Towards a Security Model for Internet of Things (IoT)

Ankur Taly, Google Inc.

November 2015

Internet of Things (IoT)

Physical devices made accessible over the network.

Exciting new possibilities!!

img source: http://www.ti.com/lsds/media/images/wireless_connectivity/50BillionThings.png

Internet of Things Security!

Goldmine for the bad guys!!

Scary new possibilities!!

This is really scary!!

Live feed from an airplane hangar in Norway!!

Found using shodan.io --- a search engine for finding devices (IoT), e.g., routers, servers, cameras, SCADA systems, HVAC systems etc.

source: http://img.wonderhowto.com/img/original/32/45/63534020036048/0/635340200360483245.jpg

In popular press

WND EXCLUSIVE

'HACKERS REMOTELY KILL A JEEP ON THE HIGHWAY'

Several baby monitors vulnerable to hacking, cybersecurity firm warns

The Associated Press Posted: Sep 02, 2015 1:53 PM ET | Last Updated: Sep 02, 2015 2:13 PM ET

Refrigerator Busted Sending Spam Emails In Massive Cyberattack

The Huffington Post | By Ryan Grenoble

Hackers Can Remotely Hack Self-Aiming Rifles to Change Its Target

Thursday, July 30, 2015 Mohit Kumar

Top IoT Vulnerabilities

HP Study Reveals 70 Percent of Internet of Things Devices Vulnerable to Attack

danielmiessler 07-29-2014 05:09 AM - edited 07-07-2015 12:33 PM

- Insufficient Authentication and Authorization (80%)
- Lack of Transport Encryption (70%)
- Insecure Web Interfaces (60%)
- Insecure Software Updates (60%)
- Insecure Defaults (70%)

Top IoT Vulnerabilities

HP Study Reveals 70 Percent of Internet of Things Devices Vulnerable to Attack

danielmiessler 07-29-2014 05:09 AM - edited 07-07-2015 12:33 PM

- Insufficient Authentication and Authorization (80%)
- Lack of Transport Encryption (70%)
- Insecure Web Interfaces (60%)
- Insecure Software Updates (60%)
- Insecure Defaults (70%)

Security First: Bake in security mechanisms from the ground up.

IoT Security Requirements

Identity and Authorization
Mutual authorization
Channel integrity and confidentiality
Forward secrecy
Fine-grained sharing and delegation
Revocation and auditing

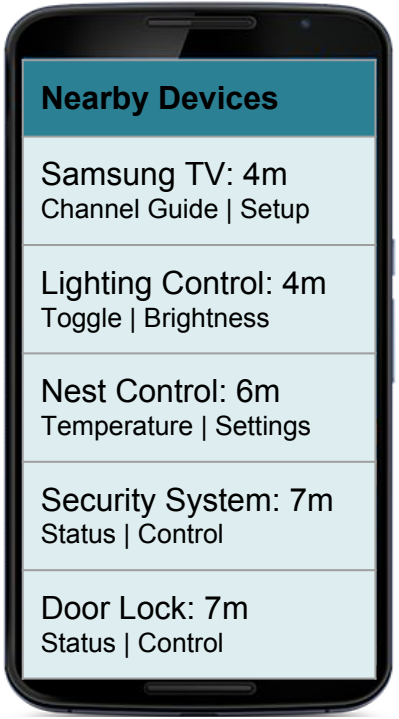
IoT Security Requirements

Identity and Authorization
Mutual authorization
Channel integrity and confidentiality
Forward secrecy
Fine-grained sharing and delegation
Revocation and auditing

Device Protection
No remote code execution
Automatic and secure updates
Verified boot

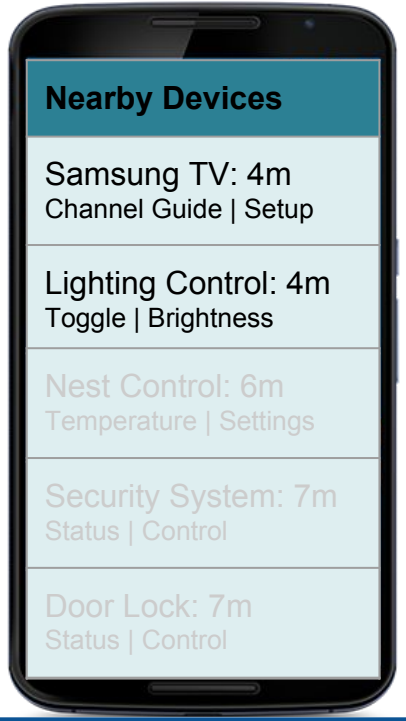
Private Discovery

courtesy: physical-web.org



Private Discovery

courtesy: physical-web.org



Private Discovery: Devices must be discoverable only by authorized parties.

IoT Security Requirements

Identity and Authorization
Mutual authorization
Channel integrity and confidentiality
Forward secrecy
Fine-grained sharing and delegation
Revocation and auditing

Device Protection
No remote code execution
Automatic and secure updates
Verified boot

Privacy
Private discovery
Anonymous communication
Transparency

This Talk

Identity and Authorization
Mutual authorization
Channel integrity and confidentiality
Forward secrecy
Fine-grained sharing and delegation
Revocation and auditing

Device Protection
No remote code execution
Automatic and secure updates
Verified boot

Privacy
Private discovery
Anonymous communication
Transparency

Plan

- Vanadium Security Model
  - Identity (Principals and Blessings)
  - Authentication and Authorization
- Case Study: Smart Lock
- Practicalities
- Conclusions

Vanadium

Open source, cross-platform framework for building secure, multi-device applications

- *Decentralized*
  "Talk when possible" ⇒ Peer-to-peer communication

- *Secure*
  Mutual authentication and authorization
  Fine-grained delegation & auditing

Mechanics

- Go, Java and JavaScript libraries
- Open-source
- All we talk about in these slides is “code-complete”

Design Docs, Tutorials, Code

- Homepage: v.io
- Concepts: v.io/concepts
- Tutorials: v.io/tutorials
- Source: vanadium.googlesource.com

Vanadium Security Model

Principals & Blessings

Principal

- An entity that can communicate externally
  - **Fine-grained**: Each App/Process/Service is a different principal

- Has a unique public/private key pair (P, S)
  - Private key is **never** shared over the network
  - Ideally held in a TPM on the device

Blessings

- Each principal has a set of hierarchical human-readable strings bound to it, called *blessings*

  Example: Alice's television (P_{tv}, S_{tv}) may have blessings:
  - `google/alice/hometv`
  - `samsung/products/tv/123`

- Principals are authenticated and authorized based on their blessings. Example: Alice's devices authorize all principals with blessing names matching `google/alice/*`

Blessings

- Each principal has a set of hierarchical human-readable strings bound to it, called *blessings*

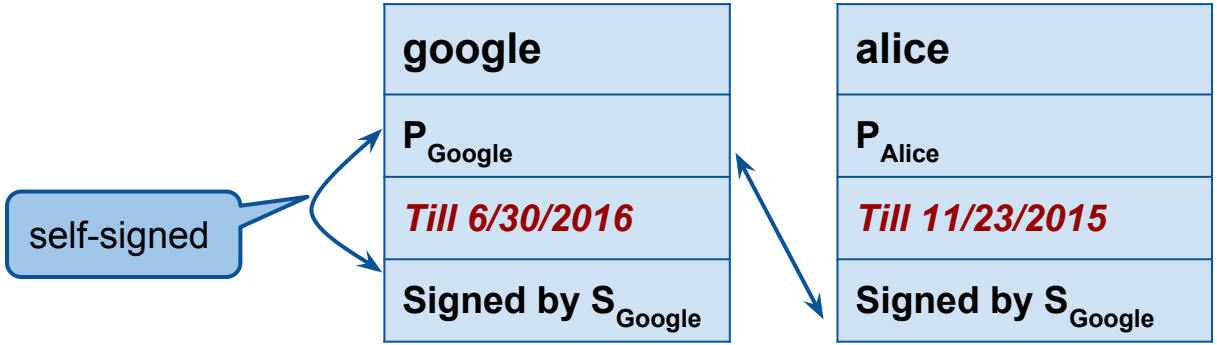
  Example: Alice's television (P_{tv}, S_{tv}) may have blessings:
  - `google/alice/hometv`
  - `samsung/products/tv/123`

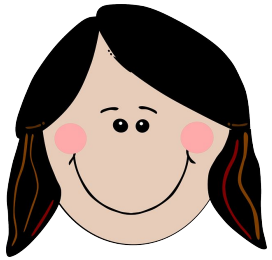
- Principals are authenticated and authorized based on their blessings. Example: Alice's devices authorize all principals with blessing names matching `google/alice/*`

Simple Distributed Security Infrastructure (**SDSI**), Lampson and Rivest, 1996

Blessings

- Blessings are certificate chains bound to the principal's public key
- Each certificate has a Name, PublicKey, Caveats and Signature





(P_{Alice}, S_{Alice})

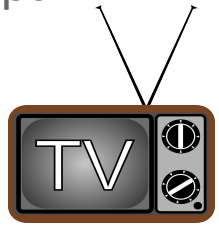
Very simple certificate format!

The Bless Operation

Extend one of your Blessings and bind it to another principal.

