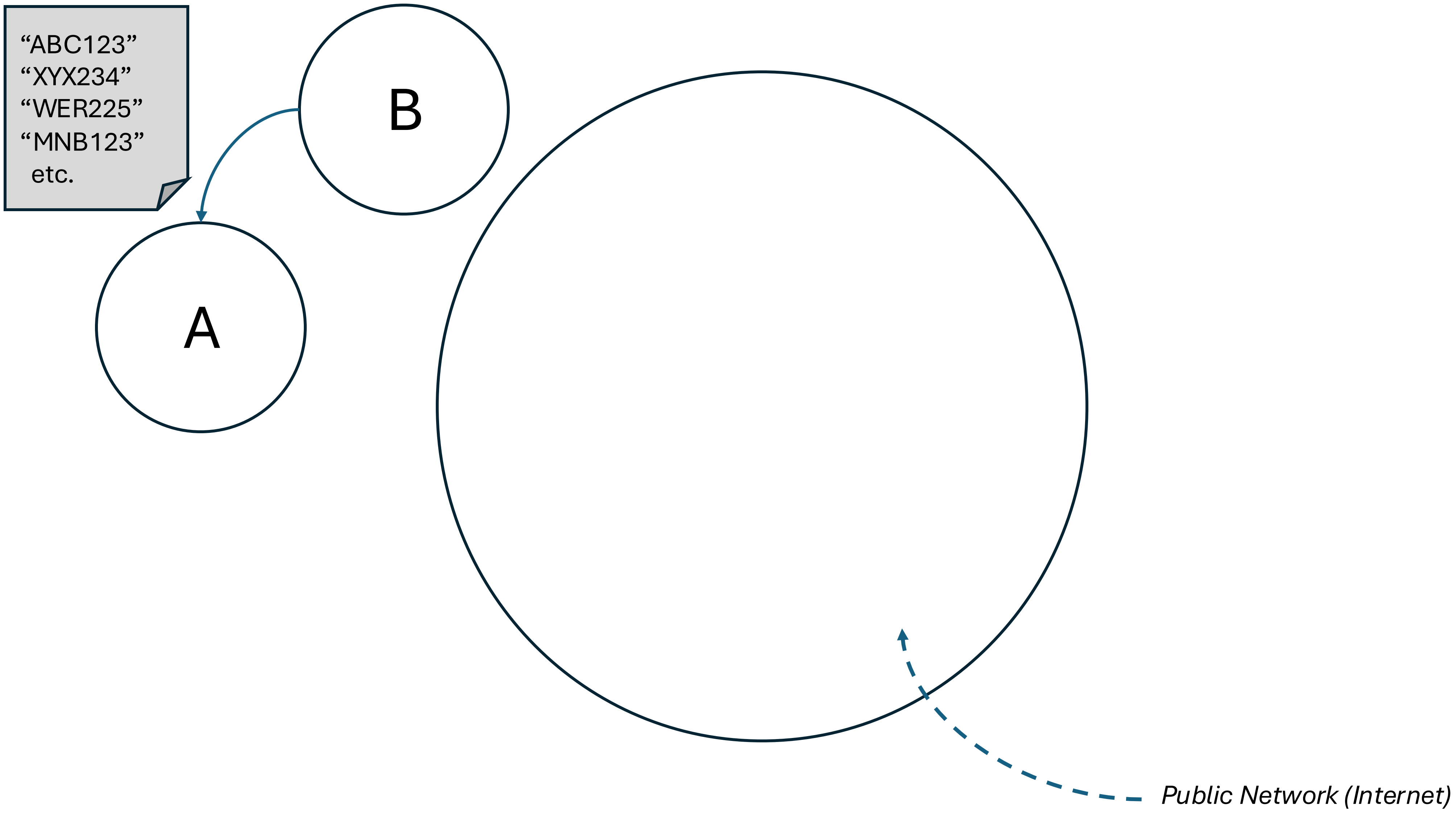
https://www.statista.com/chart/19914/estimated-revenue-loss-for-the-us-pay-tv-industry-from-piracy-and-account-sharing/



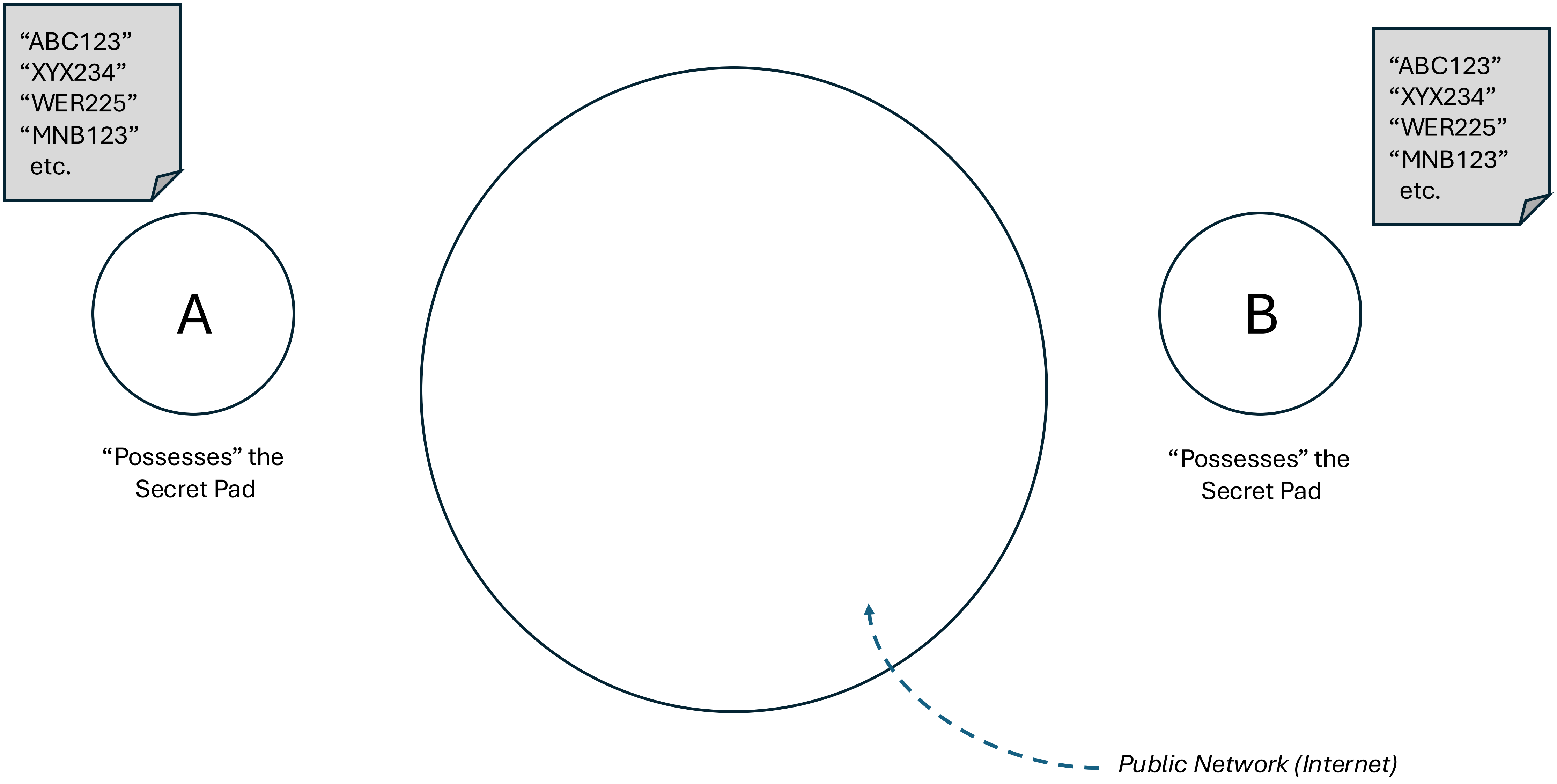
All Smart Hub settings, downloaded apps, and accounts stored on your TV will be reset (taking about 1 min). Enter your PIN to reset now.

Can we make the secret word a non-reusable, one-time password (OTP)?

Provide A with a
One Time Pad of
Secret Words



“ABC123”
“XYX234”
“WER225”
“MNB123”
etc.



"ABC123"
"XYX234"
"WER225"
"MNB123"
etc.

A

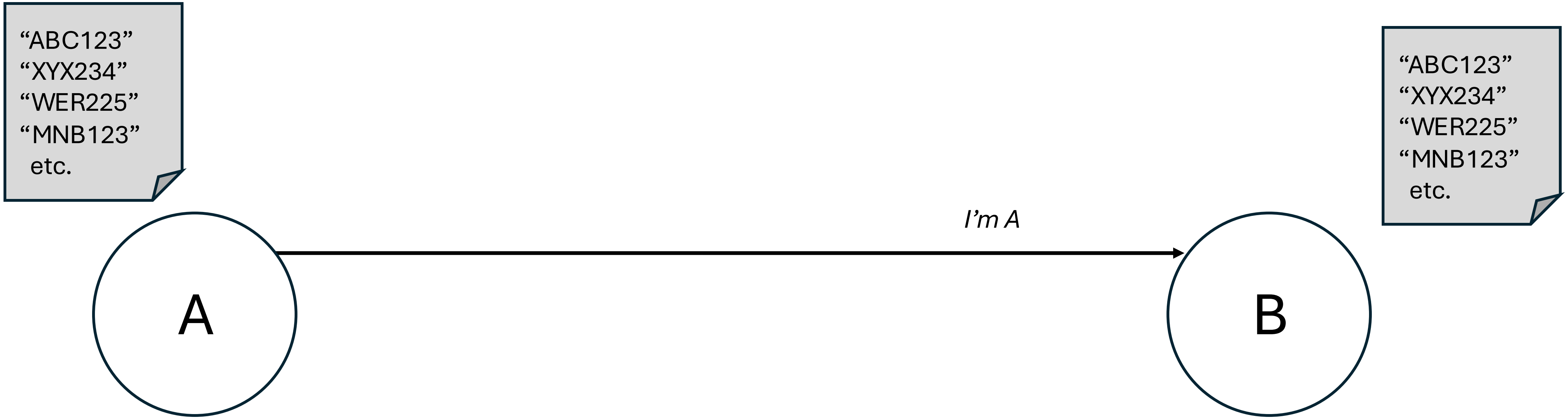
"Possesses" the
Secret Pad

"ABC123"
"XYX234"
"WER225"
"MNB123"
etc.

B

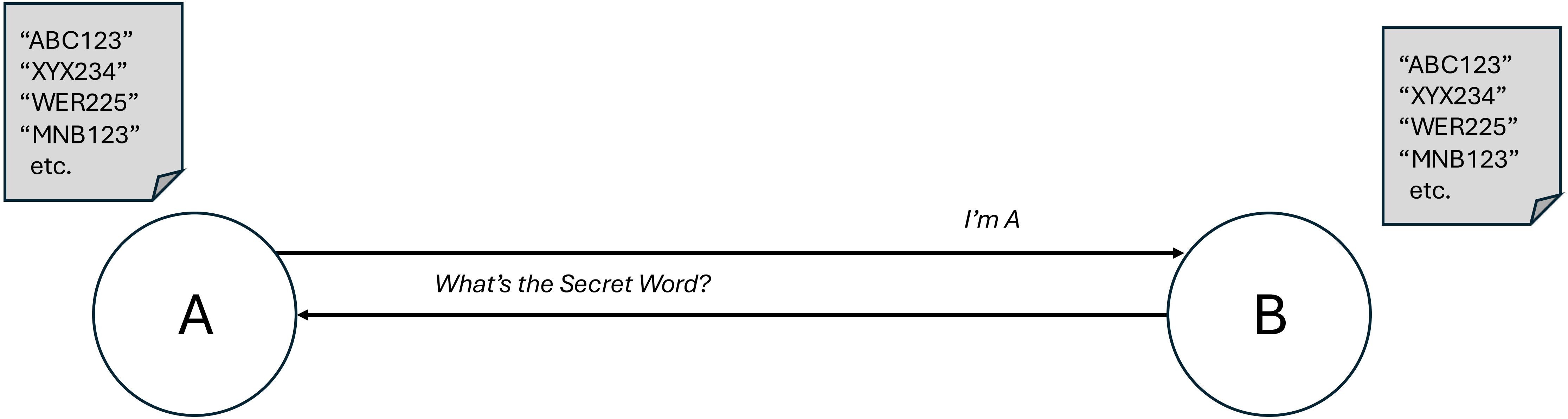
"Possesses" the
Secret Pad

Public Network (Internet)



"ABC123"
"XYX234"
"WER225"
"MNB123"
etc.

"ABC123"
"XYX234"
"WER225"
"MNB123"
etc.



“ABC123”
“XYX234”
“WER225”
“MNB123”
etc.

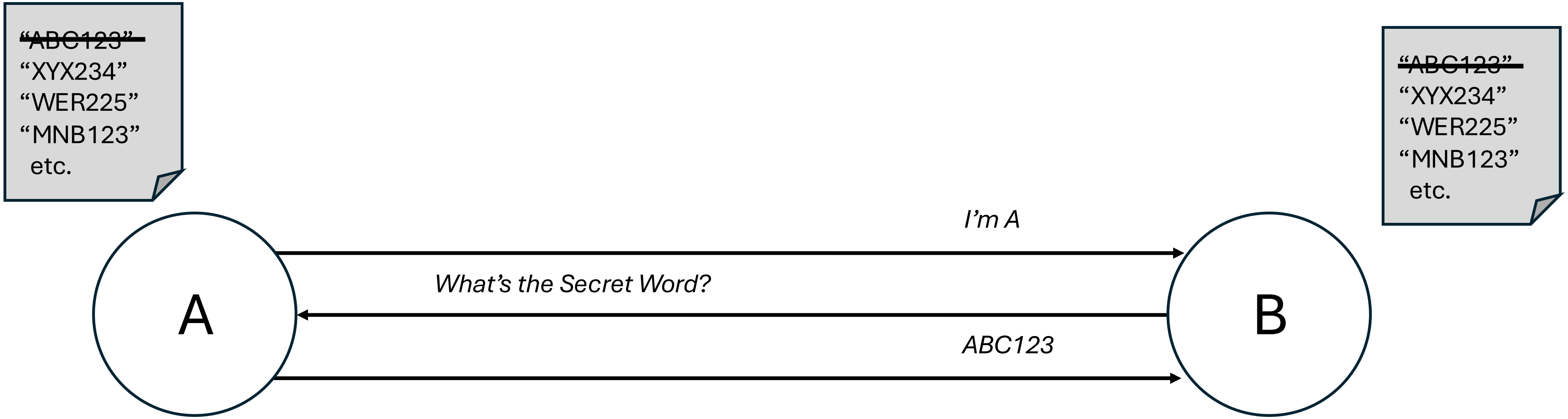
I’m A

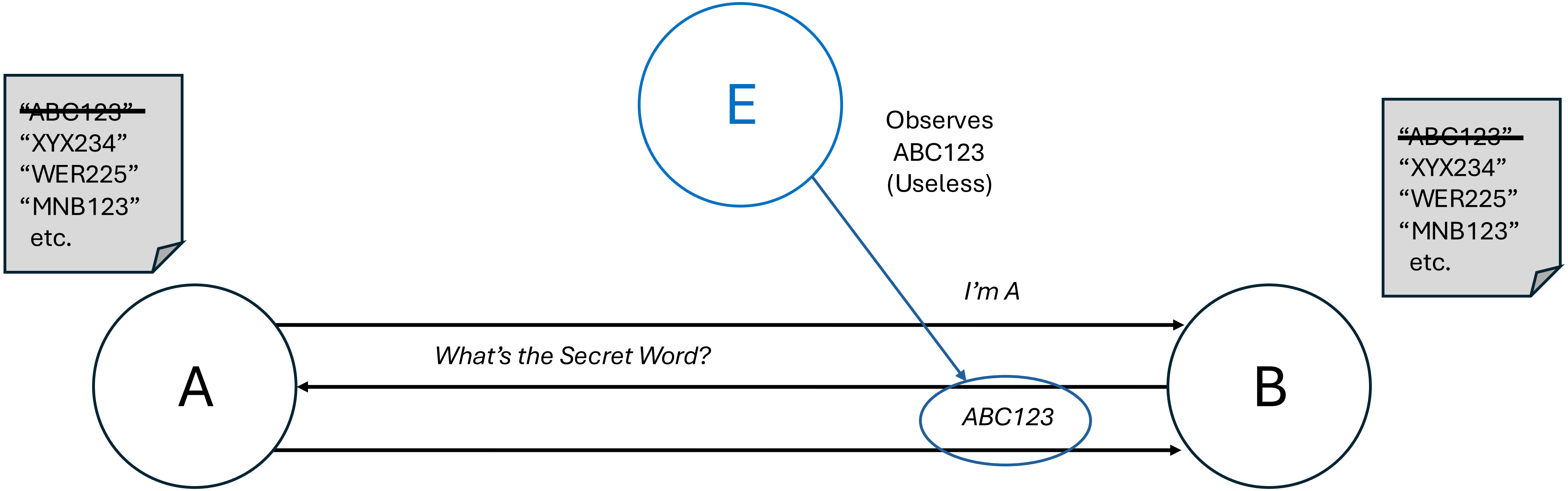
What’s the Secret Word?

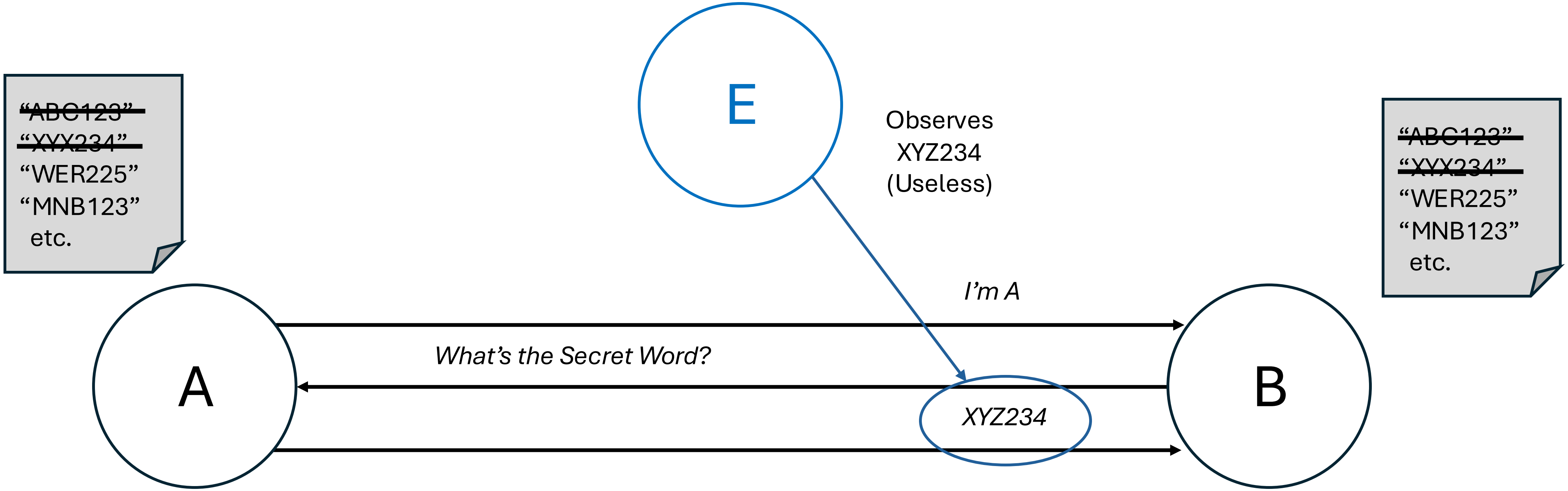
A

B

“ABC123”
“XYX234”
“WER225”
“MNB123”
etc.


