

Authentication in Distributed Systems: Theory and Practice

Butler Lampson, Martin Abadi, Michael Burrows, and Edward Wobber

A Few Comments

This is more about a system than a logic

A Few Comments

This is more about a system than a logic
The logic paper was presented last
Wednesday

A Few Comments

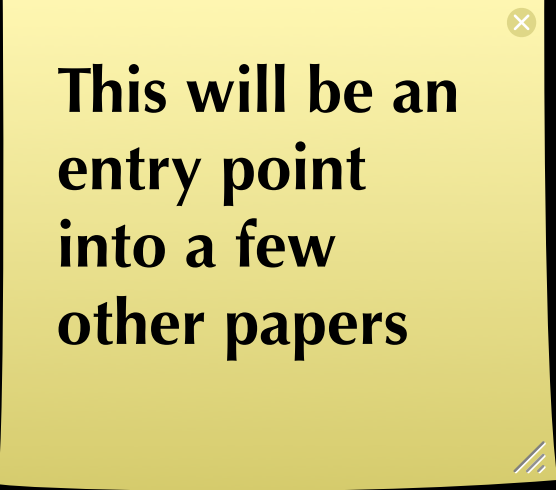
This is more about a system than a logic

There is a systems paper as well

E. Wobber, M. Abadi, M. Burrows and B. Lampson. Authentication in the Taos Operating System. *ACM Trans.Comp. Sys.*, 12(1): 3–32, Feb. 1994.

A Few Comments

This is more about a system than a logic
Connection between
(\Rightarrow , $|$, \wedge) and “what actually happens”



This will be an
entry point
into a few
other papers

A Few Comments

This is more about a system than a logic

**Understanding “lines of reasoning”
behind authentication decisions**

**All reasoning
about program
correctness
happens outside
of the logic**

**Revocation and
security vs.
availability are
talked about,
but not in logic**

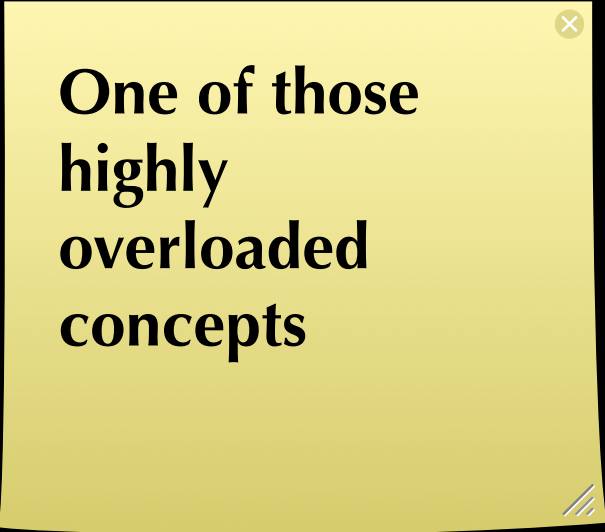
A Few Comments

This is more about a system than a logic

**Understanding “lines of reasoning”
behind authentication decisions**

**Many concepts in the logic turn out to
be heavily overloaded**

Atomic Principals



**One of those
highly
overloaded
concepts**

Atomic Principals can be:

Actors/Identities (A,B,...)

Atomic Principals can be:

Actors/Identities (**A,B,...**)
Channels (**C,...**)

Channels are the only entities that can actually SAY things - they are rarely considered directly, however

Atomic Principals can be:

Actors/Identities (**A**,**B**,...)

Channels (**C**,...)

Keys (**K**,**K_a**,**K_b**,...)

In fact, it's not **K** but **RSA(K)** or **DES(K)** that represents an "encryption channel," but this notation is not used...

Atomic Principals can be:

Actors/Identities (**A**,**B**,...)

Channels (**C**,...)

Keys (**K**,**K_a**,**K_b**,...)

Key Identifier (**K^a**,**K^b**,...)

Atomic Principals can be:

Actors/Identities (**A, B, ...**)

Channels (**C, ...**)

Keys (**K, K_a, K_b, ...**)

Key Identifier (**K^a, K^b, ...**)

If I alone know **K_{my_master}** then

K^a = Encrypt(K, K_{my_master})

Atomic Principals can be:

Actors/Identities (**A**,**B**,...)

Channels (**C**,...)

Keys (**K**,**K_a**,**K_b**,...)

Key Identifier (**K^a**,**K^b**,...)

If I have some huge array of keys
and **K** is the i^{th} one, then

K^a = **i**

Atomic Principals can be:

Actors/Identities (**A**,**B**,...)

Channels (**C**,...)

Keys (**K**,**K_a**,**K_b**,...)

Key Identifier (**K^a**,**K^b**,...)

Other variations as well

The point is that
a key must be
kept secret,
while a key
indicator

Atomic Principals can be:

Actors/Identities (**A**,**B**,...)

Channels (**C**,...)

Keys (**K**,**K_a**,**K_b**,...)

Key Identifier (**K^a**,**K^b**,...)

Roles (**R**,...)

Atomic Principals can be:

Actors/Identities (**A**,**B**,...)

Channels (**C**,...)

Keys (**K**,**K_a**,**K_b**,...)

Key Identifier (**K^a**,**K^b**,...)

Roles (**R**,...)

All I found really problematic was the difference between a superscripted and subscripted K

Atomic Principals can be:

Actors/Identities (**A**,**B**,...)

Channels (**C**,...)

Keys (**K**,**K_a**,**K_b**,...)

Key Identifier (**K^a**,**K^b**,...)

Roles (**R**,...)