(P_{Alice}, S_{Alice})

Bless



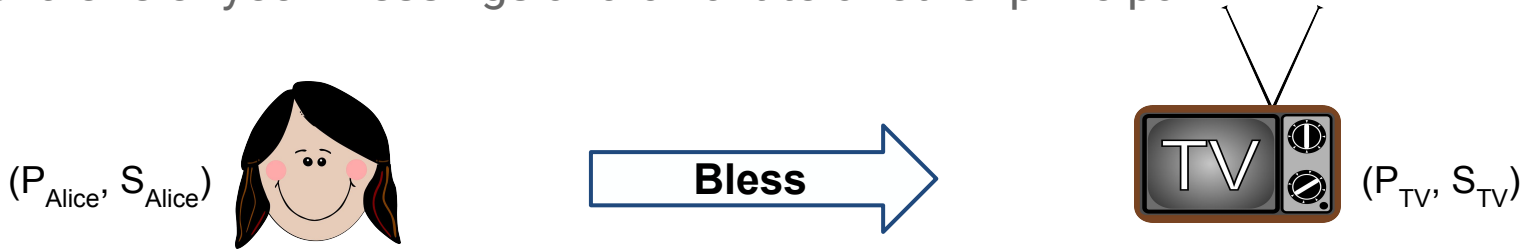
(P_{TV}, S_{TV})

google
P_{Google}
Till 6/30/2016
Signed by S_{Google}

alice
P_{Alice}
Till 11/23/2015
Signed by S_{Google}

The Bless Operation

Extend one of your Blessings and bind it to another principal.



google
P_{Google}
Till 6/30/2016
Signed by S_{Google}

alice
P_{Alice}
Till 11/23/2015
Signed by S_{Google}

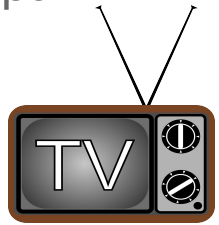
hometv
P_{TV}
Till 11/23/2015: 6PM
Signed by S_{Alice}

The Bless Operation

Extend one of your Blessings and bind it to another principal.

(P_{Alice}, S_{Alice})

Bless



(P_{TV}, S_{TV})

google
P_{Google}
Till 6/30/2016
Signed by S_{Google}

alice
P_{Alice}
Till 8/25/2015
Signed by S_{Google}

hometv
P_{TV}
Till 8/25/2015: 6PM
Signed by S_{Alice}

Dynamic Identity Creation OR Bound Capability Grant!

Blessings: Auditability and Binding

Blessings:

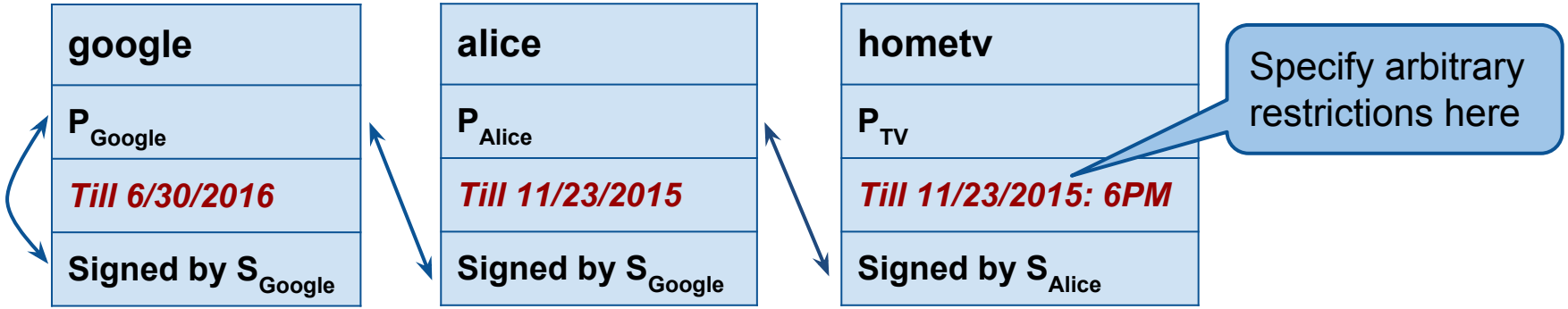
- Are bound to a private key that never leaves the device
- Can only be delegated by extending to other private keys
- Encapsulate an auditable delegation trail
- Examples:
  - `google/alice` (AccountManager app on Alice's phone)
  - `google/alice/hometv` (DeviceManager app on Alice's TV)
  - `google/alice/hometv/youtube` (Youtube app on Alice's TV)

Caveats

But Alice wants her TV to *only* access Youtube, NOT her Bank!

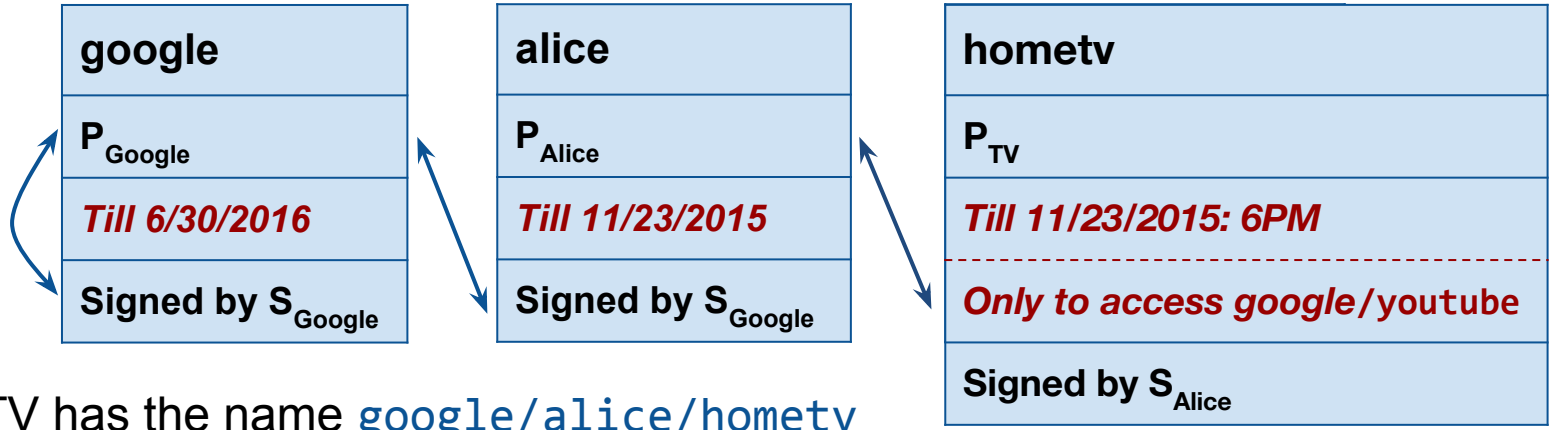
Caveats

But Alice wants her TV to *only* access Youtube, NOT her Bank!



Caveats

But Alice wants her TV to *only* access Youtube, NOT her Bank!

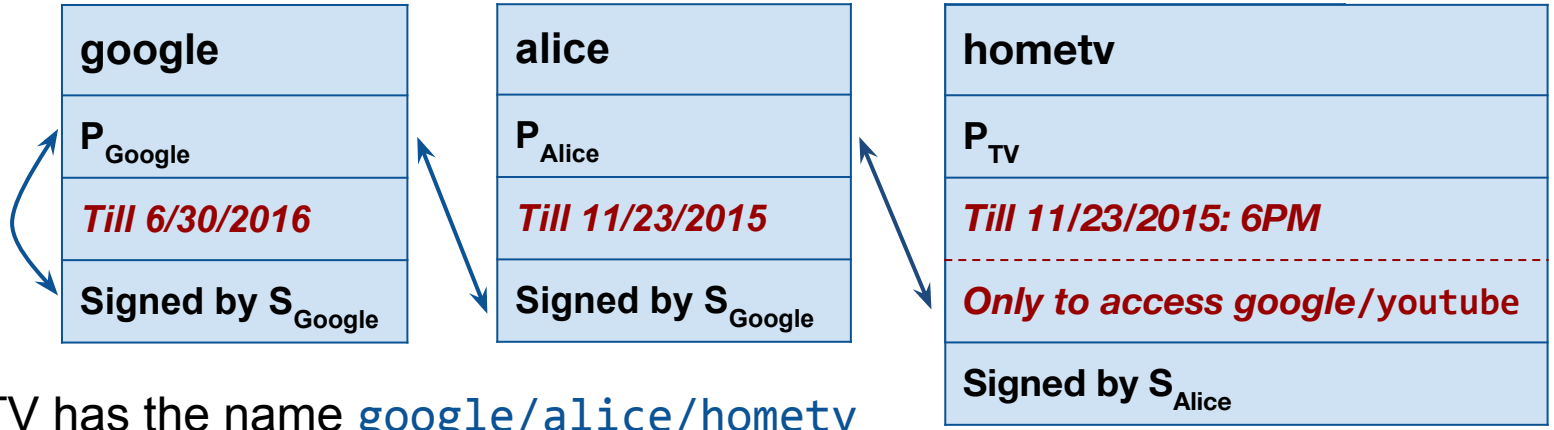


TV has the name `google/alice/hometv`

- *as long as the time is before 11/23/2015: 6PM*
- *as long as the service being accessed is* `google/youtube`

Caveats

But Alice wants her TV to *only* access Youtube, NOT her Bank!



TV has the name `google/alice/hometv`

- *as long as the time is before 11/23/2015: 6PM*
- *as long as the service being accessed is* `google/youtube`

Macaroons: Cookies with Caveats for Decentralized Authorization, Politz et al., NDSS'14

Caveats are powerful

- Services can define their own caveats

  Bless the Valet such that:

  - valet is only authorized to drive for < 5 miles
  - only for the next 3 hours
  - cannot access trunk or infotainment system
  - but can access GPS

- Validated by the target service (first-party) when the blessing is used to make a request (**first-party caveats**)

Third-party Caveats

- Caveats that must be validated by a specific third-party
- Target service (first-party) only expects a “discharge” (proof) that the caveat has been validated by the specific third-party

Third-party Caveats

- Caveats that must be validated by a specific third-party
- Target service (first-party) only expects a “discharge” (proof) that the caveat has been validated by the specific third-party

Third-party Caveat

ID: <content hash>

Restriction: within 10m proximity

Loc: google/`alice/proximity`

Verification Key: $P_{Proximity}$

Third-party Caveats

- Caveats that must be validated by a specific third-party
- Target service (first-party) only expects a “discharge” (proof) that the caveat has been validated by the specific third-party

Third-party Caveat

ID: <content hash>

Restriction: within 10m proximity

Loc: google/`alice/proximity`

Verification Key: $P_{Proximity}$

Third-party Discharge

ID: <same as caveat.ID>

Caveat: for next 1 minute

Signed by $S_{Proximity}$

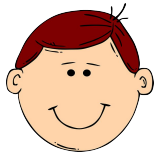
Mechanics

$(P_{Proximity}, S_{Proximity})$

Alice's proximity discharger

(P_{TV}, S_{TV})