~~"ABC123"~~
"XYX234"
"WER225"
"MNB123"
etc.
~~"ABC123"~~
"XYX234"
"WER225"
"MNB123"
etc.
A
B
I'm A
What's the Secret Word?
ABC123


~~“ABC123”~~
“XYX234”
“WER225”
“MNB123”
etc.
E
Observes
ABC123
(Useless)
~~“ABC123”~~
“XYX234”
“WER225”
“MNB123”
etc.
I’m A
What’s the Secret Word?
A
B
ABC123



~~“ABC123”~~
~~“XYX234”~~
“WER225”
“MNB123”
etc.

E

Observes
XYZ234
(Useless)

~~“ABC123”~~
~~“XYX234”~~
“WER225”
“MNB123”
etc.

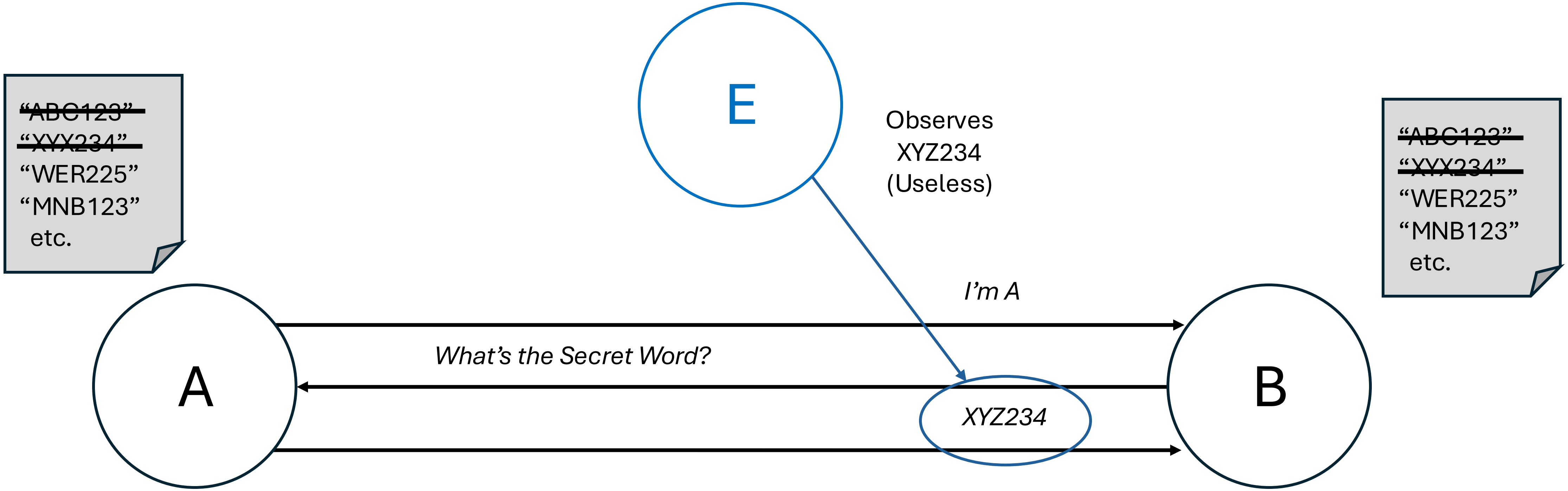
I’m A

What’s the Secret Word?

XYZ234

A

B



~~“ABC123”~~
~~“XYX234”~~
“WER225”
“MNB123”
etc.

E

Observes
XYZ234
(Useless)

~~“ABC123”~~
~~“XYX234”~~
“WER225”
“MNB123”
etc.

A

I’m A

What’s the Secret Word?

XYZ234

B

This is the One Time Pad Protocol

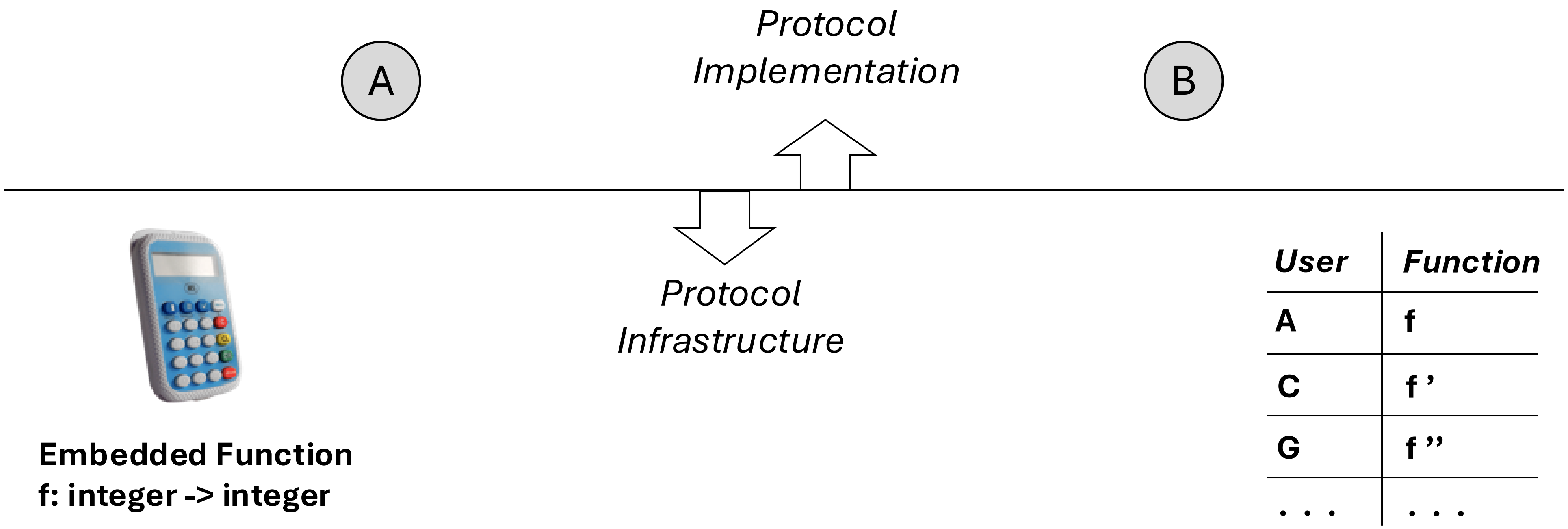
How else can we utilize non-reusable, one-time passwords (OTPs)?

A

B

Embedded Function
f: integer -> integer

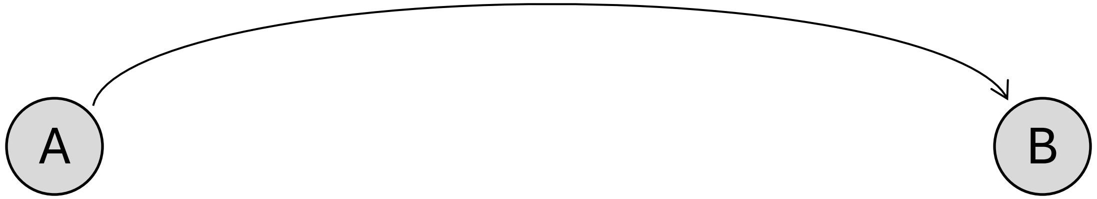
User	*Function*
A	f
C	f ’
G	f ’’
. . .	. . .



Embedded Function
f: integer -> integer

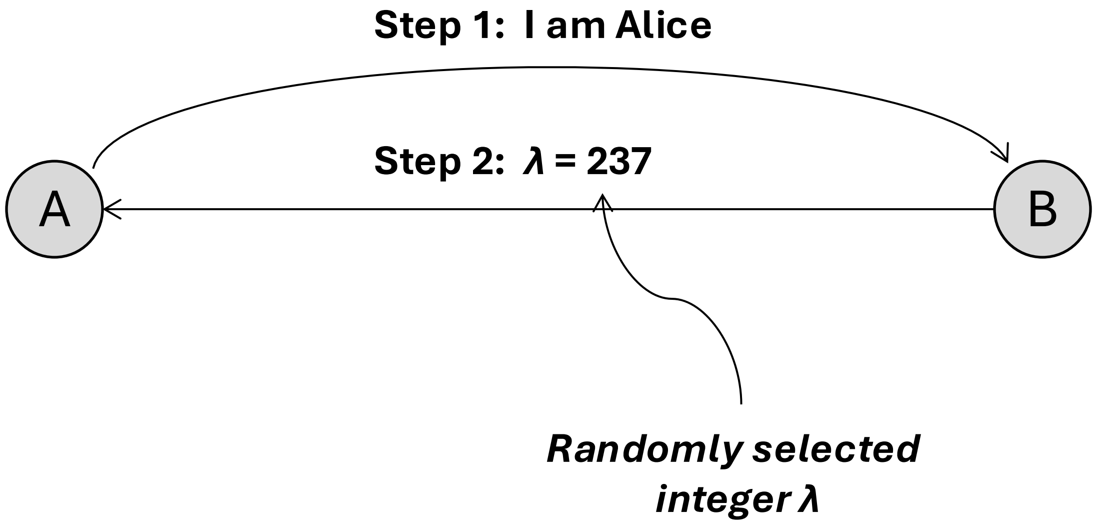
User	***Function***
A	**f**
C	**f ’**
G	**f ’’**
. . .	. . .

Step 1: I am Alice



Embedded Function
f: integer -> integer

User	*Function*
A	**f**
C	**f ’**
G	**f ’’**
. . .	. . .



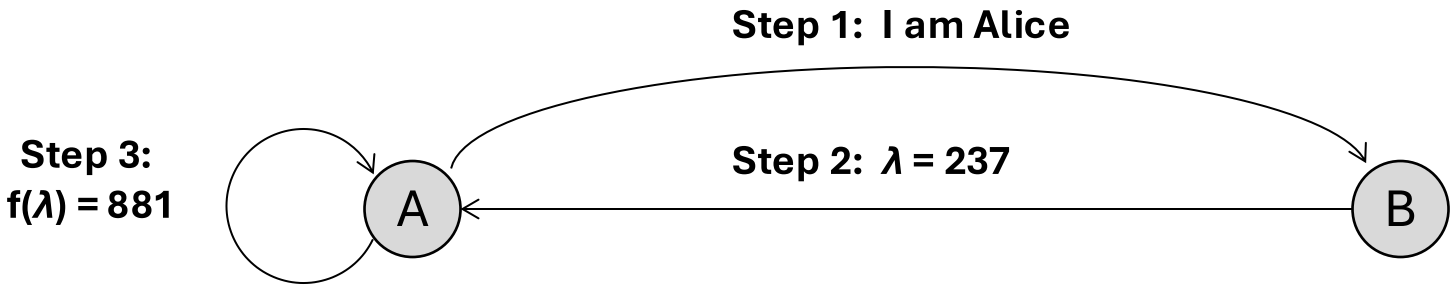
Step 1: I am Alice

Step 2: *λ* = 237

Randomly selected integer λ

Embedded Function
f: integer -> integer

User	*Function*
A	f
C	f '
G	f ''
. . .	. . .



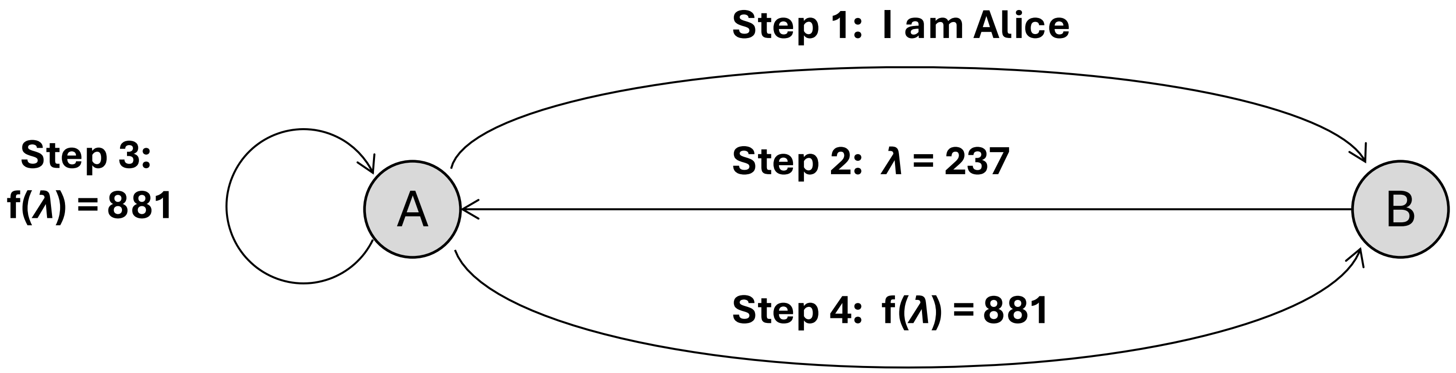
Step 1: I am Alice

Step 2: $\lambda = 237$

Step 3:
$f(\lambda) = 881$

Embedded Function
f: integer -> integer

User	*Function*
A	f
C	f ’
G	f ’’
. . .	. . .



Step 1: I am Alice

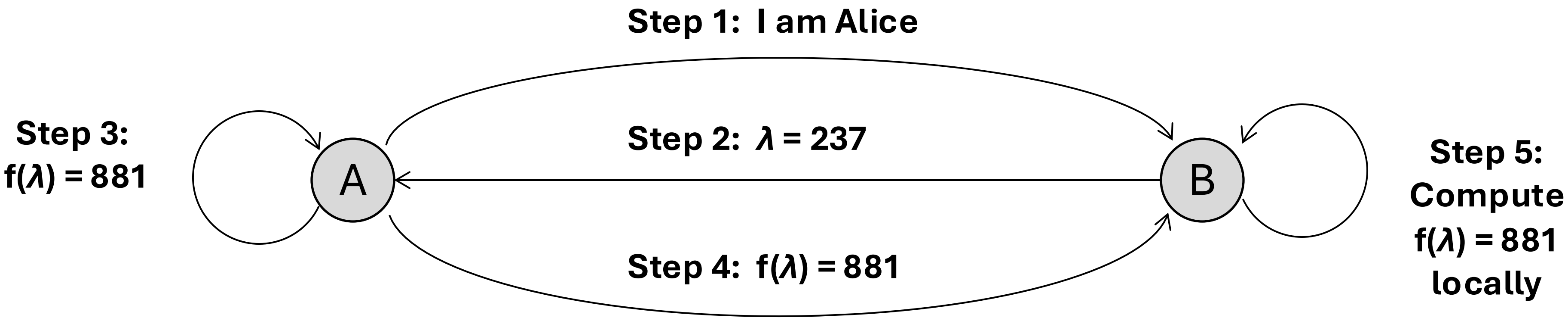
Step 2: λ = 237

Step 3:
f(λ) = 881

Step 4: f(λ) = 881

Embedded Function
f: integer -> integer

User	*Function*
A	f
C	f ’
G	f ’’
. . .	. . .



Step 1: I am Alice

Step 2: λ = 237

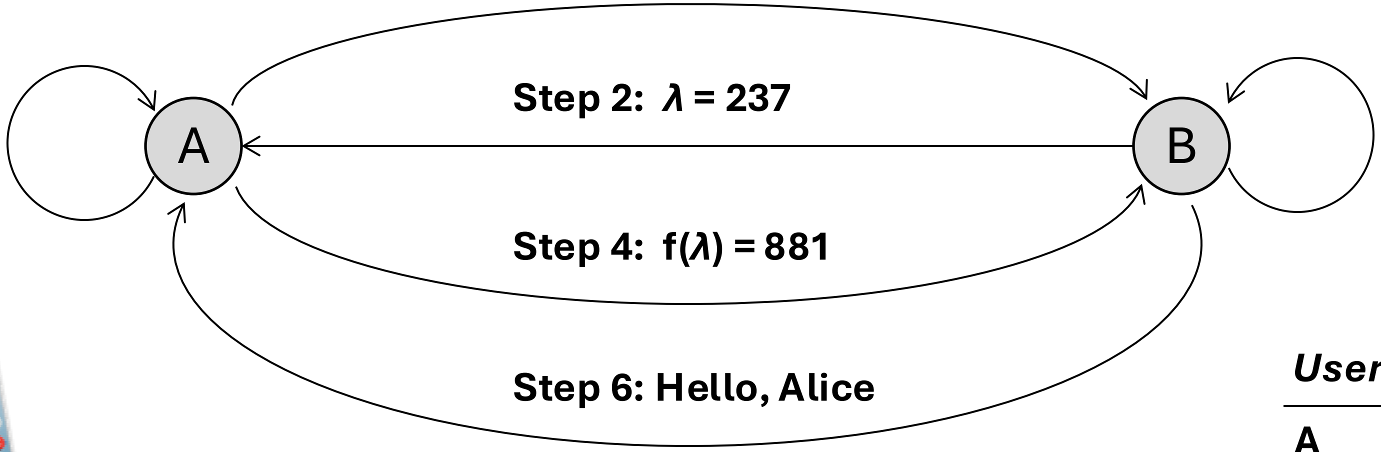
Step 3:
f(λ) = 881

Step 4: f(λ) = 881

Step 5:
Compute
f(λ) = 881
locally

Embedded Function
f: integer -> integer

User	*Function*
A	f
C	f ’
G	f ’’
. . .	. . .



