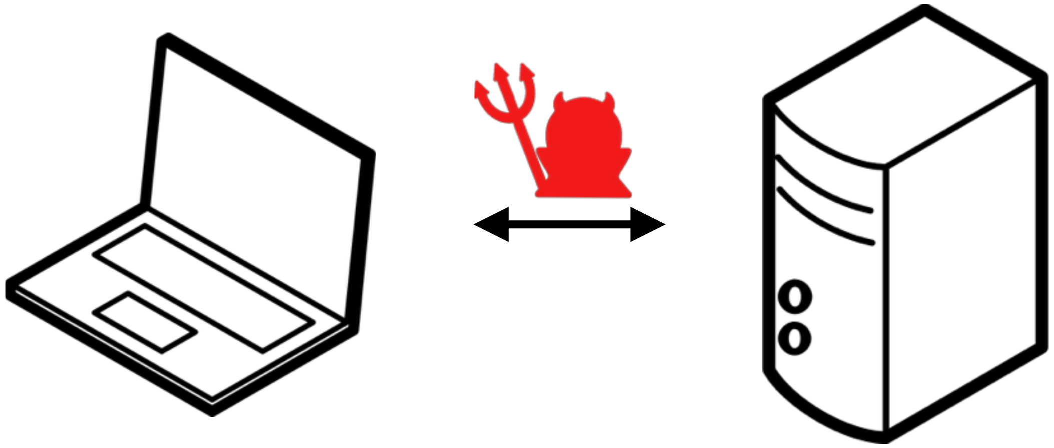
Human Errors in Security Protocols

David Basin
joint work with Saša Radomirović and Lara Schmid

Institute of Information Security

ETH zürich

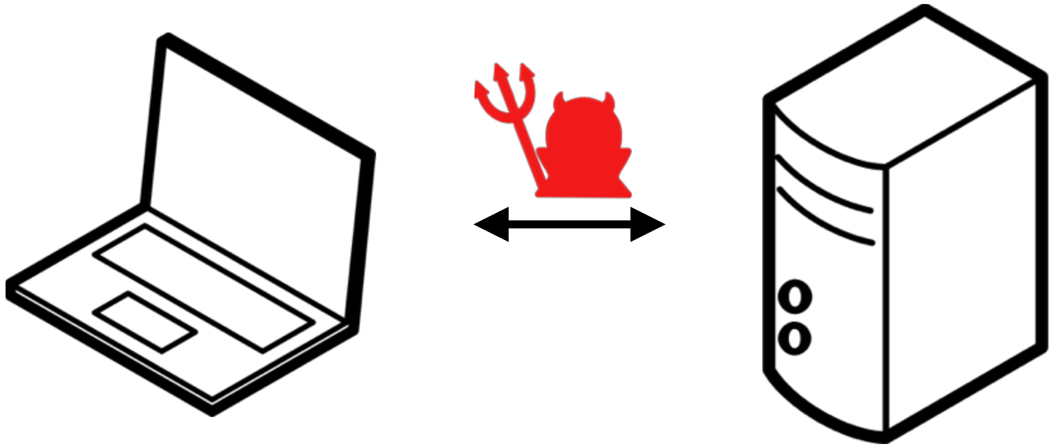
Recap Security Protocols



We have defined

- the **Dolev-Yao** adversary
- **communicating agents** that follow a **role specification**
- and the **security properties** desired

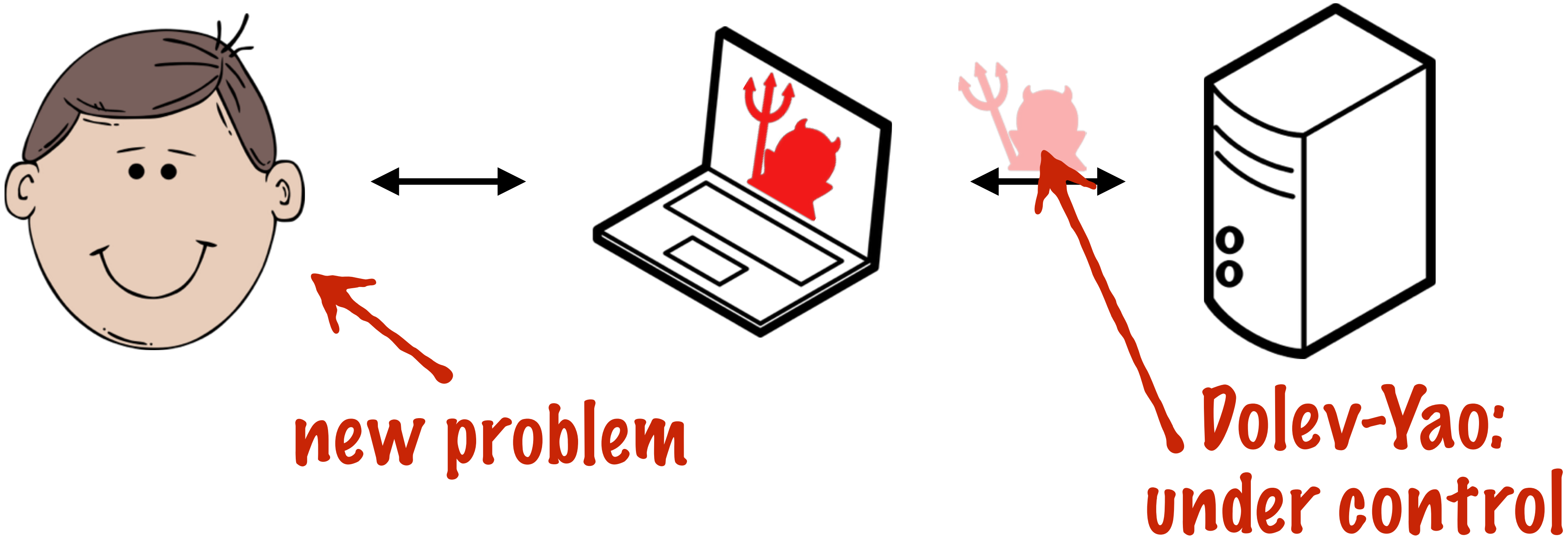
What we have not seen



We glossed over that

- a **human** is one of the communicating parties,
- humans have **limited** computational abilities, and
- they are **error-prone**.

How can we achieve secure communication between a human and a remote server?



