



Practical Distributed Authorization

Ankur Taly, Google Inc.
ataly@google.com

Two day course at the 2016 International school on Foundations of Security Analysis and Design, Bertinoro, Italy (Aug. 29 - Sep. 3)

Plan for today

- Vanadium access control policies
 - Key features
 - Algorithms and formalization
- Privacy, discovery and authentication for Vanadium and IoT
 - Problem statement
 - Review of cryptographic techniques
 - Protocol design, implementation, and benchmarks



Access Control Policies in Vanadium

Joint work with Martin Abadi, Mike Burrows, Himabindu Pucha,
Adam Sadowsky, and Asim Shankar

Recap: Vanadium authorization model

Principal and Blessings

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

Authorization is based on blessing names

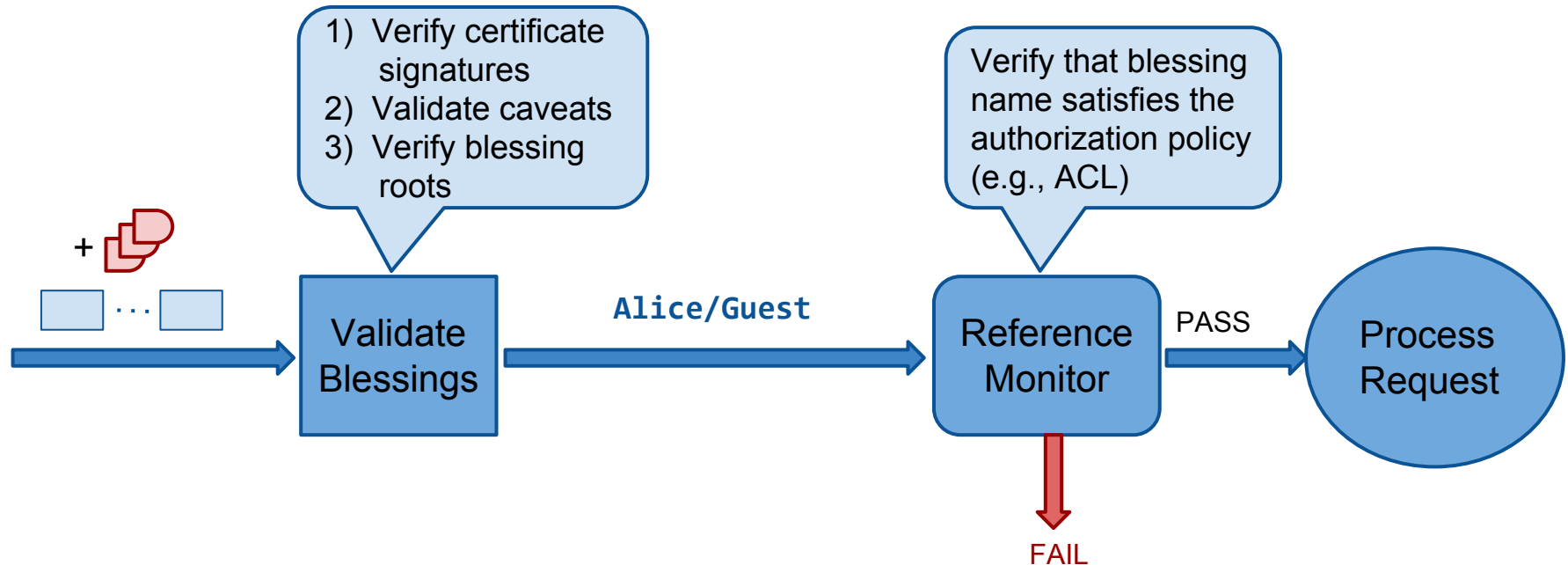
Principals authenticated and authorized based on their blessing names

Fine-grained delegation and audit

Principals can bind an extension of their blessings to another principal under caveats

Recap: Vanadium authorization model

Authorization policies are based on blessing names.



Plan

- Overview of access control policies
 - Distributed groups
 - Blessing patterns
 - Positive and negative constraints
- Formal semantics of access control
- Algorithms, implementation and trade-offs

Access control policies

Explicitly specify set of authorized blessings

Label	Policy
Photos	Allow: Alice, Alice/TV
Movies	Allow: Alice/Tablet

Access control policies

Explicitly specify set of authorized blessings

Label	Policy
Photos	Allow: Alice, Alice/TV
Movies	Allow: Alice/Tablet

Access control list (ACL)

This is the object of study for this lecture

Access control lists

Enumerate the set of authorized blessing names

ACL for Alice's device

Allow Alice, Alice/TV

How does Alice grant access to all her friends?

Access control lists

Enumerate blessing names of all friends

ACL for Alice's device

Allow Bob, Carol, Dave, ...

But, this is too cumbersome especially if it has to be done for several ACLs

Groups

Indirect through a group specifying Alice's friends

ACL for Alice's device

Allow Friends_G

Friends_G = Bob, Carol, Dave, ...

Groups

Groups may contain blessing names with multiple components

ACL for Alice's device

Allow Friends_G

Friends_G = Bob, Carol, Dave,
Carol/Friends/Eve, ...

Groups

The group can be modified independent of the ACL

ACL for Alice's device

Allow Friends_G

Friends_G = Bob, Carol, Dave,
~~Carol/Friends/Eve~~, ...

Nested groups

Groups may depend on other groups

ACL for Alice's device

Allow Friends_G

Friends_G = Bob, Carol, Dave,
BobFriends_G, ...

BobFriends_G = Mike, James, ...

Nested groups

Groups may depend on other groups

ACL for Alice's device

Allow Friends_G

$\text{Friends}_G = \text{Bob, Carol, Dave,}$
 $\text{BobFriends}_G, \dots$

$\text{BobFriends}_G = \text{Mike, James, } \dots$

Challenge: Group definitions may be spread across multiple administrative domains

Blessing patterns

Group names can be used within blessing names to form *blessing patterns*

ACL for Alice's device

Allow Friends_G/Phone

Friends_G = Bob, Carol, ...

Friends_G/Phone is matched by Bob/Phone, Carol/Phone, etc.

Blessing patterns

Group names can be used within blessing names to form *blessing patterns*

ACL for Alice's device

Allow Friends_G/Devices_G

Friends_G = Bob, Carol, ...

Devices_G = Phone, Tablet

Friends_G/Devices_G is matched by Bob/Phone, Carol/Tablet, etc.

Negative entries

Deny access to certain blessing names

ACL for Alice's device

Allow Friends_G

Deny Carol

Friends_G = Bob, Carol, ...

Negative entries

Deny access to entire groups

ACL for Alice's device

Allow Friends_G

Deny Enemies_G

Friends_G = Bob, Carol, ...

Enemies_G = Carol, Mike, ...

Negative entries

Deny access to entire groups

ACL for Alice's device

Allow Friends_G

Deny Enemies_G

$\text{Friends}_G = \text{Bob, Carol, ...}$

$\text{Enemies}_G = \text{Carol, Mike, ...}$

Challenge: Checking non-membership in a group

Blessing name extensions

ACL for Alice's device

Allow Alice, Bob

Deny Carol

Are extensions of **Alice** (e.g., **Alice/Guest**) allowed?

Are extensions of **Carol** (e.g., **Carol/Phone**) denied?

Blessing name extensions

ACL for Alice's device

Allow Alice, Bob

Deny Carol

Are extensions of **Alice** (e.g., **Alice/Guest**) allowed?

Yes

Are extensions of **Carol** (e.g., **Carol/Phone**) denied?

Yes

But the reasons are very different

Extensions of allowed names

ACL for Alice's device

Allow Alice, Bob

Deny Carol

Extensions of allowed blessing names are allowed for convenience

Simply say **Allow** Alice to allow access to all of Alice's delegates (e.g., Alice/Phone, Alice/Guest, etc.)

Stricter policy can be encoded with Alice/\$ that only matches Alice

Extensions of denied names

ACL for Alice's device

Allow Alice, Bob

Deny Carol

Extensions of denied blessing names are denied for security reasons

A principal with blessing **Carol** can easily extend it (say to **Carol/Phone**)

Thus, denying **Carol** must also imply denying all extensions of **Carol**

Recap of ACL features

- **Allow** and **Deny** clauses of blessing patterns
- Blessing patterns can contain group names
- Distributed group definitions
- Liberal matching on blessing pattern extensions

Fully decentralized with no overseeing authority

Need for formalization

Let $\text{Friends}_G = \text{Bob, Carol}$

Which of the following ACLs grant access to **Bob**?

ACL1

Allow Friends_G

Deny Friends_G

ACL2

Allow Friends_G

Deny $\text{Friends}_G / \text{Phone}$

ACL3

Allow $\text{Friends}_G / \text{Phone}$

Deny Friends_G

Need for formalization

Let $\text{Friends}_G = \text{Bob, Carol}$

Which of the following ACLs grant access to **Bob**?

ACL1

Allow Friends_G

Deny Friends_G

ACL2

Allow Friends_G

Deny $\text{Friends}_G / \text{Phone}$

ACL3

Allow $\text{Friends}_G / \text{Phone}$

Deny Friends_G

We must formally define the semantics of ACLs



Formalizing ACL semantics

Syntax

$bn ::= n$	blessing names
n / bn	
$p ::= n$	blessing patterns
g	
n / p	
g / p	
$P ::= \text{empty}$	list of blessing patterns
$P : p$	
$A ::= \text{Allow } P \text{ Deny } P$	ACL

Semantics

Objective: Define a predicate **IsAuthorized(bn, A)** that checks whether a blessing name **bn** is allowed by the ACL **A**

Plan

- Assume a semantics ρ for groups
 ρ maps a group name to the set of member blessing names
- Define “Meaning” of a blessing pattern in terms of ρ
Meaning maps a blessing pattern to the set of blessing names matched by it
- Use Meaning to define IsAuthorized

Meaning of blessing patterns

$$\begin{aligned} \text{Meaning}_\rho(p) ::= & \\ \text{case } p \text{ of} & \\ \quad n & : \{n\} \\ \quad | \quad g & : \rho(g) \\ \quad | \quad n / p & : \{n / s \mid s \in \text{Meaning}_\rho(p)\} \\ \quad | \quad g / p & : \{s / s' \mid s \in \text{Meaning}_\rho(g), s' \in \text{Meaning}_\rho(p)\} \end{aligned}$$

If $\rho(\text{Friends}_G) = \{\text{Bob}, \text{Alice/Friend}\}$ then

$\text{Meaning}(\text{Friends}_G / \text{phone}) = \{\text{Bob/Phone}, \text{Alice/Friend/Phone}\}$

Prefix matching (\sqsubseteq)

bn1 \sqsubseteq **bn2** if and only if $\text{bn1} == \text{bn2}$ OR bn1 is an extension of bn2

e.g., Alice \sqsubseteq Alice/Phone is True

Ali \sqsubseteq Alice /Phone is False

IsAuthorized

IsAuthorized(bn , **Allow** P_A **Deny** P_D) :=

$$\exists bn_1 \in \text{Meaning}_\rho(P_A). bn_1 \sqsubseteq bn$$
$$\wedge \neg \exists bn_1 \in \text{Meaning}_\rho(P_D). bn_1 \sqsubseteq bn$$

IsAuthorized

IsAuthorized(bn, **Allow** P_A **Deny** P_D) :=

$\exists bn_1 \in \text{Meaning}_p(P_A). bn_1 \sqsubseteq bn$

$\wedge \neg \exists bn_1 \in \text{Meaning}_p(P_D). bn_1 \sqsubseteq bn$

MUST be an extension of **some** blessing name matched by Allowed pattern

IsAuthorized

IsAuthorized(bn , **Allow** P_A **Deny** P_D) :=

$\exists bn_1 \in \text{Meaning}_p(P_A). bn_1 \sqsupseteq bn$

MUST be an extension of **some** blessing name matched by Allowed pattern

$\wedge \neg \exists bn_1 \in \text{Meaning}_p(P_D). bn_1 \sqsupseteq bn$

MUST NOT be an extension of **any** blessing name matched by Denied pattern



Semantics of Groups

How do we define ρ ?