Step 1: I am Alice

Step 2: λ = 237

Step 3:
f(λ) = 881

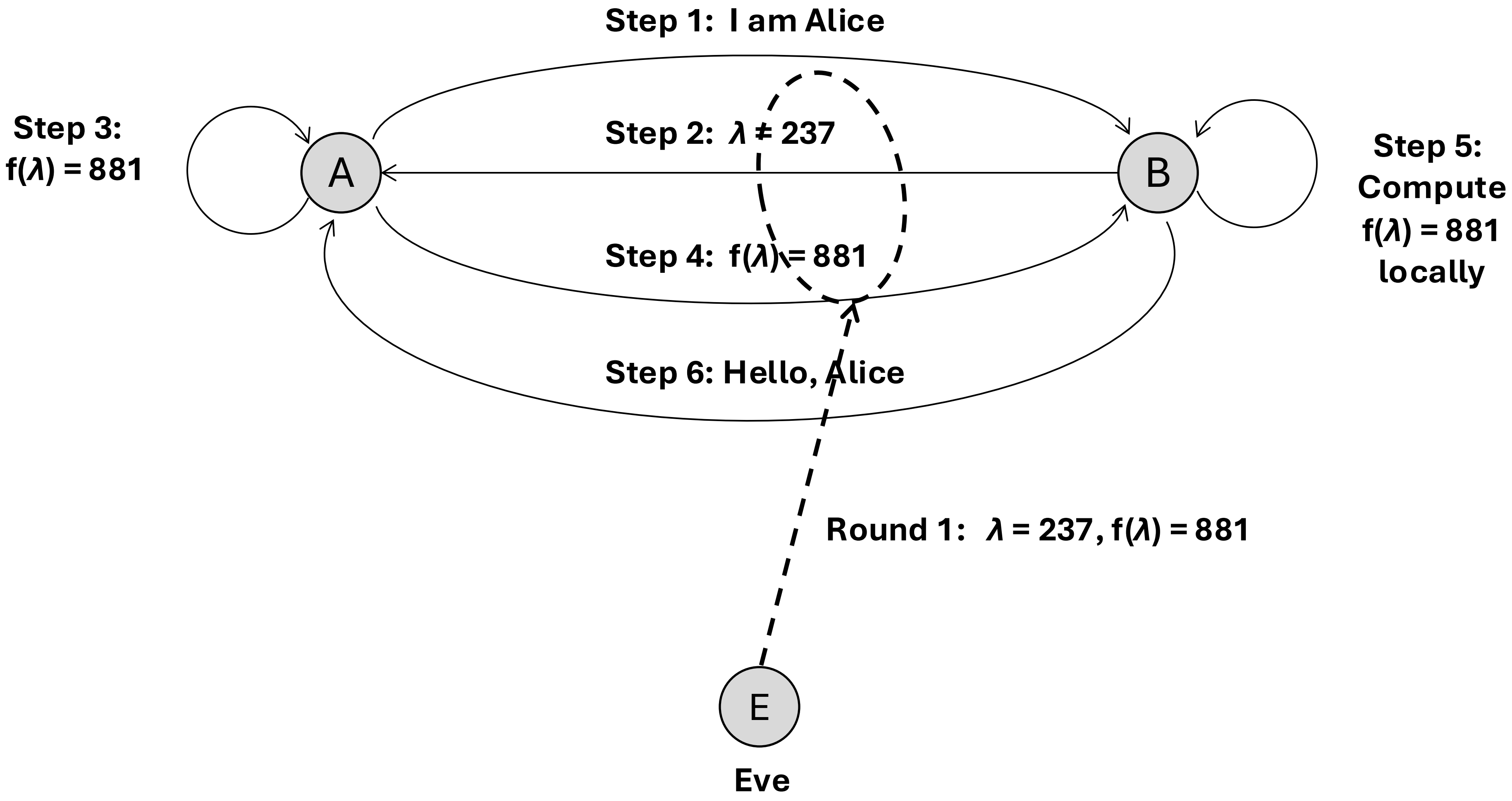
Step 4: f(λ) = 881

Step 5:
Compute
f(λ) = 881
locally

Step 6: Hello, Alice

Embedded Function
f: integer -> integer

User	*Function*
A	f
C	f ’
G	f ’’
. . .	. . .



Step 1: I am Alice

Step 2: λ = 237

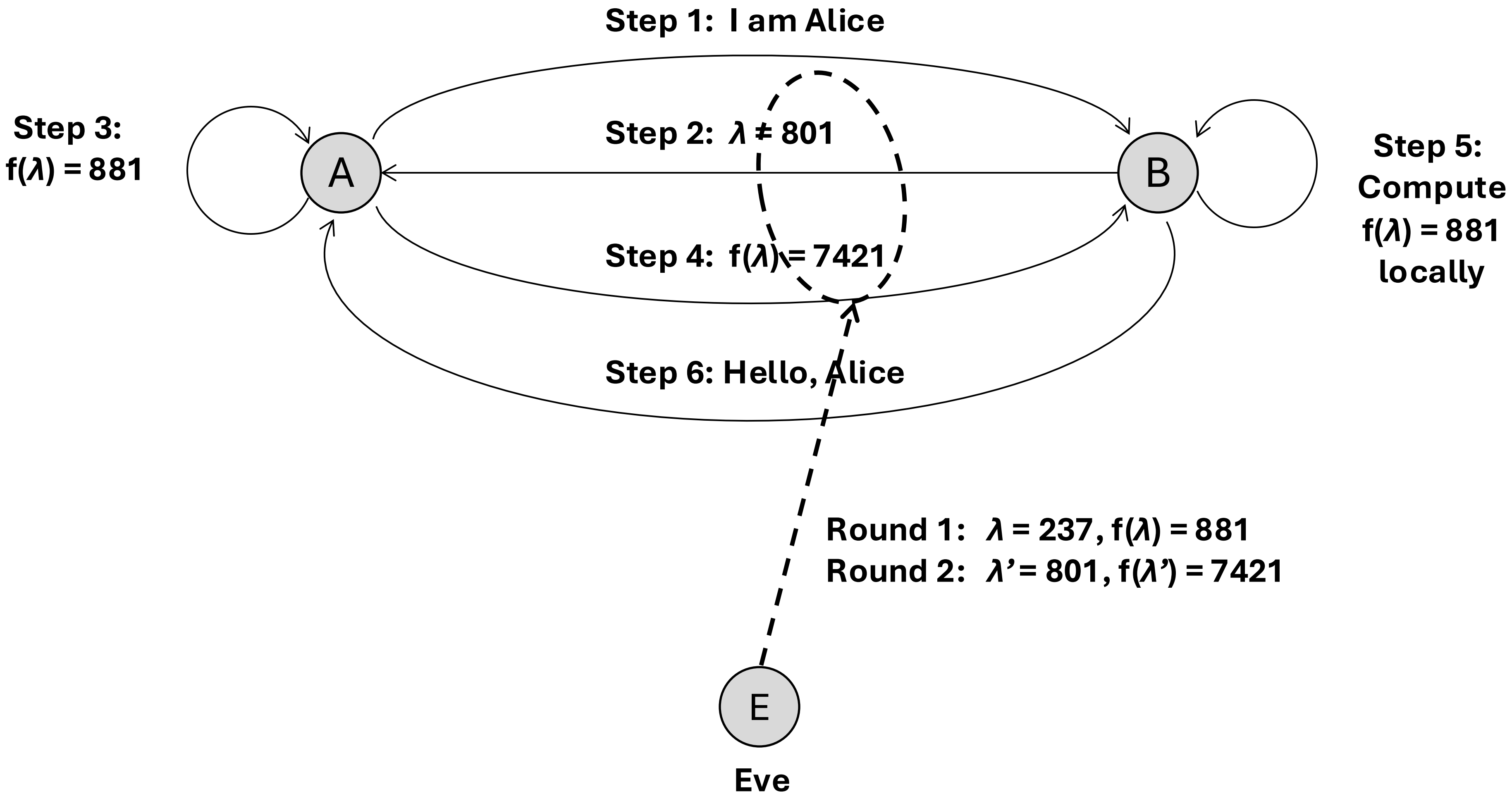
Step 3: f(λ) = 881

Step 4: f(λ) = 881

Step 5: Compute f(λ) = 881 locally

Step 6: Hello, Alice

Round 1: λ = 237, f(λ) = 881



Step 1: I am Alice

Step 2: λ = 801

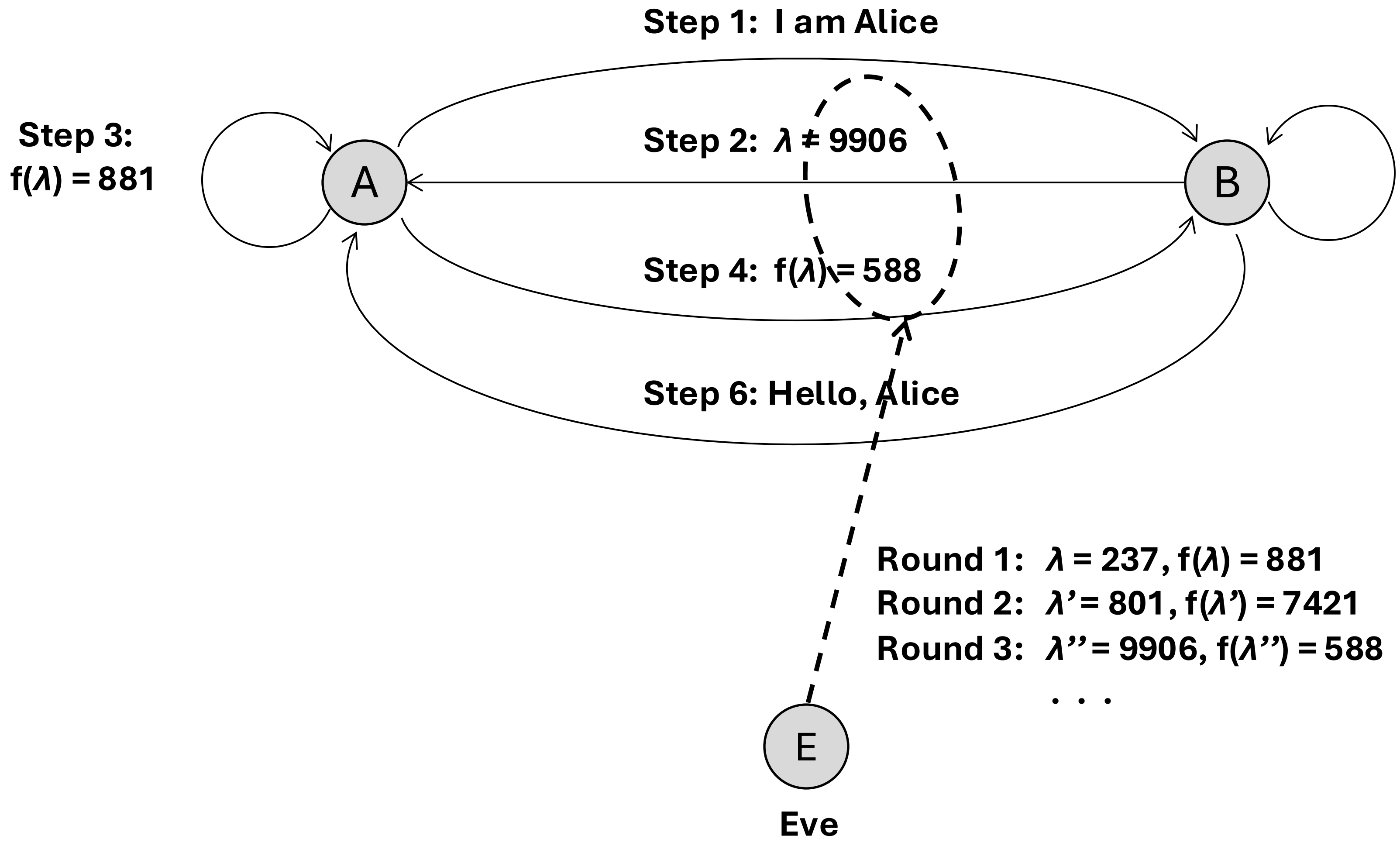
Step 3: f(λ) = 881

Step 4: f(λ) = 7421

Step 5: Compute f(λ) = 881 locally

Step 6: Hello, Alice

Round 1: λ = 237, f(λ) = 881
Round 2: λ’ = 801, f(λ’) = 7421



Step 1: I am Alice

Step 3:
f(*λ*) = 881

Step 2: *λ* = 9906

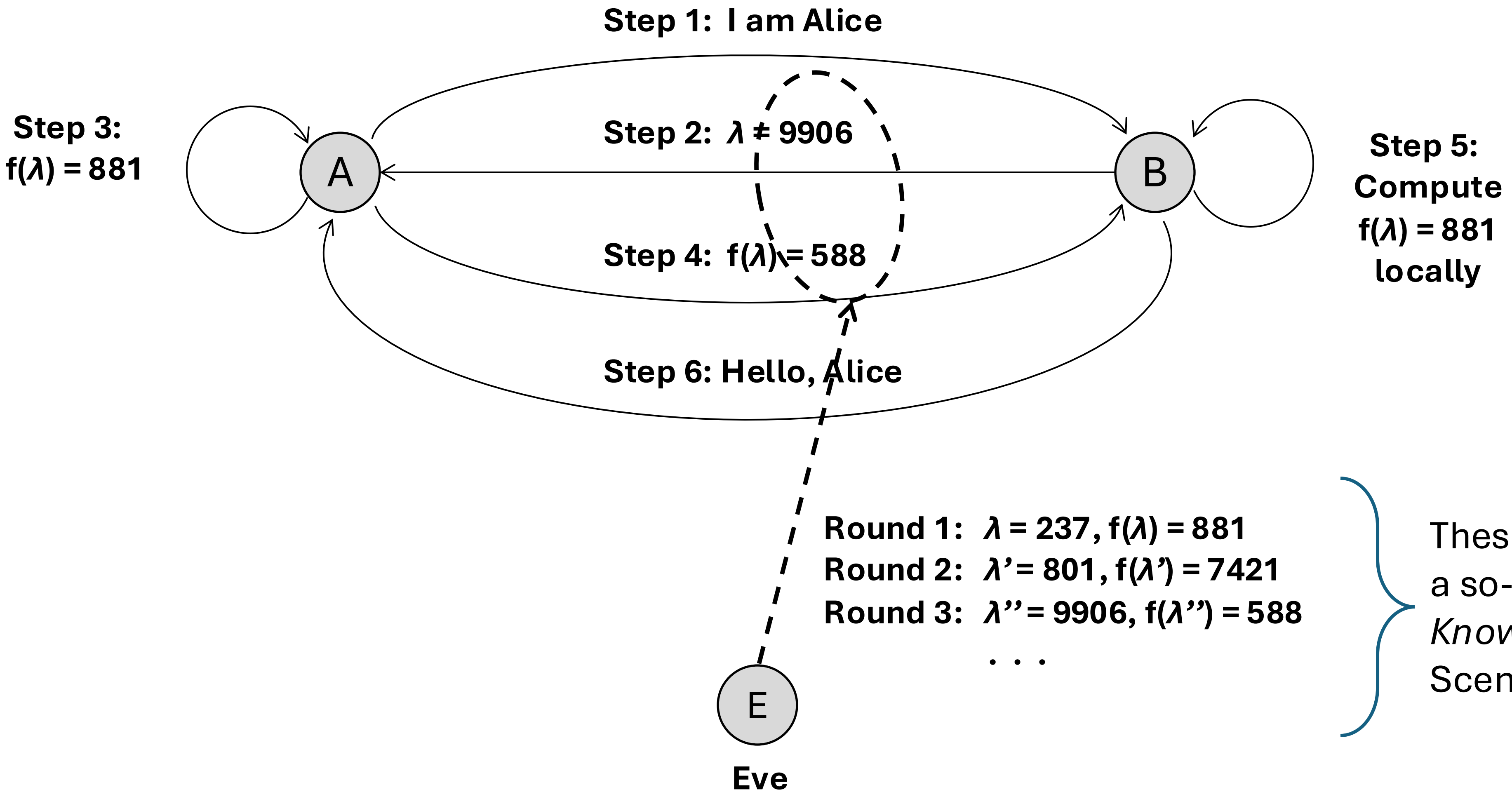
Step 5:
Compute
f(*λ*) = 881
locally

Step 4: f(*λ*) = 588

Step 6: Hello, Alice

Round 1: *λ* = 237, f(*λ*) = 881
Round 2: *λ'* = 801, f(*λ'*) = 7421
Round 3: *λ''* = 9906, f(*λ''*) = 588
. . .

Eve



Step 1: I am Alice

Step 2: λ = 9906

Step 3: f(λ) = 881

Step 4: f(λ) = 588

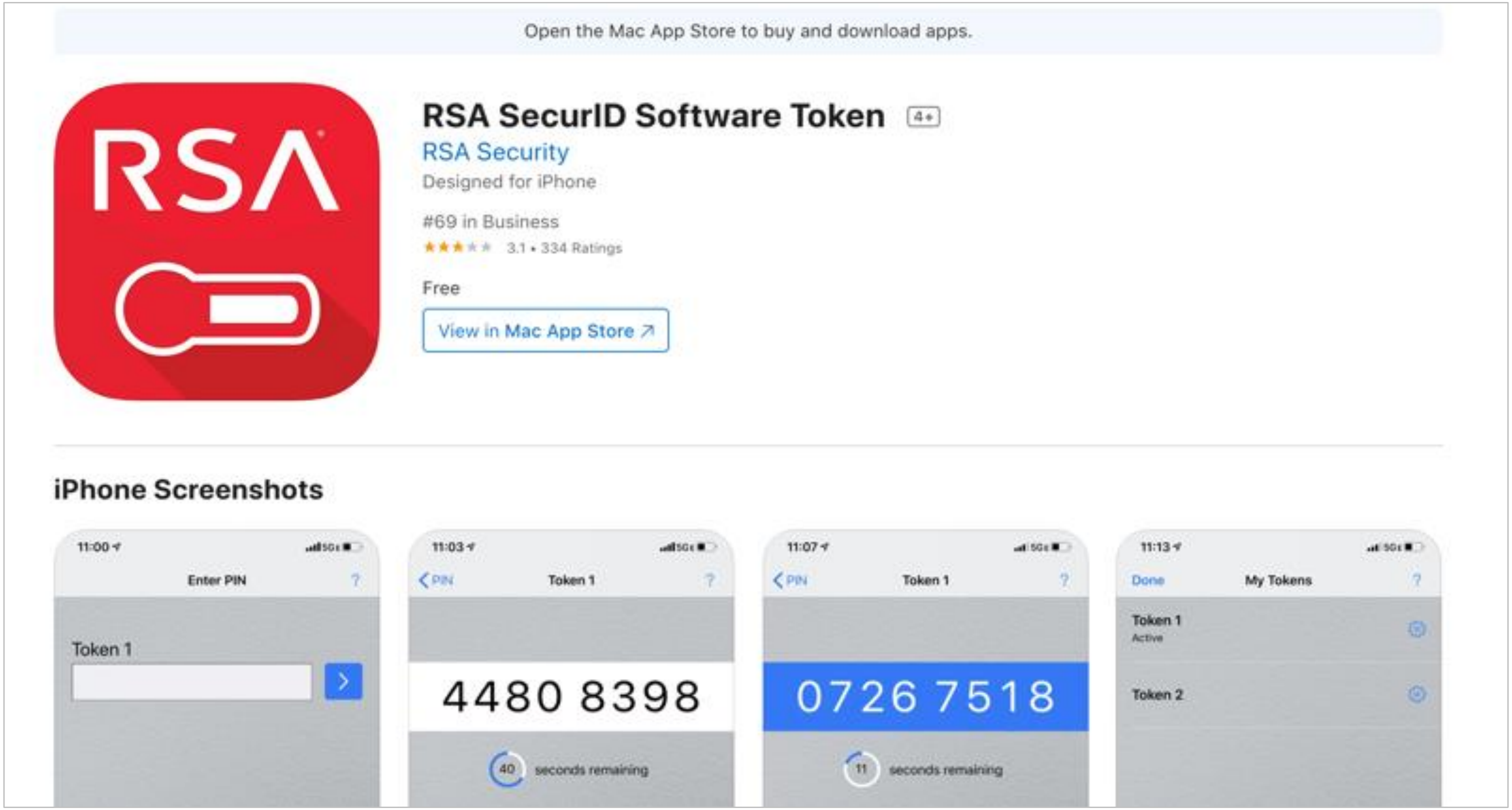
Step 5: Compute f(λ) = 881 locally

Step 6: Hello, Alice

Round 1: λ = 237, f(λ) = 881
Round 2: λ’ = 801, f(λ’) = 7421
Round 3: λ’’ = 9906, f(λ’’) = 588
. . .

These hints imply a so-called *Known Plaintext* Scenario.

Can we implement OTP without hints?



Open the Mac App Store to buy and download apps.