(P_{Bob}, S_{Bob})



google
…
…
…

alice
…
…
…

houseguest
…
…
…
proximity caveat

Mechanics

$(P_{Proximity}, S_{Proximity})$

Alice's proximity discharger

proximity caveat

1

(P_{Bob}, S_{Bob})

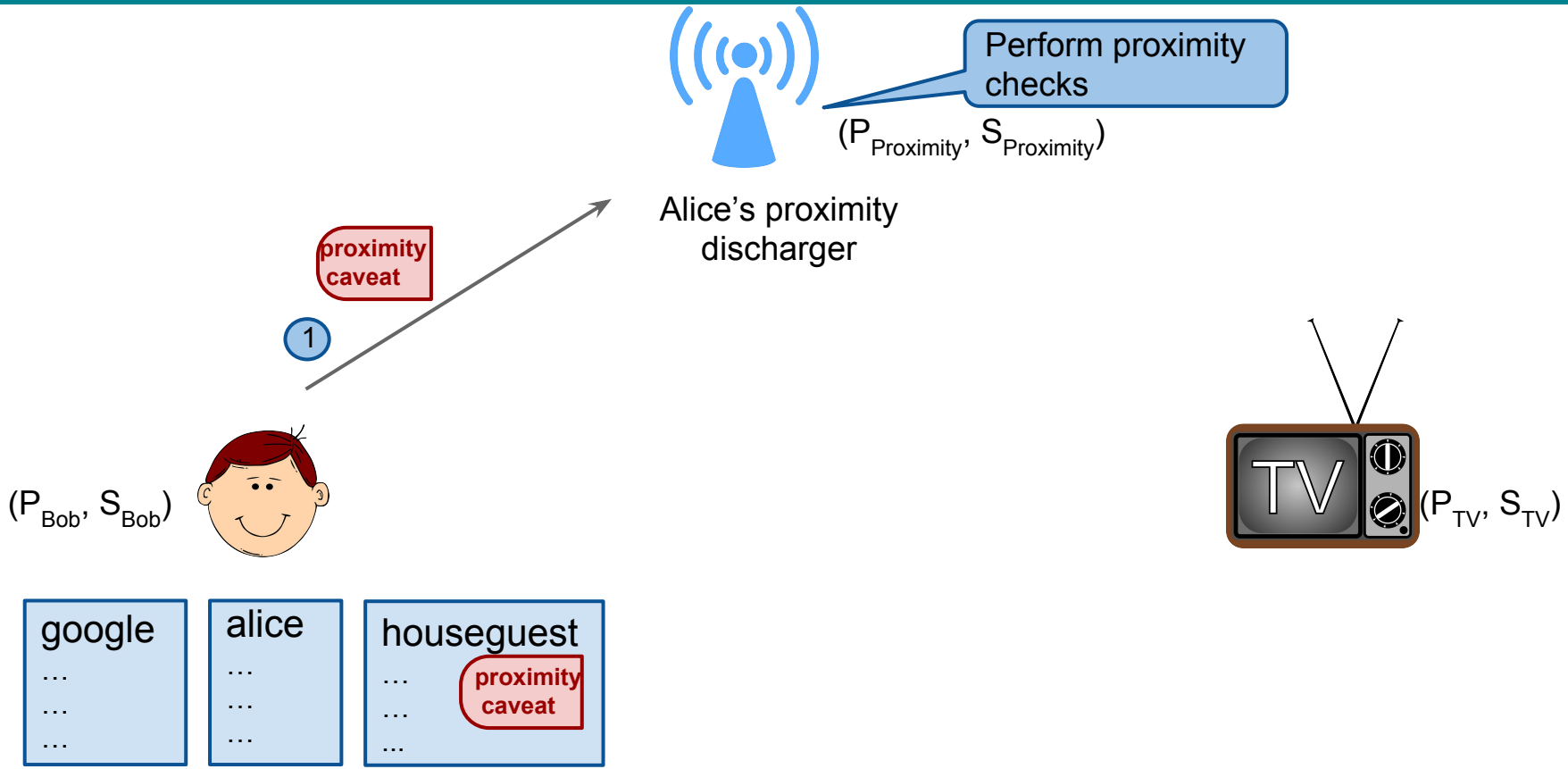
(P_{TV}, S_{TV})