Challenges

- Distributed group definitions
- Membership list must be held private (as much as possible)
- Cyclic dependencies
 - AliceFriends_G = Carol, BobFriends_G
 - BobFriends_G = Mike, AliceFriends_G
- No central overseeing authority

Observation: Group definitions induce formal grammars

ACL

Allow $\text{AliceFriends}_G / \text{Devices}_G$

Deny AliceEnemies_G

Groups

$\text{AliceFriends}_G = \text{Carol}, \text{BobFriends}_G$

$\text{BobFriends}_G = \text{Mike}, \text{AliceFriends}_G$

$\text{Devices}_G = \text{Phone}, \text{Tablet}$

$\text{AliceEnemies}_G = \text{Carol}, \text{James}$

Group names are non-terminals and group definitions are productions

$\rho(\text{AliceFriends}_G)$ is the language induced by AliceFriends_G

Checking membership in $\rho(\text{AliceFriends}_G)$ is akin to parsing

Distributed grammars

What if some group definitions (grammar productions) are unreachable?

Conservatively approximate => Define two semantics ρ^{\downarrow} and ρ^{\uparrow}

- ρ^{\downarrow} (**rho-under**): Map unreachable groups to empty set
Used in determining meaning of **Allow** patterns
- ρ^{\uparrow} (**rho-over**): Map unreachable groups to set of all blessing names
Used in determining meaning of **Deny** patterns

IsAuthorized (approximate)

IsAuthorized(bn, **Allow** P_A **Deny** P_D) :=

\exists $bn_1 \in \text{Meaning}_{\rho \downarrow}(P_A). bn_1 \sqsubseteq bn$

$\wedge \neg \exists$ $bn_1 \in \text{Meaning}_{\rho \uparrow}(P_D). bn_1 \sqsubseteq bn$

Next step: Design an algorithm for evaluating $\exists bn_1 \in \text{Meaning}_{\rho}(P_A). bn_1 \sqsubseteq bn$ without expanding the definition of ρ

The “Rest” algorithm for distributed parsing

Parsing problem: Match a blessing name against a blessing pattern
e.g., match `Mike/Phone` against `AliceFriendsG/DevicesG`

Idea: Distributed top-down parsing

- Query a group g : what part of the blessing name bn can g consume and what remains (rest)?
e.g. , $\mathbf{Rest}(\text{Mike/Phone}, \text{AliceFriends}_G) = \text{Phone}$
- Recursively make queries on the “rest” to other groups
- Conservative semantics for unreachable groups

Example

ACL

Allow AliceFriends_G/Devices_G

Deny AliceEnemies_G

Groups

AliceFriends_G = Carol, BobFriends_G

BobFriends_G = Mike, AliceFriends_G

Devices_G = Phone, Tablet

AliceEnemies_G = Carol, James

Check whether Mike/Phone is authorized

Reference
Monitor

Example

ACL

Allow AliceFriends_G/Devices_G

Deny AliceEnemies_G

Groups

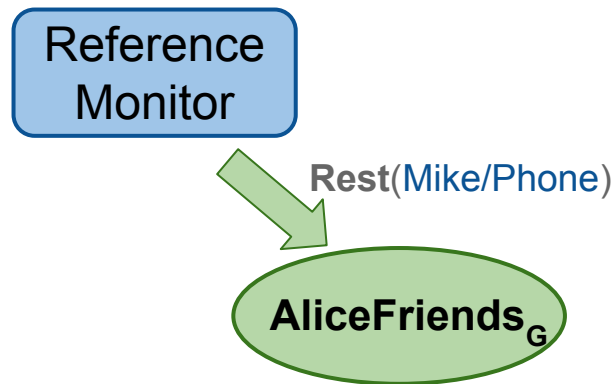
AliceFriends_G = Carol, BobFriends_G

BobFriends_G = Mike, AliceFriends_G

Devices_G = Phone, Tablet

AliceEnemies_G = Carol, James

Check whether Mike/Phone is authorized



Example

ACL

Allow AliceFriends_G/Devices_G

Deny AliceEnemies_G

Groups

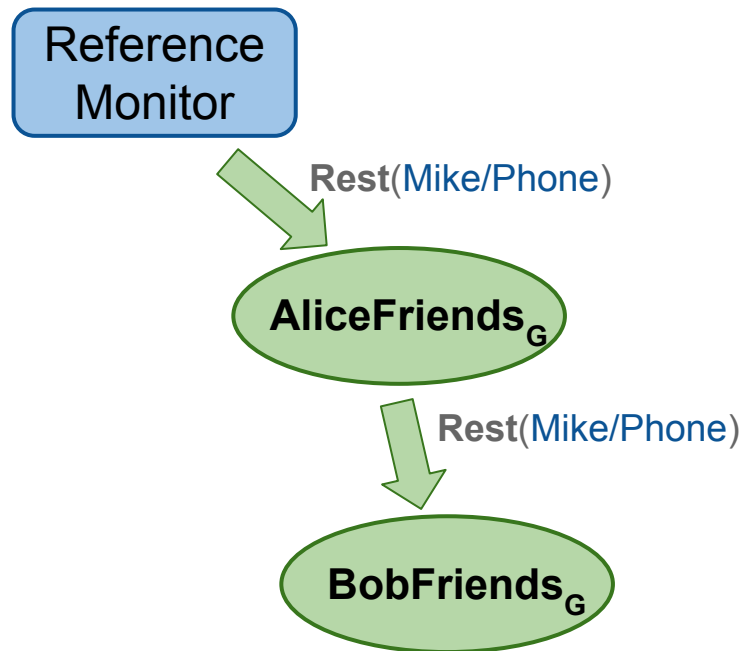
AliceFriends_G = Carol, BobFriends_G

BobFriends_G = Mike, AliceFriends_G

Devices_G = Phone, Tablet

AliceEnemies_G = Carol, James

Check whether Mike/Phone is authorized



Example

ACL

Allow AliceFriends_G/Devices_G

Deny AliceEnemies_G

Groups

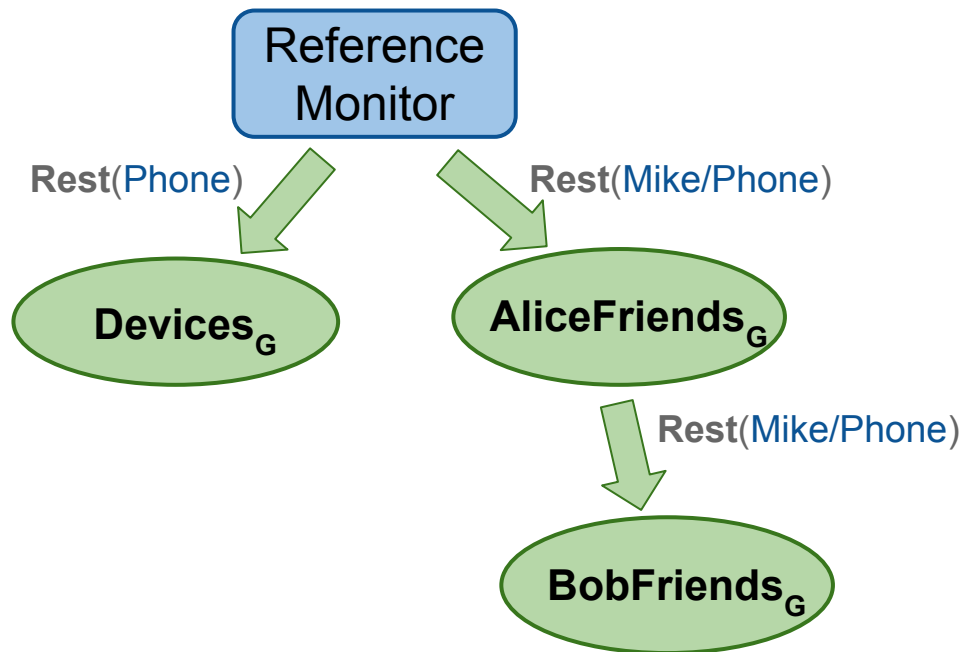
AliceFriends_G = Carol, BobFriends_G

BobFriends_G = Mike, AliceFriends_G

Devices_G = Phone, Tablet

AliceEnemies_G = Carol, James

Check whether Mike/Phone is authorized



Trade-offs

- Privacy
 - Reference monitor still reveals entire blessing name to group servers and group servers reveal at least part of blessing name that matches
 - **Open problem:** [Secure multiparty parsing](#)
- Communication costs
 - Each recursive call results in network communication
 - Left recursive definitions are a problem for top-down parsing
 - Implementations would likely operate under a communication budget
 - Caching vs. mutable group definitions

Perspective

- The combination of local names, distributed groups and deny clauses make access control policies **very expressive** **but complex to validate**
- Decentralization makes the problem even for challenging
- Several implementation and design trade-offs make the problem tractable
- Formal methods come to the rescue, yet again!

Distributed Authorization with Distributed Grammars,
Abadi et al., PLABS 2014



Privacy, Discovery and Authentication for Internet of Things (IoT)

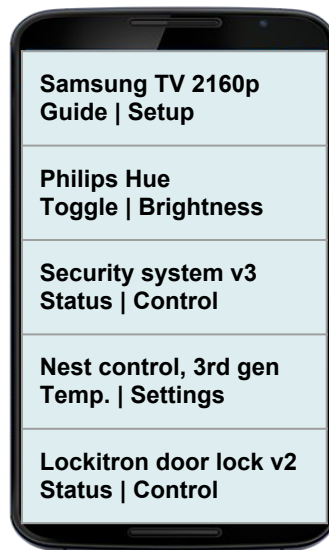
Joint work with

Asim Shankar (Google), David Wu (Stanford University) and
Dan Boneh (Stanford University)

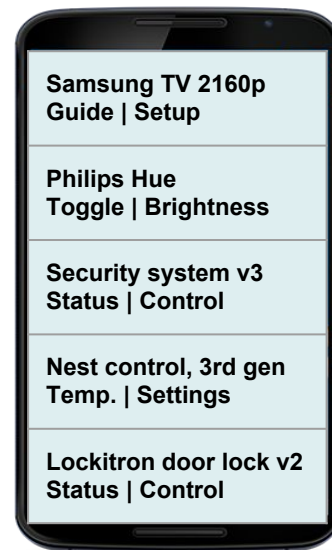
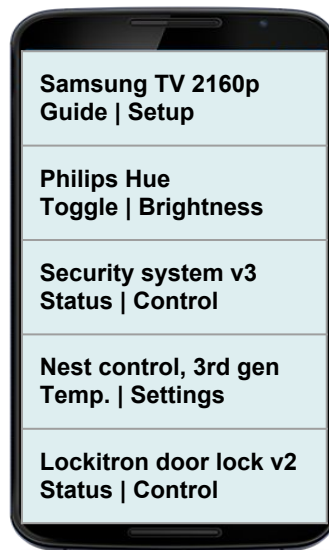
Agenda