RSA SecurID Software Token 4+

RSA Security

Designed for iPhone

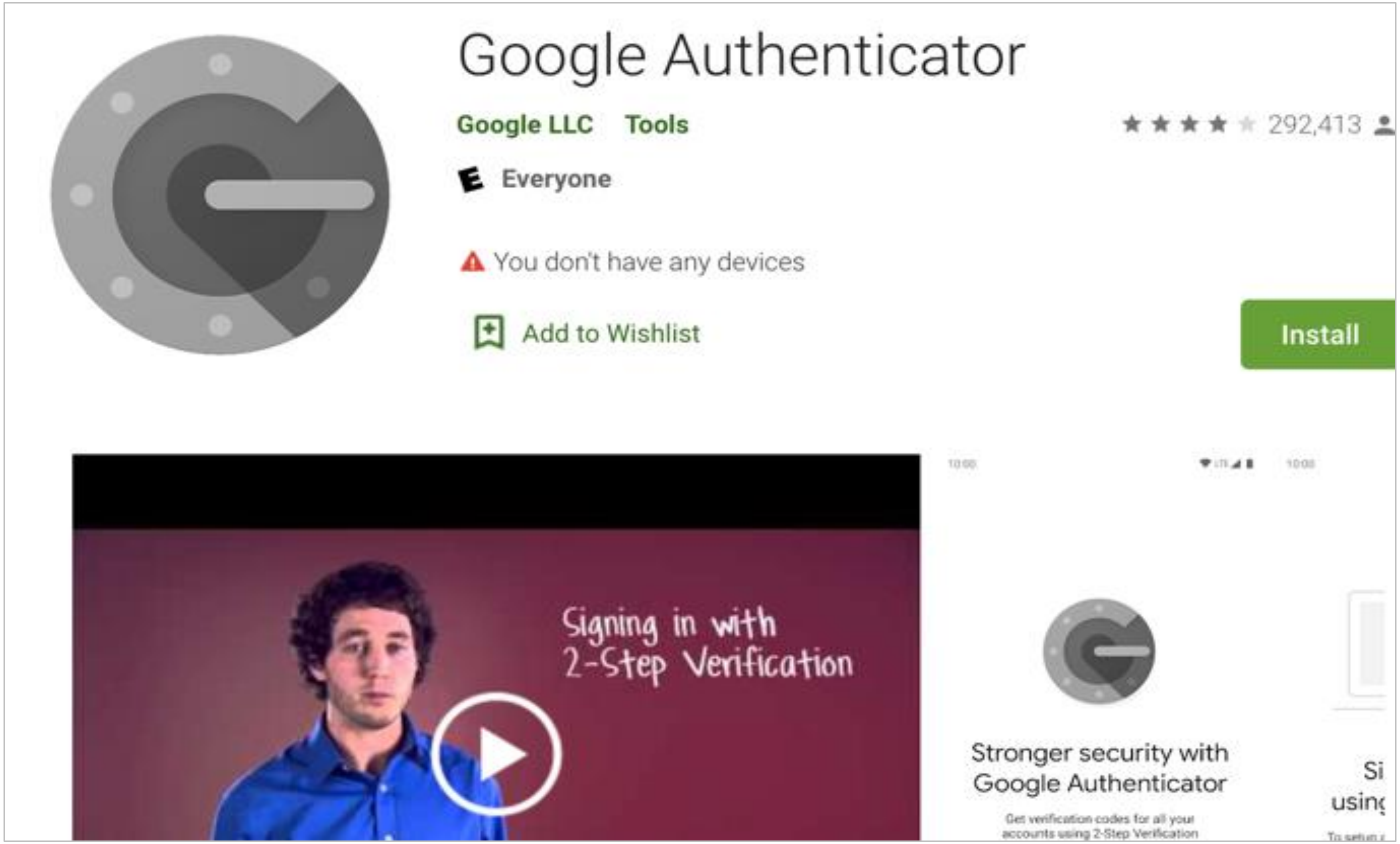
#69 in Business

★★★☆☆ 3.1 • 334 Ratings

Free

View in Mac App Store ↗

iPhone Screenshots



Google Authenticator

Google LLC Tools

★★★★☆ 292,413

Everyone

You don't have any devices

Stronger security with Google Authenticator

Get verification codes for all your accounts using 2-Step Verification

Si
usin

To setup

f: integer -> integer

λ: integer seed

t_0: initial time

t_C: current time

Δt: time interval

$n = (t_C - t_0) / \Delta t$

f: integer -> integer

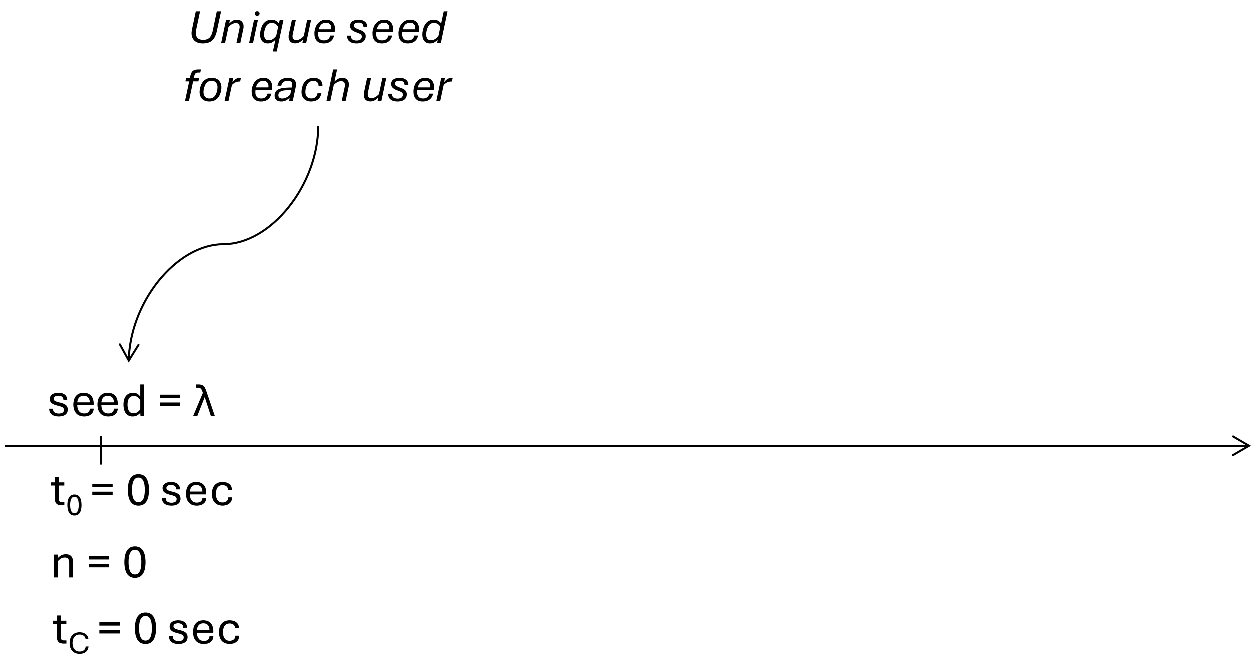
λ: integer seed

t_0: initial time

t_C: current time

Δt: time interval

$n = (t_C – t_0) / Δt$



Unique seed for each user

seed = λ

t_0 = 0 sec

n = 0

t_C = 0 sec

f: integer -> integer

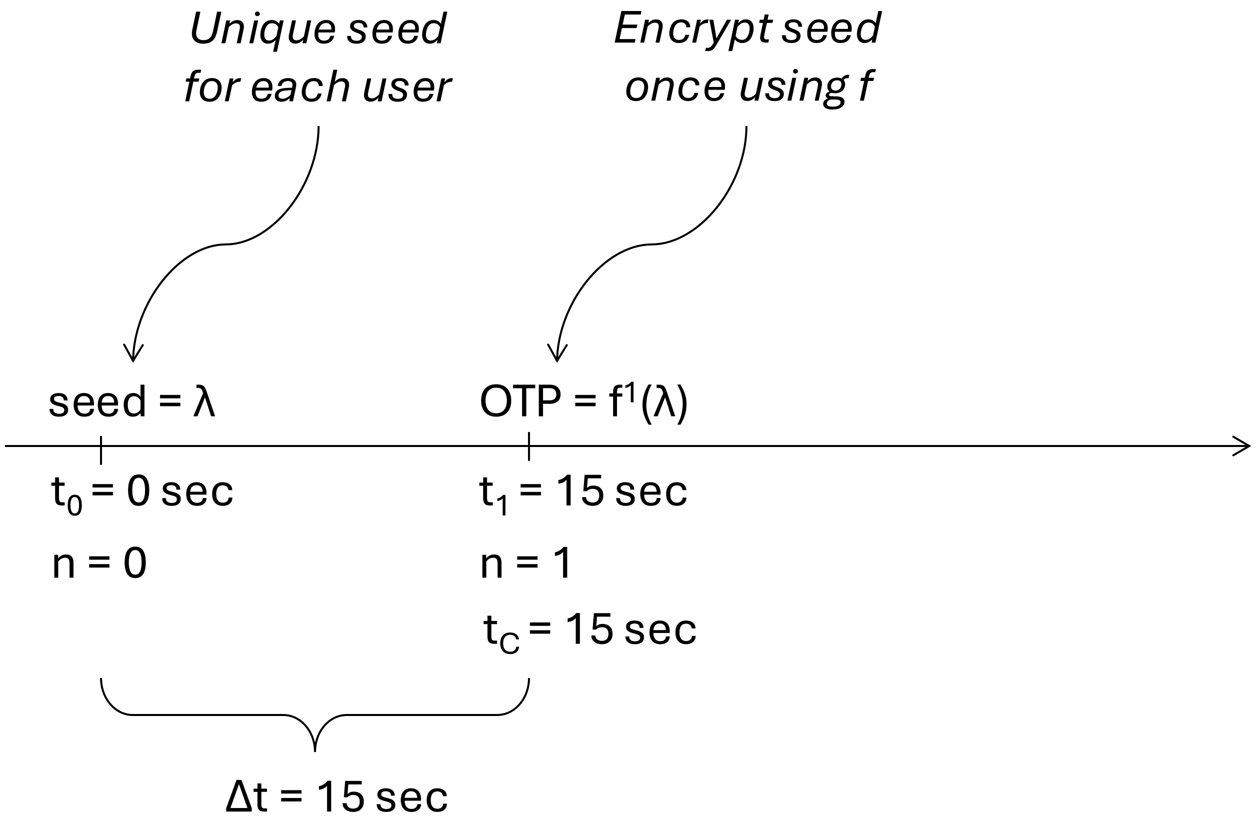
λ: integer seed

t_0: initial time

t_C: current time

Δt: time interval

$n = (t_C - t_0) / \Delta t$


Unique seed for each user

Encrypt seed once using f

seed = λ

OTP = $f^1(\lambda)$

t_0 = 0 sec

n = 0

t_1 = 15 sec

n = 1

t_C = 15 sec

Δt = 15 sec

f: integer -> integer

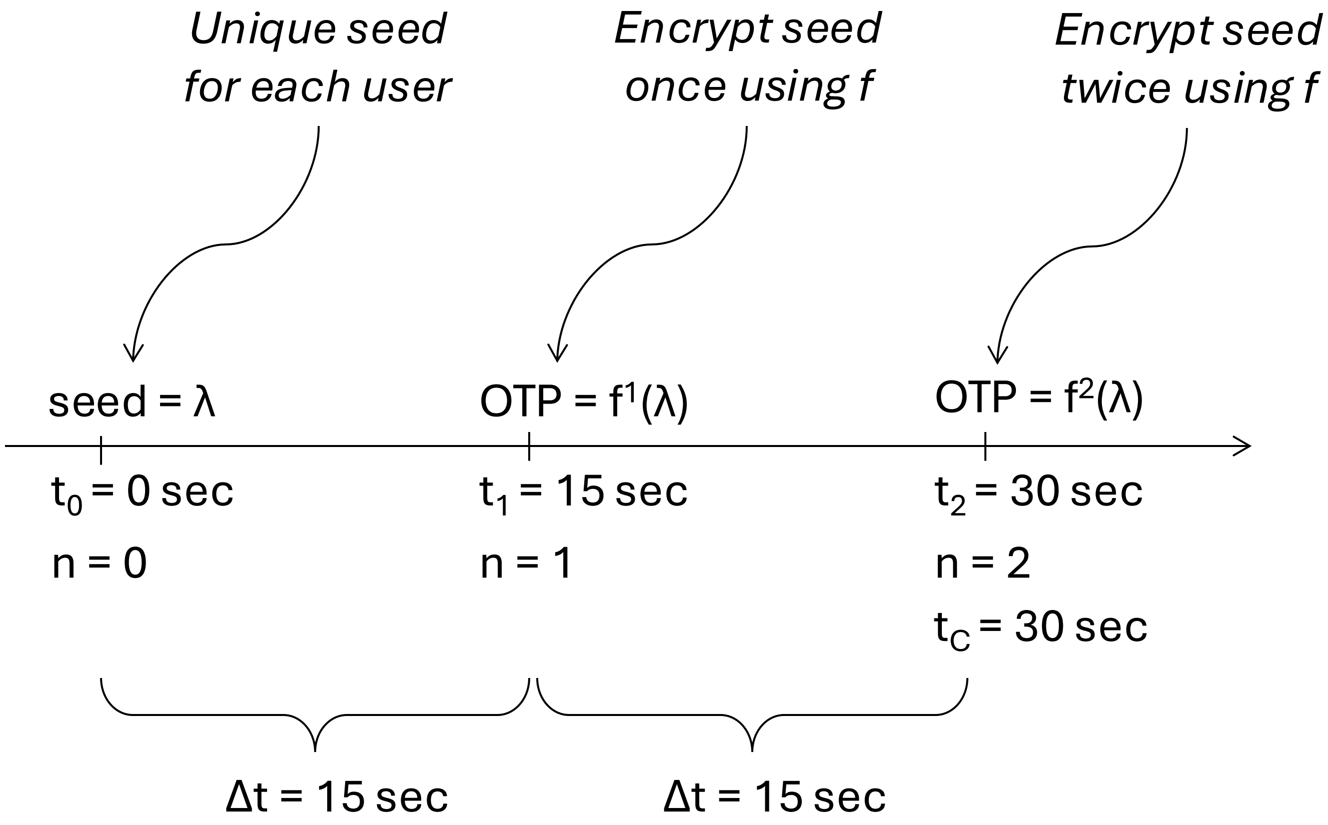
λ: integer seed

t_0: initial time

t_C: current time

Δt: time interval

$n = (t_C - t_0) / \Delta t$



Unique seed for each user

Encrypt seed once using f

Encrypt seed twice using f

seed = λ

OTP = $f^1(\lambda)$

OTP = $f^2(\lambda)$

$t_0 = 0$ sec

$n = 0$

$t_1 = 15$ sec

$n = 1$

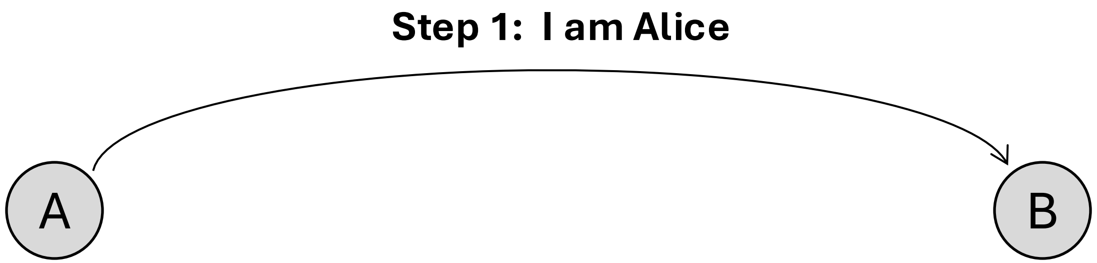
$t_2 = 30$ sec

$n = 2$

$t_C = 30$ sec

$\Delta t = 15$ sec

$\Delta t = 15$ sec



Step 1: I am Alice

f: integer -> integer

λ: integer seed

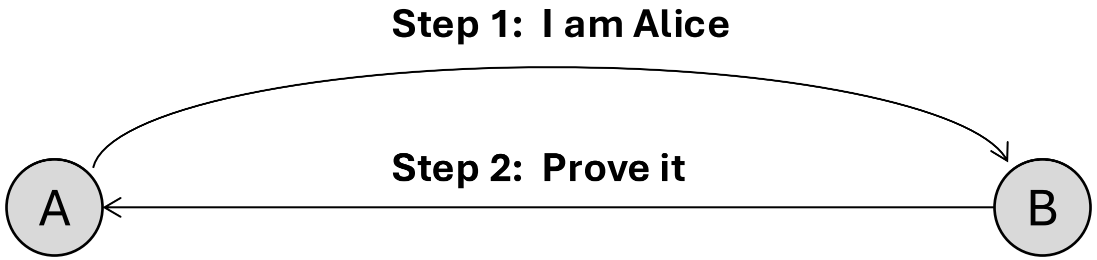
t_0: initial time

t_C: current time

Δt: time interval

$n = (t_C – t_0) / Δt$

User	*Information*
A	f: integer -> integer λ: integer seed t_0: initial time t_C: current time Δt: time interval $n = (t_C – t_0) / Δt$



Step 1: I am Alice

Step 2: Prove it

f: integer -> integer

λ: integer seed

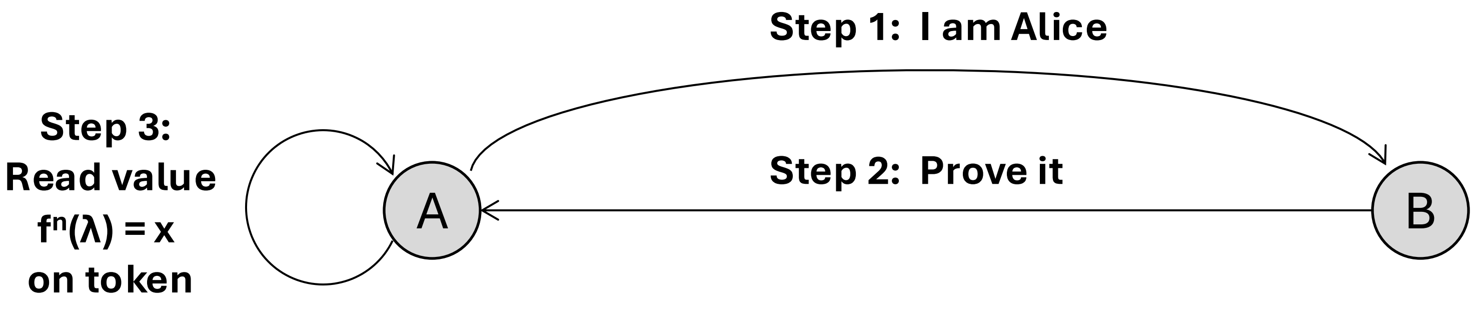
t_0: initial time

t_C: current time

Δt: time interval

$n = (t_C - t_0) / \Delta t$

User	***Information***
A	f: integer -> integer λ: integer seed t_0: initial time t_C: current time Δt: time interval $n = (t_C - t_0) / \Delta t$