Programs (**P**,...)

“Speaks For”: \Rightarrow

What is the social convention?

“Speaks For”: ⇒

What is the social convention?

“Handing over sovereignty:” USA ⇒ Me

However, this means that there is a hidden contract that is not specified whereby we're trusting, say, the Certificate Authority *not* to do certain things....

“Speaks For”: ⇒

What is the social convention?

“Handing over sovereignty:” Vsign ⇒ Me

A consequence is that we are allowing ourselves to be “ruled” by certificate authorities, keys, channels, etc. The consequences of this show up!

“Speaks For”: \Rightarrow

What is the social convention?

“Handing over sovereignty:” **Vsign** \Rightarrow **Me**

Me says **Penn** \Rightarrow **Me**

Penn \Rightarrow **Me**



This is an axiom!

“Speaks For”: \Rightarrow

What is the social convention?

“Handing over sovereignty:” $V_{\text{sign}} \Rightarrow Me$

Me says $Key \Rightarrow Me$

$Key \Rightarrow Me$

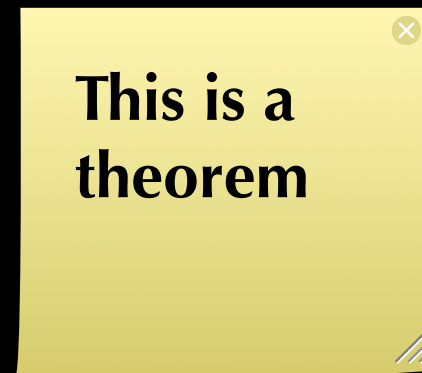
“Speaks For”: \Rightarrow

What is the social convention?

“Handing over sovereignty:” **Vsign** \Rightarrow **Me**

Me says **Key** \Rightarrow **Me**

Key \Rightarrow **Me**



USA \Rightarrow **Me** **USA** says **Penn** \Rightarrow **Me**

Penn \Rightarrow **Me**

“Speaks For”: \Rightarrow

What is the social convention?

“Handing over sovereignty:” **Vsign** \Rightarrow **Me**

Me says **Key** \Rightarrow **Me**

Key \Rightarrow **Me**

One other
rule
considered
later

Vsign \Rightarrow **Me** **Vsign** says **Key** \Rightarrow **Me**

Key \Rightarrow **Me**

Channels and encryption

Mostly like π -calculus channels

Given control over communication
passing through them (“multiplexing”)

Channels and encryption

Mostly like π -calculus channels

Given control over communication
passing through them (“multiplexing”)

USB | **OpenOffice** says (print “ABC”)

DSA(K) | **sftp** says (copy “a.foo” “b.foo”)

Channels and encryption

Channels based on encryption for their security.

Secrecy: If you know $\text{Encrypt}(K^{-1}, x)$
you can't compute x

Integrity: If you have some x but not K^{-1} ,
you can't compute $\text{Encrypt}(K^{-1}, x)$
(i.e. y such that $\text{Decrypt}(K, y) = x$)

Channels and encryption

Channels based on encryption for their security.

We're worried about integrity, which can be achieved with encryption by using a checksum...

Public and private key schemes

Channels and encryption

Encryption channels need names

K says $s \equiv \text{Encrypt}(K^{-1}, s)$

K^r says $s \equiv (\text{Encrypt}(K^{-1}, s), K^r)$

K^{me} is just some data that allows Me to remember K^{-1}

Example 1: Broadcast Encryption

Public keys are important

Could we do without them? Using a **Relay**.

Relay

OTA knows
Relay's internal
state in order to
know K^{relay}

Alice

K_a , K_a^{relay}

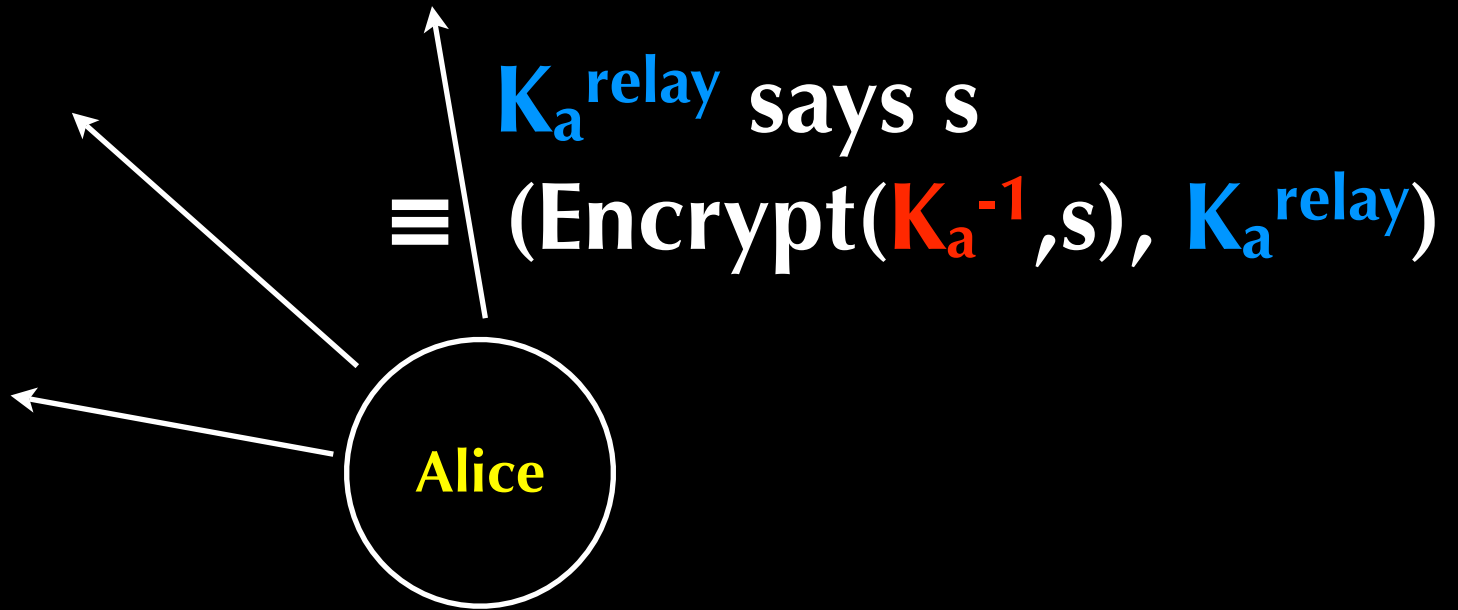
(secure channel)