google
…
…
…

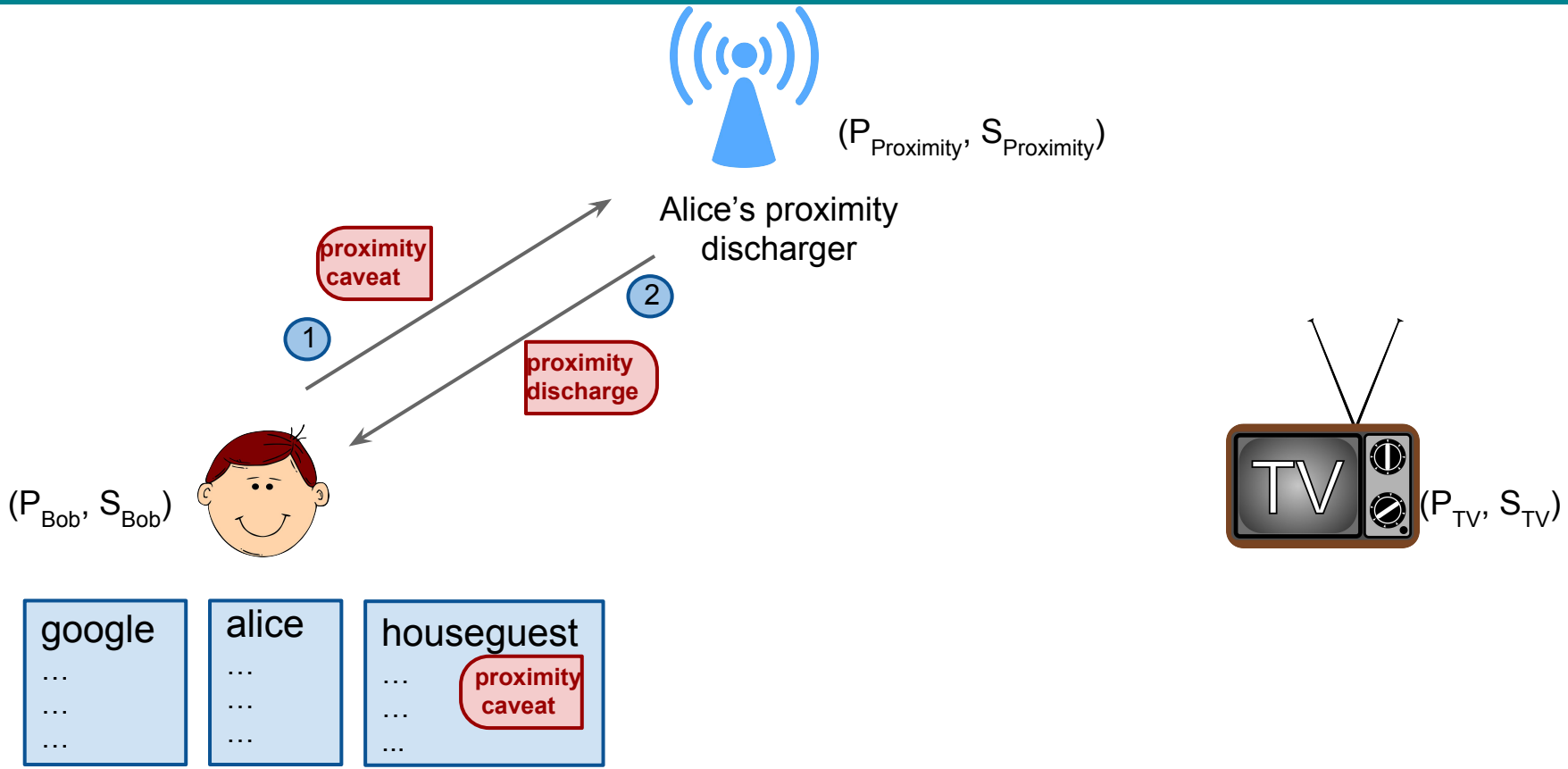
alice
…
…
…

houseguest
…
…
…
proximity caveat

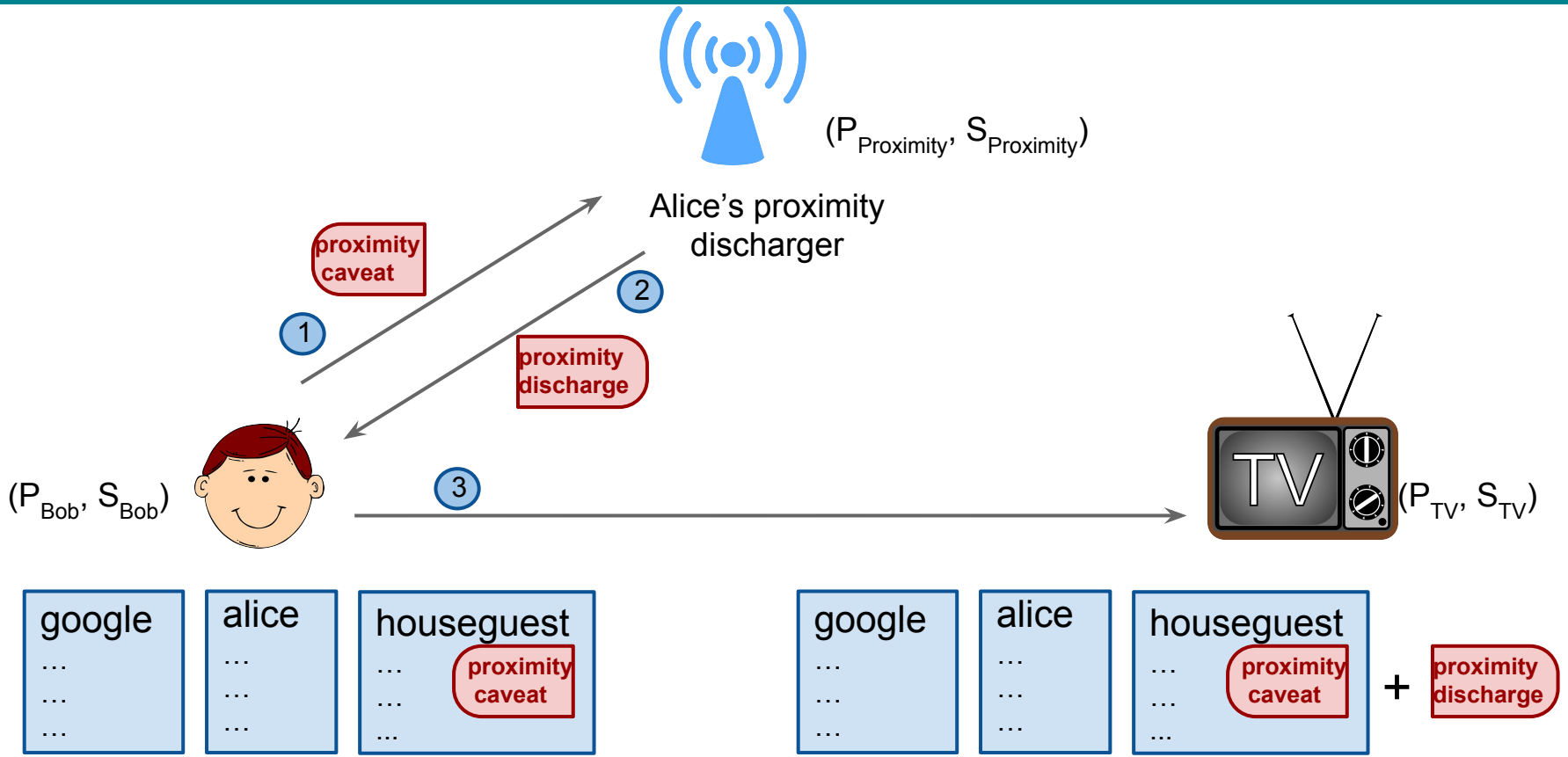
Mechanics


Perform proximity checks
$(P_{Proximity}, S_{Proximity})$
Alice's proximity discharger
proximity caveat
1
(P_{Bob}, S_{Bob})
TV
(P_{TV}, S_{TV})
google
...
...
...
alice
...
...
...
houseguest
...
...
...
proximity caveat

Mechanics


$(P_{Proximity}, S_{Proximity})$
Alice's proximity discharger
proximity caveat
1
2
proximity discharge
(P_{Bob}, S_{Bob})
TV
(P_{TV}, S_{TV})
google
…
…
…
alice
…
…
…
houseguest
…
…
…
proximity caveat

Mechanics

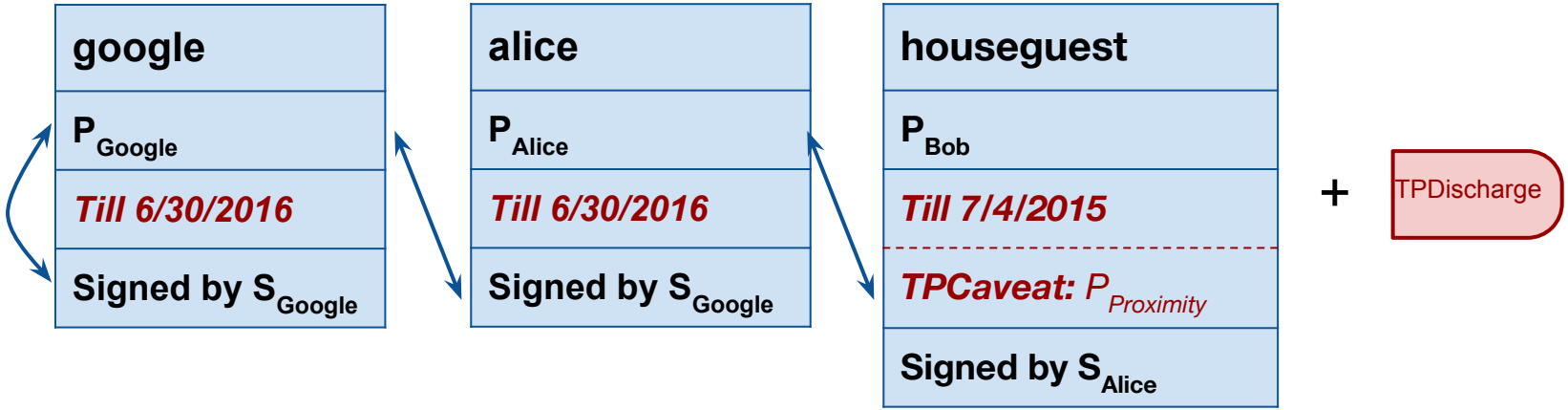

$(P_{Proximity}, S_{Proximity})$
Alice's proximity discharger
proximity caveat
1
2
proximity discharge
(P_{Bob}, S_{Bob})
3
TV
(P_{TV}, S_{TV})
google
...
...
...
alice
...
...
...
houseguest
...
...
...
proximity caveat
google
...
...
...
alice
...
...
...
houseguest
...
...
...
proximity caveat
+
proximity discharge

Third-party Caveat Examples

- Revocation
  - Third-party caveats provide a natural refresh mechanism.
  - Subsumes *Online Certificate Status Protocol (OCSP)*
- Social network
  - G+ must assert membership in “work” circle
- Parental controls
  - Kids can watch TV only if Mom approves

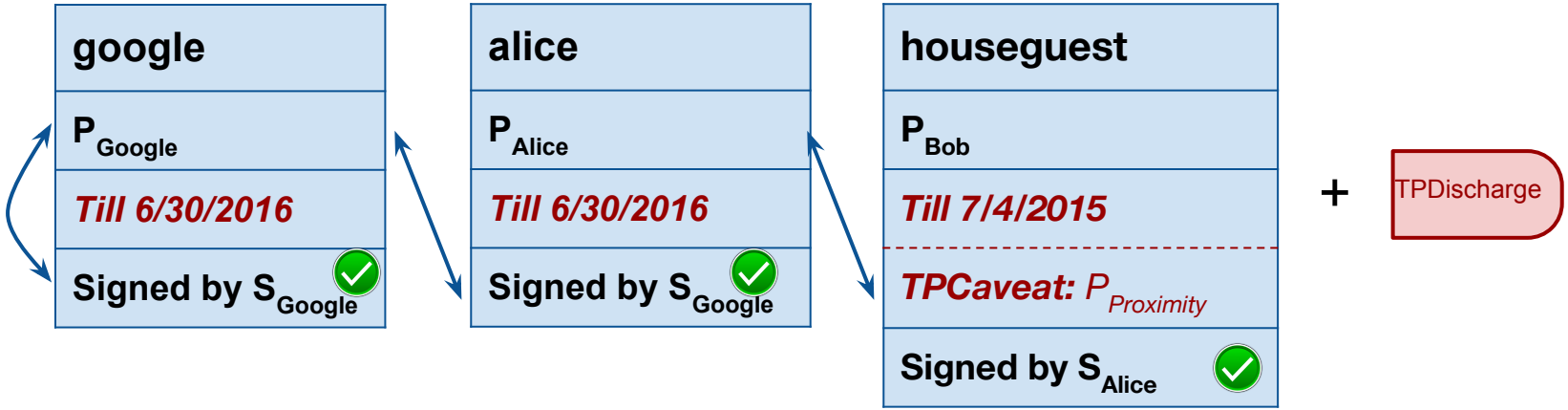
Validating Blessings

How does the TV validate Bob's blessings?



Validating Blessings

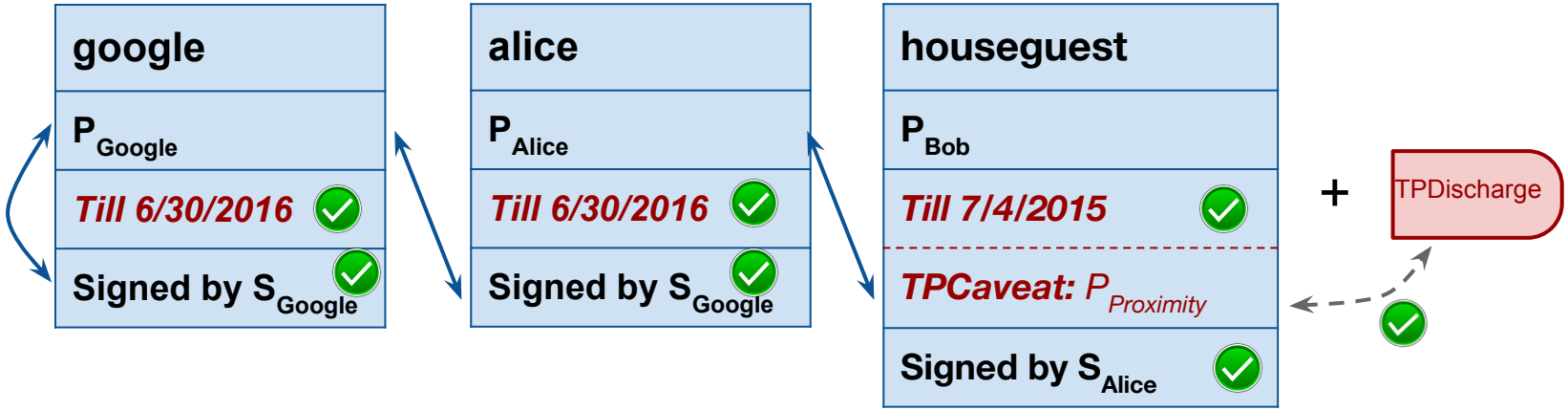
How does the TV validate Bob's blessings?