- Two Problems
 - Private service discovery
 - Private mutual authentication
- Solution
 - Identity-Based encryption (IBE)
 - Protocol design, implementation, and benchmarks
- Case study: Apple AirDrop

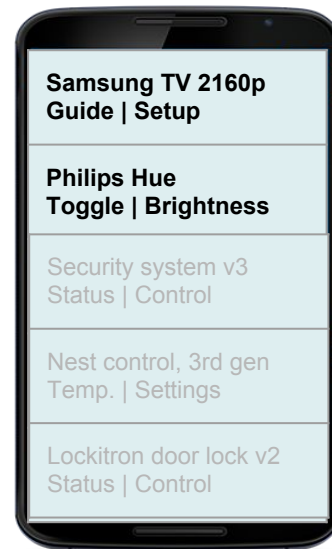
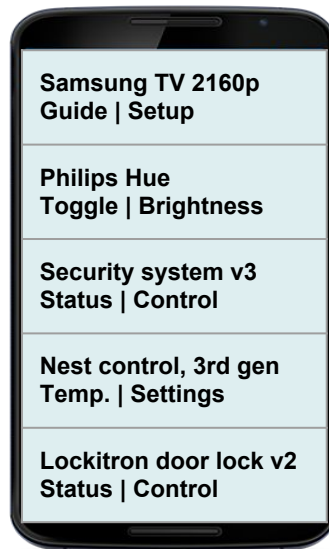
Private Discovery



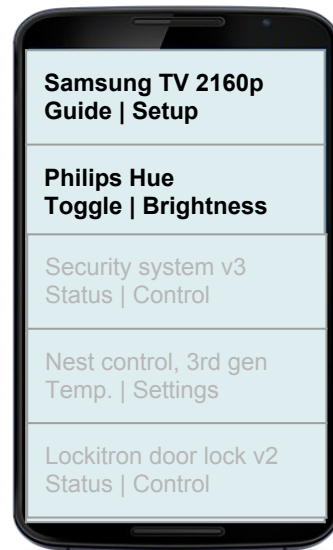
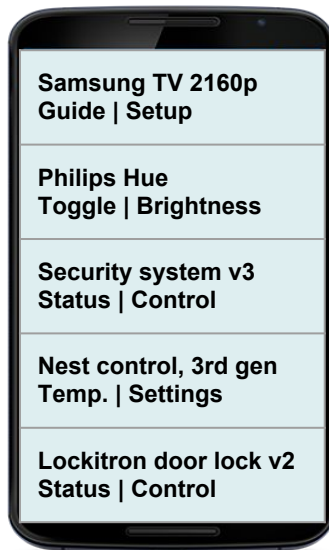
Private Discovery



Private Discovery



Private Discovery



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

Private discovery is largely unaddressed

No privacy controls in mDNS, Bonjour, and BLE

A [recent study](#) of mDNS announcements from 2957 devices in a university campus revealed that 59% devices reveal the device owner name (viewed as a private threat by 92% users)

Advertisements have no authenticity

A rogue device may forge or replay a legitimate service advertisement to determine if a client is interested in the service

Mutual Authentication (SIGMA-I)

Client



Configure



Server

Diffie-Hellman exchange

$\{ \text{Blessing}_{\text{SecSys}} + \text{Signature}_{\text{SecSys}} \}_k$

$\{ \text{Blessing}_{\text{Attacker}} + \text{Signature}_{\text{attacker}} \}_k$

Notation

k : key from DH secret

$\{m\}_k$: encryption of m under symmetric key k

Mutual Authentication (SIGMA-I)

Client



Configure



Server

At this point, attacker learns security system's blessings

Attacker may choose to send its blessing or abort.

Diffie-Hellman exchange

{ **Blessing**_{SecSys} + Signature_{SecSys} }_k

{ **Blessing**_{Attacker} + Signature_{attacker} }_k

Notation

k : key from DH secret

$\{m\}_k$: encryption of m under symmetric key k

Mutual Authentication (SIGMA-I)

Client



Configure



Server

At this point, attacker learns security system's blessings

Attacker may choose to send its blessing or abort.

Diffie-Hellman exchange

{ **Blessing**_{SecSys} + Signature_{SecSys} }_k

Notation

k : key from DH secret

$\{m\}_k$: encryption of m under symmetric key k

Mutual authentication (SIGMA-I)

Client



Configure



Server

At this point, attacker learns security system's blessings

Attacker may choose to send its blessing or abort.

Diffie-Hellman exchange

{ **Blessing**_{SecSys} + Signature_{SecSys} }_k

Notation

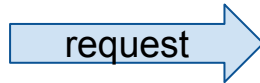
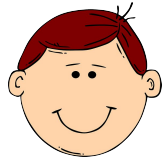
k : key from DH secret

$\{m\}_k$: encryption of m under symmetric key k

Server's blessing is revealed to *anyone* who communicates with it.
Client's blessing maintains its privacy.

Private mutual authentication

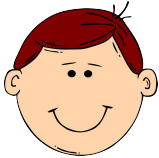
- In most existing mutual authentication protocols (e.g., TLS), one party must “go first”, i.e., reveal its identity first
- But, this is a problem when both the parties are mutually suspicious



Private mutual authentication

- In most existing mutual authentication protocols (e.g., TLS), one party must **“go first”**, i.e., reveal its identity first
- But, this is a problem when both the parties are mutually suspicious

Will only reveal my identity to Alice's security system



request



Will only reveal my identity to Alice's friends and family

Private mutual authentication

- In most existing mutual authentication protocols (e.g., TLS), one party must “go first”, i.e., reveal its identity first
- But, this is a problem when both the parties are mutually suspicious

Will only reveal my identity to Alice's security system



request



Will only reveal my identity to Alice's friends and family

Private mutual authentication: Each end authenticates to the other *if and only if* other end is authorized.

Known approaches

- Use an online trusted third-party to mediate interaction
 - Requires connectivity to third-party
 - Requires trusting the third-party
 - Conflicts with our goal of designing peer-to-peer mechanism
- Out-of-band shared secret between client and server
 - Implies pre-existing relationship between client and server
 - What's the fun in discovering services you've already seen!!

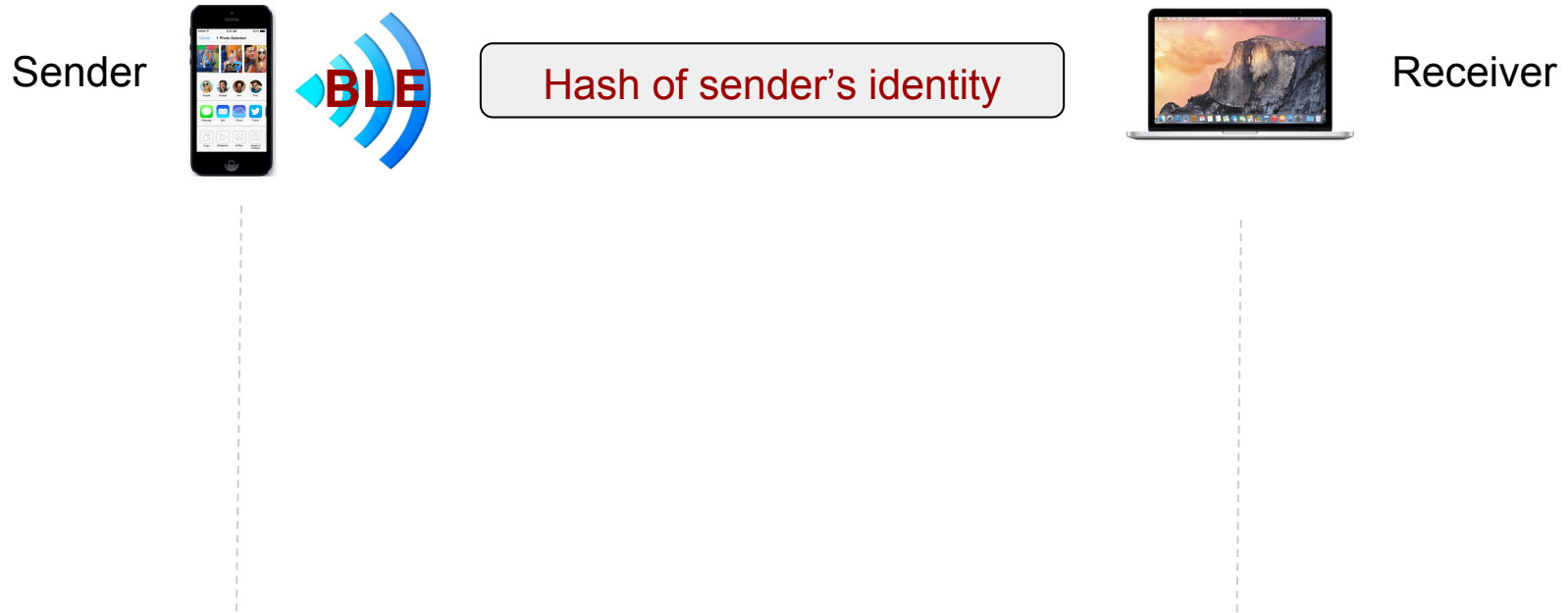


Apple AirDrop

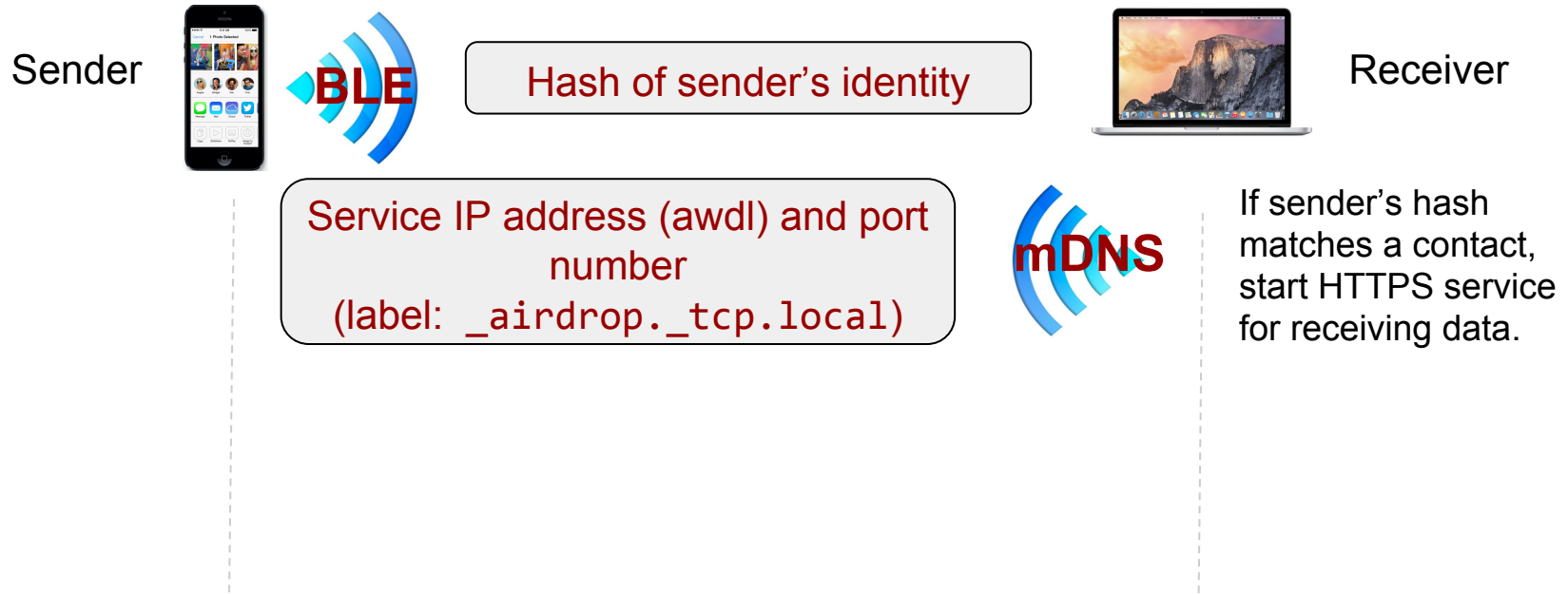
Apple AirDrop

- Peer-to-peer file sharing on iOS (> 7) and OSX (> 10.7) devices
- Uses BLE and Apple's peer-to-peer WiFi technology (awdl)
- Two modes
 - Everyone (accept files from everyone)
 - **Contacts-Only** (accept files only from contacts)