Offline
Trusted
Agent

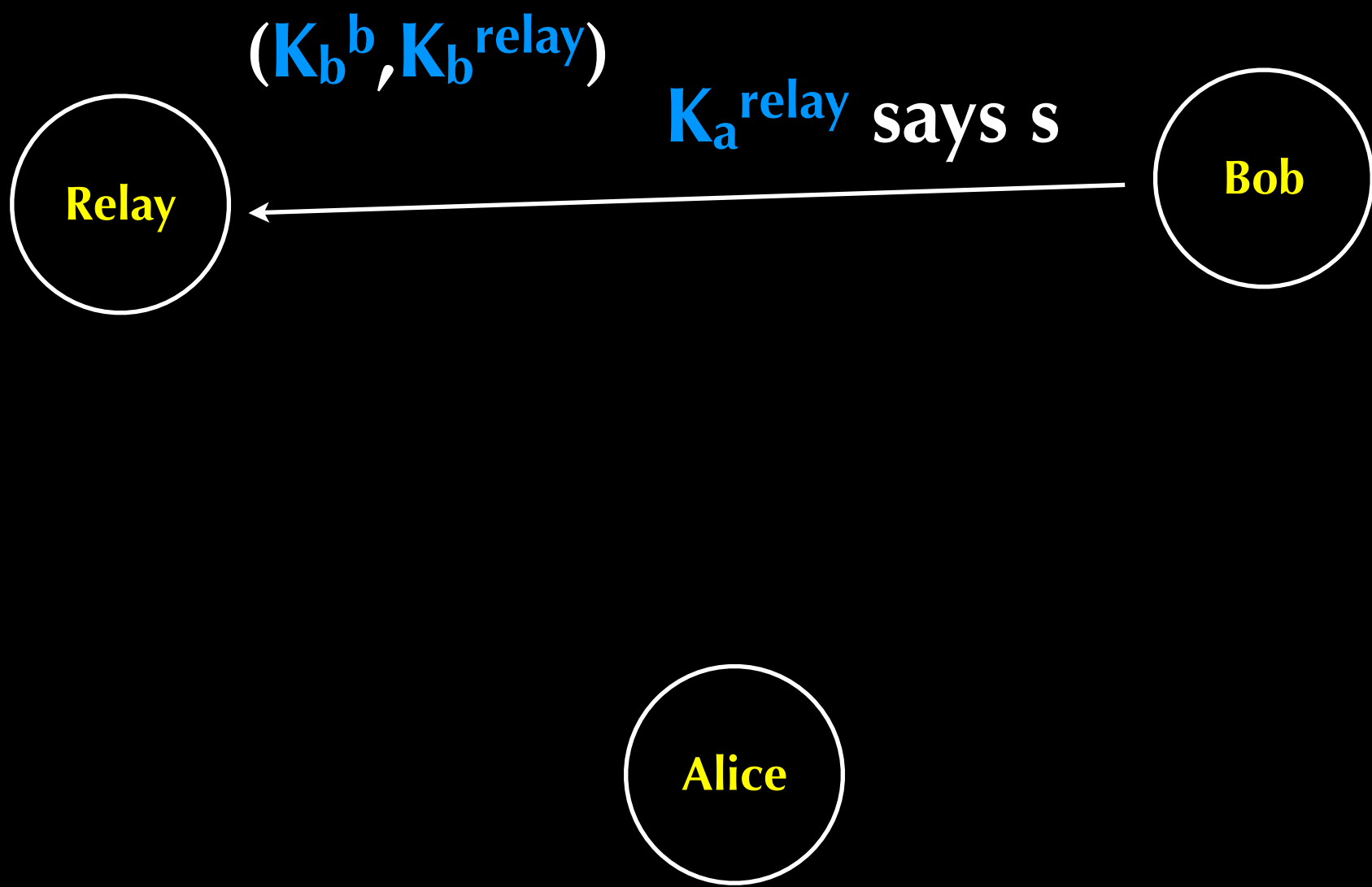
Cert.
Auth.