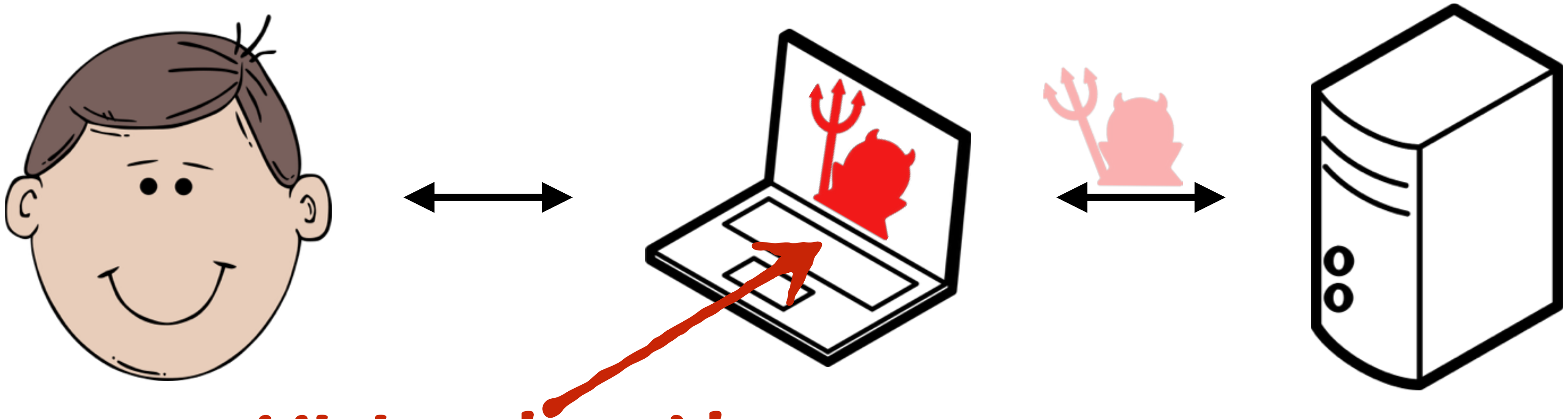
new problem

Dolev-Yao: under control

- Examples: Online Banking, Internet Voting, Electronic Tax Returns, …

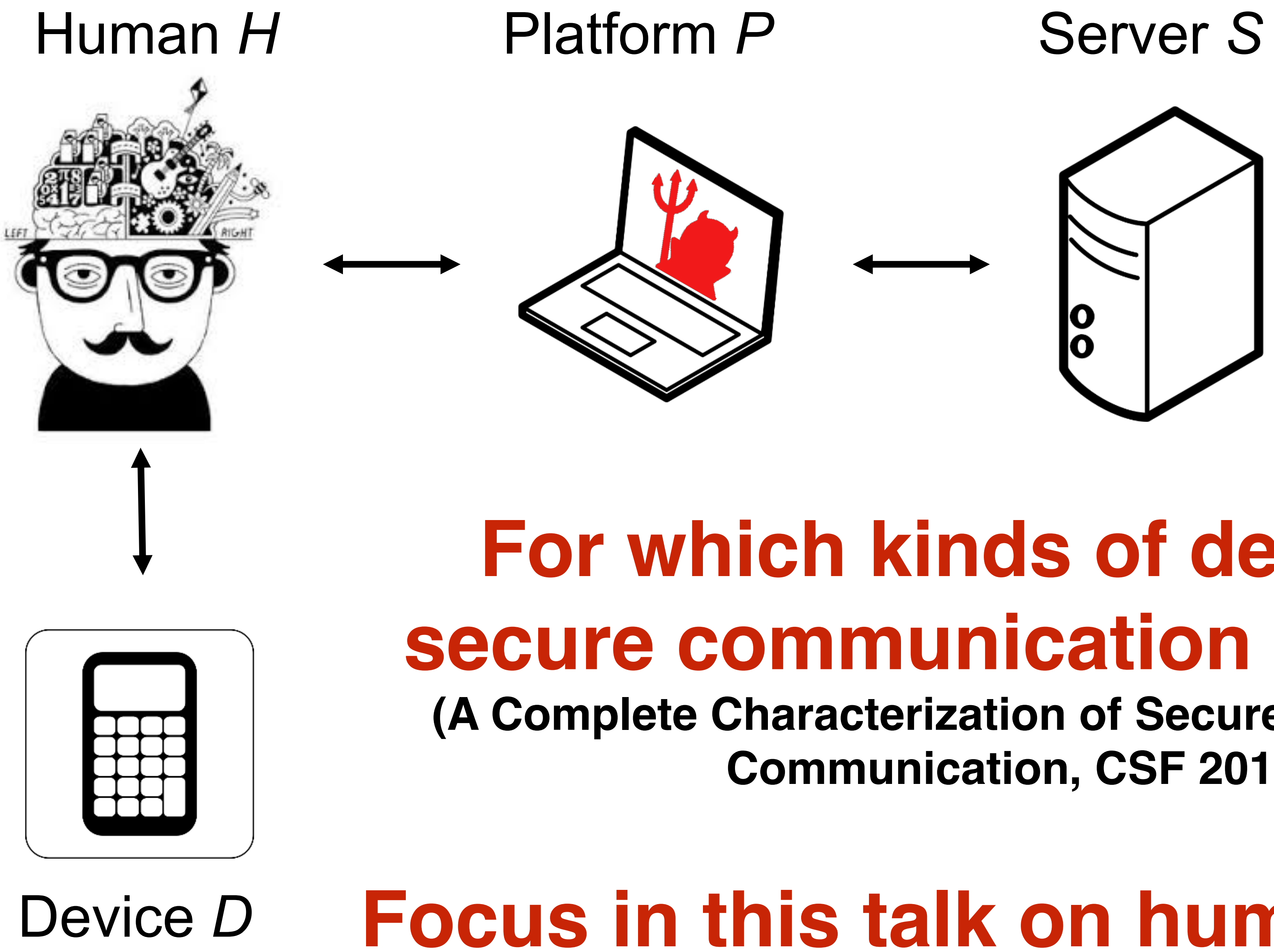
- How do we model and reason about interaction between humans and computers?

How can we achieve secure communication between a human and a remote server?



additional problem: compromised platform

- If platform is compromised: no useful secure communication is possible.
- A trusted device is necessary.



For which kinds of devices is secure communication possible?

(A Complete Characterization of Secure Human-Server Communication, CSF 2015)

Focus in this talk on human errors

(Modeling Human Errors in Security Protocols, CSF 2016)

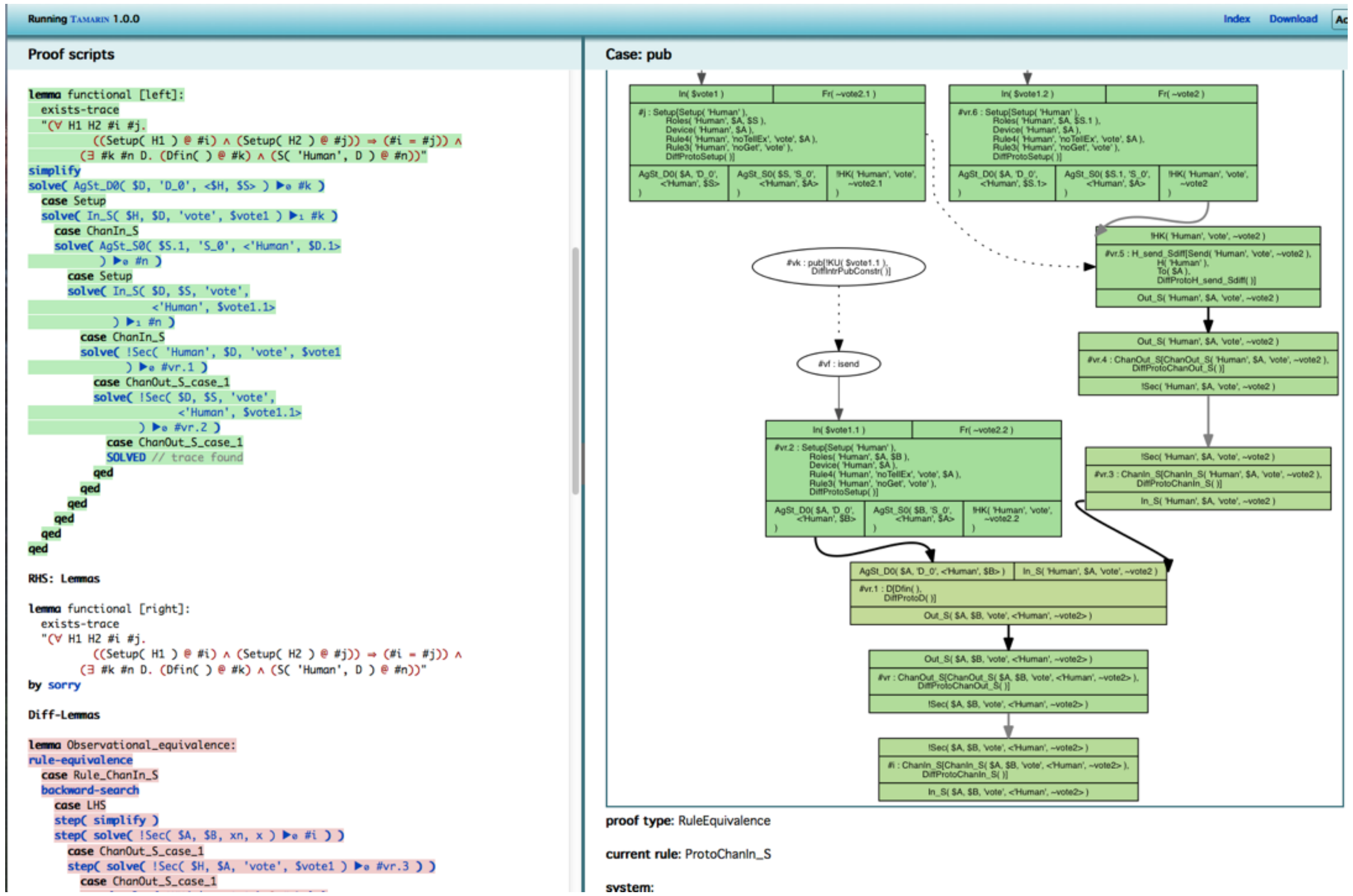
Possible “devices”:

Overview

1. **Security protocol model**
2. Modelling Human Error
3. Applications

Security Protocol Model — Tamarin

- Symbolic formal model specified using **multiset rewriting**
- **Dolev-Yao adversary** controlling communication network.
- Possible executions modeled by **traces**
- **Tool support**



Proof scripts

```
lemma functional [left]:
  exists-trace
  "(∀ H1 H2 #i #j.
         ((Setup( H1 ) @ #i) ∧ (Setup( H2 ) @ #j)) ⇒ (#i = #j)) ∧
        (∃ #k #n D. (Dfin( ) @ #k) ∧ (S( 'Human', D ) @ #n))"
simplify
solve( AgSt_D0( $D, 'D_0', <$H, $S> ) ▶₀ #k )
  case Setup
  solve( In_S( $H, $D, 'vote', $vote1 ) ▶₁ #k )
    case ChanIn_S
    solve( AgSt_S0( $S.1, 'S_0', <'Human', $D.1>
                ) ▶₀ #n )
      case Setup
      solve( In_S( $D, $S, 'vote',
                      <'Human', $vote1.1>
                 ) ▶₁ #n )
        case ChanIn_S
        solve( !Sec( 'Human', $D, 'vote', $vote1
                  ) ▶₀ #vr.1 )
          case ChanOut_S_case_1
          solve( !Sec( $D, $S, 'vote',
                          <'Human', $vote1.1>
                   ) ▶₀ #vr.2 )
            case ChanOut_S_case_1
            SOLVED // trace found
          qed
        qed
      qed
    qed
  qed
qed

RHS: Lemmas

lemma functional [right]:
  exists-trace
  "(∀ H1 H2 #i #j.
         ((Setup( H1 ) @ #i) ∧ (Setup( H2 ) @ #j)) ⇒ (#i = #j)) ∧
        (∃ #k #n D. (Dfin( ) @ #k) ∧ (S( 'Human', D ) @ #n))"
by sorry

Diff-Lemmas

lemma Observational_equivalence:
rule-equivalence
  case Rule_ChanIn_S
  backward-search
    case LHS
    step( simplify )
    step( solve( !Sec( $A, $B, xn, x ) ▶₀ #i ) )
      case ChanOut_S_case_1
      step( solve( !Sec( $H, $A, 'vote', $vote1 ) ▶₀ #vr.3 ) )
        case ChanOut_S_case_1
```