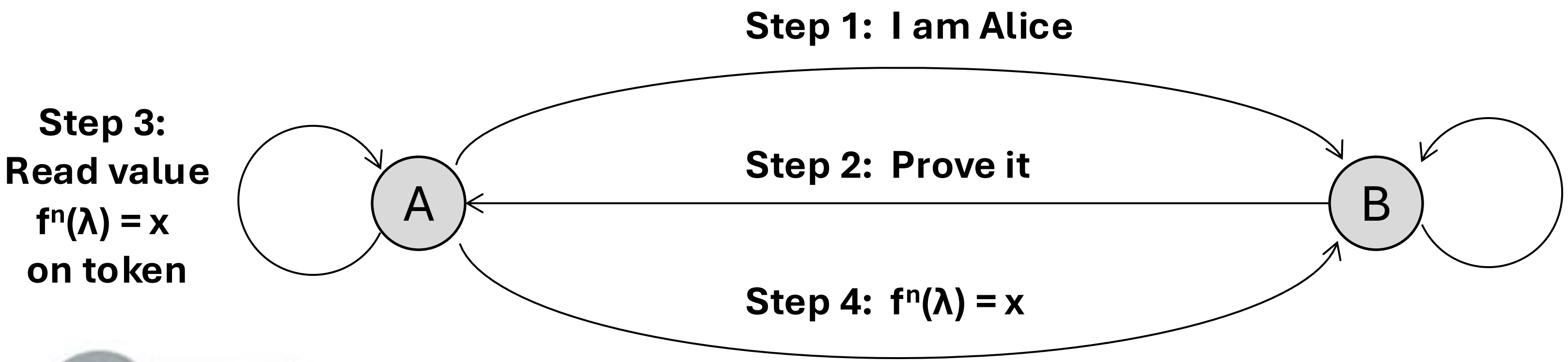
Step 1: I am Alice

Step 2: Prove it

Step 3: Read value $f^n(\lambda) = x$ on token

f: integer -> integer
λ: integer seed
t_0: initial time
t_C: current time
Δt: time interval
$n = (t_C - t_0) / \Delta t$

User	***Information***
A	f: integer -> integer λ: integer seed t_0: initial time t_C: current time Δt: time interval $n = (t_C - t_0) / \Delta t$



Step 1: I am Alice

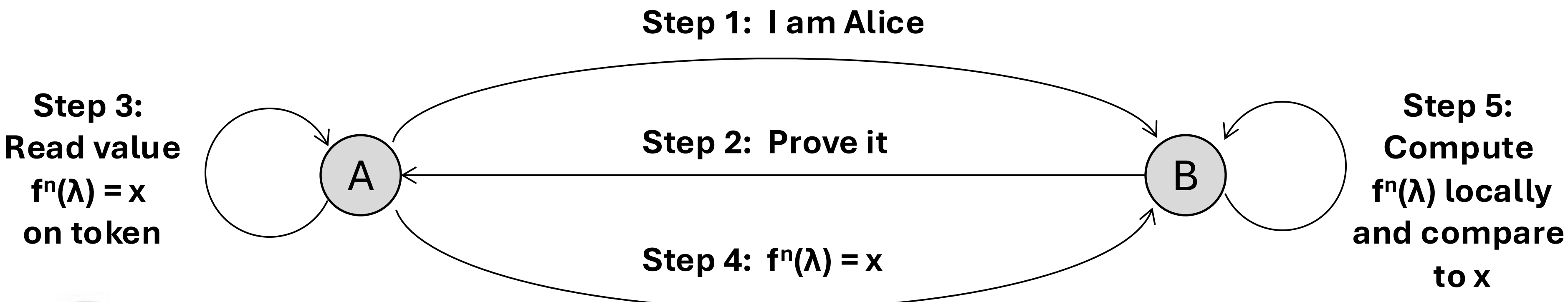
Step 2: Prove it

Step 3: Read value $f^n(\lambda) = x$ on token

Step 4: $f^n(\lambda) = x$

f: integer -> integer
λ: integer seed
t_0: initial time
t_C: current time
Δt: time interval
$n = (t_C – t_0) / \Delta t$

User	*Information*
A	f: integer -> integer λ: integer seed t_0: initial time t_C: current time Δt: time interval $n = (t_C – t_0) / \Delta t$



Step 1: I am Alice

Step 2: Prove it

Step 3: Read value $f^n(\lambda) = x$ on token

Step 4: $f^n(\lambda) = x$

Step 5: Compute $f^n(\lambda)$ locally and compare to x

f: integer -> integer

λ: integer seed

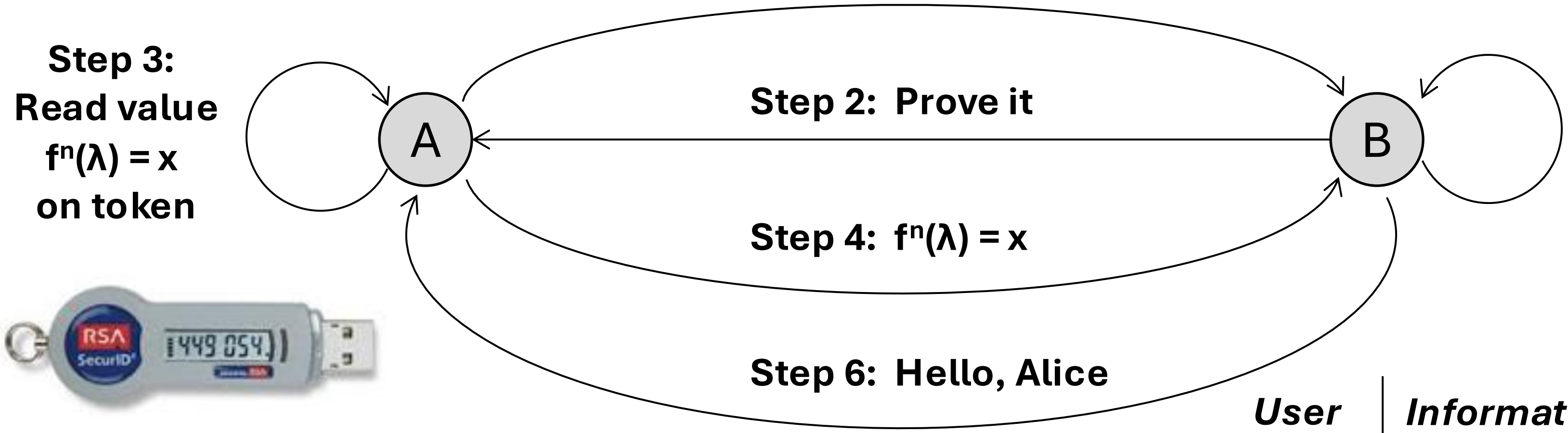
t_0: initial time

t_C: current time

Δt: time interval

$n = (t_C – t_0) / \Delta t$

User	*Information*
A	f: integer -> integer λ: integer seed t_0: initial time t_C: current time Δt: time interval $n = (t_C – t_0) / \Delta t$



Step 1: I am Alice

Step 2: Prove it

Step 3: Read value $f^n(\lambda) = x$ on token

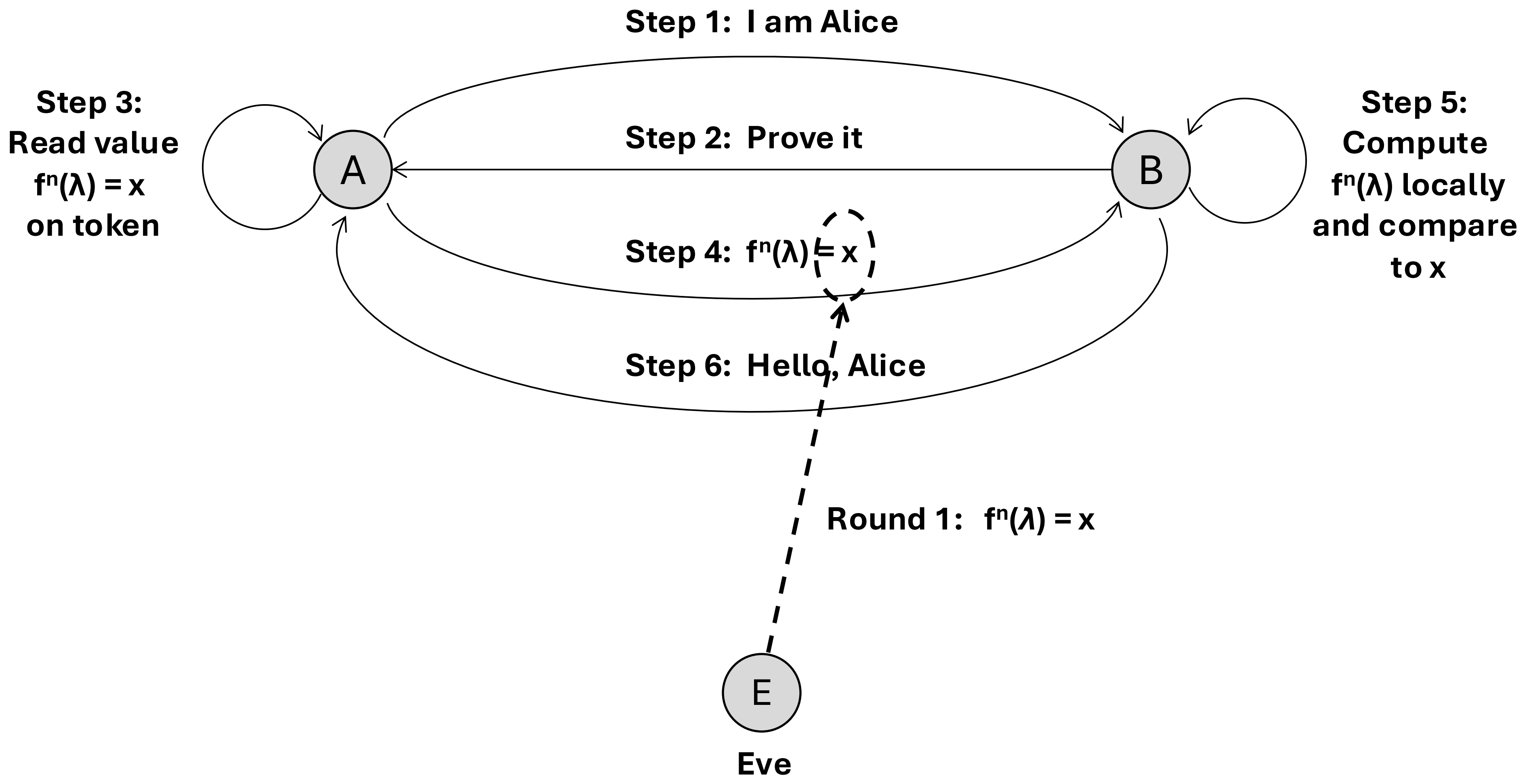
Step 4: $f^n(\lambda) = x$

Step 5: Compute $f^n(\lambda)$ locally and compare to x

Step 6: Hello, Alice

f: integer -> integer
λ: integer seed
t_0: initial time
t_C: current time
Δt: time interval
$n = (t_C - t_0) / \Delta t$

User	*Information*
A	f: integer -> integer λ: integer seed t_0: initial time t_C: current time Δt: time interval $n = (t_C - t_0) / \Delta t$



Step 1: I am Alice

Step 2: Prove it

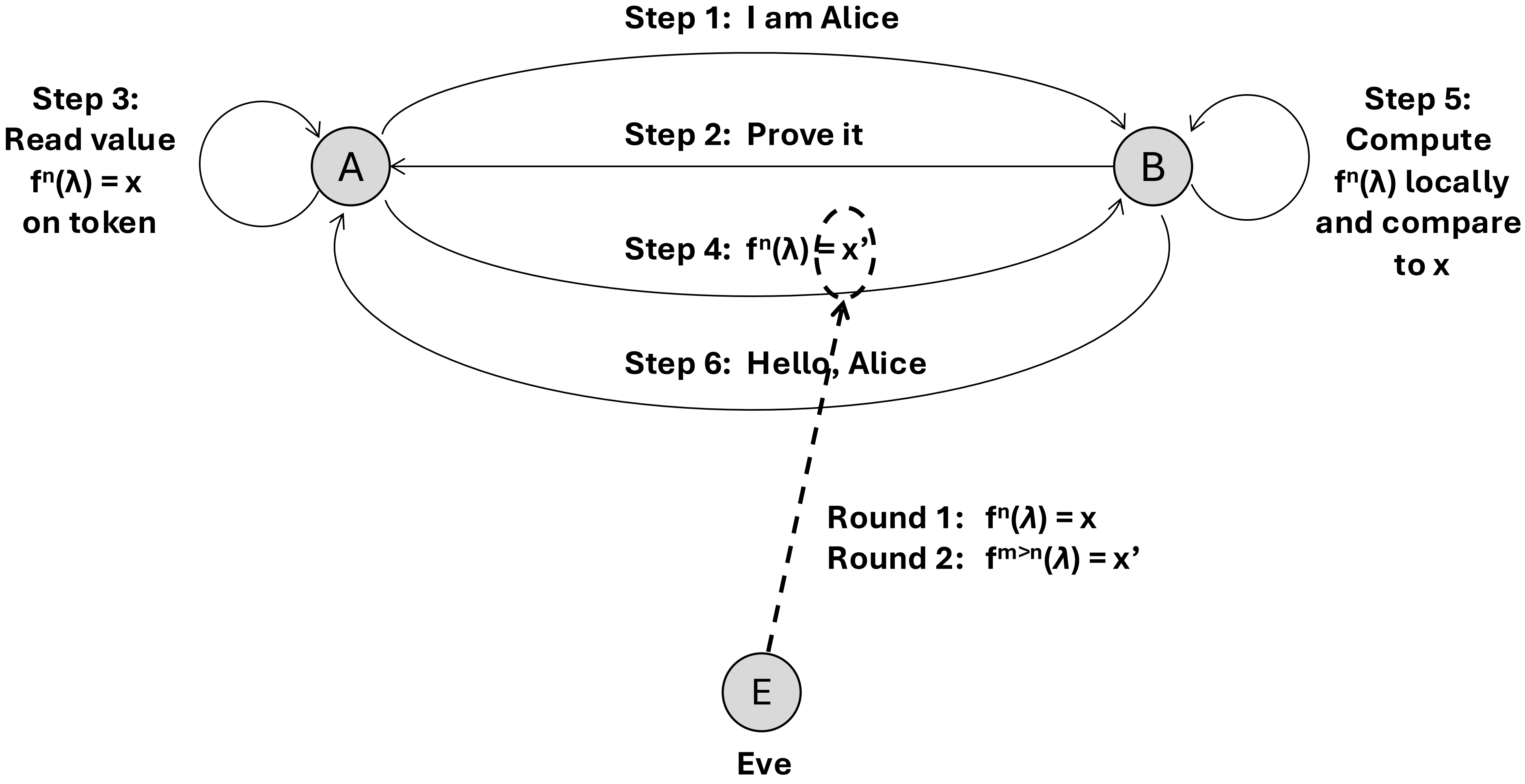
Step 3:
Read value
$f^n(\lambda) = x$
on token

Step 4: $f^n(\lambda) = x$

Step 5:
Compute
$f^n(\lambda)$ locally
and compare
to x

Step 6: Hello, Alice

Round 1: $f^n(\lambda) = x$



Step 1: I am Alice

Step 2: Prove it

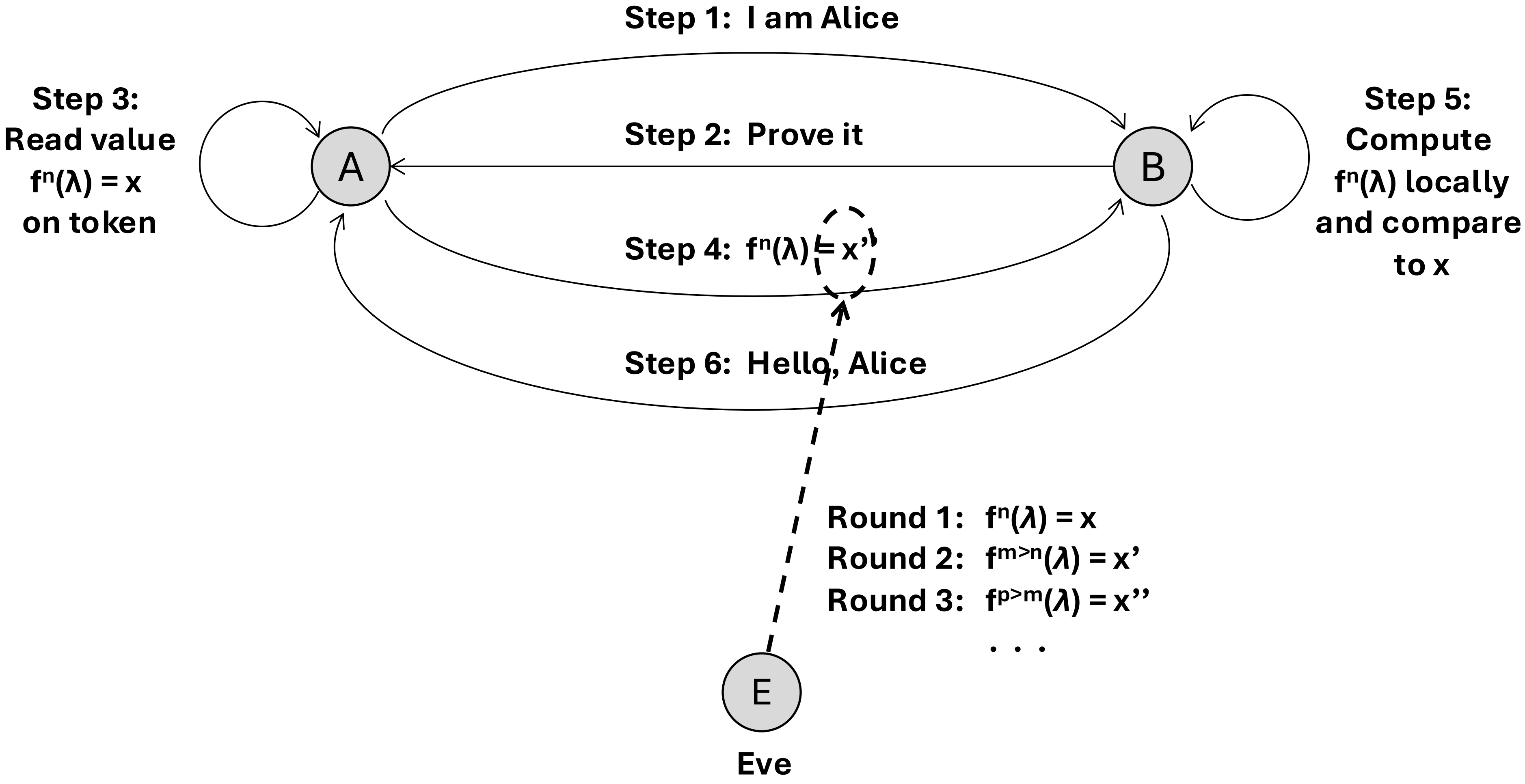
Step 3:
Read value
$f^n(\lambda) = x$
on token