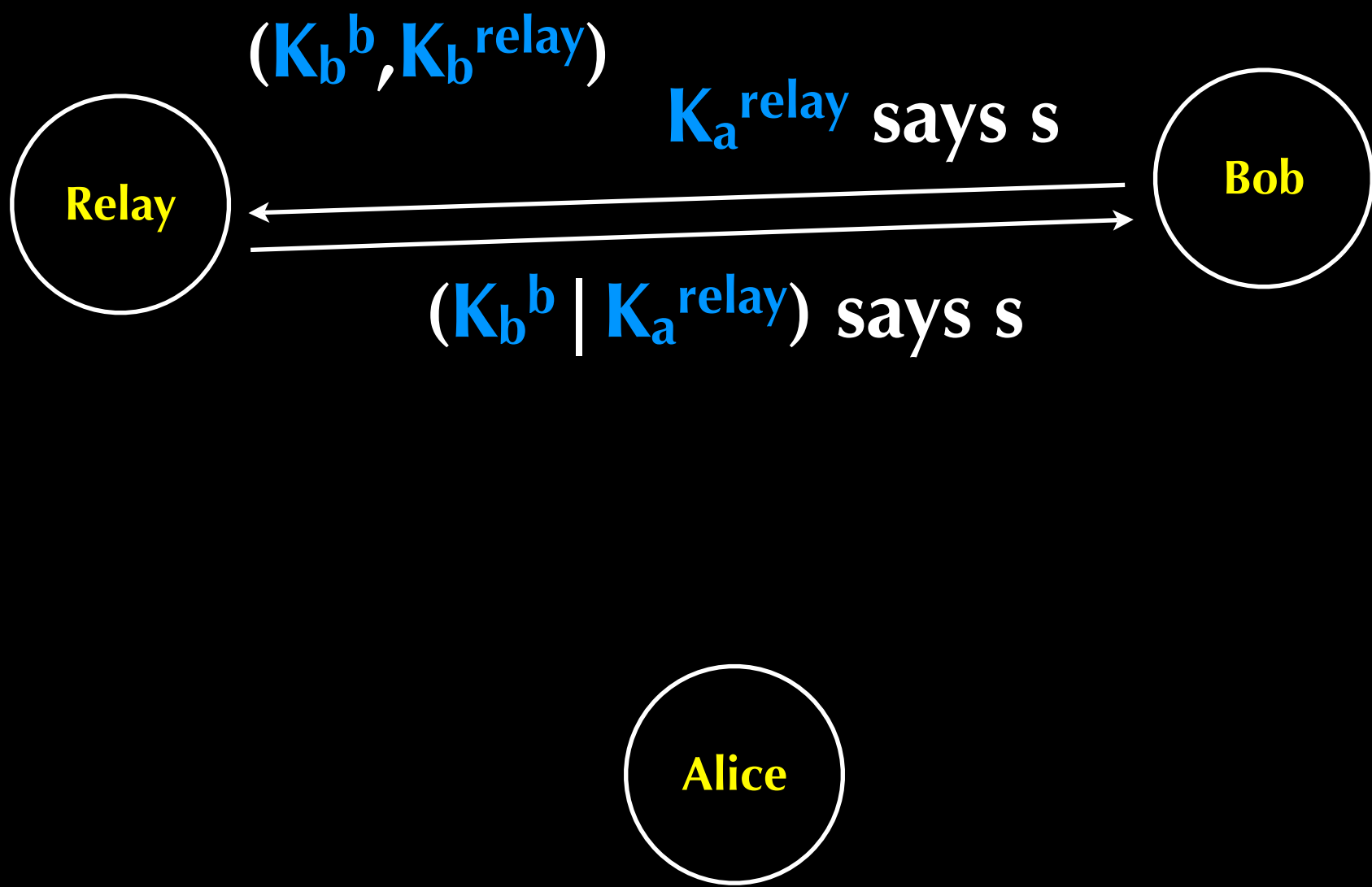
Relay

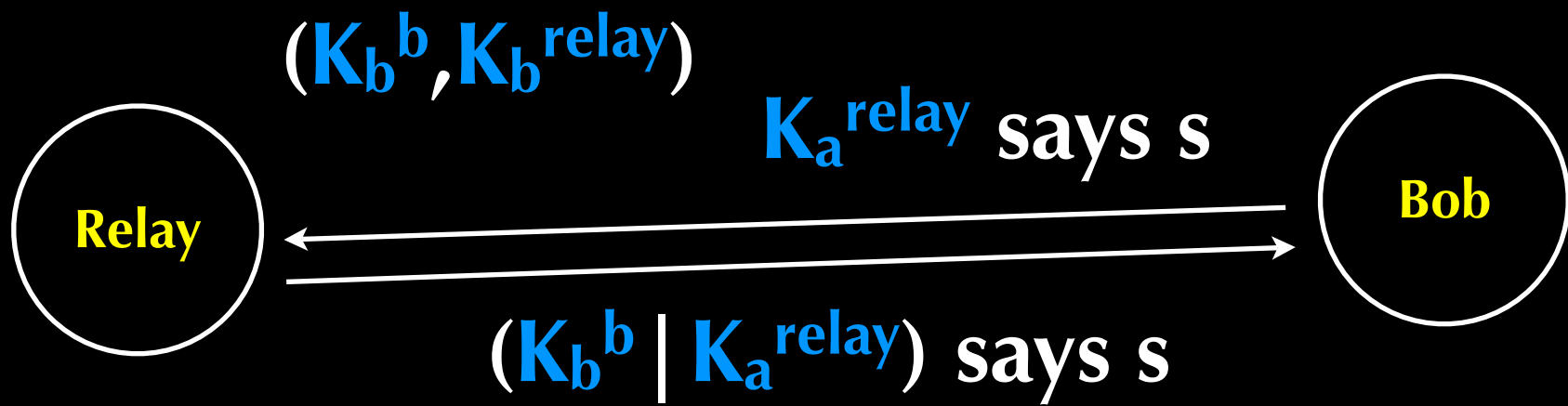
Bob



Cert.
Auth.