Case: pub

In($vote1)
Fr(~vote2.1)
#j : Setup[Setup('Human'), Roles('Human', $A, $S), Device('Human', $A), Rule4('Human', 'noTellEx', 'vote', $A), Rule3('Human', 'noGet', 'vote'), DiffProtoSetup( )]
AgSt_D0($A, 'D_0', <'Human', $S>)
AgSt_S0($S, 'S_0', <'Human', $A>)
!HK('Human', 'vote', ~vote2.1)
In($vote1.2)
Fr(~vote2)
#vr.6 : Setup[Setup('Human'), Roles('Human', $A, $S.1), Device('Human', $A), Rule4('Human', 'noTellEx', 'vote', $A), Rule3('Human', 'noGet', 'vote'), DiffProtoSetup( )]
AgSt_D0($A, 'D_0', <'Human', $S.1>)
AgSt_S0($S.1, 'S_0', <'Human', $A>)
!HK('Human', 'vote', ~vote2)
!HK('Human', 'vote', ~vote2)
#vr.5 : H_send_Sdiff[Send('Human', 'vote', ~vote2), H('Human'), To($A), DiffProtoH_send_Sdiff( )]
Out_S('Human', $A, 'vote', ~vote2)
#vk : pub[!KU($vote1.1), DiffIntrPubConstr( )]
#vf : isend
Out_S('Human', $A, 'vote', ~vote2)
#vr.4 : ChanOut_S[ChanOut_S('Human', $A, 'vote', ~vote2), DiffProtoChanOut_S( )]
!Sec('Human', $A, 'vote', ~vote2)
In($vote1.1)
Fr(~vote2.2)
#vr.2 : Setup[Setup('Human'), Roles('Human', $A, $B), Device('Human', $A), Rule4('Human', 'noTellEx', 'vote', $A), Rule3('Human', 'noGet', 'vote'), DiffProtoSetup( )]
AgSt_D0($A, 'D_0', <'Human', $B>)
AgSt_S0($B, 'S_0', <'Human', $A>)
!HK('Human', 'vote', ~vote2.2)
!Sec('Human', $A, 'vote', ~vote2)
#vr.3 : ChanIn_S[ChanIn_S('Human', $A, 'vote', ~vote2), DiffProtoChanIn_S( )]
In_S('Human', $A, 'vote', ~vote2)
AgSt_D0($A, 'D_0', <'Human', $B>)
In_S('Human', $A, 'vote', ~vote2)
#vr.1 : D[Dfin( ), DiffProtoD( )]
Out_S($A, $B, 'vote', <'Human', ~vote2>)
Out_S($A, $B, 'vote', <'Human', ~vote2>)
#vr : ChanOut_S[ChanOut_S($A, $B, 'vote', <'Human', ~vote2>), DiffProtoChanOut_S( )]
!Sec($A, $B, 'vote', <'Human', ~vote2>)
!Sec($A, $B, 'vote', <'Human', ~vote2>)
#i : ChanIn_S[ChanIn_S($A, $B, 'vote', <'Human', ~vote2>), DiffProtoChanIn_S( )]
In_S($A, $B, 'vote', <'Human', ~vote2>)

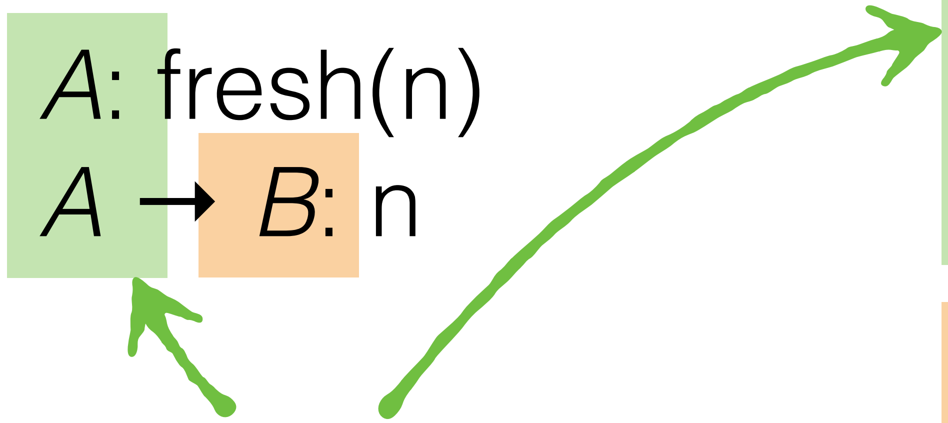
proof type: RuleEquivalence

current rule: ProtoChanIn_S

system:

Protocol Specification Example

Alice & Bob specification



A: fresh(n)

A → *B*: n

Role specification of A

Protocol rules

$$[\ \mathrm{Fr}(n)\ ] \longrightarrow [\ \mathrm{AgSt}(n)\ ]$$

$$[\ \mathrm{AgSt}(n)\ ] \xrightarrow{S(n)} [\ \mathrm{Out}(n)\ ]$$

$$[\ \mathrm{In}(n)\ ] \xrightarrow{R(n)} [\ ]$$

Fresh rule

$$[\ \ ] \longrightarrow [\ \mathrm{Fr}(n)\ ]$$

Adversary rules (simplified)

$$[\ \mathrm{Out}(n)\ ] \xrightarrow{K(n)} [\ !K(n)\ ]$$

$$[\ !K(n)\ ] \longrightarrow [\ \mathrm{In}(n)\ ]$$

$$[\ !K(n),\ !K(m)\ ] \longrightarrow [\ !K(\ \mathrm{pair}(n,m)\ )\ ]$$

$$[\ \ ] \longrightarrow [\ \ !K(\$x)\ ] \qquad (\$x\text{: public term})$$

…

Protocol Execution Example

State	Term Rewriting Rule	Instantiation	Trace
[ ]	[ ] ⟶ [Fr(n)]		
[Fr(~1)]	[Fr(n)] ⟶ [AgSt(n)]		
[AgSt(~1)]	[AgSt(n)] $\xrightarrow{S(n)}$ [Out(n)]		S(~1)
[Out(~1)]	[Out(n)] $\xrightarrow{K(n)}$ [!K(n)]		K(~1)
[!K(~1)]	[!K(n)] ⟶ [In(n)]	[!K(~1)] ⟶ [In(~1)]	
[!K(~1), In(~1)]	[In(n)] $\xrightarrow{R(n)}$ [ ]	[In(n)] $\xrightarrow{R(\sim 1)}$ [ ]	R(~1)

Specified rules:

[ ] ⟶ [Fr(n)]

[Fr(n)] ⟶ [AgSt(n)]

[AgSt(n)] ⟶ [Out(n)]

[Out(n)] ⟶ [!K(n)]

[!K(n)] ⟶ [In(n)]

…

Linear vs persistent facts

Communication Channels

Authentic •→○, confidential ○→•, and secure •→• channel rules are used to restrict capabilities of Dolev-Yao adversary.

Example: Confidential channel rules

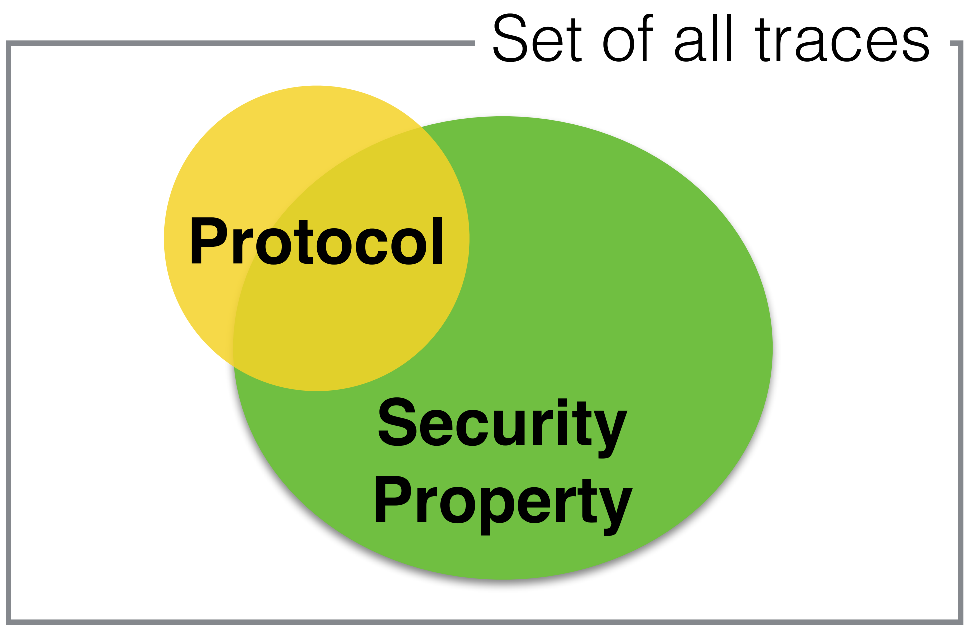
[SndC($A,$B,m)] ⟶ [!Conf($B,m)]

$ sign: public term. Agent names are public knowledge.