Step 4: $f^n(\lambda) = x'$

Step 5:
Compute
$f^n(\lambda)$ locally
and compare
to x

Step 6: Hello, Alice

Round 1: $f^n(\lambda) = x$
Round 2: $f^{m>n}(\lambda) = x'$



Step 1: I am Alice

Step 2: Prove it

Step 3: Read value $f^n(\lambda) = x$ on token

Step 4: $f^n(\lambda) = x'$

Step 5: Compute $f^n(\lambda)$ locally and compare to x

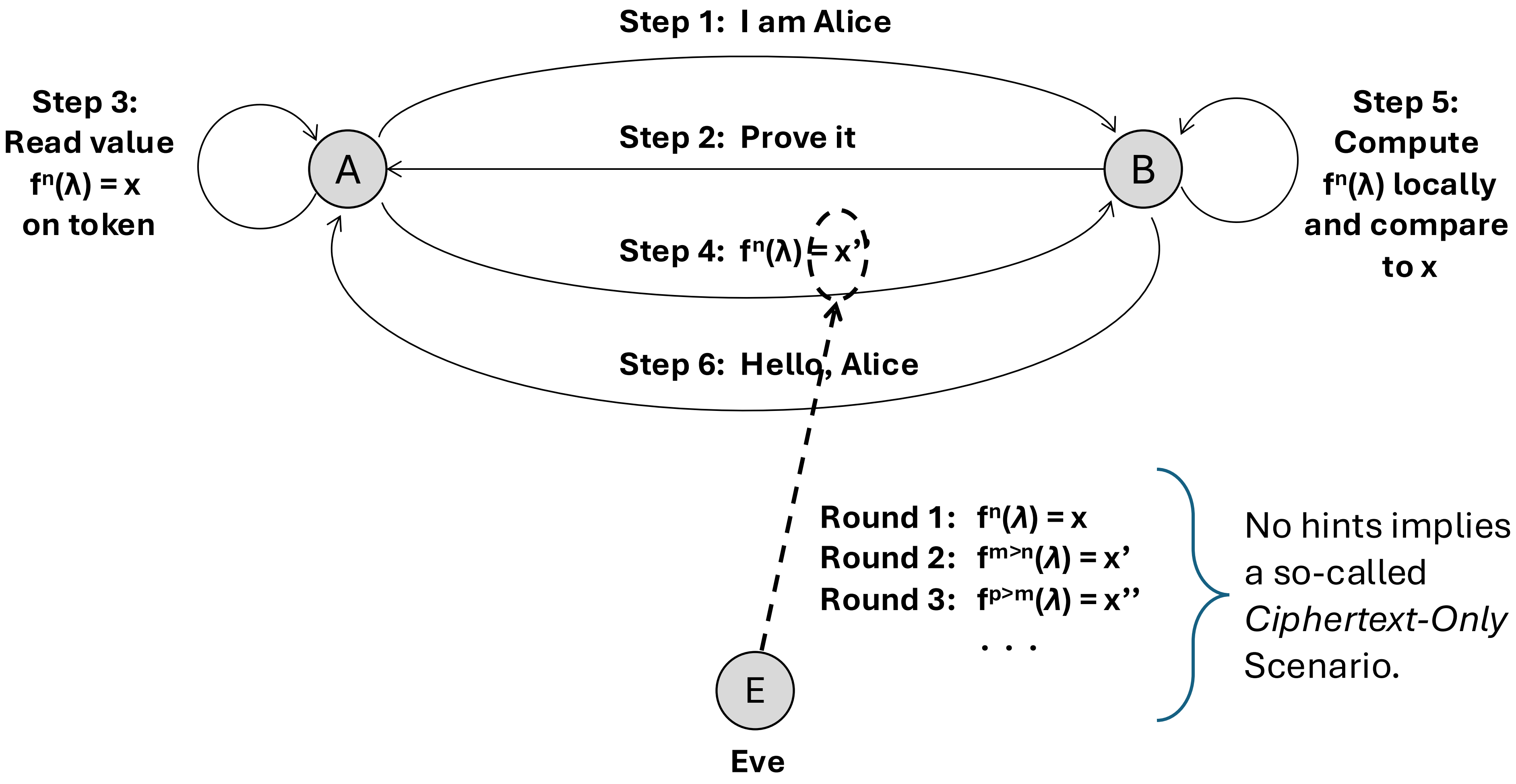
Step 6: Hello, Alice

Round 1: $f^n(\lambda) = x$

Round 2: $f^{m>n}(\lambda) = x'$

Round 3: $f^{p>m}(\lambda) = x''$

. . .



Step 1: I am Alice

Step 2: Prove it

Step 3:
Read value
$f^n(\lambda) = x$
on token

Step 4: $f^n(\lambda) = x'$

Step 5:
Compute
$f^n(\lambda)$ locally
and compare
to x

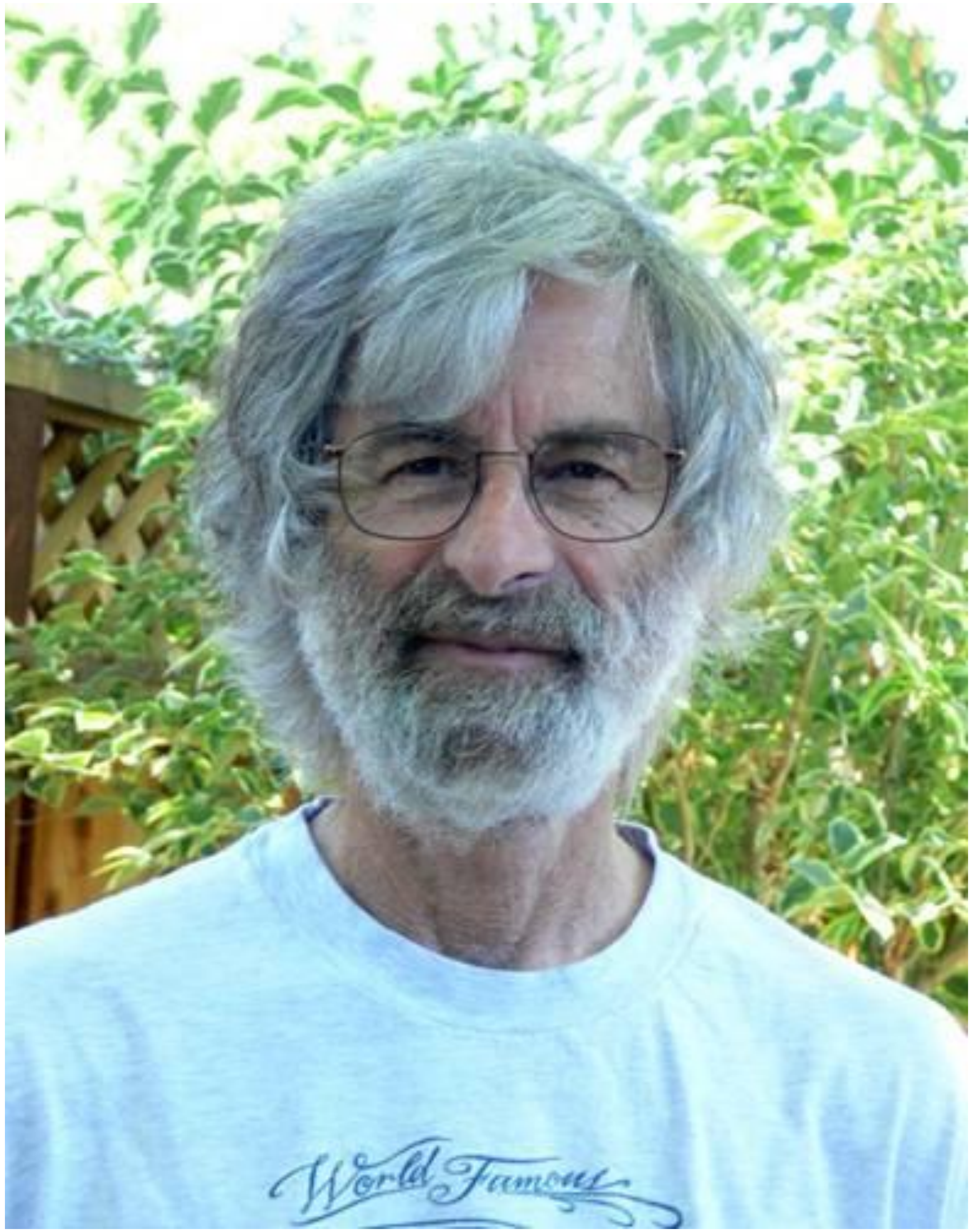
Step 6: Hello, Alice

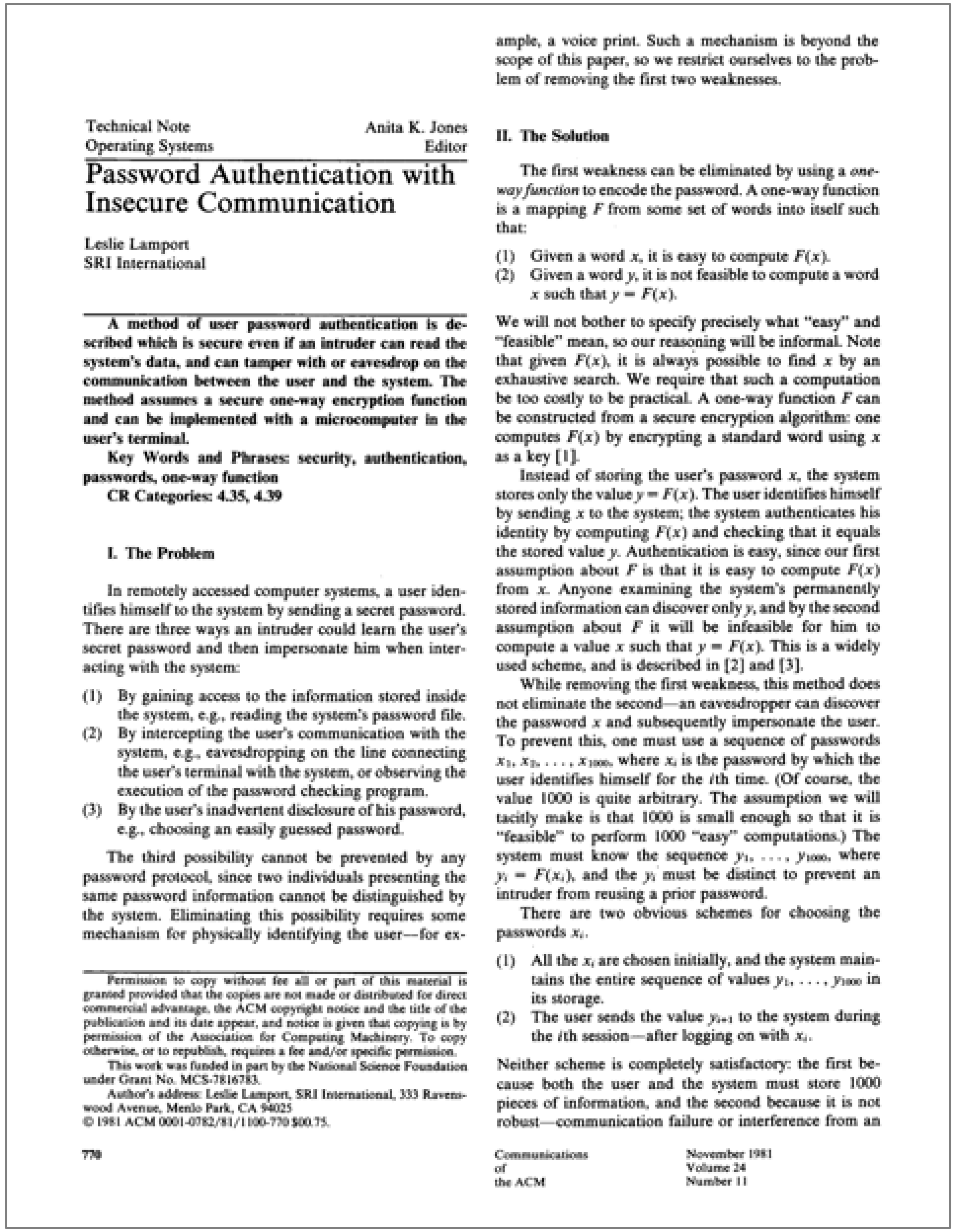
Eve

Round 1: $f^n(\lambda) = x$
Round 2: $f^{m>n}(\lambda) = x'$
Round 3: $f^{p>m}(\lambda) = x''$
. . .

No hints implies a so-called *Ciphertext-Only* Scenario.

Does avoidance of challenges ensure ciphertext only cases?





Technical Note
Operating Systems

Anita K. Jones
Editor

Password Authentication with Insecure Communication

Leslie Lamport
SRI International

A method of user password authentication is described which is secure even if an intruder can read the system's data, and can tamper with or eavesdrop on the communication between the user and the system. The method assumes a secure one-way encryption function and can be implemented with a microcomputer in the user's terminal.

Key Words and Phrases: security, authentication, passwords, one-way function

CR Categories: 4.35, 4.39

I. The Problem

In remotely accessed computer systems, a user identifies himself to the system by sending a secret password. There are three ways an intruder could learn the user's secret password and then impersonate him when interacting with the system:

(1) By gaining access to the information stored inside the system, e.g., reading the system's password file.
(2) By intercepting the user's communication with the system, e.g., eavesdropping on the line connecting the user's terminal with the system, or observing the execution of the password checking program.
(3) By the user's inadvertent disclosure of his password, e.g., choosing an easily guessed password.

The third possibility cannot be prevented by any password protocol, since two individuals presenting the same password information cannot be distinguished by the system. Eliminating this possibility requires some mechanism for physically identifying the user—for example, a voice print. Such a mechanism is beyond the scope of this paper, so we restrict ourselves to the problem of removing the first two weaknesses.

II. The Solution

The first weakness can be eliminated by using a *one-way function* to encode the password. A one-way function is a mapping F from some set of words into itself such that:

(1) Given a word x, it is easy to compute $F(x)$.
(2) Given a word y, it is not feasible to compute a word x such that $y = F(x)$.

We will not bother to specify precisely what "easy" and "feasible" mean, so our reasoning will be informal. Note that given $F(x)$, it is always possible to find x by an exhaustive search. We require that such a computation be too costly to be practical. A one-way function F can be constructed from a secure encryption algorithm: one computes $F(x)$ by encrypting a standard word using x as a key [1].

Instead of storing the user's password x, the system stores only the value $y = F(x)$. The user identifies himself by sending x to the system; the system authenticates his identity by computing $F(x)$ and checking that it equals the stored value y. Authentication is easy, since our first assumption about F is that it is easy to compute $F(x)$ from x. Anyone examining the system's permanently stored information can discover only y, and by the second assumption about F it will be infeasible for him to compute a value x such that $y = F(x)$. This is a widely used scheme, and is described in [2] and [3].

While removing the first weakness, this method does not eliminate the second—an eavesdropper can discover the password x and subsequently impersonate the user. To prevent this, one must use a sequence of passwords $x_1, x_2, \ldots, x_{1000}$, where x_i is the password by which the user identifies himself for the ith time. (Of course, the value 1000 is quite arbitrary. The assumption we will tacitly make is that 1000 is small enough so that it is "feasible" to perform 1000 "easy" computations.) The system must know the sequence $y_1, \ldots, y_{1000}$, where $y_i = F(x_i)$, and the y_i must be distinct to prevent an intruder from reusing a prior password.

There are two obvious schemes for choosing the passwords x_i.

(1) All the x_i are chosen initially, and the system maintains the entire sequence of values $y_1, \ldots, y_{1000}$ in its storage.
(2) The user sends the value y_{i+1} to the system during the ith session—after logging on with x_i.

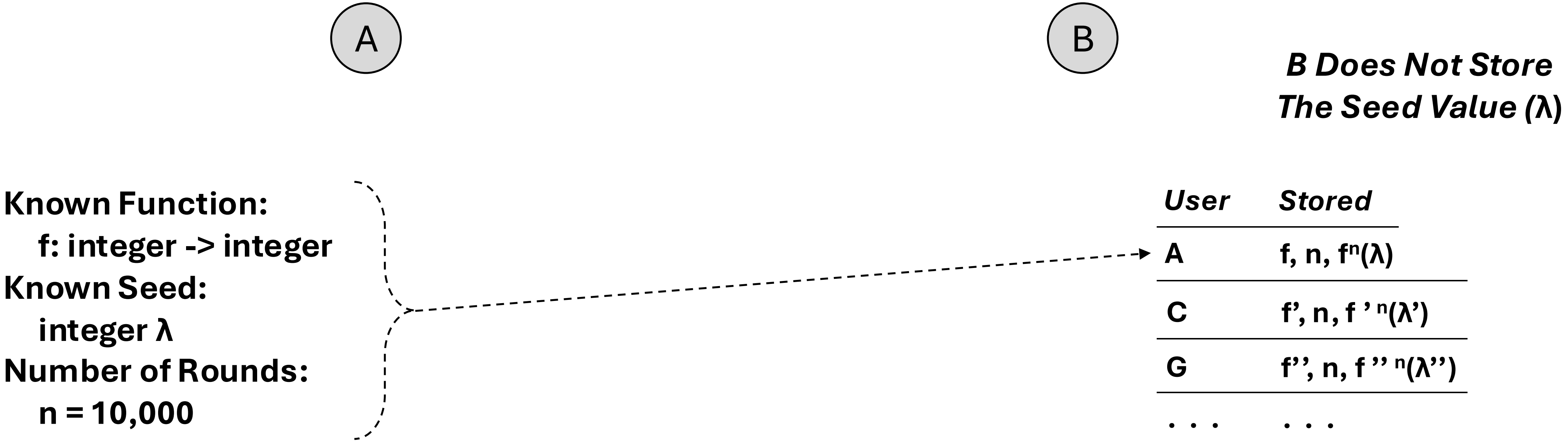
Neither scheme is completely satisfactory: the first because both the user and the system must store 1000 pieces of information, and the second because it is not robust—communication failure or interference from an

This work was funded in part by the National Science Foundation under Grant No. MCS-7816783.

Author's address: Leslie Lamport, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025

A is reporting its identity to B

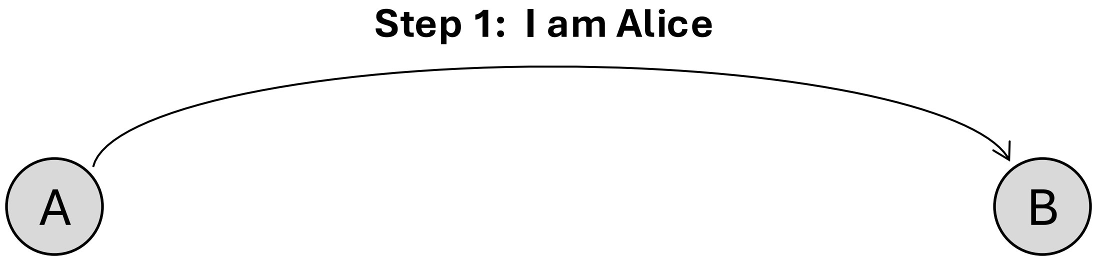
B is attempting to validate A's reported identity (i.e., authenticating A)



B Does Not Store The Seed Value (λ)

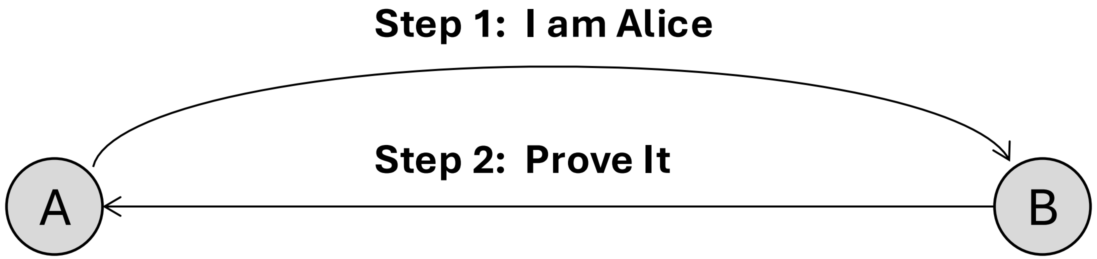
Known Function:
f: integer -> integer
Known Seed:
integer λ
Number of Rounds:
n = 10,000

User	*Stored*
A	$f, n, f^{n}(\lambda)$
C	$f', n, f'^{n}(\lambda')$
G	$f'', n, f''^{n}(\lambda'')$
. . .	. . .