1. Verify Certificate Signatures

Validating Blessings

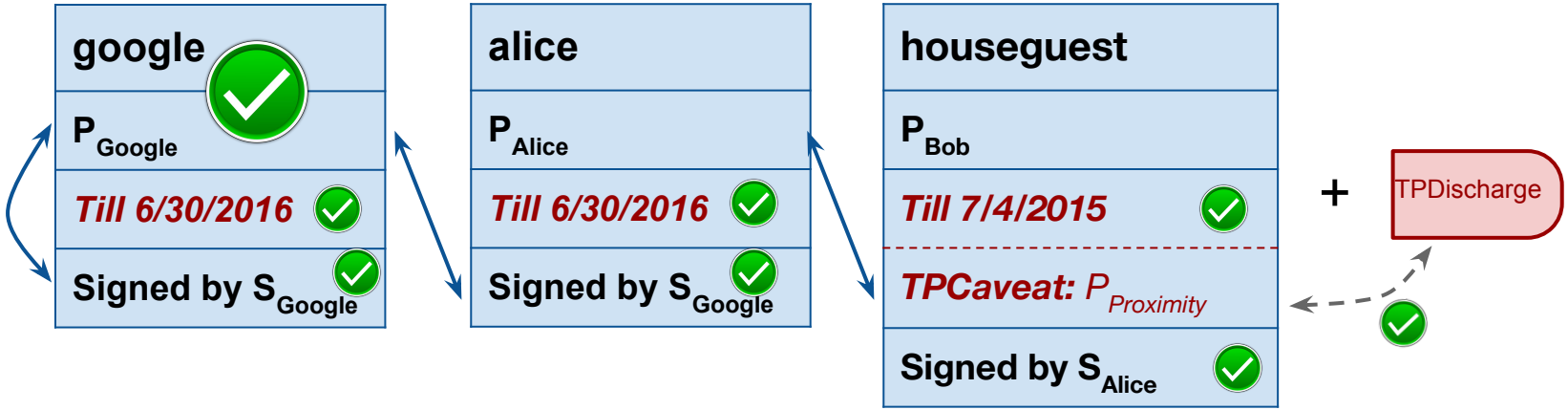
How does the TV validate Bob's blessings?



1. Verify Certificate Signatures
2. Validate all first-party and third-party caveats

Validating Blessings

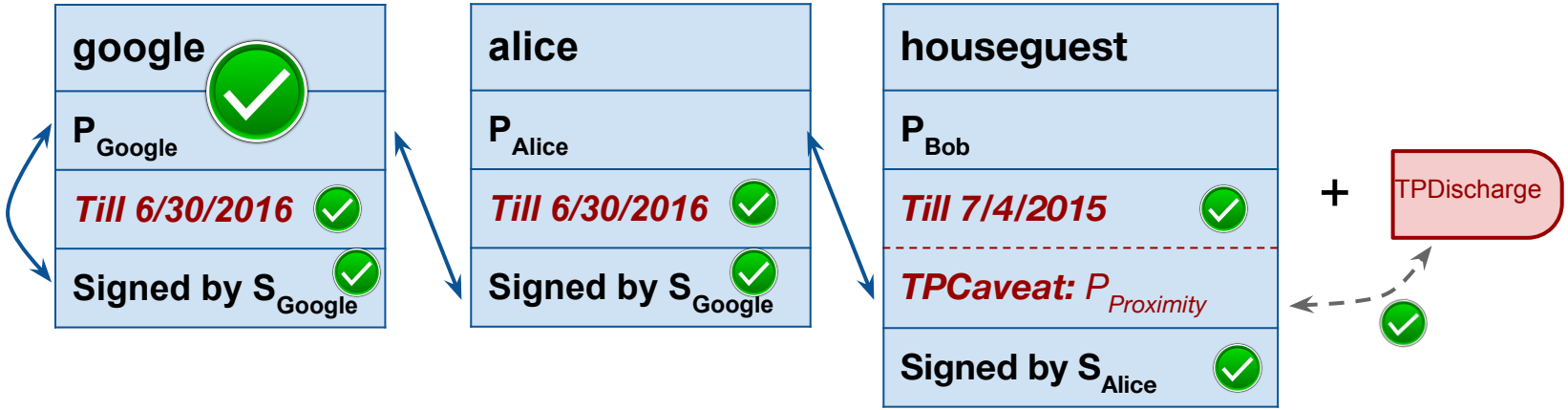
How does the TV validate Bob's blessings?



1. Verify Certificate Signatures
2. Validate all first-party and third-party caveats
3. Verify that the blessing root is recognized

Validating Blessings

How does the TV validate Bob's blessings?



1. Verify Certificate Signatures
2. Validate all first-party and third-party caveats
3. Verify that the blessing root is recognized

Bob can be recognized as `google/alice/houseguest`

Authentication and Authorization

All communication must be encrypted, mutually authenticated and authorized

Authentication and Authorization

Client: Initiator of request

Server: Responder of request

Mutual Authentication

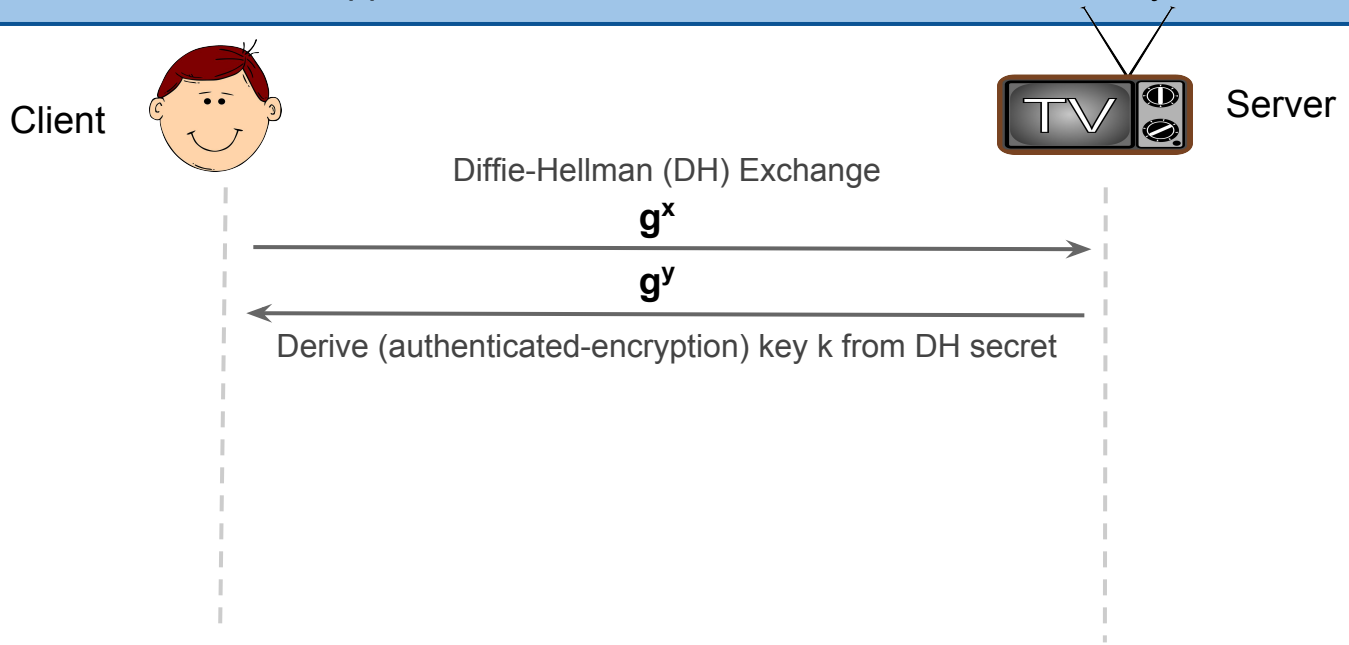
- Each end *learns* the other end's blessings and is convinced that the other end possesses the corresponding private key

Mutual Authorization

- Each end validates the other end's blessings and evaluates the blessing names against an authorization policy

Mutual Authentication Protocol

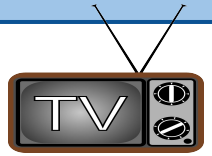
SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman, Krawczyk et al., CRYPTO'03



Mutual Authentication Protocol

SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman, Krawczyk et al., CRYPTO'03

Client



Server

Diffie-Hellman (DH) Exchange

g^x →

← g^y

Derive (authenticated-encryption) key k from DH secret

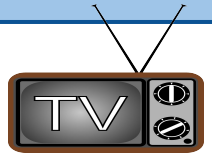
← $\{ \text{Blessings}_{TV},\ \text{Sign}_{TV}(<\text{"s"}, g^x, g^y>) \}_k$

Bob learns Blessings_{TV} and authorizes them

Mutual Authentication Protocol

SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman, Krawczyk et al., CRYPTO'03

Client



Server

Diffie-Hellman (DH) Exchange

g^x →

← g^y

Derive (authenticated-encryption) key k from DH secret

Bob learns Blessings_{TV} and authorizes them

← $\{ \text{Blessings}_{TV}, \text{Sign}_{TV}(<\text{"s"}, g^x, g^y>) \}_k$

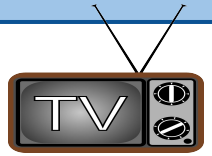
→ $\{ \text{Blessings}_{Bob}, \text{Sign}_{Bob}(<\text{"c"}, g^x, g^y>) \}_k$

TV learns Blessings_{Bob} and authorizes them

Mutual Authentication Protocol

SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman, Krawczyk et al., CRYPTO'03

Client



Server

Diffie-Hellman (DH) Exchange

g^x →

← g^y

Derive (authenticated-encryption) key k from DH secret

Bob learns Blessings_{TV} and authorizes them

← $\{ \text{Blessings}_{TV}, \text{Sign}_{TV}(<\text{"s"}, g^x, g^y>) \}_k$

→ $\{ \text{Blessings}_{Bob}, \text{Sign}_{Bob}(<\text{"c"}, g^x, g^y>) \}_k$

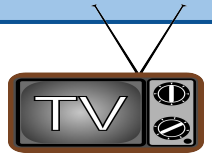
TV learns Blessings_{Bob} and authorizes them

Server presents its blessings *before* the client

Mutual Authentication Protocol

SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman, Krawczyk et al., CRYPTO'03

Client



Server

Diffie-Hellman (DH) Exchange

g^x

Derive (authenti… k from DH secret

{ Bless… , g^y>) }$_k$

{ Blessings$_{Bob}$, S… $_{Bob}$(<"c… , g^x, g^y>) }$_k$

Formally verified in ProVerif

Bob learns Blessings$_{TV}$ and authorizes them

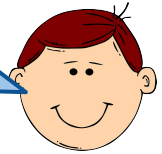
TV learns Blessings$_{Bob}$ and authorizes them

Server presents its blessings *before* the client

Private Mutual Authentication*

- Neither the server nor the client wants to present its blessings first.