[!Conf($B,m), !K($A)] ⟶ [RcvC($A,$B,m)]

[!K(<$A,$B,m>)] ⟶ [RcvC($A,$B,m)]

Security Properties

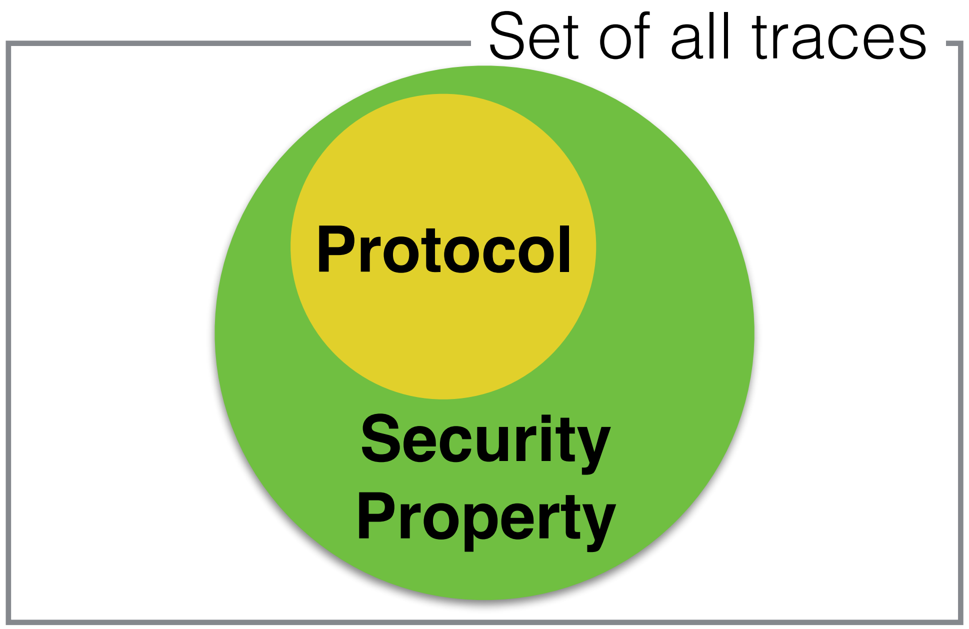


Set of all traces

Protocol

Security Property

Protocol does not satisfy security property.



Set of all traces

Protocol

Security Property

Protocol satisfies security property.

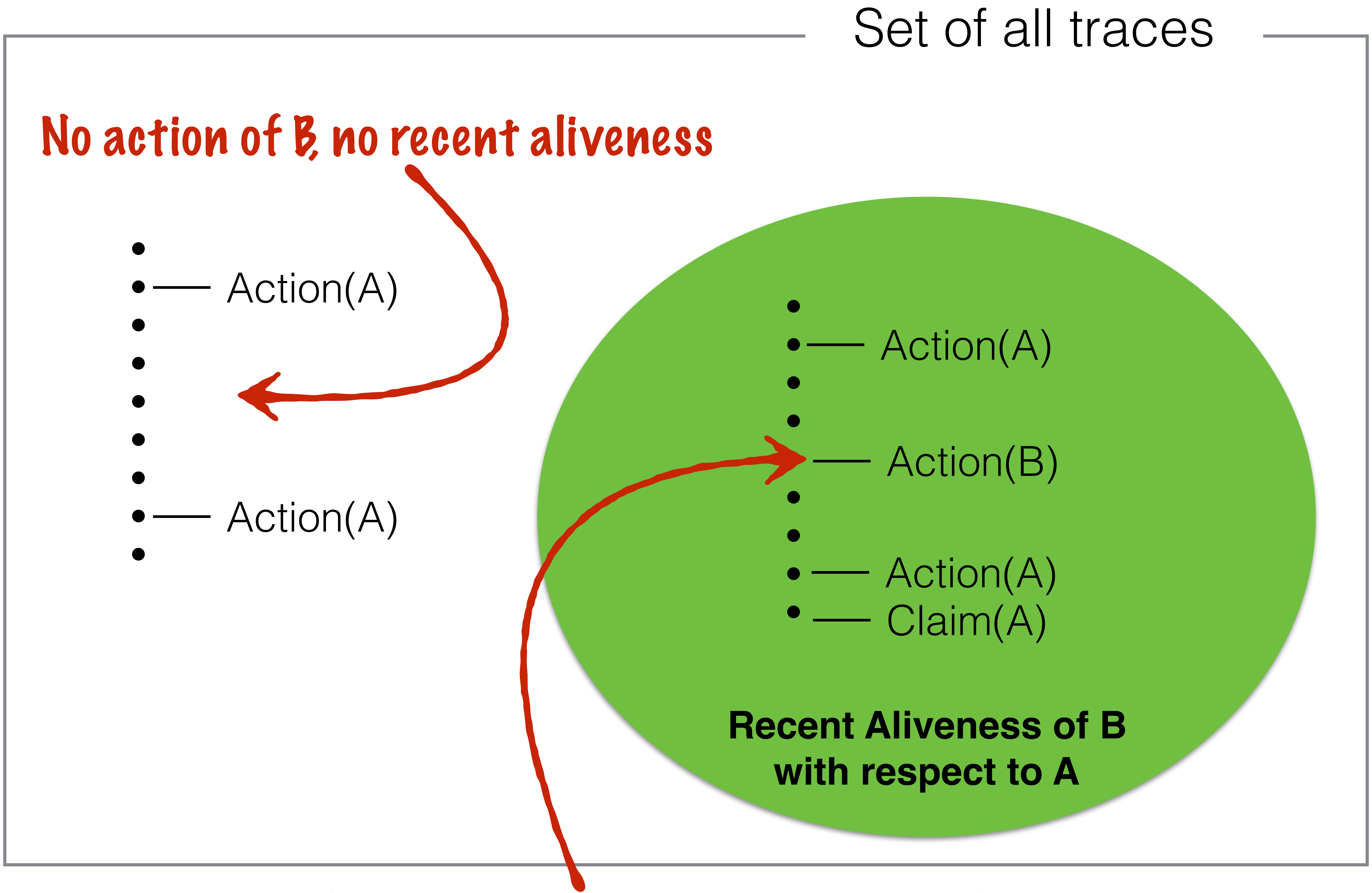
Confidentiality

If *m* is claimed to be secret, then the adversary does not learn *m*.

Set of traces:

$$\forall\ m\ \#i\ \#j.\ \ \mathrm{Secret}(m)@i\ \ \Rightarrow\ \ \mathrm{not}\ \ \mathrm{K}(m)@j$$

Authentication Properties: Recent Aliveness



Set of all traces

No action of B, no recent aliveness

— Action(A)

— Action(A)

— Action(A)

— Action(B)

— Action(A)

— Claim(A)

Recent Aliveness of B with respect to A

Action of B occurs between two events of A.

Authentication Properties

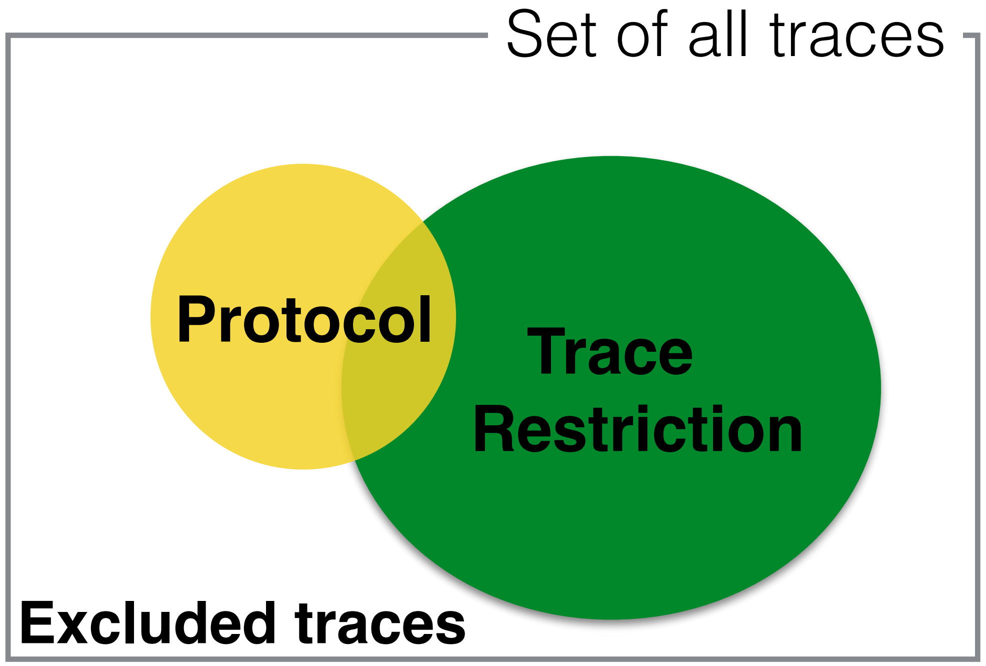
Entity Authentication: Recent aliveness of an entity *H*, with respect to verifier (remote server *S*).

Device Authentication: Recent aliveness of a device *D.* We generally assume exclusive access of human *H* to *D*.

Message Authentication: If verifier claims that *H* has sent *m*, then *H* has indeed sent *m*.

Trace Restrictions

Exclude traces that violate the specification.



Only traces in intersection are considered.

Example: A trusted agent was not previously dishonest.

Set of traces:

$$\forall\ A\ \#i\ \#j.\ \ (\text{Trusted}(A)@i\ \wedge\ \text{Dishonest}(A)@j\ )\ \Rightarrow\ i < j$$

Modelling Humans

- Humans can **communicate** over provided interfaces.
- Human **knowledge** is modelled with $!\mathrm{HK}(\overset{\text{agent,}}{H},\overset{\text{type,}}{t},\overset{\text{message}}{m})$ facts.
  E.g.: $!\mathrm{HK}(H,\text{'pw'},p)$ means human *H* knows password *p*.
- Humans can **concatenate** and **split** messages:

$$[\ !\mathrm{HK}(H,t_1,m_1),\ !\mathrm{HK}(H,t_2,m_2)\ ] \longrightarrow [\ !\mathrm{HK}(H,<t_1,t_2>,<m_1,m_2>)\ ]$$

$$[\ !\mathrm{HK}(H,<t_1,t_2>,<m_1,m_2>)\ ] \longrightarrow [\ !\mathrm{HK}(H,t_1,m_1),\ !\mathrm{HK}(H,t_2,m_2)\ ]$$

(simplified rules)

Overview

1. Security protocol model
2. **Modelling Human Error**
3. Applications

Modelling Human Error

- Users **don’t know protocol** specifications
- **Mistakes are made**, even experts slip up
- We are susceptible to **social engineering**
- So how should we analyze security of systems in view of human errors?