Known Function:
f: integer -> integer
Known Seed:
integer λ
Number of Rounds:
n = 10,000

User	*Stored*
A	**f, n, $f^n(\lambda)$**



Known Function:

f: integer -> integer

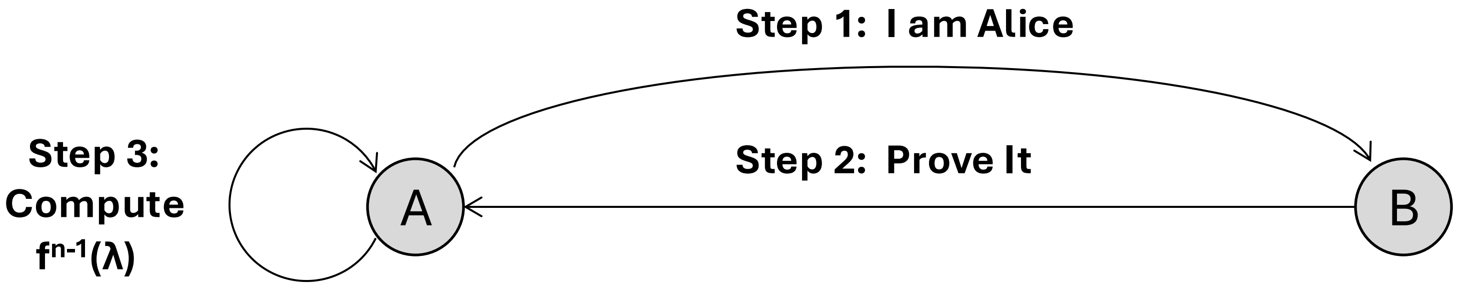
Known Seed:

integer λ

Number of Rounds:

n = 10,000

User	*Stored*
A	**f, n, $f^n(\lambda)$**

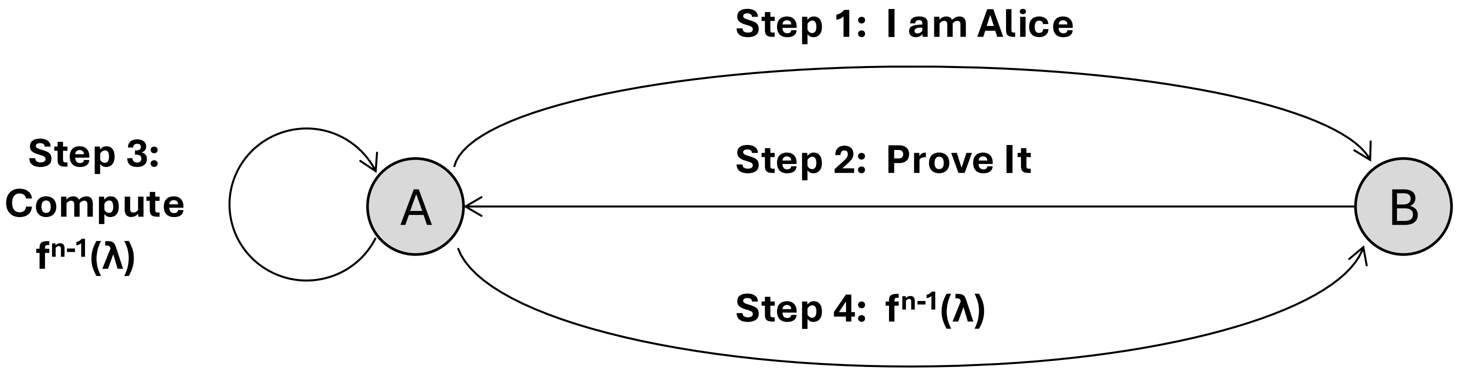


Known Function:
f: integer -> integer

Known Seed:
integer λ

Number of Rounds:
n = 10,000

User	*Stored*
A	**f, n, $f^{n}(\lambda)$**



Step 1: I am Alice

Step 2: Prove It

Step 3: Compute $f^{n-1}(\lambda)$

Step 4: $f^{n-1}(\lambda)$

A

B

Known Function:

f: integer -> integer

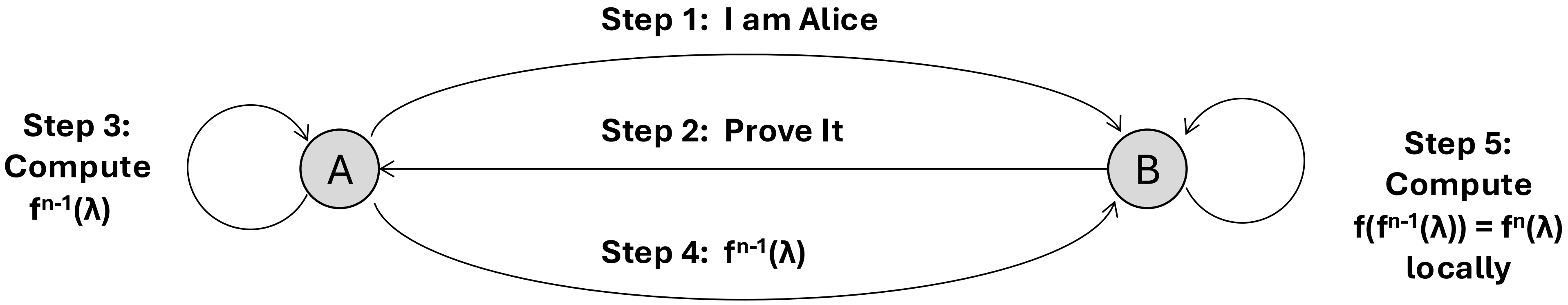
Known Seed:

integer λ

Number of Rounds:

n = 10,000

User	*Stored*
A	**f, n, $f^n(\lambda)$**



Step 1: I am Alice

Step 2: Prove It

Step 3: Compute $f^{n-1}(\lambda)$

Step 4: $f^{n-1}(\lambda)$

Step 5: Compute $f(f^{n-1}(\lambda)) = f^{n}(\lambda)$ locally

Known Function:

f: integer -> integer

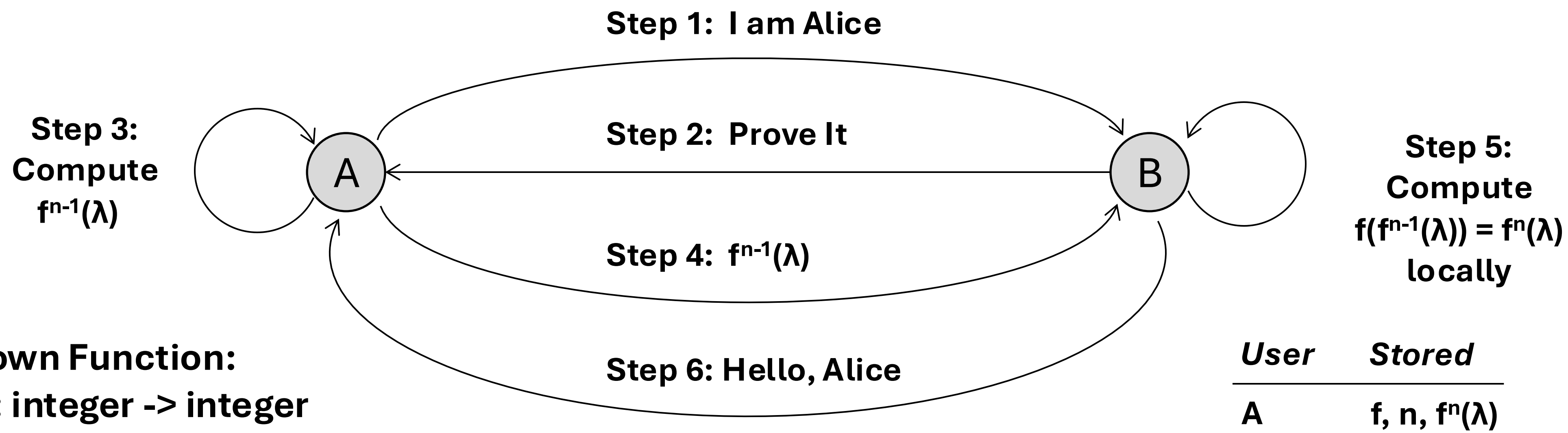
Known Seed:

integer λ

Number of Rounds:

$n = 10{,}000$

User	*Stored*
A	$f, n, f^{n}(\lambda)$



A

B

Step 1: I am Alice

Step 2: Prove It

Step 3: Compute $f^{n-1}(\lambda)$

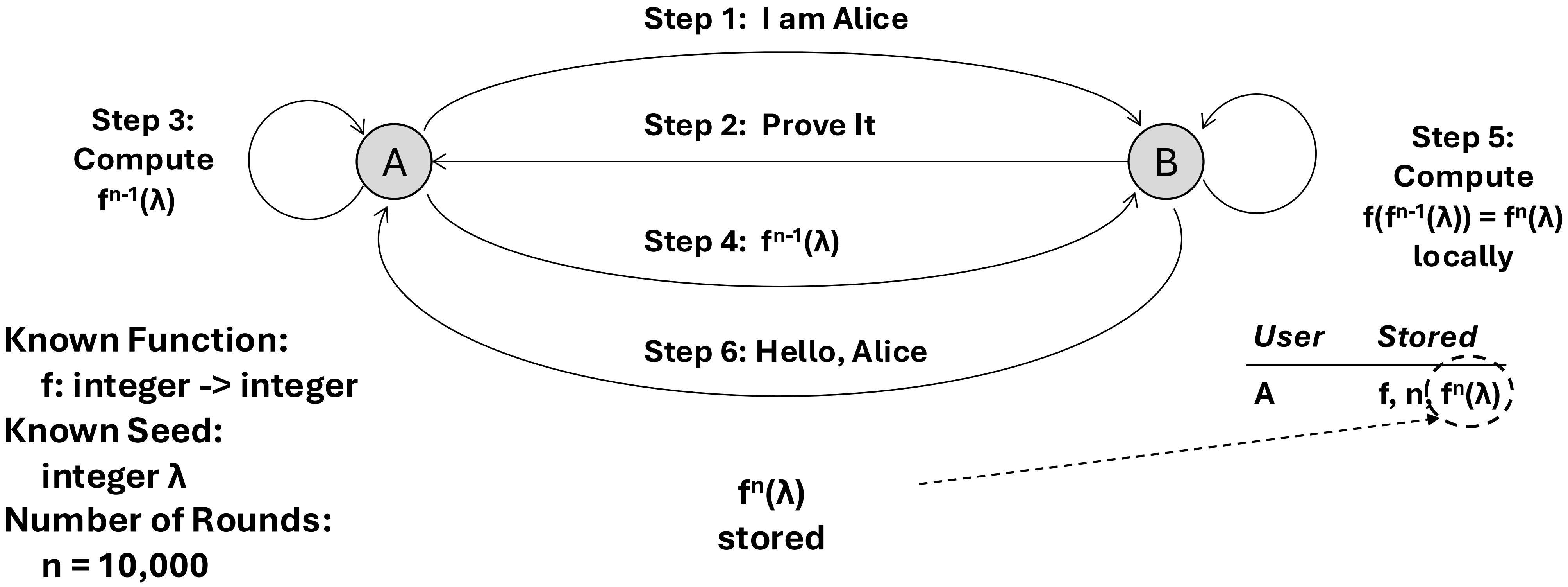
Step 4: $f^{n-1}(\lambda)$

Step 5: Compute $f(f^{n-1}(\lambda)) = f^{n}(\lambda)$ locally

Step 6: Hello, Alice

Known Function:
f: integer -> integer
Known Seed:
integer λ
Number of Rounds:
n = 10,000

User	*Stored*
A	f, n, $f^{n}(\lambda)$



Step 1: I am Alice

Step 2: Prove It

Step 3: Compute $f^{n-1}(\lambda)$

Step 4: $f^{n-1}(\lambda)$

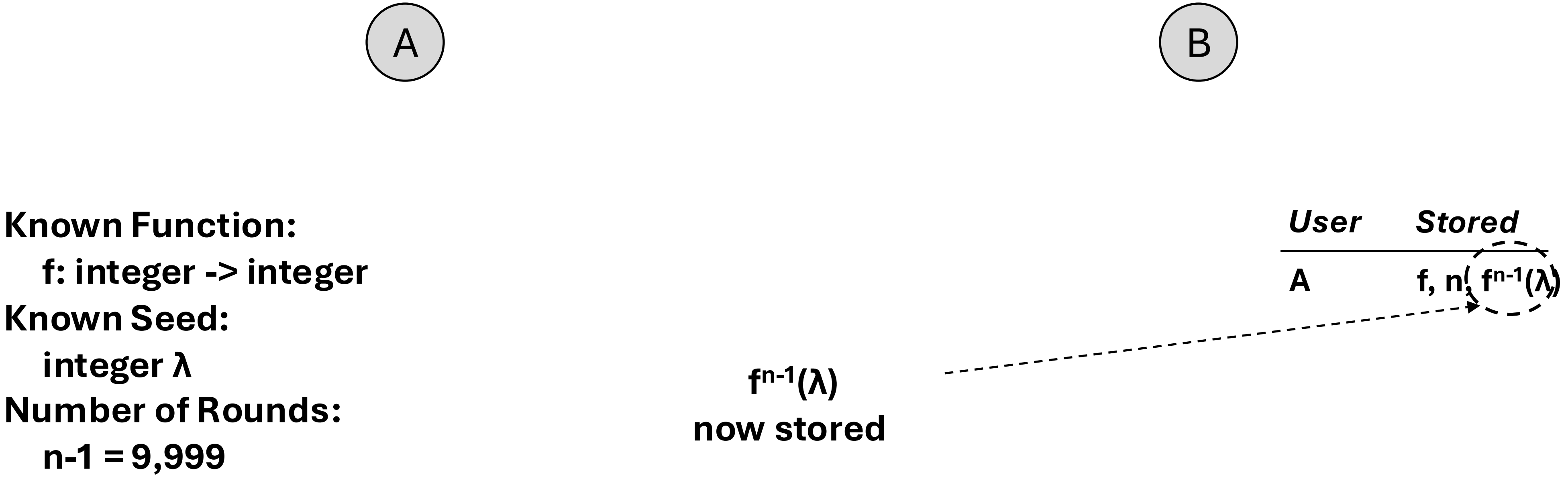
Step 5: Compute $f(f^{n-1}(\lambda)) = f^{n}(\lambda)$ locally

Step 6: Hello, Alice

Known Function:
f: integer -> integer
Known Seed:
integer λ
Number of Rounds:
n = 10,000

User	*Stored*
A	f, n, $f^{n}(\lambda)$

$f^{n}(\lambda)$
stored



Known Function:

f: integer -> integer

Known Seed:

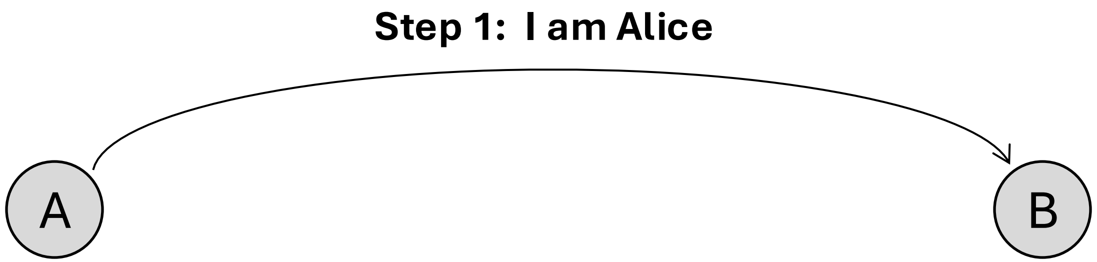
integer λ

Number of Rounds:

n-1 = 9,999

$f^{n-1}(\lambda)$ now stored

User	*Stored*
A	f, n, $f^{n-1}(\lambda)$



Step 1: I am Alice

Known Function:

f: integer -> integer

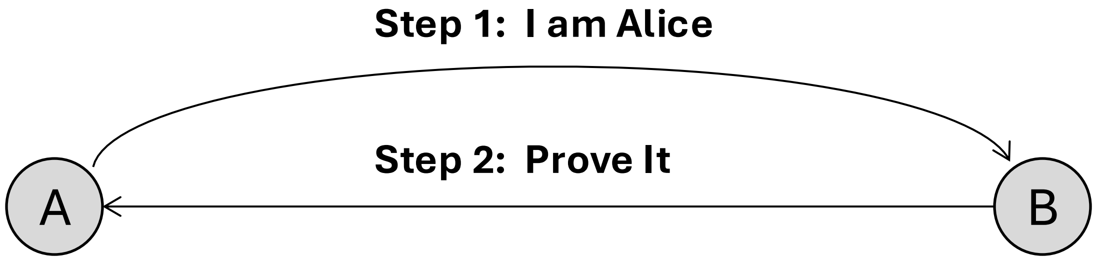
Known Seed:

integer λ

Number of Rounds:

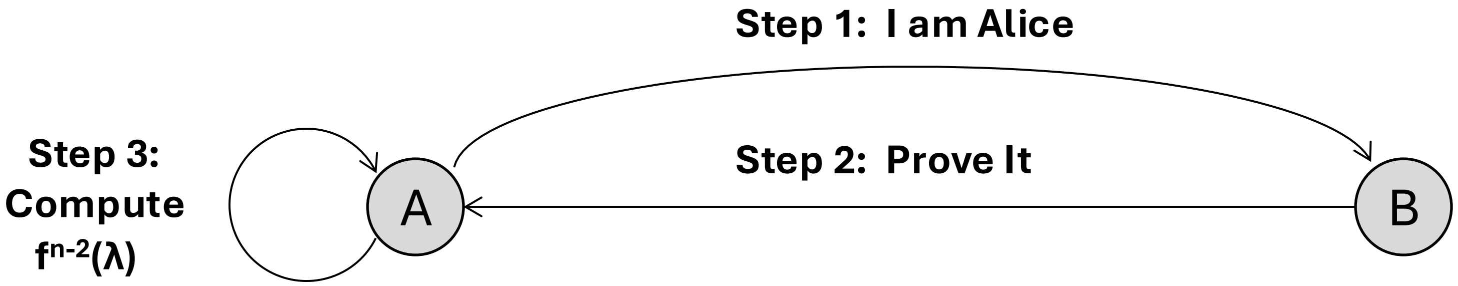
n-1 = 9,999

User	*Stored*
A	f, n, $f^{n-1}(\lambda)$



Known Function:
f: integer -> integer
Known Seed:
integer λ
Number of Rounds:
n-1 = 9,999