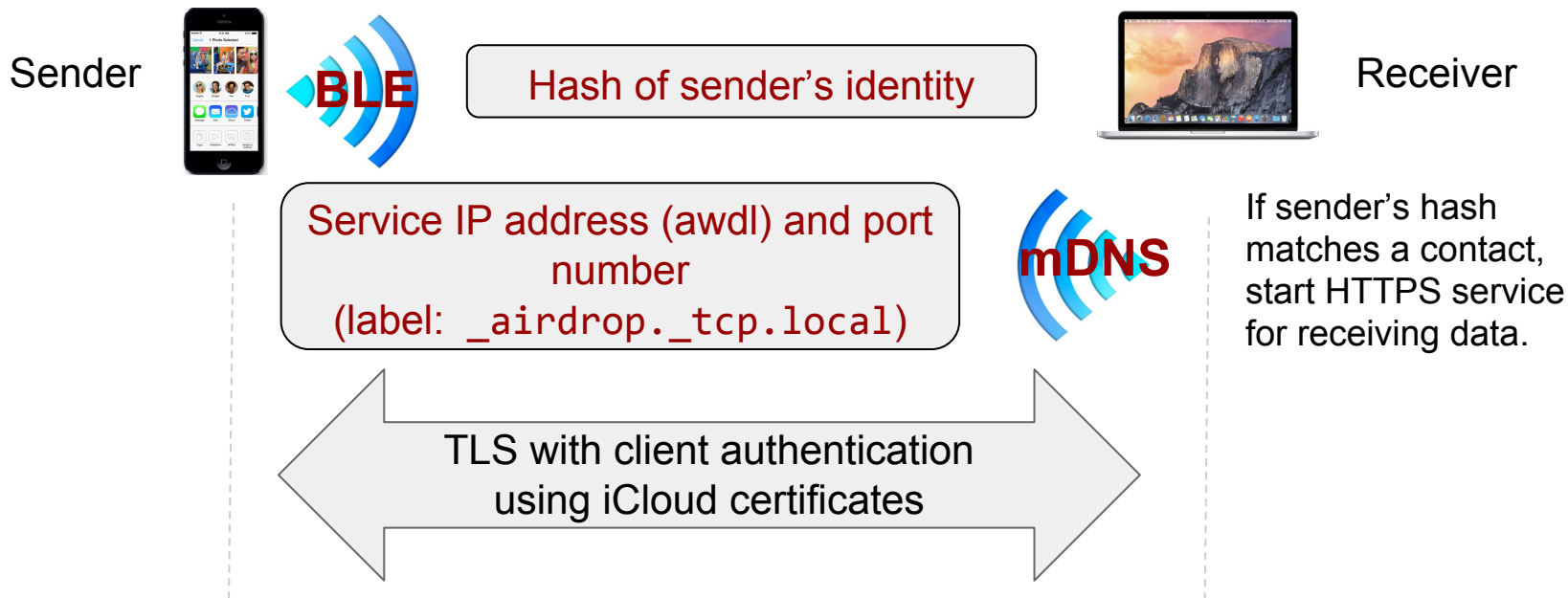
AirDrop (contacts-only mode)



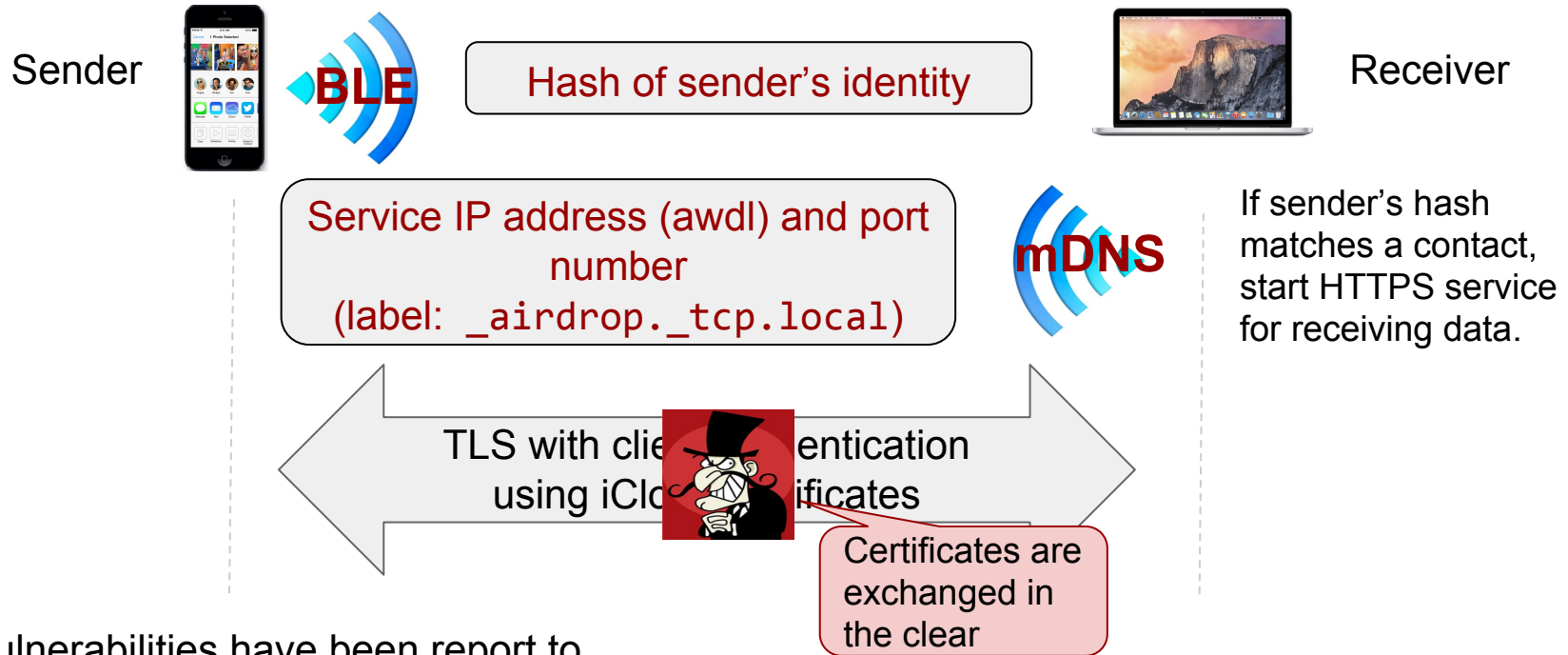
AirDrop (contacts-only mode)



AirDrop (contacts-only mode)