I will only reveal my name to `google/alice/hometv`



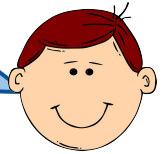
I only reveal my name to delegates of `google/alice`

- How do we resolve this deadlock?

Private Mutual Authentication*

- Neither the server nor the client wants to present its blessings first.

I will only reveal my name to `google/alice/hometv`



I only reveal my name to delegates of `google/alice`

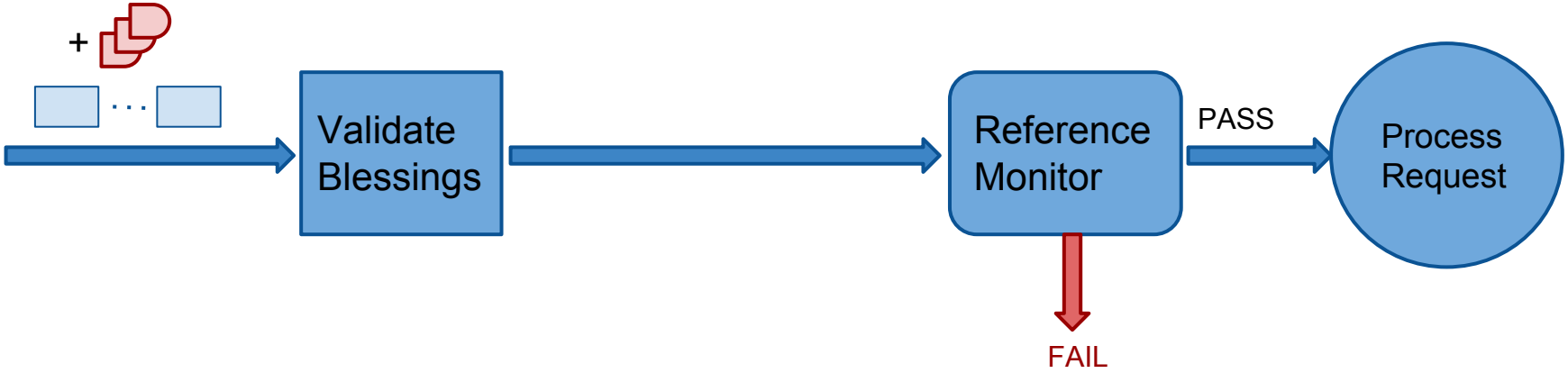
- How do we resolve this deadlock?

Relevant cryptography research (carried out in the context of secret agents)

- Oblivious Signature-Based Envelope - Li et al., PODC‘03
- Secret Handshakes from Pairing-Based Key Agreements - Balfanz et al., S&P‘03

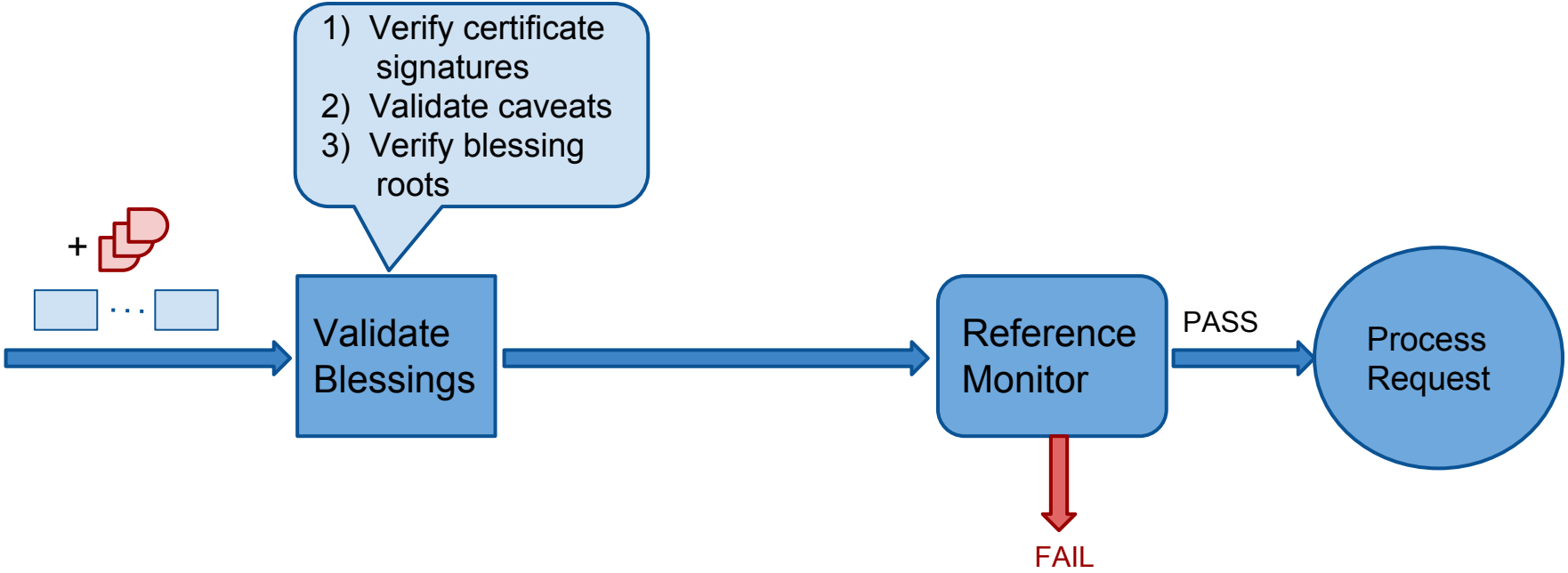
Authorization

Authorization policies are based on blessing names.



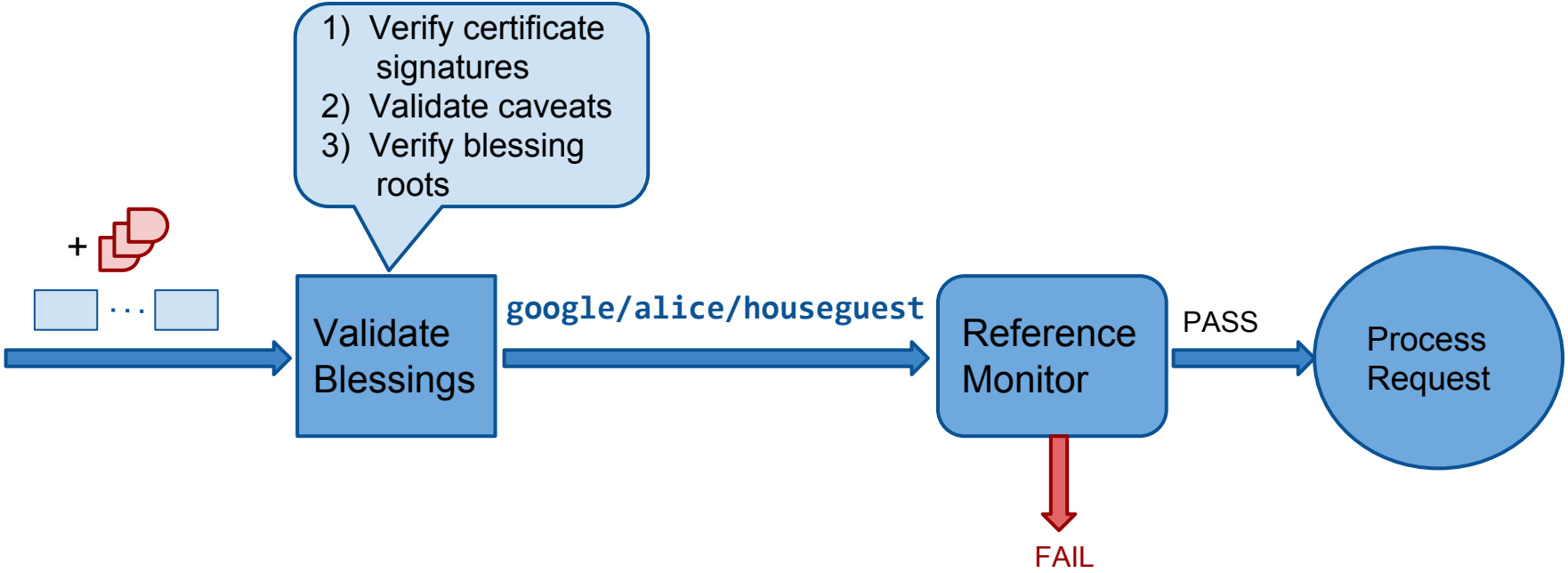
Authorization

Authorization policies are based on blessing names.



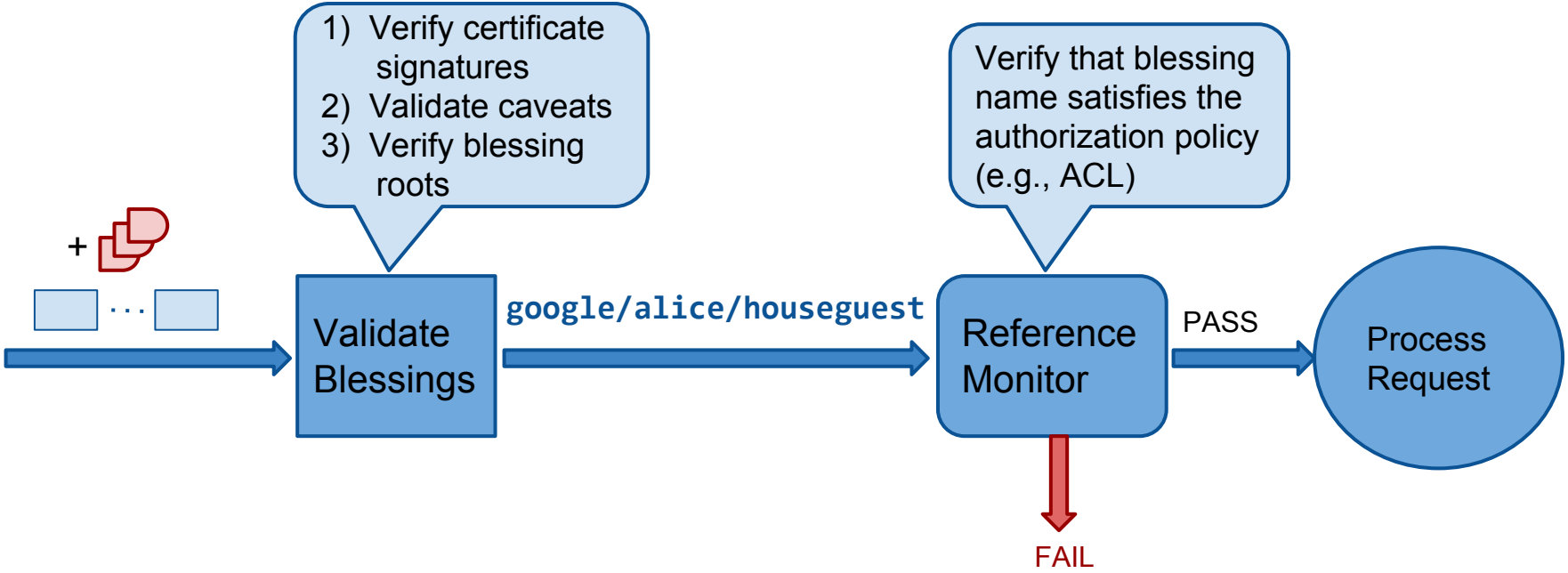
Authorization

Authorization policies are based on blessing names.