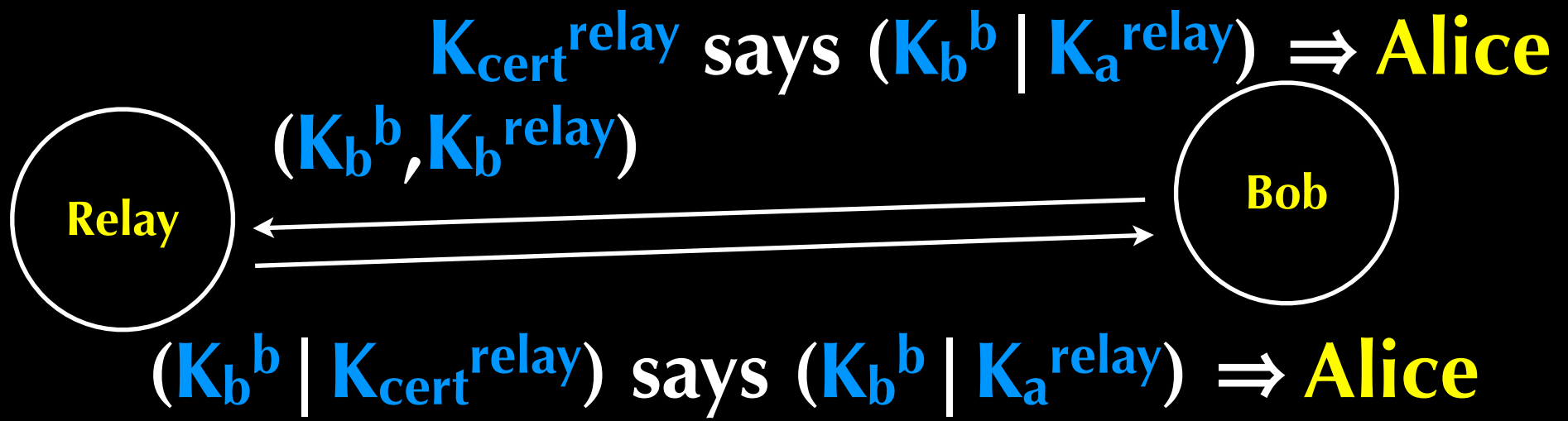


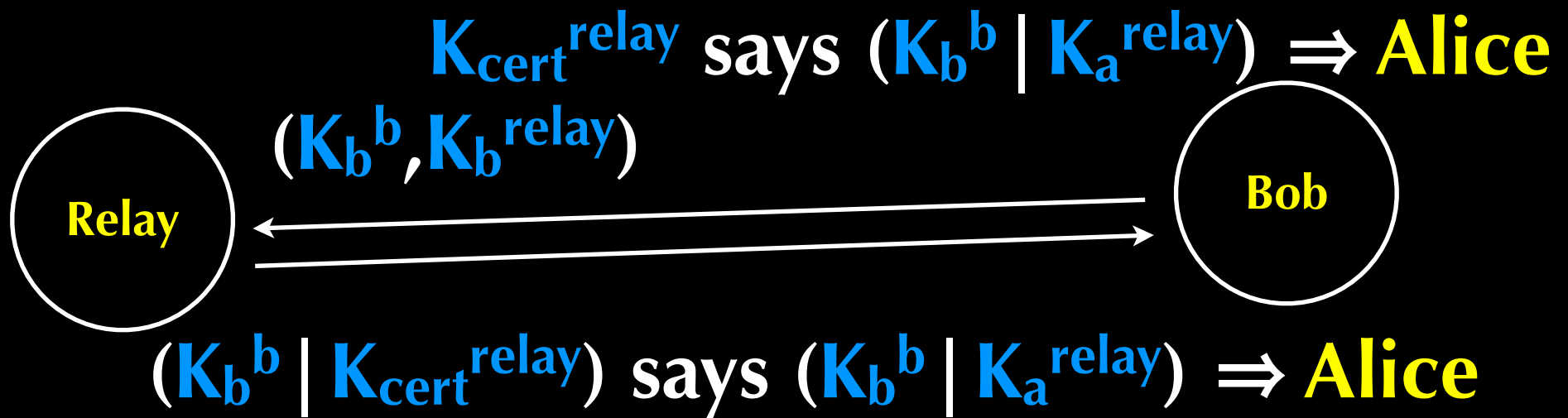




This means **(Bob | K_a^{relay})** says s
 because $K_b^b \Rightarrow K_b \Rightarrow \mathbf{Bob}$