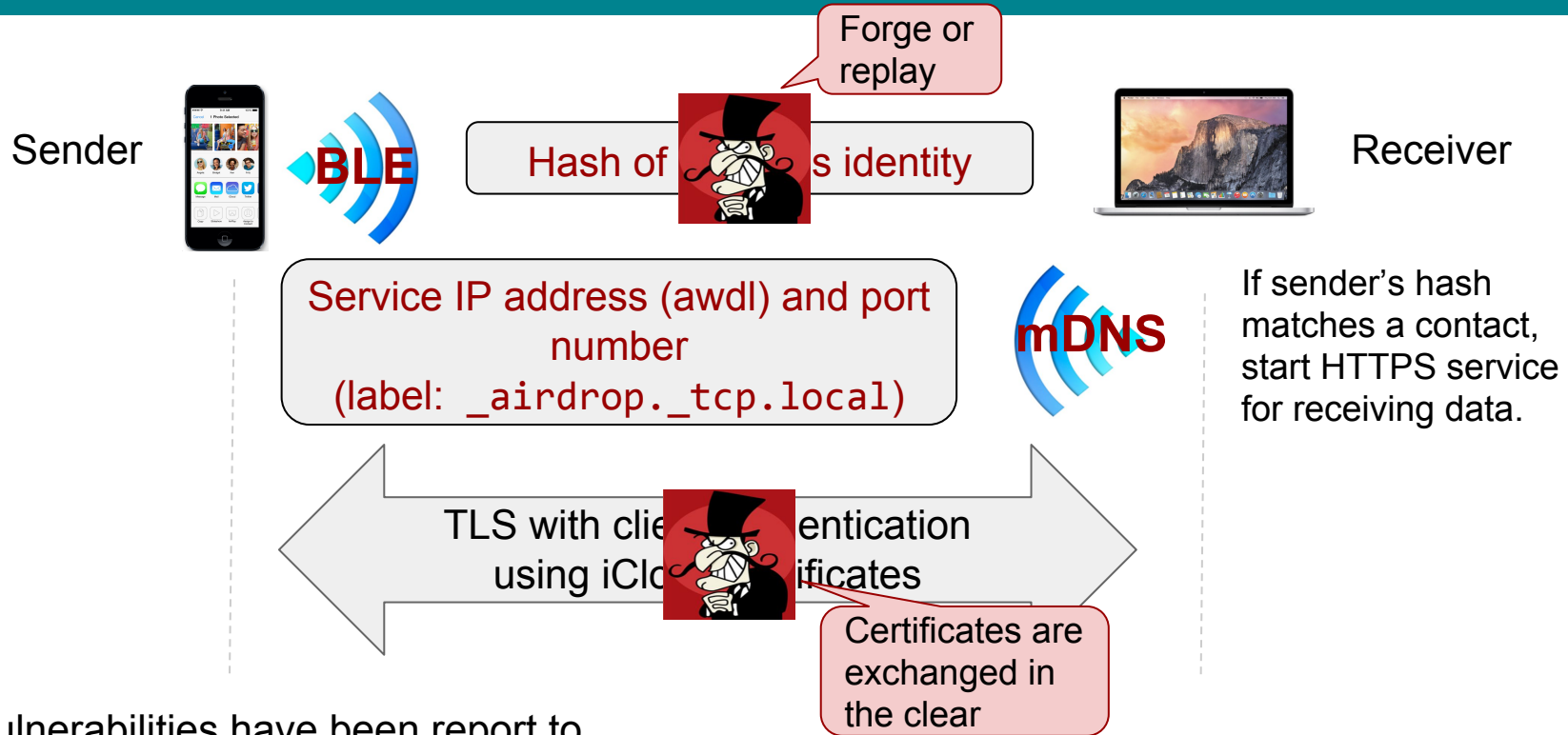


AirDrop (contacts-only mode) vulnerabilities



Vulnerabilities have been reported to Apple, awaiting response

AirDrop (contacts-only mode) vulnerabilities



Vulnerabilities have been report to Apple, awaiting response

Our design goals

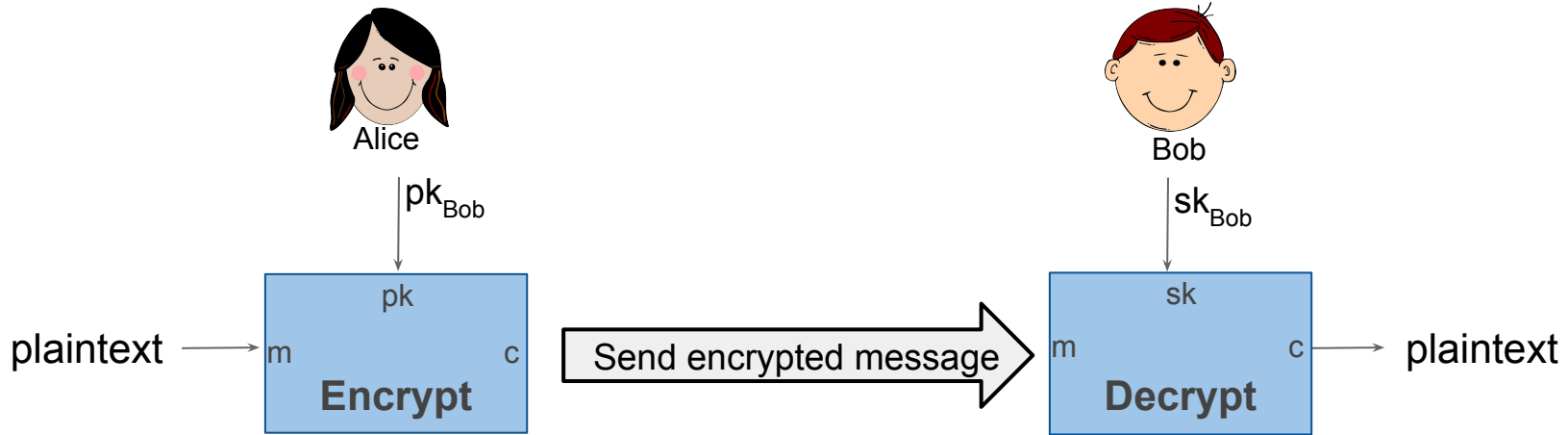
- **Mutual privacy**
Protocol participants reveal their identities only to **authorized recipients**
- **Authentic advertisements (for service discovery)**
Service advertisements should be unforgeable and authentic
- **No dependence on global services during protocol execution**
- **No out-of-band pairing for participants**