Authorization

Authorization policies are based on blessing names.



Access Control Lists (ACL)

Set of authorized blessings specified using explicit ACLs.

Alice's TV

Label	**Blessing Patterns**
Admin	**In**: `google/alice`
Photos	**In**: `google/alice/*`
Movies	**In**: `google/alice/spouse`

Access Control Lists (ACL)

Set of authorized blessings specified using explicit ACLs.

Alice's TV

Label	Blessing Patterns
Admin	**In**: `google/alice`
Photos	**In**: `google/alice/*`
Movies	**In**: `google/alice/spouse`

`google/alice/houseguest` can browse photos but cannot watch movies

Access Control Lists (ACL)

Set of authorized blessings specified using explicit ACLs.

Alice's TV

Label	Blessing Patterns
Admin	**In**: `google/alice`
Photos	**In**: `google/alice/*` **`NotIn: google/alice/housemaid`**
Movies	**In**: `google/alice/spouse`

`google/alice/houseguest` can browse photos but cannot watch movies

Distributed Groups

ACLs can also contain groups.

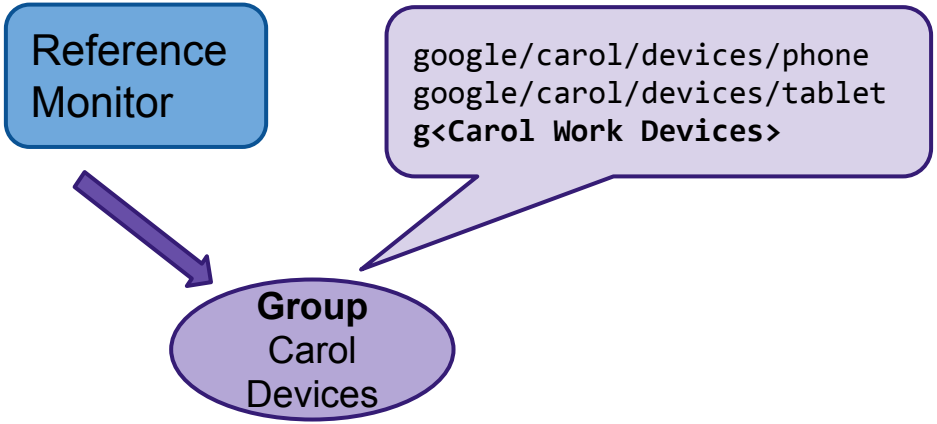
Label	Blessing Patterns
Admin	In: `alice`
Photos	In: `g<Carol Devices>`
Movies	In: `alice/devices/*`

Reference Monitor

Distributed Groups

ACLs can also contain groups.

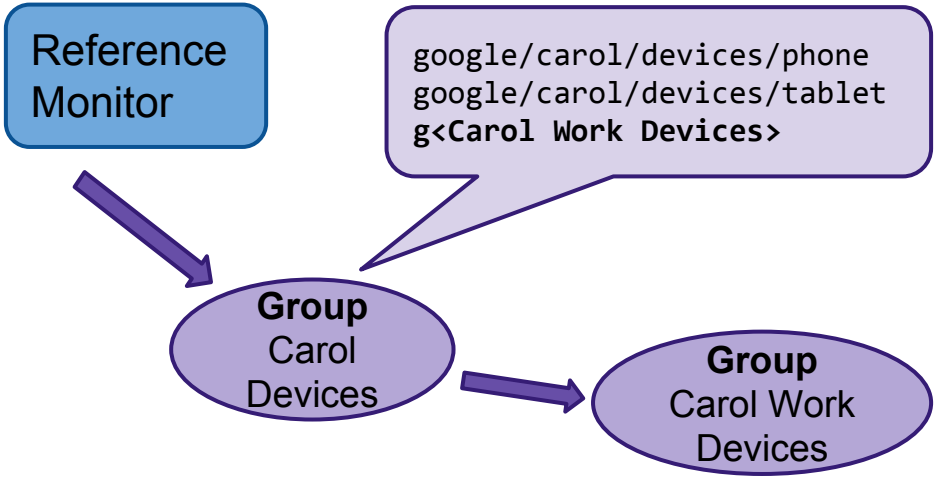
Label	Blessing Patterns
Admin	In: `alice`
Photos	In: `g<Carol Devices>`
Movies	In: `alice/devices/*`



Distributed Groups

ACLs can also contain groups.

Label	Blessing Patterns
Admin	In: `alice`
Photos	In: `g<Carol Devices>`
Movies	In: `alice/devices/*`



Distributed Authorization with Distributed Grammars, Abadi et al. PLABS'15

Case Study: Smart Lock

Lockitron

GOji

UNIKEY

August

Why Smart Locks?

- Remote locking/unlocking.
- Keyless proximity-based access.
- Maintain an audit log of who got in.
- Mint new (virtual) keys and share with others.
- Some also have a camera that will take the visitors picture.

Lock Setup

Blessings
google/alice

Blessings
lockcorp/1234

Lock Setup

Blessings
google/alice

Out of Band

<token>, <LockWiFi>

Blessings
lockcorp/1234

Lock Setup

Blessings
google/alice

I am **lockcorp/1234**

Blessings
lockcorp/1234

Authorization Policy
Claim: *Allow Everyone*

Lock Setup

Blessings
google/alice

Blessings
lockcorp/1234

Claim(“alice-door”, <token>, <Wifi>)

Authorization Policy
Claim: *Allow Everyone*

Lock Setup

Create self-signed blessing with name **`alice-door`**

Blessings
`google/alice`

Blessings
`lockcorp/1234`

`Claim(“alice-door”, <token>, <Wifi>)`

Authorization Policy
`Claim`: *Allow Everyone*

Lock Setup

Blessings
google/alice

Blessings
lockcorp/1234 **alice-door**

Claim(“alice-door”, <token>, <Wifi>)

Authorization Policy
Claim: *Allow Everyone*

Lock Setup

Blessings
google/alice

Now I am **alice-door**

Blessings
lockcorp/1234 **alice-door**

Claim(“alice-door”, <token>, <Wifi>)

Authorization Policy
Claim: *Allow Everyone*

Lock Setup

Blessings
google/alice **alice-door/key**

Blessings
lockcorp/1234 **alice-door**

Claim(“alice-door”, <token>, <Wifi>)

Authorization Policy
Claim: *Allow Everyone*

Lock Setup

Blessings
google/alice **alice-door/key**

Blessings
lockcorp/1234 **alice-door**

Claim(“alice-door”, <token>, <Wifi>)

Authorization Policy
Lock: alice-door/key/* **Unlock**: alice-door/key/*

Lock Setup

Blessings
google/alice **alice-door/key**

Blessings
lockcorp/1234 **alice-door**

Claim(“alice-door”, <token>, <Wifi>)

Lock using **alice-door/key**

Unlock using **alice-door/key**

Authorization Policy
Lock: alice-door/key/* **Unlock**: alice-door/key/*

Lock Setup

Blessings

google/alice
alice-door/key

Blessings

lockcorp/1234
alice-door

Claim(“alice-door”, <token>, <Wifi>)

Authorization Policy

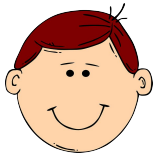
Lock: alice-door/key/*
Unlock: alice-door/key/*