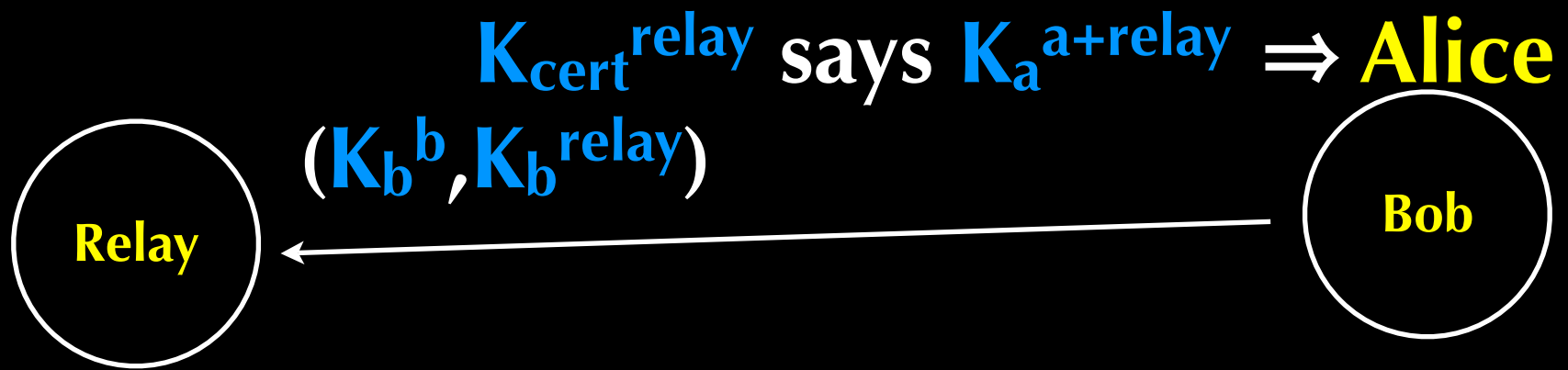


This means **Bob** says $K_a^{relay} \Rightarrow \text{Alice}$

because $K_b^b \Rightarrow K_b \Rightarrow \text{Bob}$

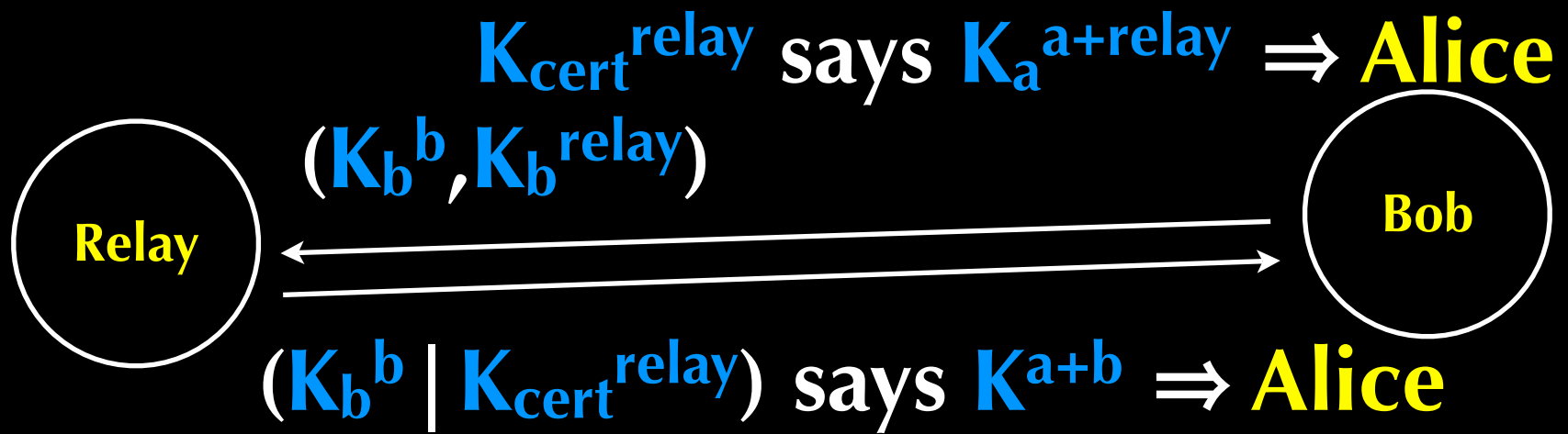
and because $K_{cert}^b \Rightarrow \text{Bob}$

Not too good -
means cert.
authority must
be always be
active



What is the status of that +?

It is just a pair, but $K_a^{a+relay} \Rightarrow \text{Alice}$ seems to imply $K_a^a \Rightarrow \text{Alice}$ and $K_a^{relay} \Rightarrow \text{Alice}$



Perhaps all users of the relay cede authority to it...
crappy EULA if that is the case...

Relay invents **K**

Relay justifies **K** $\Rightarrow K_a^a$ and **K** $\Rightarrow K_b^b$?

K^{a+b} is a pair (K^a, K^b) which allow Alice and Bob, respectively, to decrypt **K**.

Example 2: Node-to-node channel

Uses public key encryption

Their take on Needham-Schroeder

Flawed in some similar ways...

Speaks of “higher-level protocols that reuse sequence numbers and connection identifiers” - this seems problematic...