Definition

A **human error** in a protocol execution is **any deviation** of a human from his or her **role specification**.

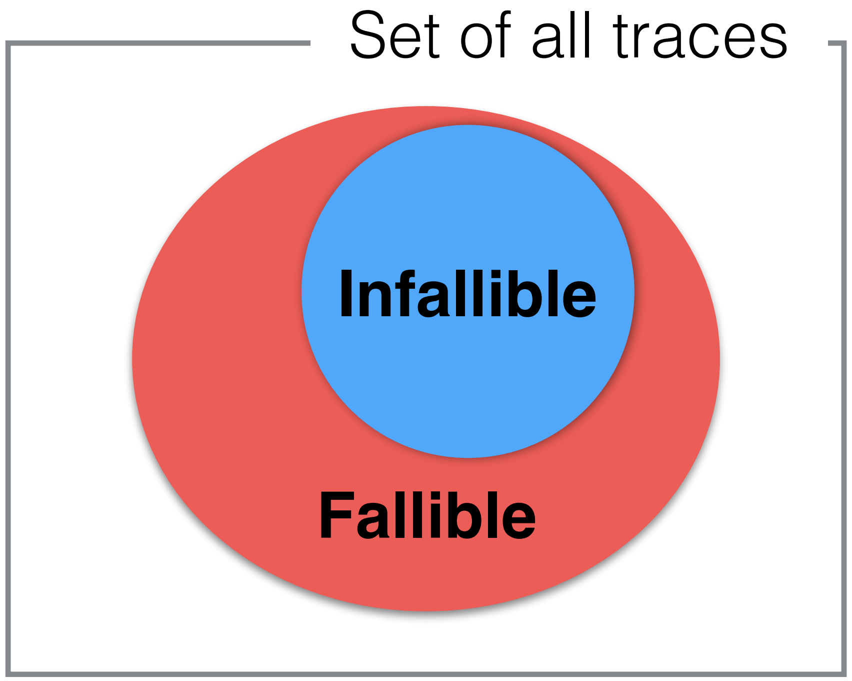
Two Classes of Human Error

- Distinguish between slips and lapses by **skilled users** and mistakes by **inexperienced users**.
- Model slips and lapses: Allow an **infallible agent** to make a small number of mistakes.
- Model rule-based behaviour: Allow for arbitrary behaviour of an **untrained agent** up to a few simple rules (guidelines).

IF YOU DIDN'T EXPECT IT, REJECT IT
REJECT
DON'T CLICK!
CONSULT@BERKELEY.EDU • 510-664-9000 • SECURITY.BERKELEY.EDU/PHISHING

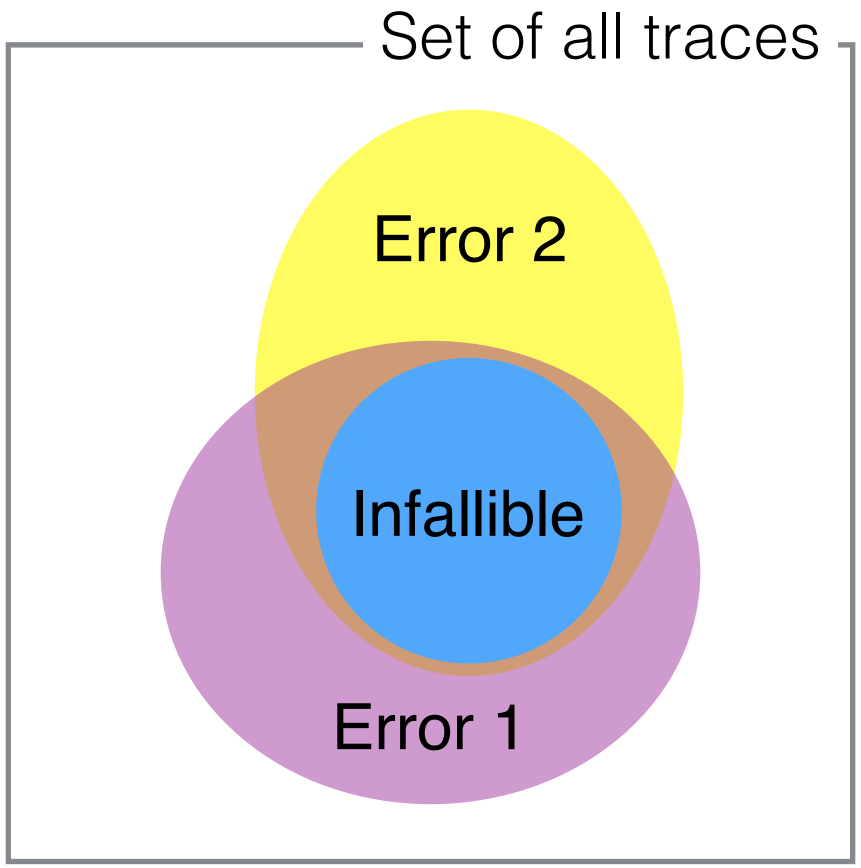
Infallible vs Fallible Humans

- **Infallible** human follows protocol specification.
- **Fallible** human *may* deviate from protocol specification.
- Fallible humans give rise to **more system behaviours** than the infallible human.

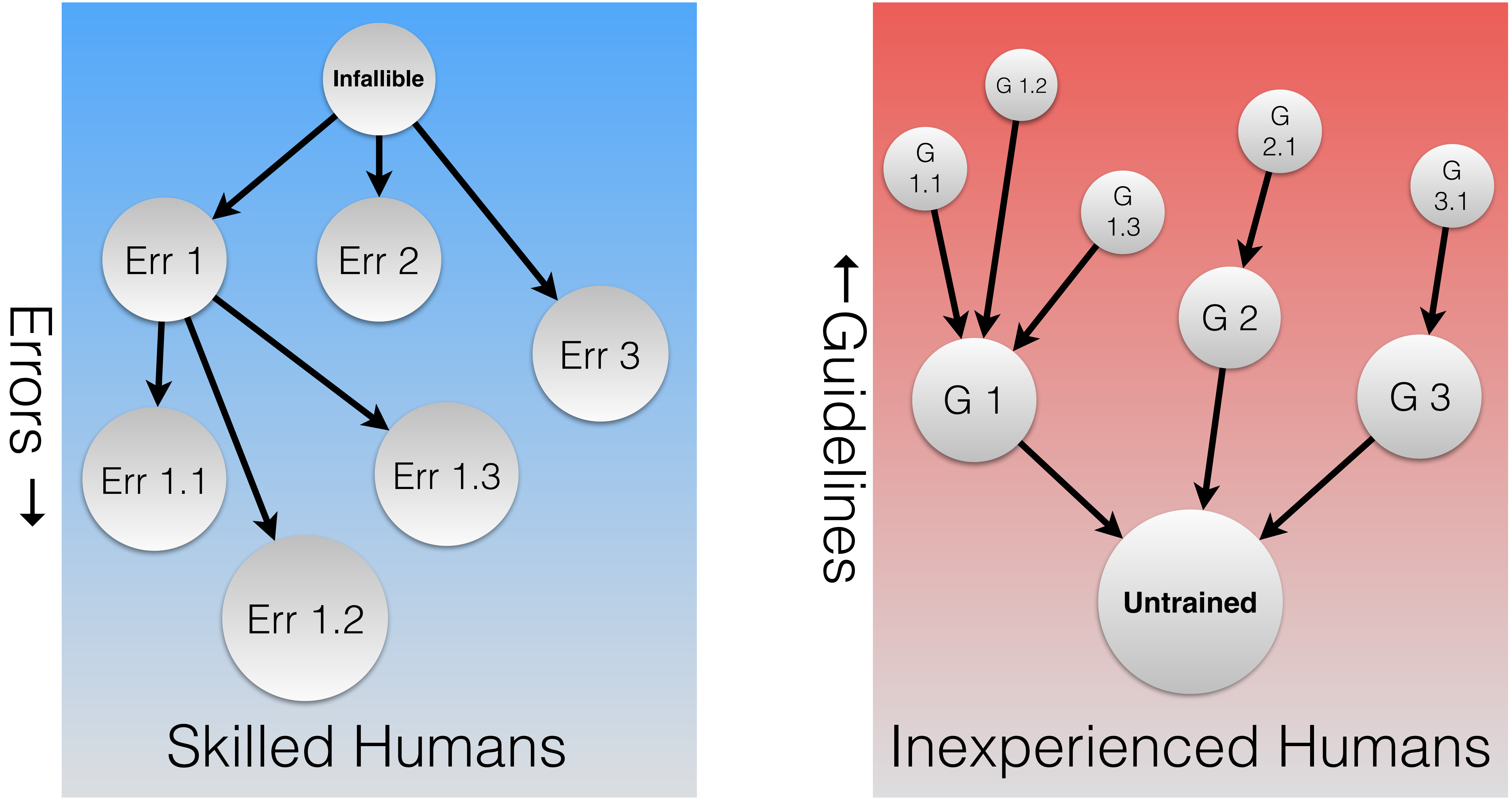


Comparing Specific Errors

Partial order of human errors by comparing sets of induced traces.