Lock using **alice-door/key**

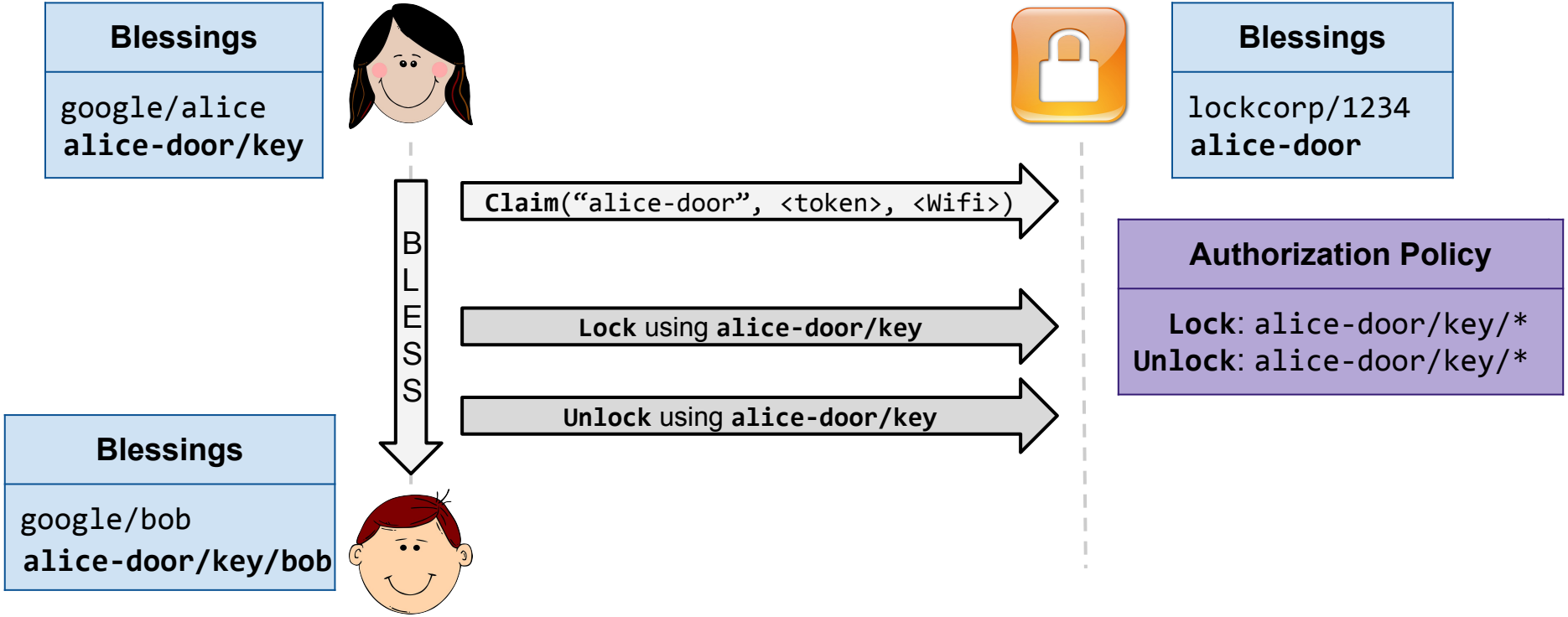
Unlock using **alice-door/key**

Blessings

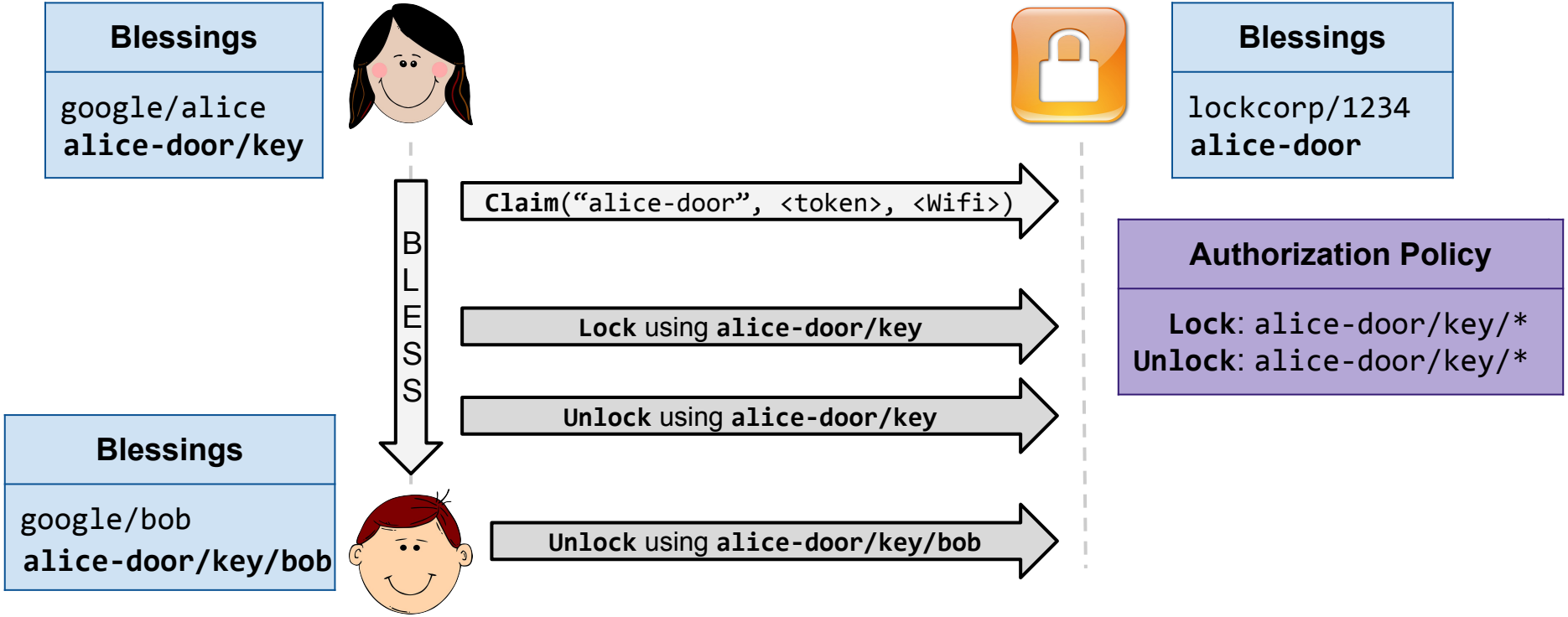
google/bob



Lock Setup


Blessings
google/alice
alice-door/key
Blessings
lockcorp/1234
alice-door
Claim("alice-door", <token>, <Wifi>)
BLESS
Lock using alice-door/key
Unlock using alice-door/key
Authorization Policy
Lock: alice-door/key/*
Unlock: alice-door/key/*
Blessings
google/bob
alice-door/key/bob

Lock Setup


Blessings
google/alice
alice-door/key
Blessings
lockcorp/1234
alice-door
Claim("alice-door", <token>, <Wifi>)
BLESS
Authorization Policy
Lock: alice-door/key/*
Unlock: alice-door/key/*
Lock using alice-door/key
Unlock using alice-door/key
Blessings
google/bob
alice-door/key/bob
Unlock using alice-door/key/bob

Lock Setup

- Works Offline

  No internet access required to interact with the lock
- Fully Decentralized

  No cloud server controls access to all locks
- Fine-grained Auditing

  Each lock device can keep track of who accessed it (including delegation trail)
- No bearer tokens involved

Practicalities

Vanadium Identity Provider

- We run our own identity provider that grants blessings by authenticating users using Google OAuth2.
  - Blessings are of the form dev.v.io/u/alice@gmail.com.
  - They carry third-party revocation caveats.
  - Blessing root: https://dev.v.io/auth/blessing-root.

- Other organizations (e.g., Facebook, Stanford University) can also become Identity Providers.

Blessings Management

- Devices and apps would accumulate multiple blessings over time.
- How should users visualize and grant blessings?

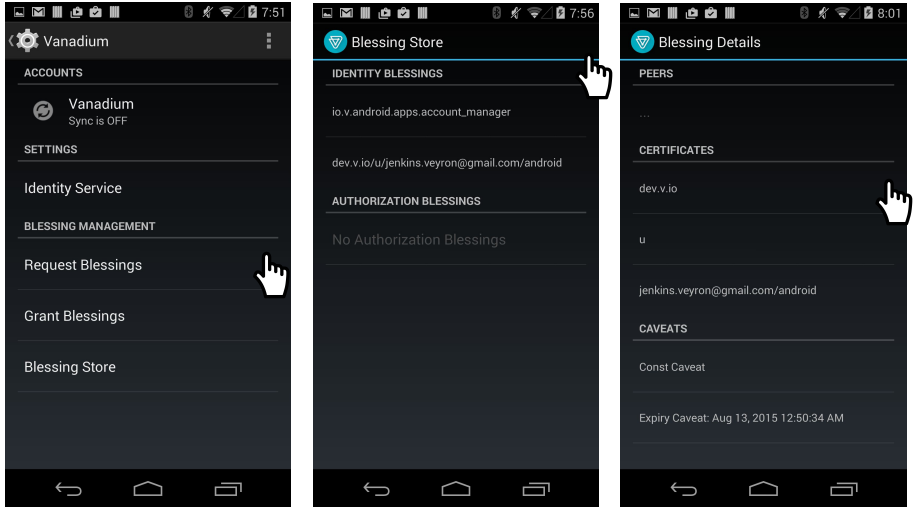
Blessings Management

- Devices and apps would accumulate multiple blessings over time.
- How should users visualize and grant blessings?

Vanadium Blessings Manager App

- UI for visualizing blessings
- Grant blessings over NFC, Bluetooth

<u>Future work</u>: Blessing mailbox in the cloud



Private Key Management

- Securely storing private keys on device.
- Many different hardware architectures and operating systems.
- Multiple private keys per device (one for each app).

Private Key Management

- Securely storing private keys on device.
- Many different hardware architectures and operating systems.
- Multiple private keys per device (one for each app).

<u>An Approach</u>: Use a security agent (e.g., Plan9's factotum)

- Special process that holds private keys and performs crypto.
- May store private keys in a TPM, if available.
- May adjust itself based on the hardware.

Conclusions

Vanadium Security Model: Summary

Principal and Blessings

A principal is a unique public/private key pair with human-readable names bound to it

All communication is encrypted & mutually authenticated

Forward-secrecy safe protocol, client and service identity privacy

Blessing names based Authorization

Principals authenticated and authorized based on their blessing names

Fine-grained Delegation and Audit

Bind an extension of a blessing to another principal under caveats

What goals have we met?

Identity and Authorization
Mutual authorization
Channel integrity and confidentiality
Forward secrecy
Fine-grained sharing and delegation
Revocation and auditing

Device Protection
No remote code execution
Automatic and secure updates
Verified boot

Privacy
Private discovery
Anonymous communication
Transparency

What goals have we met?

Identity and Authorization	
Mutual authorization	✓
Channel integrity and confidentiality	✓
Forward secrecy	✓
Fine-grained sharing and delegation	✓
Revocation and auditing	✓

Device Protection	
No remote code execution	⌛
Automatic and secure updates	⌛
Verified boot	⌛

Privacy	
Private discovery	⌛
Anonymous communication	⌛
Transparency	✗

Thank You!

Users, Devices, Applications

- The actors in our model are apps/processes
- Each app on each device is a separate principal
  - It may be acting-on-behalf-of of a user depending on its blessings
- Examples:
  - `google/alice` (AccountManager app on Alice's phone)
  - `google/alice/hometv` (DeviceManager app on Alice's TV)
  - `google/alice/hometv/youtube` (Youtube app on Alice's TV)