Alice

$K_a, K_a^{-1},$

K_{am}

Bob

$K_b, K_b^{-1},$

K_{bm}

Alice

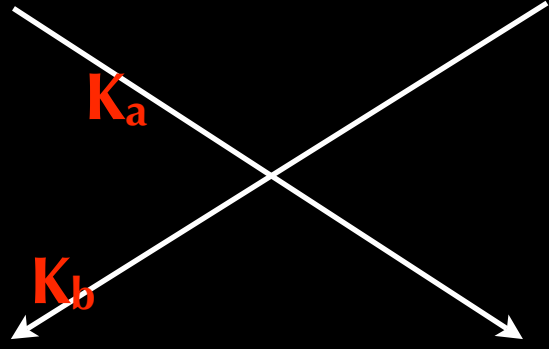
Bob

K_a, K_a^{-1}
 K_{am}

K_b, K_b^{-1}
 K_{bm}

K_b, N_a

K_a, N_b



Alice

Bob

$K_a, K_a^{-1},$
 K_{am}

$K_b, K_b^{-1},$
 K_{bm}

K_b, N_a

K_a, N_b

N_b

N_a

K_a

K_b

$Enc(K_b, N_a)$

$Enc(K_a, N_b)$

Alice

Bob

$K_a, K_a^{-1},$
 K_{am}

$K_b, K_b^{-1},$
 K_{bm}

K_b, N_a

K_a, N_b

N_b

N_a

$K =$

$K =$

$\text{Hash}(N_a, N_b)$

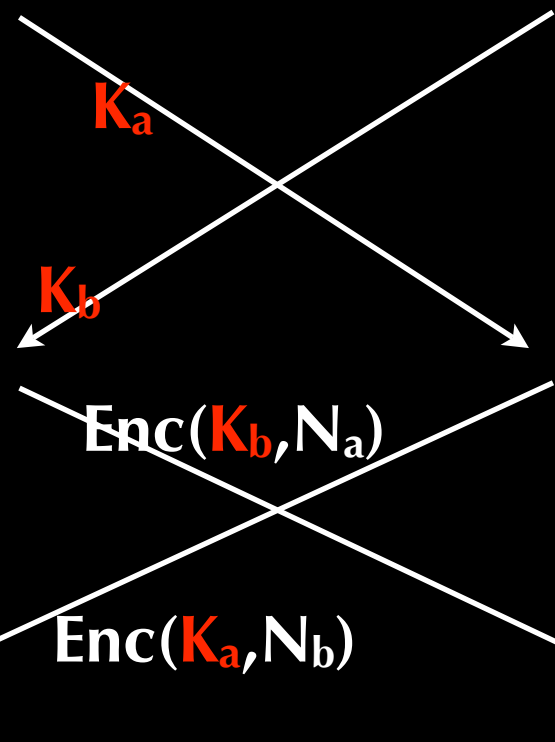
$\text{Hash}(N_a, N_b)$

$K^a =$

$K^b =$

$\text{Enc}(K_{am}, K)$

$\text{Enc}(K_{bm}, K)$



Alice

Bob

$K_a, K_a^{-1},$
 K_{am}

$K_b, K_b^{-1},$
 K_{bm}

K_b, N_a

K_a, N_b

N_b

N_a

$K =$

$K =$

$\text{Hash}(N_a, N_b)$

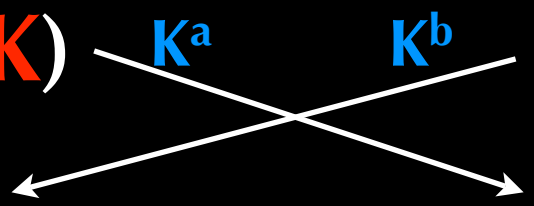
$\text{Hash}(N_a, N_b)$

$K^a =$

$K^b =$

$\text{Enc}(K_{am}, K)$

$\text{Enc}(K_{bm}, K)$



Alice

Zelda

$K_a, K_a^{-1},$
 K_{am}

$K_b,$
 $Enc(K_a, N_b)$

K_b, N_a

K_a

N_b

$Enc(K_b, N_a)$

$K =$

$Hash(N_a, N_b)$

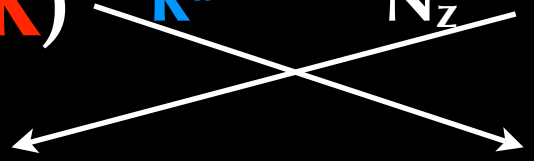
N_z

$K^a =$

$Enc(K_{am}, K)$

K^a

N_z



Alice

$K_a, K_a^{-1},$

K_{am}

K_b, N_a

N_b

$K =$

$\text{Hash}(N_a, N_b)$

$K^a =$

$\text{Enc}(K_{am}, K)$

“**A** believes that only someone who can decrypt $\text{Encrypt}(K_b, J_a)$ could share its knowledge of **K**”

Alice

$K_a, K_a^{-1},$

K_{am}

K_b, N_a

N_b

$K =$

$\text{Hash}(N_a, N_b)$

$K^a =$

$\text{Enc}(K_{am}, K)$

“**A** believes that only someone who can decrypt $\text{Encrypt}(K_b, J_a)$ could share its knowledge of **K**”

This might, in fact, be the case if J_a (i.e. N_a) is generated correctly

Alice

$K_a, K_a^{-1},$

K_{am}

K_b, N_a

“**A** believes that only someone who can decrypt $\text{Encrypt}(K_b, J_a)$ could share its knowledge of **K**. In other words, A believes that $K \Rightarrow K_b$.”

N_b

$K =$

$\text{Hash}(N_a, N_b)$

$K^a =$

$\text{Enc}(K_{am}, K)$

Alice

$K_a, K_a^{-1},$

K_{am}

K_b, N_a

N_b

$K =$

Hash(

K

Enc(

“**A** believes that only someone who can decrypt $\text{Encrypt}(K_b, J_a)$ could share its knowledge of K . In other words, A believes that $K \Rightarrow K_b$.”

This seems like a problem. Either we say K_b intrinsically is able to cede power to K , or else we can't justify $K \Rightarrow \text{Bob}$.

...Or else my metaphor is a bad one!

Alice

$K_a, K_a^{-1},$

K_{am}

K_b, N_a

N_b

$K =$

$\text{Hash}(N_a, N_b)$

$K^a =$

$\text{Enc}(K_{am}, K)$

“A believes that only someone who can decrypt $\text{Encrypt}(K_b, J_a)$ could share its knowledge of K . In other words, A believes that $K \Rightarrow K_b$.”

“...an assumption of the theory; we can't prove it...”

The important logical reasoning (and the “bug”) is, in some sense, happening “outside”

Example 3: Revocation

Uses the notion of “short lived hypothesis” that was explicitly not included in the logic (an “until” construct is mentioned)

The construct $O \mid K_a$ reads like O is a butler-like figure...