User	*Stored*
A	**f, n, $f^{n-1}(\lambda)$**



Known Function:

f: integer -> integer

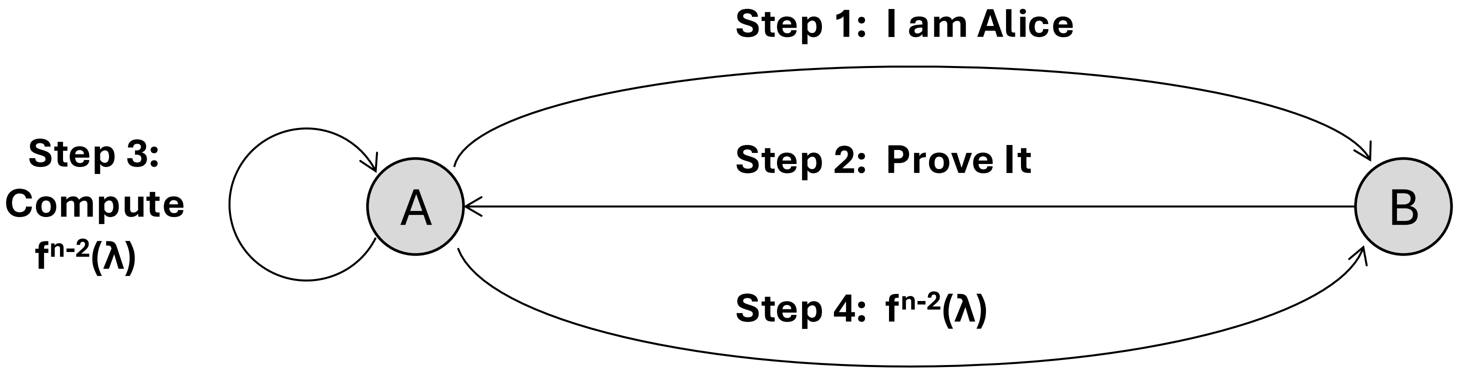
Known Seed:

integer λ

Number of Rounds:

$n-1 = 9{,}999$

User	*Stored*
A	**f, n, $f^{n-1}(\lambda)$**



Step 1: I am Alice

Step 2: Prove It

Step 3: Compute $f^{n-2}(\lambda)$

Step 4: $f^{n-2}(\lambda)$

Known Function:

f: integer -> integer

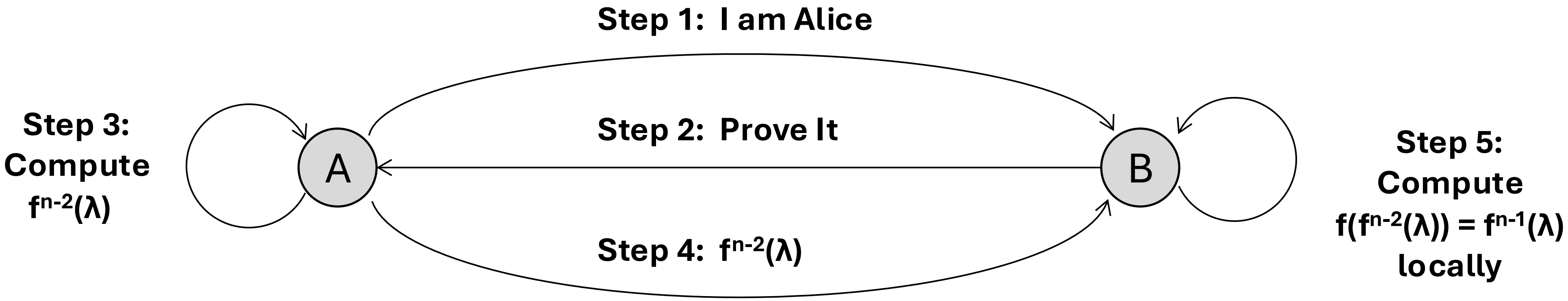
Known Seed:

integer λ

Number of Rounds:

n-1 = 9,999

User	*Stored*
A	**f, n, $f^{n-1}(\lambda)$**



Step 1: I am Alice

Step 2: Prove It

Step 3: Compute $f^{n-2}(\lambda)$

Step 4: $f^{n-2}(\lambda)$

Step 5: Compute $f(f^{n-2}(\lambda)) = f^{n-1}(\lambda)$ locally

Known Function:

f: integer -> integer

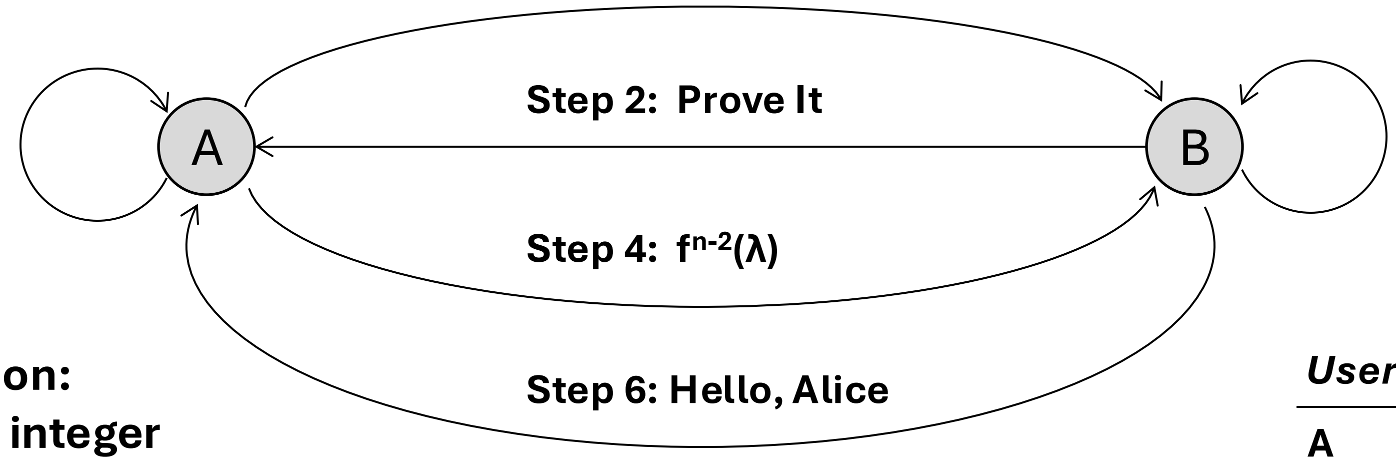
Known Seed:

integer λ

Number of Rounds:

n-1 = 9,999

User	***Stored***
A	**f, n, $f^{n-1}(\lambda)$**



Step 1: I am Alice

Step 2: Prove It

Step 3:
Compute
$f^{n-2}(\lambda)$

Step 4: $f^{n-2}(\lambda)$

Step 5:
Compute
$f(f^{n-2}(\lambda)) = f^{n-1}(\lambda)$
locally

Step 6: Hello, Alice

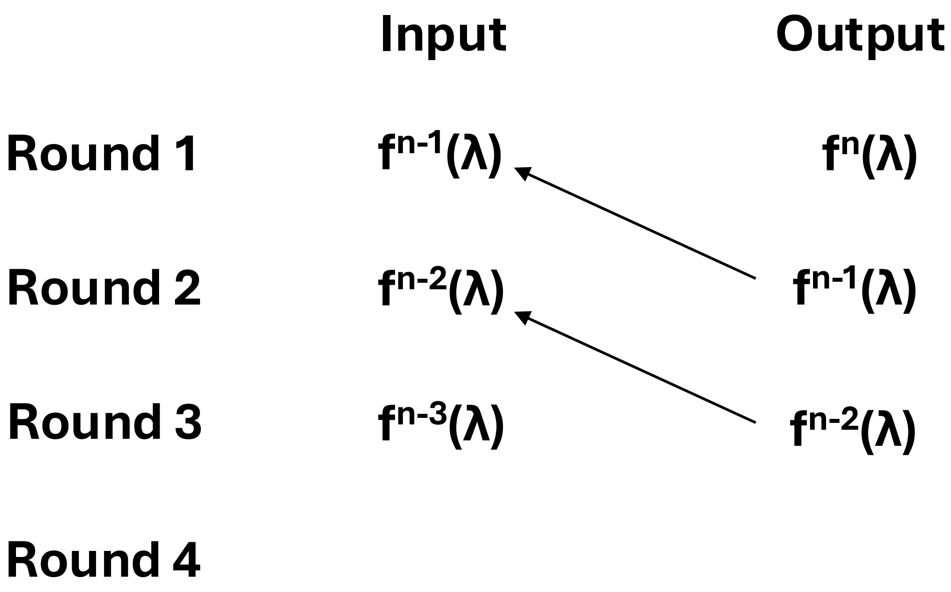
Known Function:
f: integer -> integer
Known Seed:
integer λ
Number of Rounds:
n-1 = 9,999

User	***Stored***
A	**f, n, $f^{n-1}(\lambda)$**

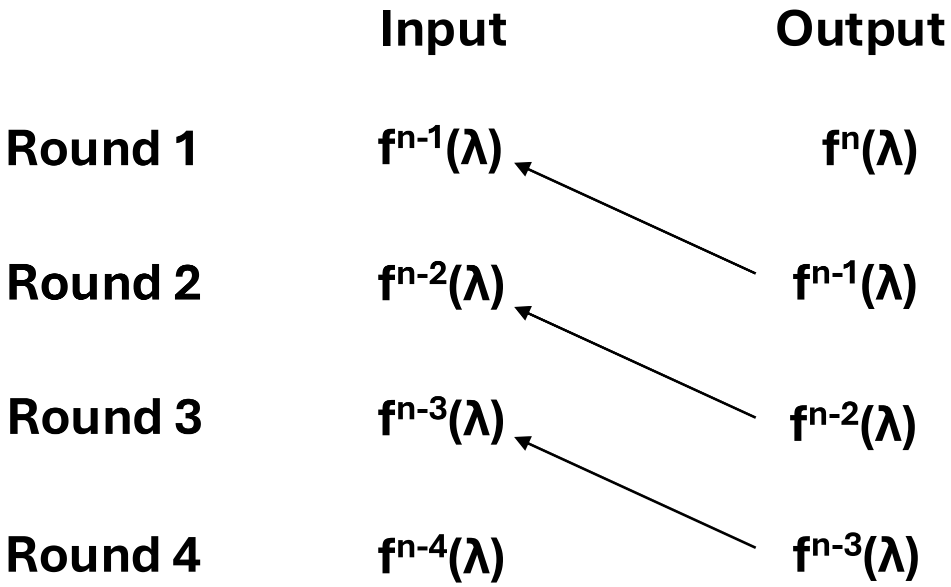
	Input	Output
Round 1		$f^n(\lambda)$
Round 2		
Round 3		
Round 4		

	Input	Output
Round 1	$f^{n-1}(\lambda)$	$f^{n}(\lambda)$
Round 2		
Round 3		
Round 4		

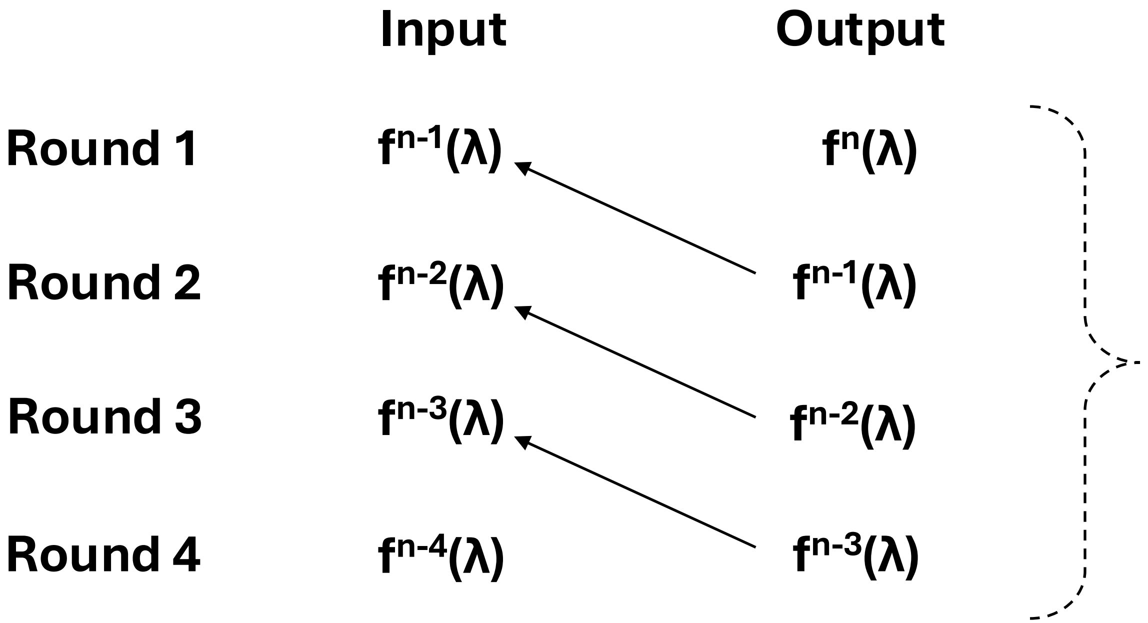
$$\begin{tikzcd}
 & \textbf{Input} & \textbf{Output} \\
\textbf{Round 1} & f^{n-1}(\lambda) & f^{n}(\lambda) \\
\textbf{Round 2} & f^{n-2}(\lambda) & f^{n-1}(\lambda) \arrow[ul] \\
\textbf{Round 3} & & \\
\textbf{Round 4} & &
\end{tikzcd}$$



	Input	Output
Round 1	$f^{n-1}(\lambda)$	$f^{n}(\lambda)$
Round 2	$f^{n-2}(\lambda)$	$f^{n-1}(\lambda)$
Round 3	$f^{n-3}(\lambda)$	$f^{n-2}(\lambda)$
Round 4		



	Input	Output
Round 1	$f^{n-1}(\lambda)$	$f^{n}(\lambda)$
Round 2	$f^{n-2}(\lambda)$	$f^{n-1}(\lambda)$
Round 3	$f^{n-3}(\lambda)$	$f^{n-2}(\lambda)$
Round 4	$f^{n-4}(\lambda)$	$f^{n-3}(\lambda)$



	Input	Output
Round 1	$f^{n-1}(\lambda)$	$f^{n}(\lambda)$
Round 2	$f^{n-2}(\lambda)$	$f^{n-1}(\lambda)$
Round 3	$f^{n-3}(\lambda)$	$f^{n-2}(\lambda)$
Round 4	$f^{n-4}(\lambda)$	$f^{n-3}(\lambda)$

By waiting for successive rounds, observer Eve can see the plaintext for the previous round

Implies *Known Plaintext* Cryptanalysis

What are the Three Strategies for Cryptanalysis?

Three Methods for Cryptanalysis

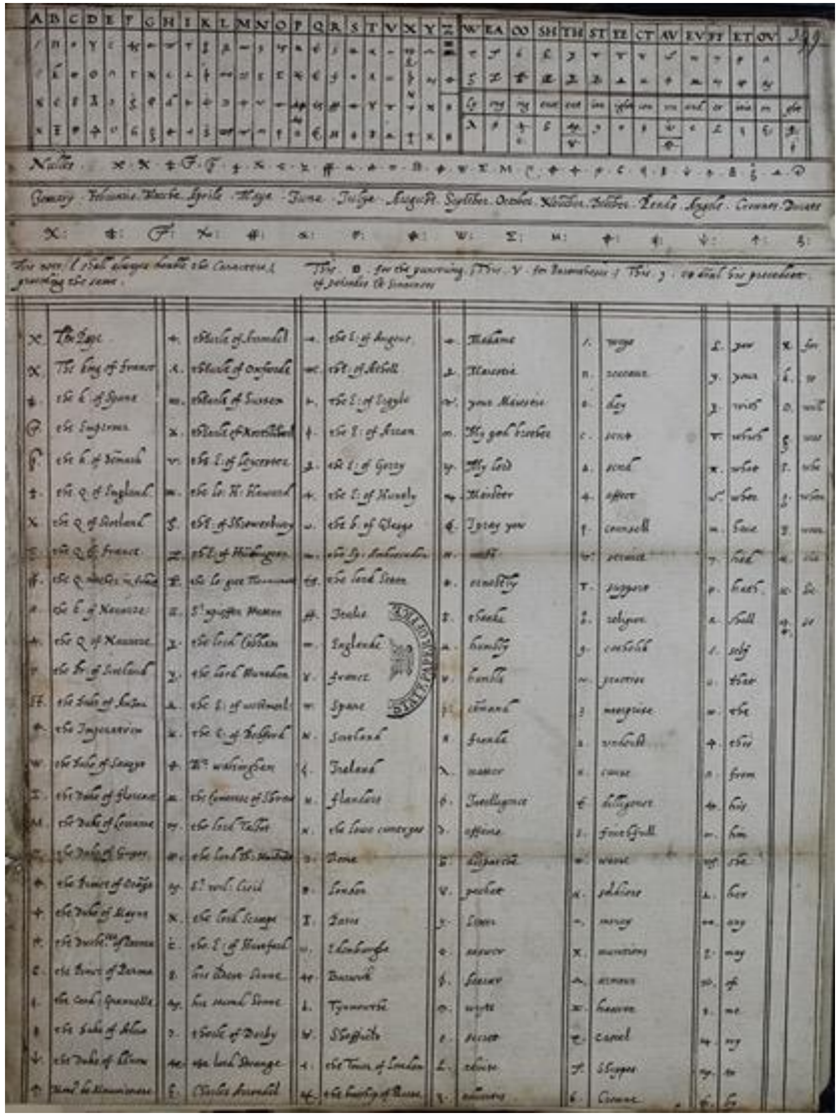
- **Ciphertext Only**
  - Cryptanalyst only has encrypted text
  - No hints or codebooks

Three Methods for Cryptanalysis

- **Ciphertext Only**
    - Cryptanalyst only has encrypted text
    - No hints or codebooks
- **Known Plaintext**
    - Cryptanalyst observes hints (no control)
    - Tiny codebook examples can be developed

Three Methods for Cryptanalysis

- **Ciphertext Only**
  - Cryptanalyst only has encrypted text
  - No hints or codebooks
- **Known Plaintext**
  - Cryptanalyst observes hints (no control)
  - Tiny codebook examples can be developed
- **Chosen Plaintext**
  - Cryptanalyst has the encryption function
  - Codebook can be developed



Three Methods for Cryptanalysis

- **Ciphertext Only**
  - Cryptanalyst only has encrypted text
  - No hints or codebooks
- **Known Plaintext**
  - Cryptanalyst observes hints (no control)
  - Tiny codebook examples can be developed
- **Chosen Plaintext**
  - Cryptanalyst has the encryption function
  - Codebook can be developed

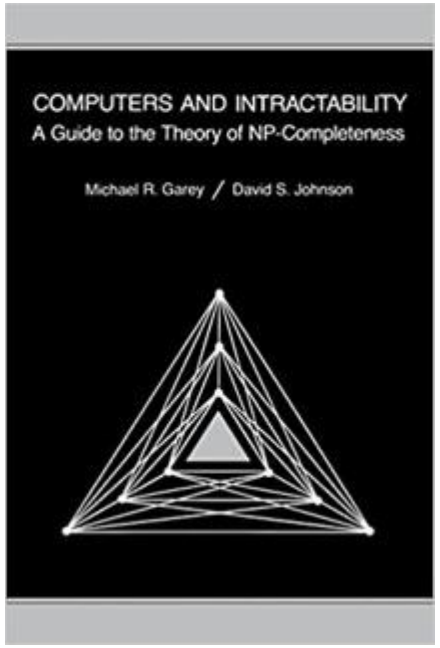
Two requirements protect encrypted text:

1. The encryption function must be cryptographically har
2. The cleartext and ciphertext domains must be huge

Two Implications:

You must try every possible case to find the encryption function

The number of possible cases cannot feasibly be covered


COMPUTERS AND INTRACTABILITY
A Guide to the Theory of NP-Completeness
Michael R. Garey / David S. Johnson