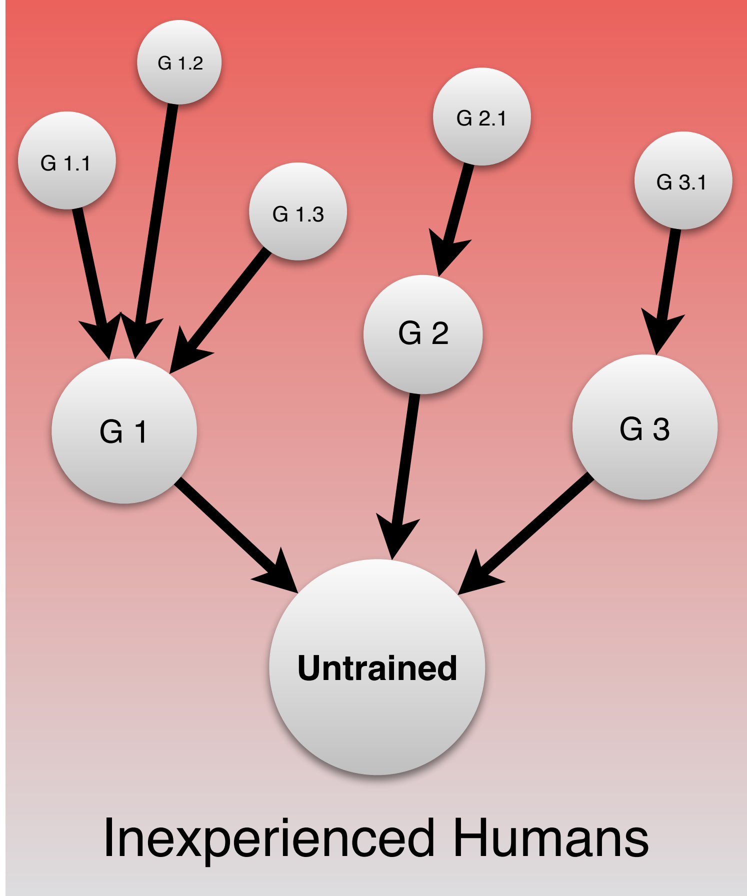


Two Classes of Human Error



Arrows indicate trace-set containment
(node at arrowhead contains more behaviors than node at tail)

Untrained Humans



We focus on this class

- They are **ignorant of** and **may deviate arbitrarily** from protocol specification.

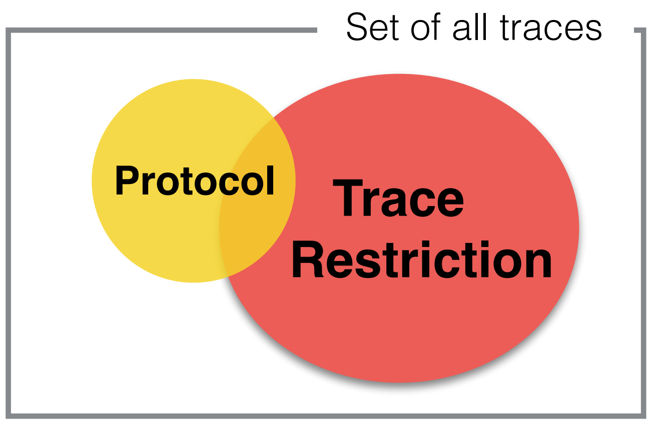
- They accept any message received and send any message requested.

$$[\ \mathrm{In}(<\mathit{tag},\mathit{msg}>)\ ] \longrightarrow [\ !\mathrm{HK}(H,\mathit{tag},\mathit{msg})\ ]$$

$$[\ !\mathrm{HK}(H,\mathit{tag},\mathit{msg})\ ] \longrightarrow [\ \mathrm{Out}(<\mathit{tag},\mathit{msg}>)\ ]$$

(Trace labels omitted.)

- But they can be trained, given guidelines!



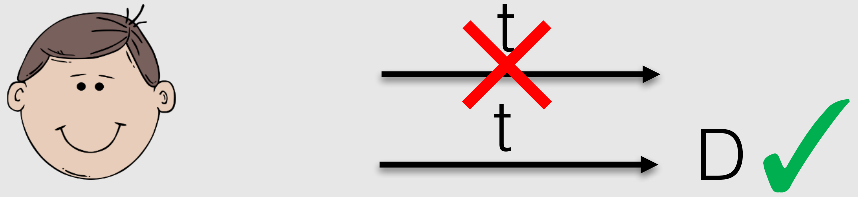
Guidelines

Guidelines are modelled by trace restrictions.

IF YOU DIDN'T EXPECT IT, REJECT IT
REJECT
DON'T CLICK!
CONSULT@BERKELEY.EDU . 510-664-9000 . SECURITY.BERKELEY.EDU/PHISHING

PASSWORDS IN EMAIL=EPIC FAIL
***
F
DON'T SEND IT!
CONSULT@BERKELEY.EDU . 510-664-9000 . SECURITY.BERKELEY.EDU/PHISHING

Exemplary Guidelines I

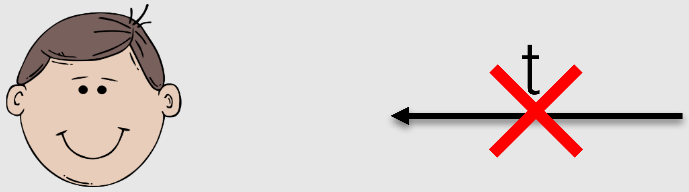
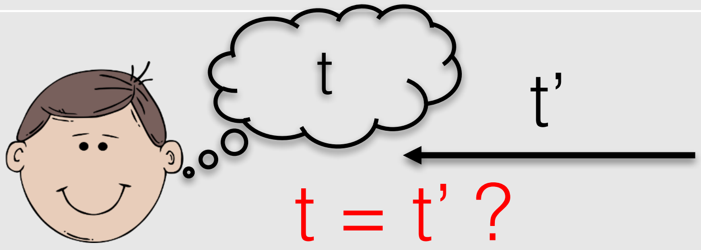
NoTell(H,t)	
NoTellExcept(H,t,D)	

- **NoTell**(*H*,*tag*):
  $\forall\ m\ \#i\ \#j.\ \mathrm{NoTell}(H,tag)@i \ \Rightarrow\ \text{not Snd}(H,<tag,m>)@j$

  Human *H* does not send information of type tag to anyone.
  E.g.: Never reveal your private key.

- **NoTellExcept**(*H*,*tag*,*D*):
  Human *H* does not send information of type *tag* to anyone except *D*.
  E.g.: Only enter your password into your own device.

Exemplary Guidelines II

NoGet(H,t)	
ICompare(H,t)	

- **NoGet**(*H*,*tag*): Human *H* rejects information of type *tag* from everyone.
  E.g.: Never click on links in emails.

- **ICompare**(*H*,*tag*): Human *H* always compares received information of type *tag* with information in his initial knowledge.
  E.g.: Always check the website's URL.

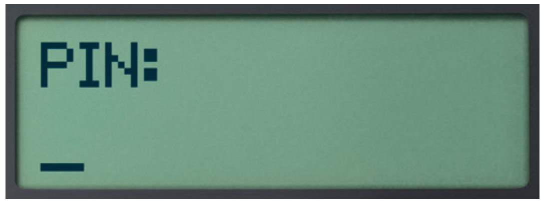
Concrete example — ebanking

Login with the Access Card and card reader

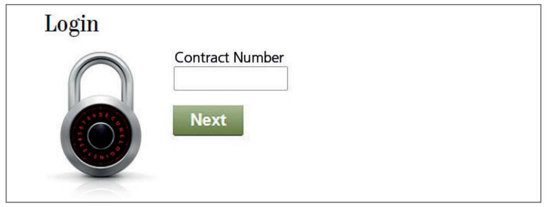
Access the desired online service via ubs.com/online and initiate the login process (self-authorization).

1. Activate the card reader by inserting the Access Card.

2. Enter your PIN and press OK .


PIN:
_

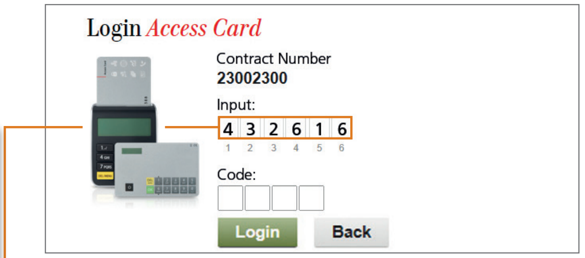
3. Enter your contract number on the login page and click **Next.**


Login

Contract Number

Next

4. Enter the six-digit code displayed on the login page into the card reader and press OK .

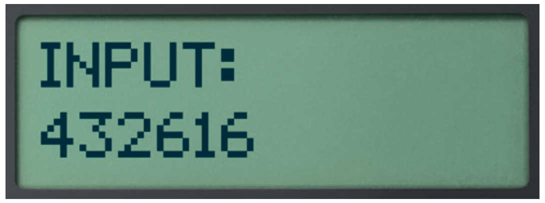

Login *Access Card*

Contract Number
23002300

Input:
4 3 2 6 1 6
1 2 3 4 5 6

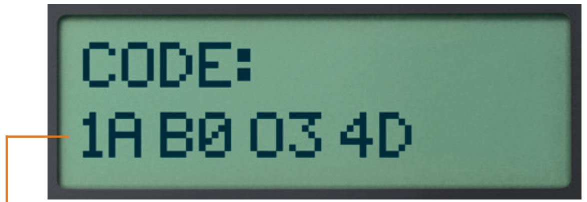
Code:
□□□□

Login Back


INPUT:
432616

Security note: The login number displayed by UBS **always has six digits.** If it has fewer digits, this could be a case of attempted fraud. Contact the support team as soon as possible in this case.