The certificate authority's keys essentially have sovereignty over everyone...

But the CA can choose to say K_{ca} says $K_a \Rightarrow A$

or they can say K_{ca} says $((O | K_a) \wedge K_a) \Rightarrow A$

In a strange use of the handoff axiom, O "backs you up" by saying

$(O | K_a)$ says $K_a \Rightarrow (O | K_a)$

An alternative would be a notion of blacklist

New Concept: Path Names

Supposed to be “decentralized,” sort of

Tree structure requires trust up to “least common ancestor”

Root/CMU/Me is a path name

Incomplete?

I'm mostly ignoring this - the “tree structure” of authority seems to have a very specific use model - it's not a general solution to anything

Roles and Programs

A **role** is something honest parties can quote?

Treating roles as principals seems unclear,
especially as roles are “kind-of-program

Singularity
agrees!

An operating system goes to program by
using the hash (digest) of a program’s code:

“**OS** as **Digest1571** says s”

Roles and Programs

A **role** is something honest parties can quote?

Treat... principals seems unclear,
espe... "kind-

An o... es to
using the hash (digest) of a p...:
"OS as **Digest1571** says s"

You probably need to allow multiple digests (corresponding to different versions of a program) to speak for the same abstract program...

Extends to trusting the OS in a straightforward fashion

Digest1571 ⇒ **MSWord**, **Digest64** ⇒ **MSWord**

Delegation - "for"

Some combination of handoff (\Rightarrow) and roles

Only really makes sense w.r.t timeouts

GS6177 as **Linux** as **Web** as **Print** for **Me**

Delegation - “for”

Some combination of handoff (\Rightarrow) and roles

Only really makes sense w.r.t timeouts

GS6177 as **Linux** as **Web** as **Print** for **Me**

Singularity models this more directly:

`login@rjsimmon+firefox+lpr`

Authorizing applications in singularity. Wobber, T. and Yumerefendi, A. and Abadi, M. and Birrell, A. and Simon, D. R. 355--368 (2007)

Access Control

“Handoff” happens in terms of “for”

Procedures happen in terms of “as”

Channels only appear as the last “as”

(A as R₁ as R₂) for (B as R₃ as R₄ as C₂)

Access Control

principal ::= forList | principal \wedge forList

forList ::= asList | forList for asList

asList ::= properPrincipal | asList as Role

role ::= pathName

properPrincipal ::= pathName | channel

The algorithm for access control happens on this fragment of the language...

Access Control

The algorithm uses a “PULL” model - the resource monitor is the only one that can construct a proof, and must gather the premises when needed. (Idea: caching)

Access Control

The algorithm uses a “PULL” model - the resource monitor is the only one that can construct a proof, and must gather the premises when needed. (Idea: caching)

The same thing can be done from a fully “PUSH” model - the client constructs a proof.

Proof-Carrying Authentication.

Appel, A. W. and Felten, E. W. 52--62 (1999)

Conclusions

Glossed over Section 8 on interprocess communication

Network examples: \Rightarrow is really strong!

OS examples: Roles are quite complicated

Ends up looking like two different kinds of reasoning!

Proof-Carrying Authentication

What might happen naturally if you were to write this system down in Twelf as closely as possible.

(Forced to use higher-order logic, for one thing)

Proof-Carrying Authentication

More primitive “real world” things get encoded than in Lampson et. al.

`digital_signature(s,k,F)`, i.e. “s represents signing the formula F with the key k,” is the primary primitive

Everything else is coded up as definitions

Proof-Carrying Authentication

A “worldview” is a N of type $\text{form} \rightarrow \text{form}$.

A principal is just a worldview that accepts all true things and all consequences of other things it accepts:

$$\text{prin}(P) \equiv (\forall F.(F \rightarrow P(F))) \\ \wedge (\forall F,G.(P(F) \rightarrow P(F \rightarrow G) \rightarrow P(G)))$$

Proof-Carrying Authen

If controls is just a defn., then controls_e can be a lemma, etc...

```
controls_e: pf (controls @ A @ F) -> pf (A @ F) -> pf F.
keybind_e: pf (keybind @ K @ W) -> pf (digital_signature S K F)
           -> pf (W @ F).
trusted_ca_e: pf (trusted_ca @ Ca)
              -> pf (controls @ Ca @ (keybind @ K @ A)).

abc: pf (txmsted_ca @ Charlie) ->
     pf (keybind @ Kc @ Chaxlie) ->
     pf (controls @ Alice @ ReadFoo) ->
     pf (digital_signature BI0010111 Ka ReadFoo) ->
     pf (digital_signature BOi001110 Kc (keybind @ Ka @ Alice)) ->
     pf ReadFoo =

[A1] [A23 [A3] [S1] [$2]
controls_e A3
      (keybind_e (controls_e (trusted_ca_e Ai) (keybind_e A2 S2)) S1).
```

Proof-Carrying Authentication

No Kripke models, not “modeling something.”

All parties agree on higher order logic, the meaning of the signature rule rule, and whatever definitions you use

Authorizing Applications in Singularity

Don't use "for" and "as" as primitives, use them as the way they were *actually used*.

Programs have immutable names (no multiple identities!)

`login@rjsimmon+firefox+java+tinyim`

Access control in terms of regular expressions

Authorizing Applications in Singularity

Seems to boil down to the decidable part of the logic in “Authentication in Distributed Systems”

Needs to deal less with “speaks for”

No clear story for remote login and distributed programming