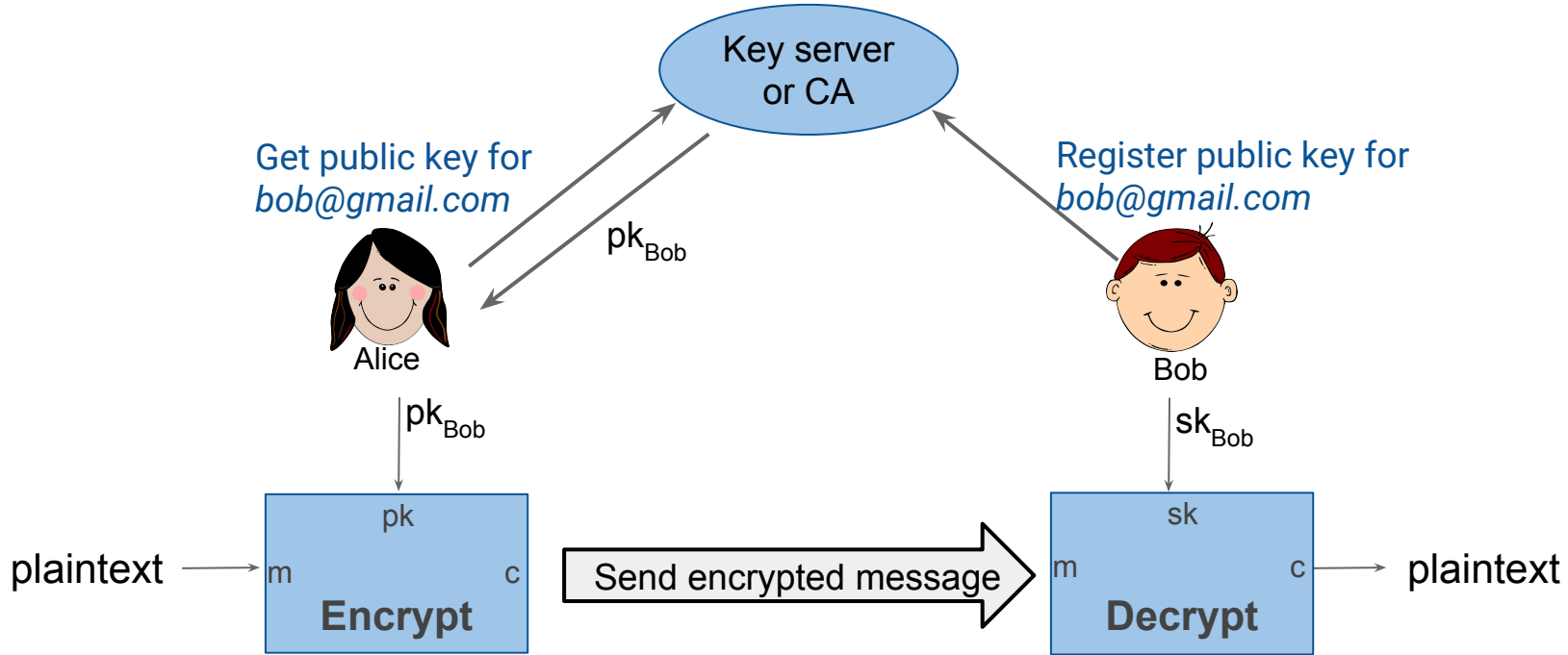
Identity-Based Encryption

Shamir (1984): *“What if your name could be your public key?”*

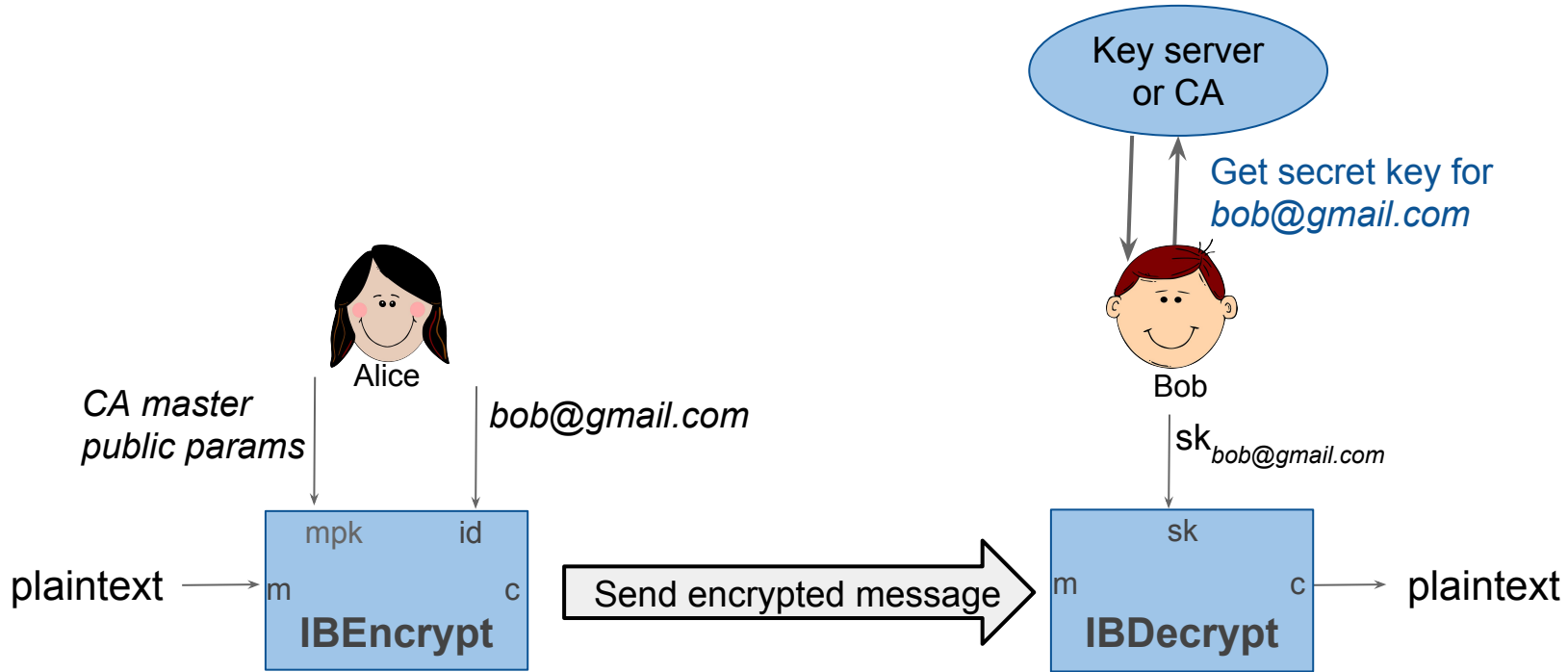
Public-key encryption



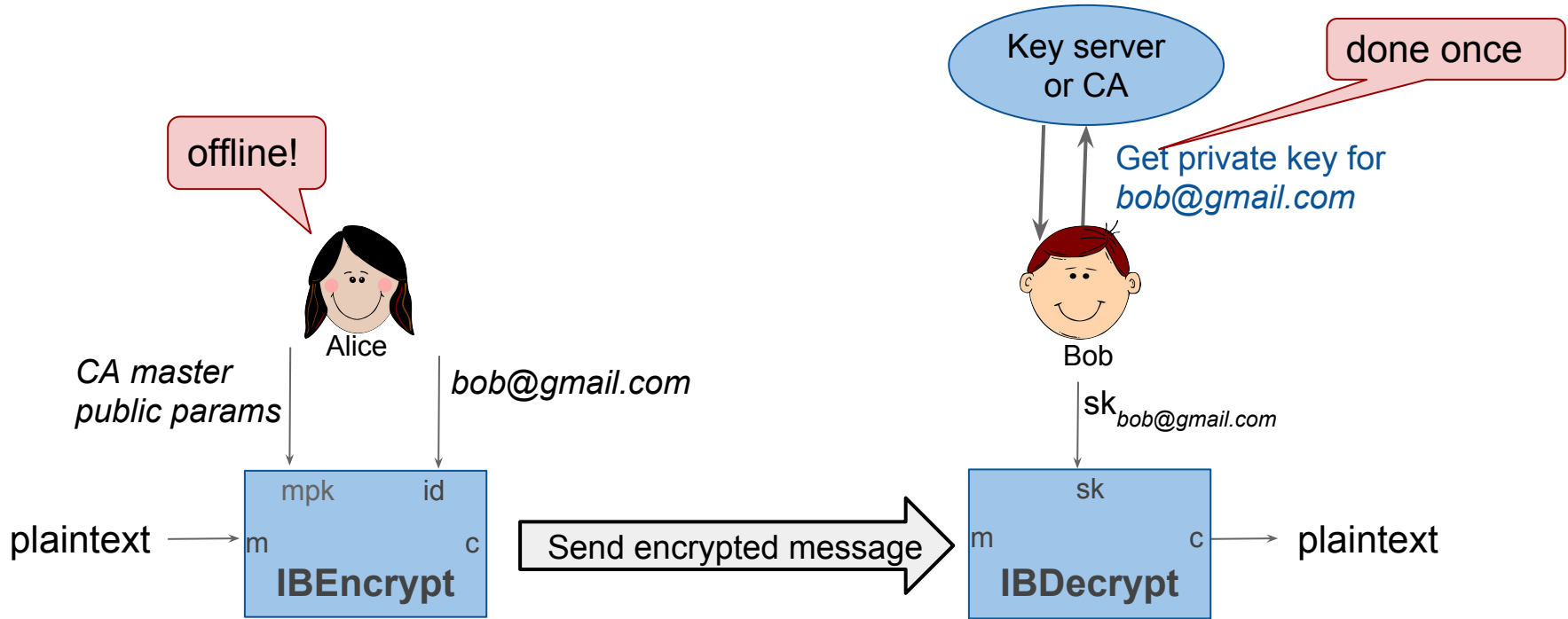
Public-key encryption



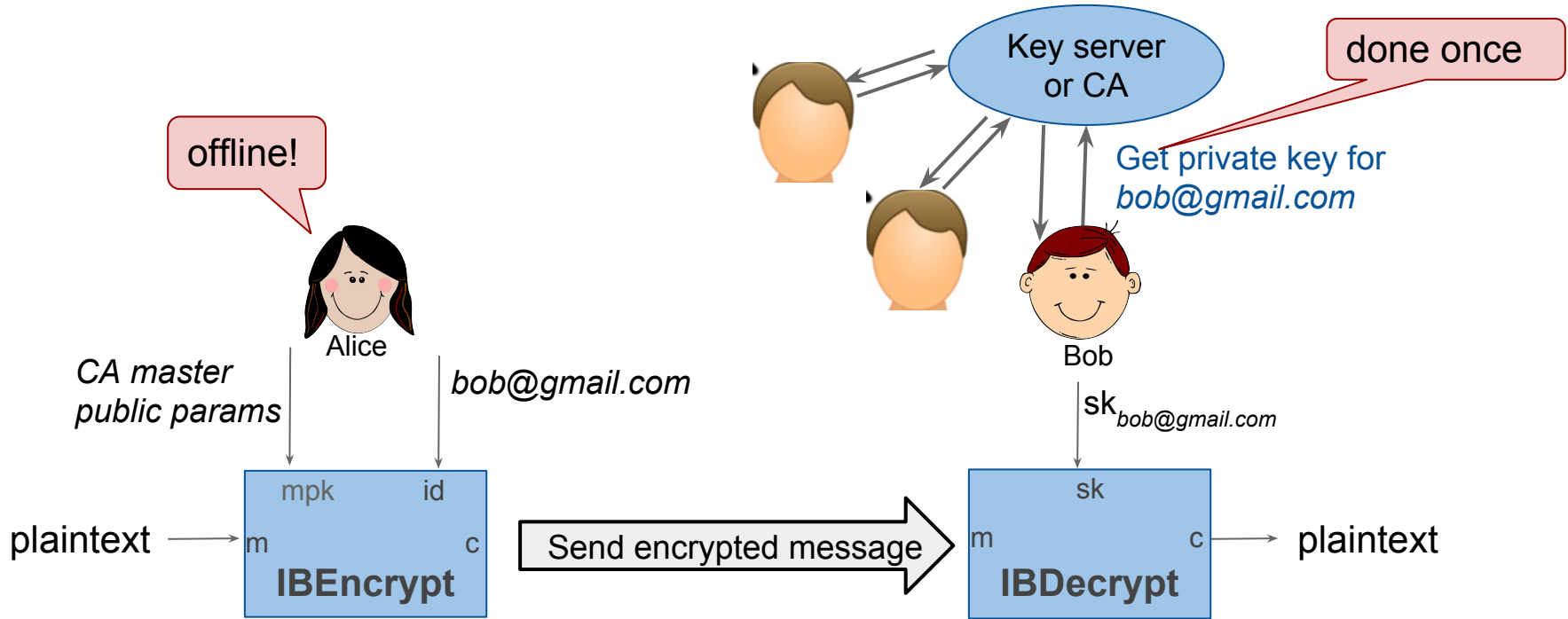
Identity-based encryption



Identity-based encryption



Identity-based encryption



IBE cryptosystem

Operations

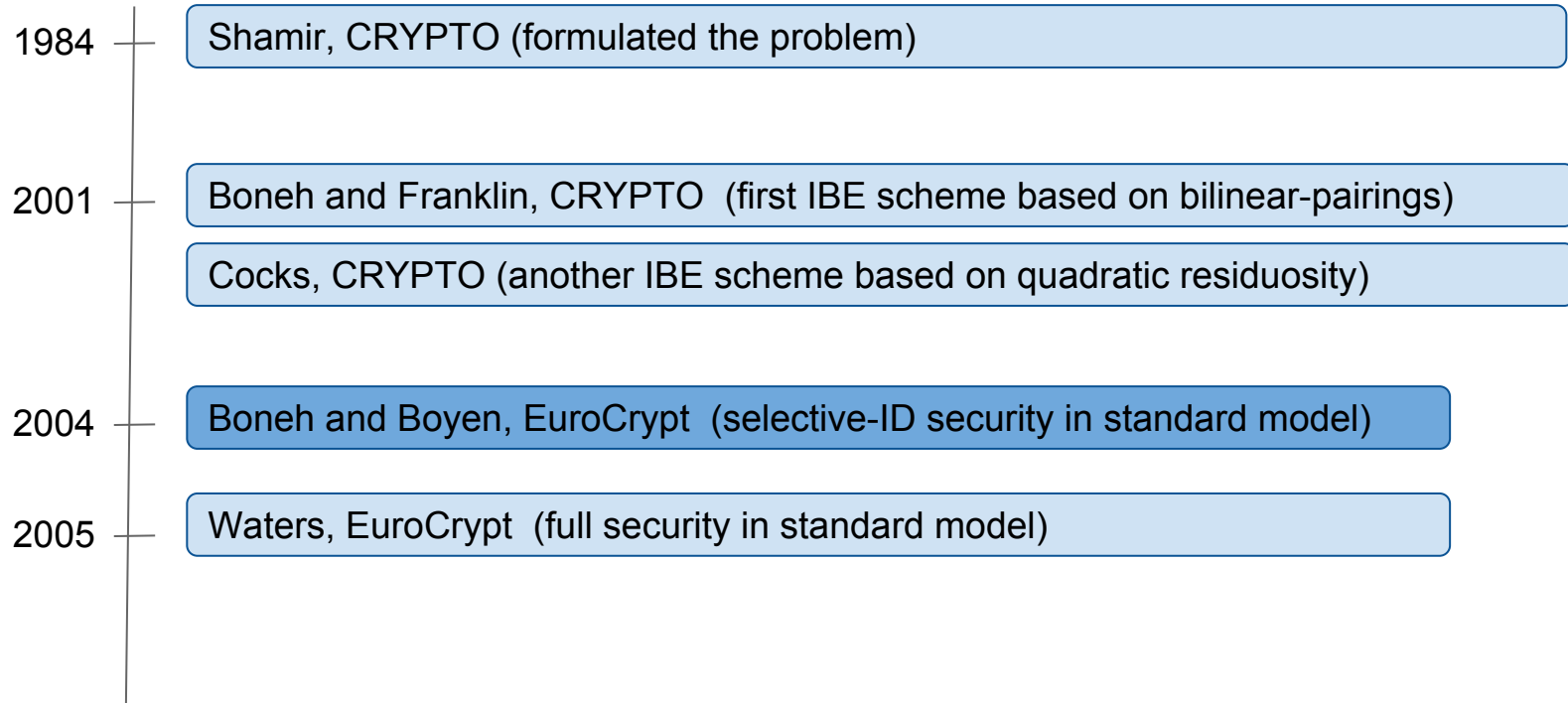
- $\text{Encrypt}(\text{mpk}, \text{ID}, \text{ptxt})$
- $\text{Decrypt}(\text{sk}_{\text{ID}}, \text{ctxt})$
- $\text{Extract}(\text{msk}, \text{ID})$

Notation

- mpk : master public params (well-known)
- msk : master private key (known to IBE root)
- sk_{ID} : private key for ID
- ptxt : plaintext
- ctxt : ciphertext

$\forall \text{ptxt} \forall \text{ID}. \text{Decrypt}(\text{Extract}(\text{msk}, \text{ID}), \text{Encrypt}(\text{mpk}, \text{ID}, \text{ptxt})) = \text{ptxt}$

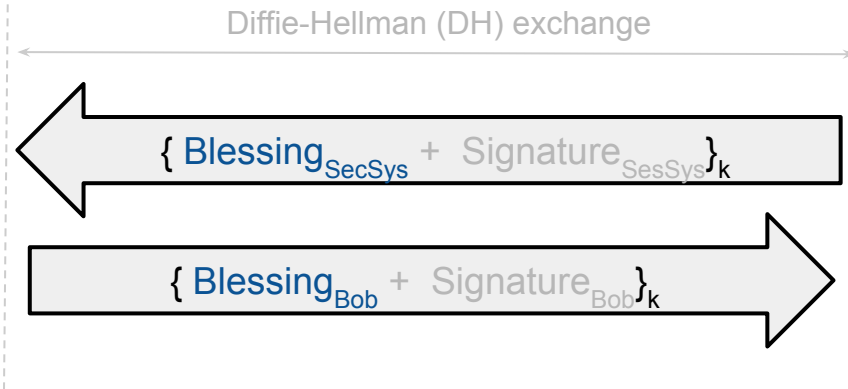
Important IBE results





Private Mutual Authentication

Key Idea



Notation

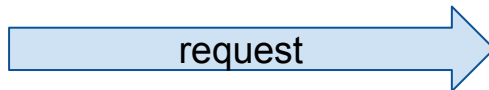
k : key from DH secret

$\{m\}_k$: authenticated-encryption of m under key k .

Key idea



Blessing: Alice/Family/Bob
Policy: **Allow** Alice/SecSys



Blessing: Alice/SecSys
Policy: **Allow** Alice/Family

Bob can read security system's blessings *only if* he satisfies the system's policy.

Bob responds *only if* the security system satisfies his policy.

Diffie-Hellman (DH) exchange



Notation

k : key from DH secret

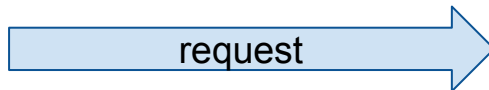
$\{m\}_k$: authenticated-encryption of m under key k .

$\llbracket m \rrbracket_{\text{pol}}$: encryption of m under policy 'pol'.

Key idea



Blessing: Alice/Family/Bob
Policy: **Allow** Alice/SecSys

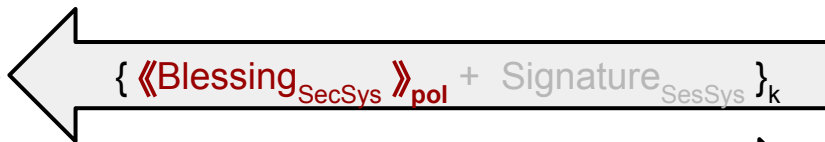


Blessing: Alice/SecSys
Policy: **Allow** Alice/Family

Diffie-Hellman (DH) exchange

Bob can read security system's blessings *only if* he satisfies the system's policy.

Bob responds *only if* the security system satisfies his policy.



Notation

k : key from DH secret

$\{m\}_k$: authenticated-encryption of m under key k .