5. Enter the eight-digit code from the card reader on the login page and click **Login.**


CODE:
1A B0 03 4D

Overview

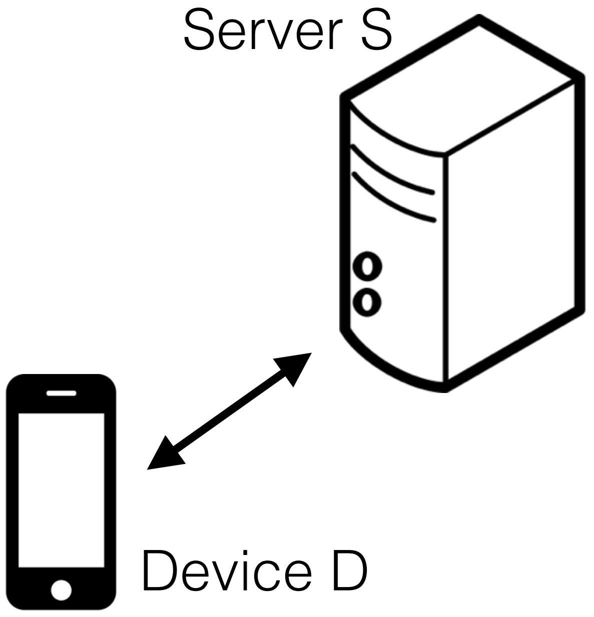
1. Security protocol model
2. Modelling Human Error
3. **Applications**

Phone-based Authentication

- **Cronto**: Scan a code on platform, decrypted by mobile device, enter code + password on platform
- **Google 2-step**: login/password + SMS
- **MP-Auth**: Enter password into mobile device
- **One-time passwords over SMS**: single-factor authentication
- **Phoolproof**: choose server on device, device-server communication, then enter password on the platform
- **Sound-Proof**: ambient noise recorded by platform and mobile

MP-Auth* without Human Role

D knows: H, pk(S), pw
S knows: H, sk(S), pw



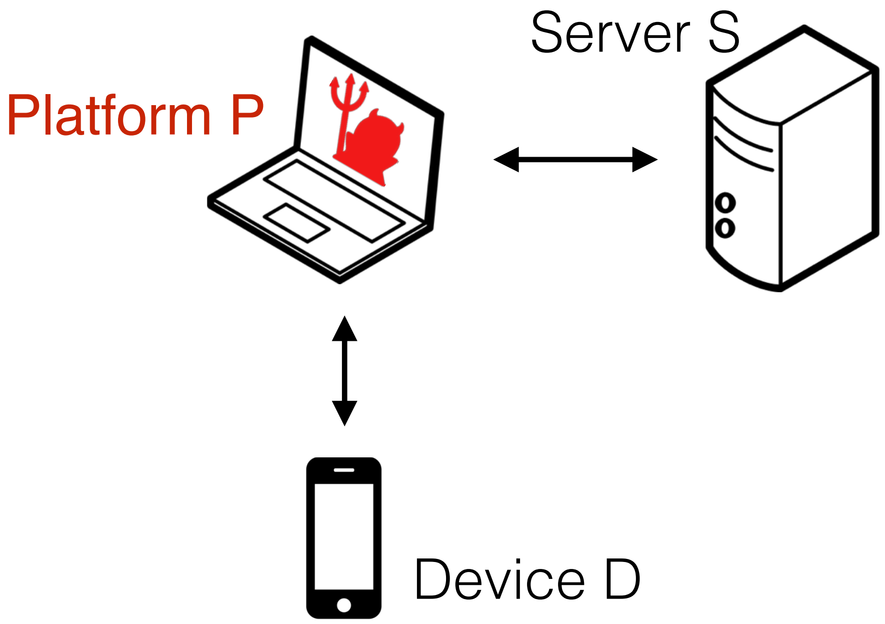
S: fresh(r_S)
S → D: S, r_S fresh session, no replay
D: fresh(r_D) only S can read this
D → S: $\{r_D\}_{pk(S)}$, $\{h(r_S)$, H, pw$\}_{h(r_S,r_D)}$ shared key: only D and S can compute this
D must have sent this
S → D: $\{h(r_D)\}_{h(r_S,r_D)}$
S must have sent this

satisfies confidentiality & authenticity of $h(r_S, r_D)$

. (*) Mohammad Mannan and Paul C. van Oorschot. Leveraging personal devices for stronger password authentication from untrusted computers. *Journal of Computer Security*, 19(4):703–750, 2011.

MP-Auth without Human Role

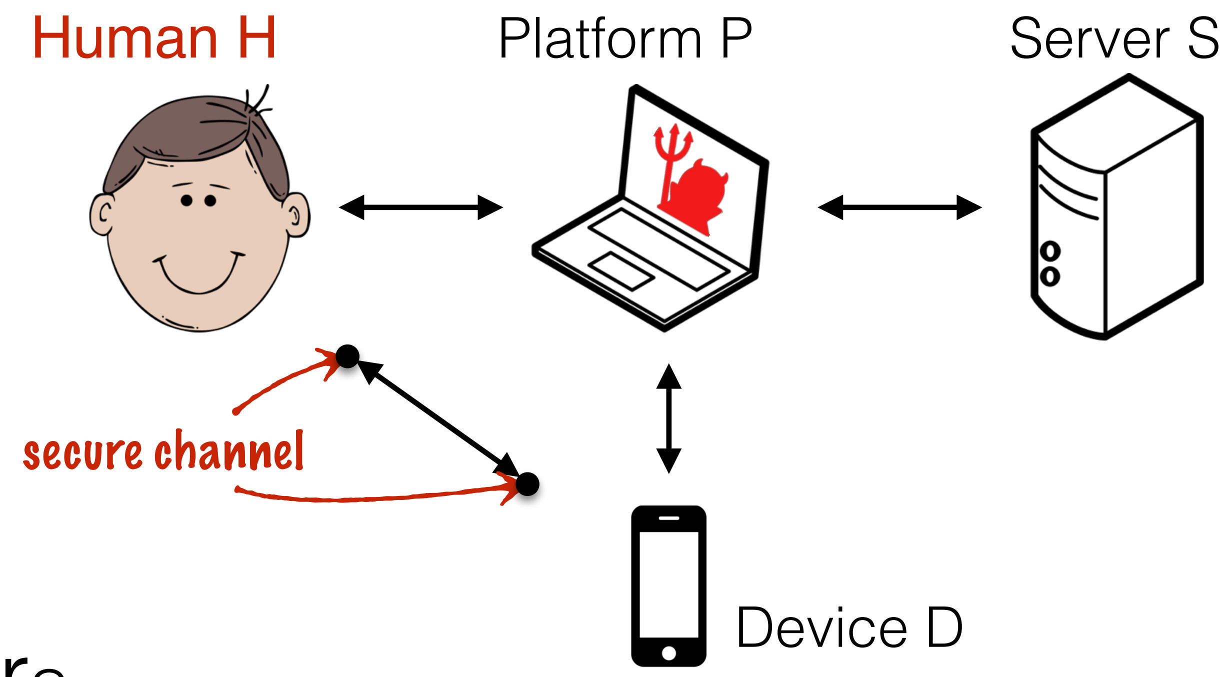
D knows: H, pk(S), pw
S knows: H, sk(S), pw



S: fresh(r_S)
S → P → D: S, r_S
D: fresh(r_D)
D → P → S: $\{r_D\}_{pk(S)}$, $\{h(r_S), H, pw\}_{h(r_S, r_D)}$
S → P → D: $\{h(r_D)\}_{h(r_S, r_D)}$

D: trusted device, P: untrusted platform

MP-Auth



H knows: D, P, S, pw

D knows: H, pk(S)

S knows: H, sk(S), pw

H → P: S

P → S: ‘start’

S → P → D: fresh(r_S) . S, r_S

D •→• H: S

H •→• D: pw, H

D → P → S: fresh(r_D) . $\{r_D\}_{pk(S)}$, $\{h(r_S), H, pw\}_{h(r_S,r_D)}$

S → P → D: $\{h(r_D)\}_{h(r_S,r_D)}$

D •→• H: ‘success’

MP-Auth

H knows: D, P, S, pw

D knows: H, pk(S)

S knows: H, sk(S), pw

H → S: ‘start’