$\llbracket m \rrbracket_{\text{pol}}$: encryption of m under policy 'pol'.

How do we build policy-based encryption?

Policy-based encryption

Problem: *Encrypt a message under an authorization policy so that it can be decrypted only by principals with blessings satisfying the policy*

Policies in Vanadium are blessing prefixes

e.g., **Alice/Family** is matched by **Alice/Family/Bob**, **Alice/Family/Carol**, ...

(We simplify and do not consider groups in this work)

We can build prefix-based encryption using IBE [Lewko-Waters '14].

- Encrypt - Use `IBEncrypt` with prefix as the identity
- Extract - Use `IBExtract` to obtain a key for each prefix of the blessing

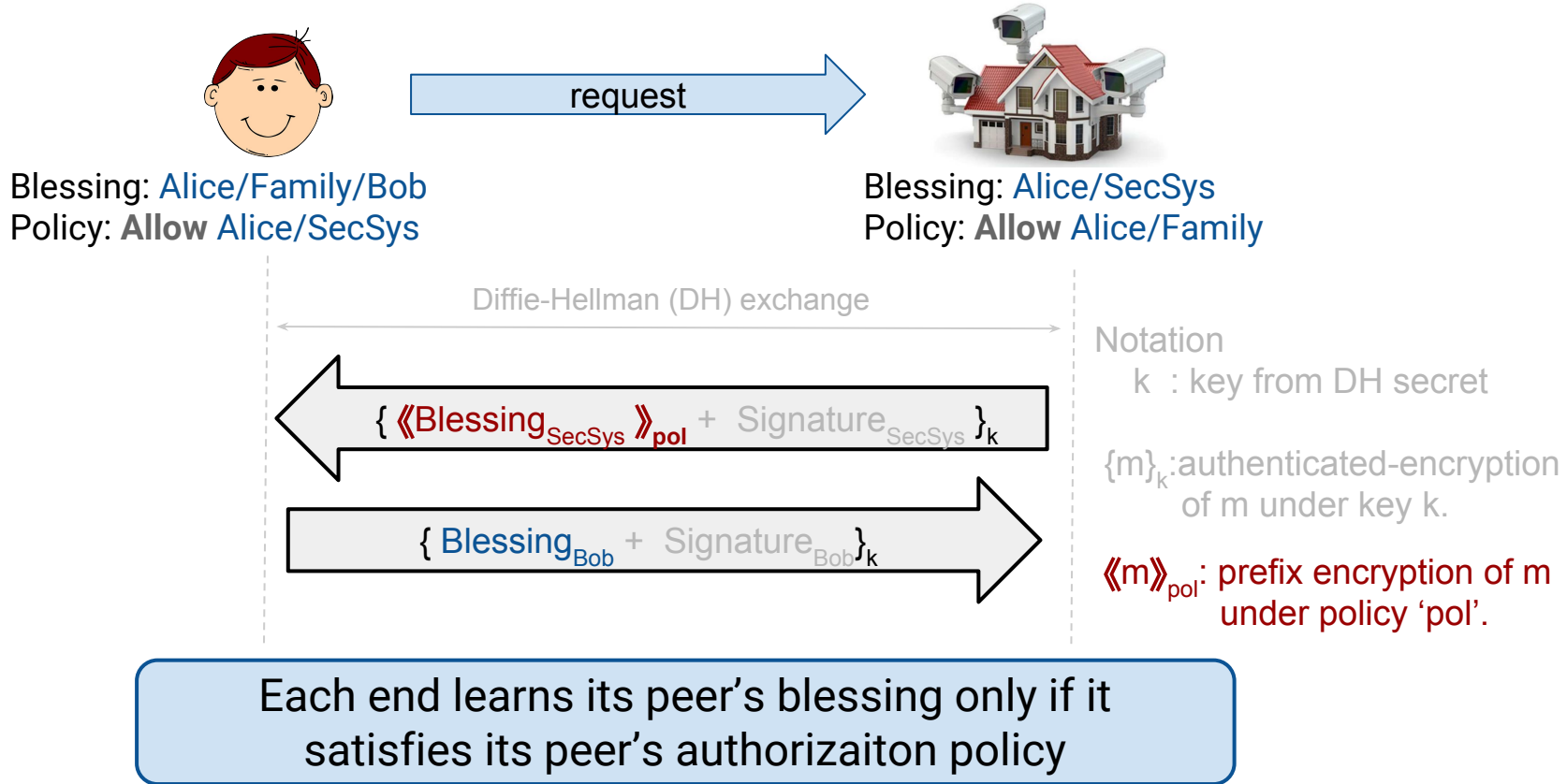
Prefix-based encryption

`PrefixEncrypt(mpk, m, "Alice/Family")` returns
`IBEncrypt(mpk, m, "Alice/Family")`

`PrefixExtract(msk, "Alice/Family/Bob")` returns a vector of private keys
extracted for identities

- "Alice"
- "Alice/Family"
- "Alice/Family/Bob"
- "Alice/Family/Bob/\$"

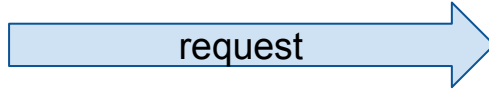
Private mutual authentication



Private mutual authentication



Blessing: Alice/Family/Bob
Policy: **Allow** Alice/SecSys



Blessing: Alice/SecSys
Policy: **Allow** Alice/Family

Server overhead: None

Client overhead: One
IBE Decryption per
handshake

Diffie-Hellman (DH) exchange



Notation

k : key from DH secret

$\{m\}_k$: authenticated-encryption
of m under key k .

$\llbracket m \rrbracket_{pol}$: prefix encryption of m
under policy 'pol'.

Each end learns its peer's blessing only if it
satisfies its peer's authorization policy

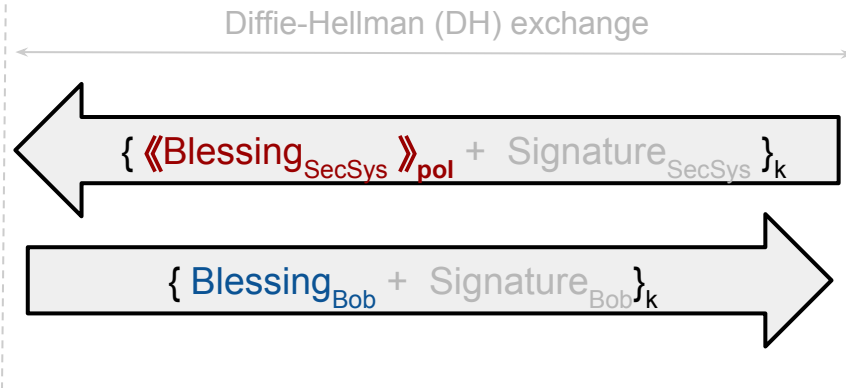
Analysis

Theorem: *Protocol satisfies key-exchange security and mutual identity privacy in the Canetti-Krawczyk model*

Technique may be more generally applicable

- Simply replace server's blessing with prefix-encrypted blessings under the server's policy
- **Future work:** Mutual privacy in TLS 1.3

Unlinkability



Notation

k : key from DH secret

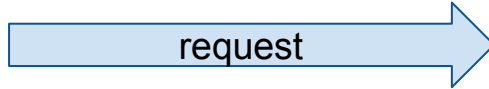
$\{m\}_k$: authenticated-encryption of m under key k .

$\llbracket m \rrbracket_{pol}$: prefix encryption of m under policy 'pol'.

Unlinkability



Blessing: Alice/Family/Bob
Policy: **Allow** Alice/SecSys



Blessing: Alice/SecSys
Policy: **Allow** Alice/Family

Diffie-Hellman (DH) exchange



Linkability Issues:

Signature reveals the server's public key to everyone

Prefix encryption reveals the server's policy to everyone

Notation

k : key from DH secret

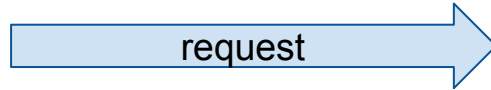
$\{m\}_k$: authenticated-encryption of m under key k .

$\llbracket m \rrbracket_{pol}$: prefix encryption of m under policy 'pol'.

Unlinkability



Blessing: Alice/Family/Bob
Policy: **Allow** Alice/SecSys



Blessing: Alice/SecSys
Policy: **Allow** Alice/Family

Diffie-Hellman (DH) exchange

Linkability Issues:

Signature reveals the server's public key to everyone

Prefix encryption reveals the server's policy to everyone



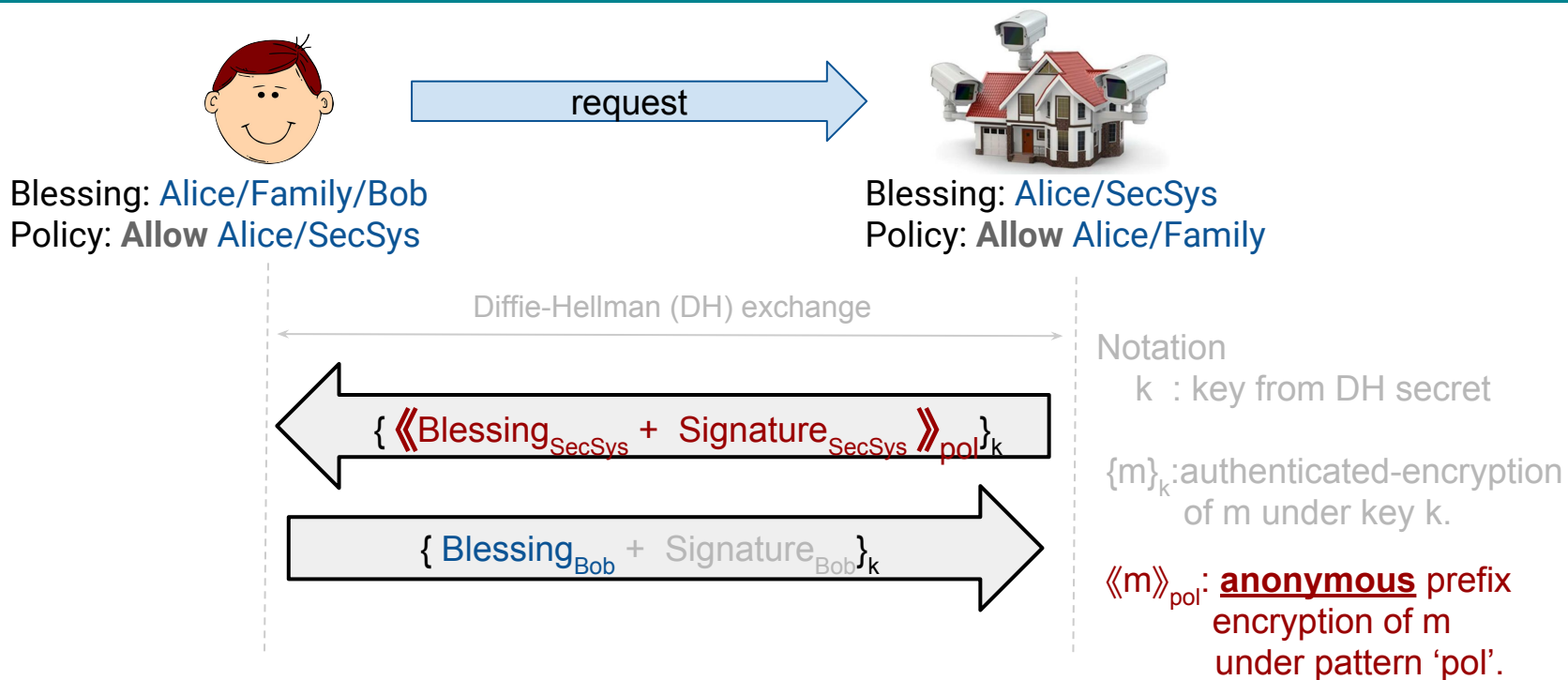
Notation

k : key from DH secret

$\{m\}_k$: authenticated-encryption of m under key k .