S → D: fresh(r_S) . S, r_S

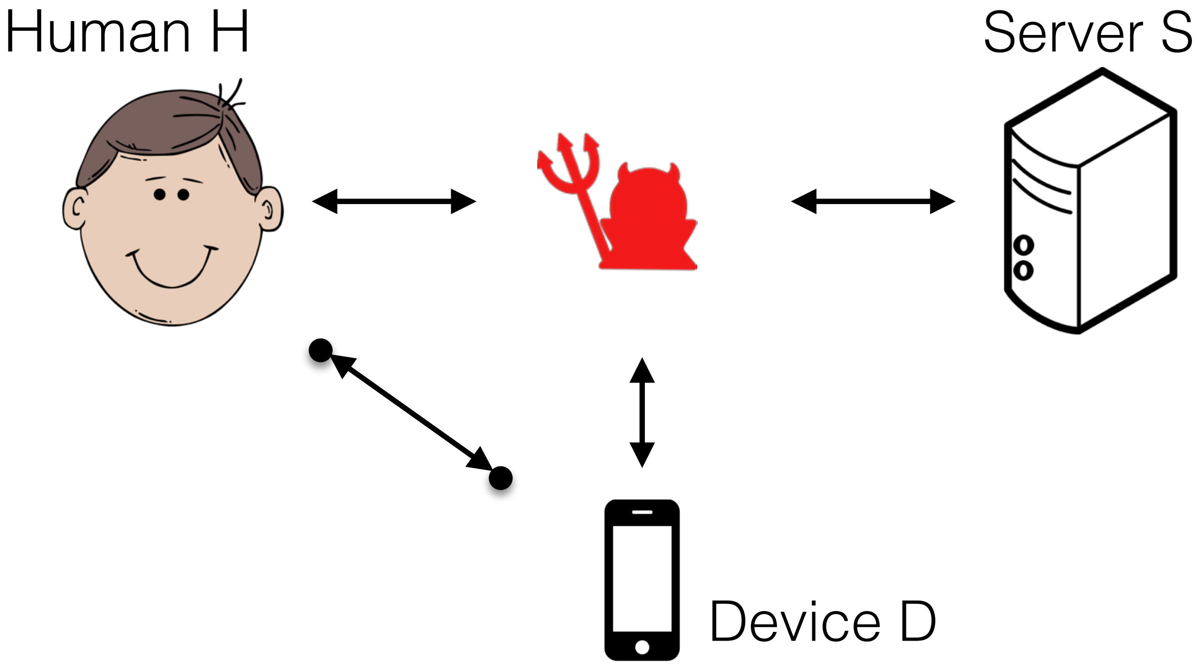
D •→• H: S

H •→• D: pw, H

D → S: fresh(r_D) . $\{r_D\}_{pk(S)}$, $\{h(r_S), H, pw\}_{h(r_S,r_D)}$

S → D: $\{h(r_D)\}_{h(r_S,r_D)}$

D •→• H: ‘success’



Modelling untrusted platform with insecure channels.

Comparison: Phone-based Authentication Protocols

MP-Auth Analysis

	Entity Authentication			Device Authentication		
	Infallible	Untrained	With Guidelines	Infallible	Untrained	With Guidelines
MP-Auth	✓	×	✓	✓	×	✓

Guideline:
NoTellExcept(H,'pw','D')

Guideline:
NoTellExcept(H,'pw','D')

Adversary impersonates H and D to server,
after untrained H enters password on corrupted platform.

Entity Authentication vs Message Authentication

- Both are important.
    - E.g., message (origin) authentication used to authenticate transactions in online banking.
- Some entity authentication protocols can be extended for message authentication
    - Extensions not always possible or straightforward

MP-Auth Message Authentication

H knows: D, P, S, m
D knows: H, pk(S), S, k ← derived from shared key established in login protocol
S knows: sk(S), H, k

H → S: m "please wire 10€ to account #123"
S → D: fresh(r_S) . $\{m, r_S\}_k$ confirm: 10€ to #123
D •→• H: m transfer 10€ to #123 ?
H •→• D: 'ok'
D → S: $\{h(m, r_S)\}_k$ confirmed: 10€ to #123

replay protection

MP-Auth Message Authentication Analysis

- MP-Auth MA with **infallible human** ✓

- MP-Auth MA with **untrained human** ✗

  H presses OK without reading display, confirms message *m* sent by adversary.

- **Guidelines** NoTell, NoTellExcept, NoGet, and ICompare **insufficient** to prevent attack. ✗

Improved MP-Auth Message Authentication

H knows: D, P, S, m
D knows: H, pk(S), S, k
S knows: sk(S), H, k

H → S: m
S → D: fresh(r_S) . $\{m, r_S\}_k$
D •→• H: fresh(vc) . vc, m
H •→• D: vc
D → S: $\{h(m,r_S)\}_k$

H must read display in order to proceed

Satisfies message authentication with human following ICompare guideline.

Google-2-step

with message authentication

H : knows(P,D,S,pw,m,idH)

D : knows(H)

S : knows(H,pw,D,idH)

$H \rightarrow P : S, \boxed{idH, pw,} \boxed{m}$ ← **enters name/password**

$P \circ\!\!\rightarrow\!\bullet\ S : idH, m$ + **message to authenticate**

$S \circ\!\!\rightarrow\!\bullet\ D : fresh(c).\ c,m$

$D \bullet\!\!\rightarrow\!\bullet\ H : \boxed{c,}m$ ← **gets code on device (SMS)**

$H \rightarrow P : S, \boxed{c}$ ← **code entered on platform**

$P \rightarrow S : \boxed{c,}\ pw, m$ ← **and forwarded to server**

Authenticity of m from H to S?

Authenticity in Google-2-step

For an infallible human: verified.

For a fallible human: falsified.

Human does not know he has to compare message on phone with the m that he sent.

For a human with rule *ICompare(H,`m')*: verified.

Comparison: Message Authentication

	Infallible	Untrained	With Guidelines
Cronto MA	✓	×	✓
Google 2-Step*	✓	×	✓
OTP over SMS*	✓	×	✓
MP-Auth VC	✓	×	✓
MP-Auth MA	✓	×	×
Phoolproof*	✓	✓	
Sound-Proof	×		

Guideline:
ICompare(H,'m')

* Our extension based on HISP design guidelines.

Conclusion

- First formal model of human errors in security protocols, providing systematic approach for reasoning about human errors
- Applications to authentication protocols:
    - Finding attacks arising from human errors.
    - Identifying protocol techniques that provide effective protection against various mistakes.
    - Ranking protocols WRT their robustness to human errors

Future Work

- What are good **guidelines**?
- Verify protocols in **combination** of **skilled** and **untrained** human error models.
- **Apply** the model to improve security in the real world:
  - **Improve** system and protocol design.
  - **Identify** critical user actions that must be monitored.
  - Identify critical concepts to **teach** to untrained users.

Literature

- **Modeling Human Errors in Security Protocols**
  D.B., Sasa Radomirovic, Lara Schmid, CSF 2016.

- **A Complete Characterisation of Secure Human-Server Communication**
  D.B., Sasa Radomirovic, Michael Schläpfer, CSF 2015.

Details

Skilled Humans

- Skilled humans **follow protocol specification**, may make a small number of mistakes (slips & lapses).

- **Slips & lapses**: Inattentiveness, routine behaviour in an unusual situation. E.g, clicking “OK” w/o reading an alert.

- Modelled by adding **failure rules** to protocol model.

Specifying Skilled Human Role

Skilled Human H follows protocol specification, keeps state information: AgSt(H,$step$,$knownTerms$)

Pattern for **receiving** messages:

$$[\mathrm{AgSt}(H,s_1,k),\ \mathrm{Rcv}(H,<t,m>)\ ] \longrightarrow$$
$$[\ !\mathrm{HK}(H,t,m),\ \mathrm{AgSt}(H,s_2,<k,m>)\ ]$$

Pattern for **sending** messages:

$$[\ \mathrm{AgSt}(H,s_1,<k,m>),\ !\mathrm{HK}(H,t,m)\ ] \longrightarrow$$
$$[\ \mathrm{Snd}(H,<t,m>),\ \mathrm{AgSt}(H,s_2,<k,m>)\ ]$$

(Trace labels omitted.)

Example of a Failure Rule (Skilled Human Error)

Message confusion: Human H intends to send message m_1, sends instead message $m_{2.}$

$$[\mathrm{Snd}(H,<t_1,m_1>),\ !\mathrm{HK}(H,t_2,m_2),\ \mathbf{Fail}(H,\text{'msc'})] \longrightarrow [\ \mathrm{Snd}(H,<t_2,m_2>)\ ]$$

Fail fact: allows control over type and number of errors.

(Trace labels omitted.)

Related Work

- Beckert and Beuster (2006), Rukšėnas et al. (2008) formally model humans and human error in *human-machine interfaces*.
- Their models correspond to our **skilled human** approach, but capture only *finite scenarios.*
- We model human error in *unbounded protocol executions.*
- A set of failure rules for **skilled human agents** in security protocols are given by Schläpfer (2016).
- Our **untrained human** approach is new.

HISP Channel Assumptions

Authentic Channel:

$$[\mathrm{Snd_A}(A, B, m)] -\!\![\ \mathrm{Snd_A}(A, B, m)\ ]\!\!\rightarrow [!\mathrm{Auth}(A, m), \mathrm{Out}(\langle A, B, m\rangle)]$$
$$[!\mathrm{Auth}(A, m), \mathrm{In}(B)] -\!\![\ \mathrm{Rcv_A}(A, B, m)\ ]\!\!\rightarrow [\mathrm{Rcv_A}(A, B, m)]$$

Confidential Channel:

$$[\mathrm{Snd_C}(A, B, m)] -\!\![\ \mathrm{Snd_C}(A, B, m)\ ]\!\!\rightarrow [!\mathrm{Conf}(B, m)]$$
$$[!\mathrm{Conf}(B, m), \mathrm{In}(A)] -\!\![\ \mathrm{Rcv_C}(A, B, m)\ ]\!\!\rightarrow [\mathrm{Rcv_C}(A, B, m)]$$
$$[\mathrm{In}(\langle A, B, m\rangle)] -\!\![\ \mathrm{Rcv_C}(A, B, m)\ ]\!\!\rightarrow [\mathrm{Rcv_C}(A, B, m)]$$

Secure Channel:

$$[\mathrm{Snd_S}(A, B, m)] -\!\![\ \mathrm{Snd_S}(A, B, m)\ ]\!\!\rightarrow [!\mathrm{Sec}(A, B, m)]$$
$$[!\mathrm{Sec}(A, B, m)] -\!\![\ \mathrm{Rcv_S}(A, B, m)\ ]\!\!\rightarrow [\mathrm{Rcv_S}(A, B, m)]$$