$\ll m \rangle\rangle_{\text{pol}}$: **anonymous** prefix encryption of m under pattern 'pol'.

Unlinkability

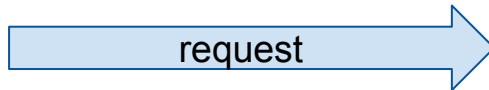


This protocol offers both mutual privacy and unlinkability

Unlinkability



Blessing: Alice/Family/Bob
Policy: **Allow** Alice/SecSys



Blessing: Alice/SecSys
Policy: **Allow** Alice/Family

Server overhead: One IBE Encryption per handshake

Client overhead: One or more IBE Decryptions per handshake

Diffie-Hellman (DH) exchange



Notation

k : key from DH secret

$\{m\}_k$: authenticated-encryption of m under key k .

$\langle\langle m \rangle\rangle_{pol}$: **anonymous** prefix encryption of m under pattern 'pol'.

Efficiency-Unlinkability trade-off



Private Service Discovery

Private and authentic service advertisements

Simple protocol

- Service signs its advertisement and IBEncrypts it under its policy
- Clients then connect to service using private mutual authentication
 - Why private mutual authentication?
 - Answer: To defend against port-scanning attacker

But this is too expensive,

Can we avoid a full blown private mutual authentication?

Idea

« ServiceData + g^s + Blessing_{SecSys} + Signature_{SecSys} »
pol

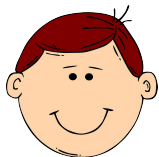


- Service includes a (semi-static) DH share (g^s) in the advertisement
- It signs the advertisement and encrypts it under its prefix policy

This allows clients to authenticate to the server and set up an encrypted channel to it in **zero roundtrips (0-RTT)**

Private Service Discovery

《 ServiceData + g^s + Blessing_{SecSys} + Signature_{SecSys} 》
pol



bid + sid + g^x + { Blessing_{SecSys} + Blessing_{Bob} + Signature_{Bob} }_{k1}

$k1 := \text{KDF}(g^x, g^s, g^{xs}, 1)$

$k2 := \text{KDF}(g^x, g^s, g^{xs}, 2)$

$k3 := \text{KDF}(g^x, g^s, g^{xs}, 3)$

Private Service Discovery

« ServiceData + g^s + Blessing_{SecSys} + Signature_{SecSys} »

pol



bid + sid + g^x + { Blessing_{SecSys} + Blessing_{Bob} + Signature_{Bob} }_{k1}

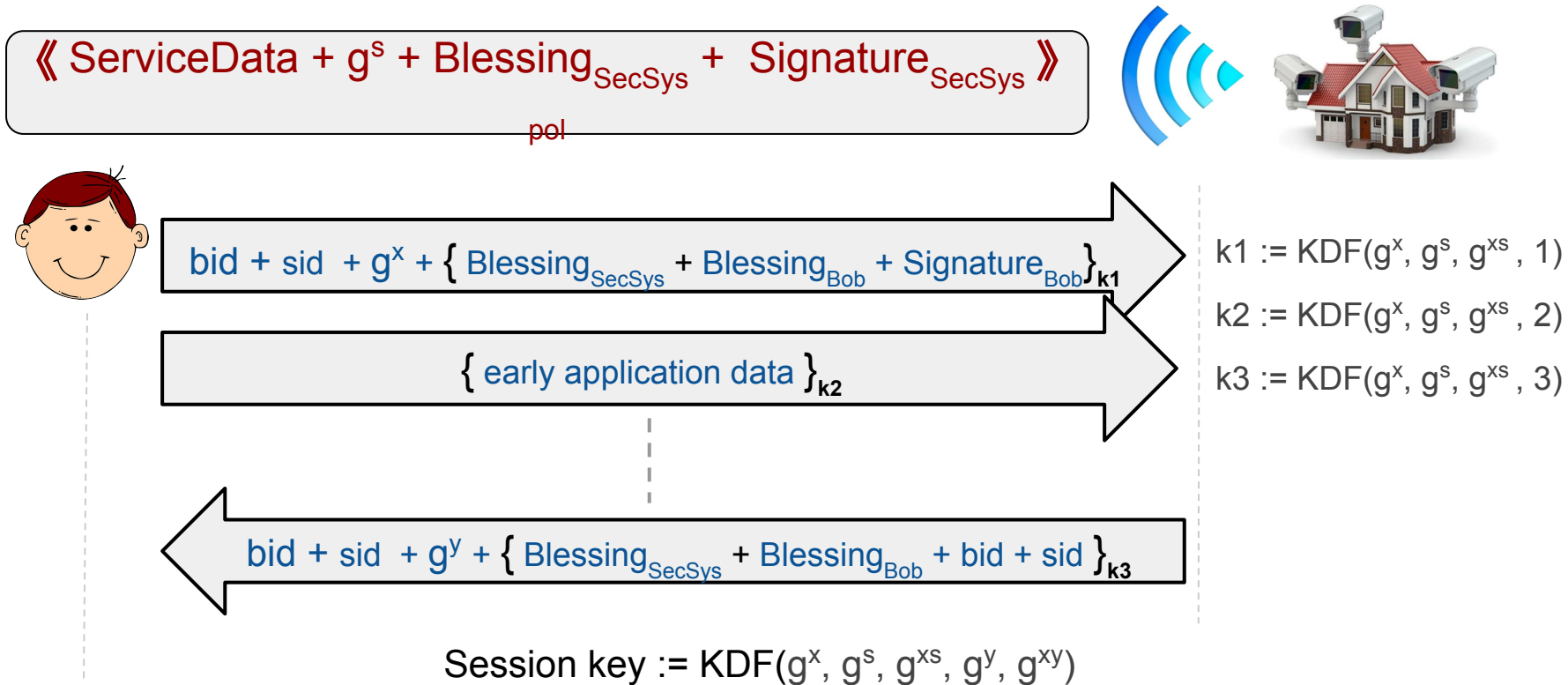
{ early application data }_{k2}

$k1 := \text{KDF}(g^x, g^s, g^{xs}, 1)$

$k2 := \text{KDF}(g^x, g^s, g^{xs}, 2)$

$k3 := \text{KDF}(g^x, g^s, g^{xs}, 3)$

Private Service Discovery



Analysis

Theorem: Protocol satisfies key-exchange security and mutual identity privacy in the Canetti-Krawczyk model

- Private and authentic service advertisements
- 0-RTT mutual authentication

Limitations

- Perfect forward secrecy lost for early application data
- Replay attacks against server's advertisements
 - Add an expiry time to the broadcast
 - Maintain a log of "used" client nonces at the server
 - This is a challenge for all 0-RTT protocols (e.g., TLS 1.3, QUIC)



Implementation and Benchmarks

Implementation

We implemented the following in the Vanadium framework

- Boneh-Boyen (BB1, BB2) IBE schemes over “bn256” pairings along with the Fujisaki-Okamoto transformation for CCA2 security
- Prefix-encryption on top of IBE
- Private mutual authentication protocol (without unlinkability) using prefix encryption
- Private discovery protocol using prefix encryption

IBE benchmarks

	Intel Edison	Raspberry Pi 2	Nexus 5X*	Laptop	Desktop
Pairing	254.5 ms	101.2 ms	219.2 ms	5.0 ms	1.0 ms
Encrypt	406.7 ms	157.2 ms	161.3 ms	8.2 ms	3.5 ms
Decrypt	623.0 ms	235.2 ms	235.7 ms	11.6 ms	3.7 ms
Extract	107.5 ms	41.6 ms	47.2 ms	2.1 ms	0.5 ms

* Unoptimized Go implementation of bn256.

Intel Edison : 0.5GHz Intel Atom processor

RaspberryPi2 : 0.9GHz ARM Cortex-A7

Nexus5X : 3.1GHz Intel Core i7

Private mutual authentication benchmarks

- Run client and server on the same device
- Pre-compute encryption of server blessings under server's policy

	Intel Edison	Raspberry Pi 2	Nexus 5X	Laptop	Desktop
SIGMA-I	252.1 ms	88.0 ms	91.6 ms	6.3 ms	5.3 ms
Private Mutual Auth.	1694.3 ms	326.1 ms	360.4 ms	19.6 ms	9.5 ms
Slowdown	6.7x	3.7x	3.9x	3.1x	1.8x

Intel Edison : 0.5GHz Intel Atom processor

RaspberryPi2 : 0.9GHz ARM Cortex-A7

Nexus5X : 3.1GHz Intel Core i7

Private Service Discovery Benchmarks

bid + \llbracket ServiceData + g^s + Blessing_{SecSys} + Signature_{SecSys} \rrbracket

pol

Advertisement processing time

1 IBE Decryption + 1 ECDSA verification

E.g., on a Nexus 5x: 236ms + 11ms = 247ms

Private Service Discovery Benchmarks

bid + \llbracket ServiceData + g^s + Blessing_{SecSys} + Signature_{SecSys} \rrbracket

pol

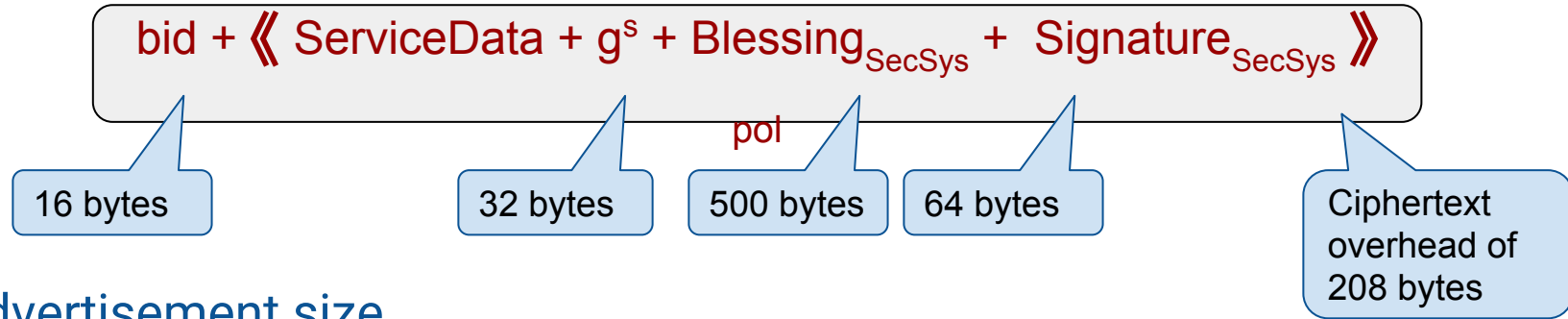
Advertisement size

- 820bytes for a single policy pattern
- $628 + 240 \cdot n$ when there are n patterns

But, mDNS packet limit is 1300 bytes and BLE is 30 bytes

- Start an auxiliary service to host the advertisement
- Advertise the endpoint of this auxiliary service over BLE or Wifi

Private Service Discovery Benchmarks



Advertisement size

- 820bytes for a single policy pattern
- $628 + 240 \cdot n$ when there are n patterns

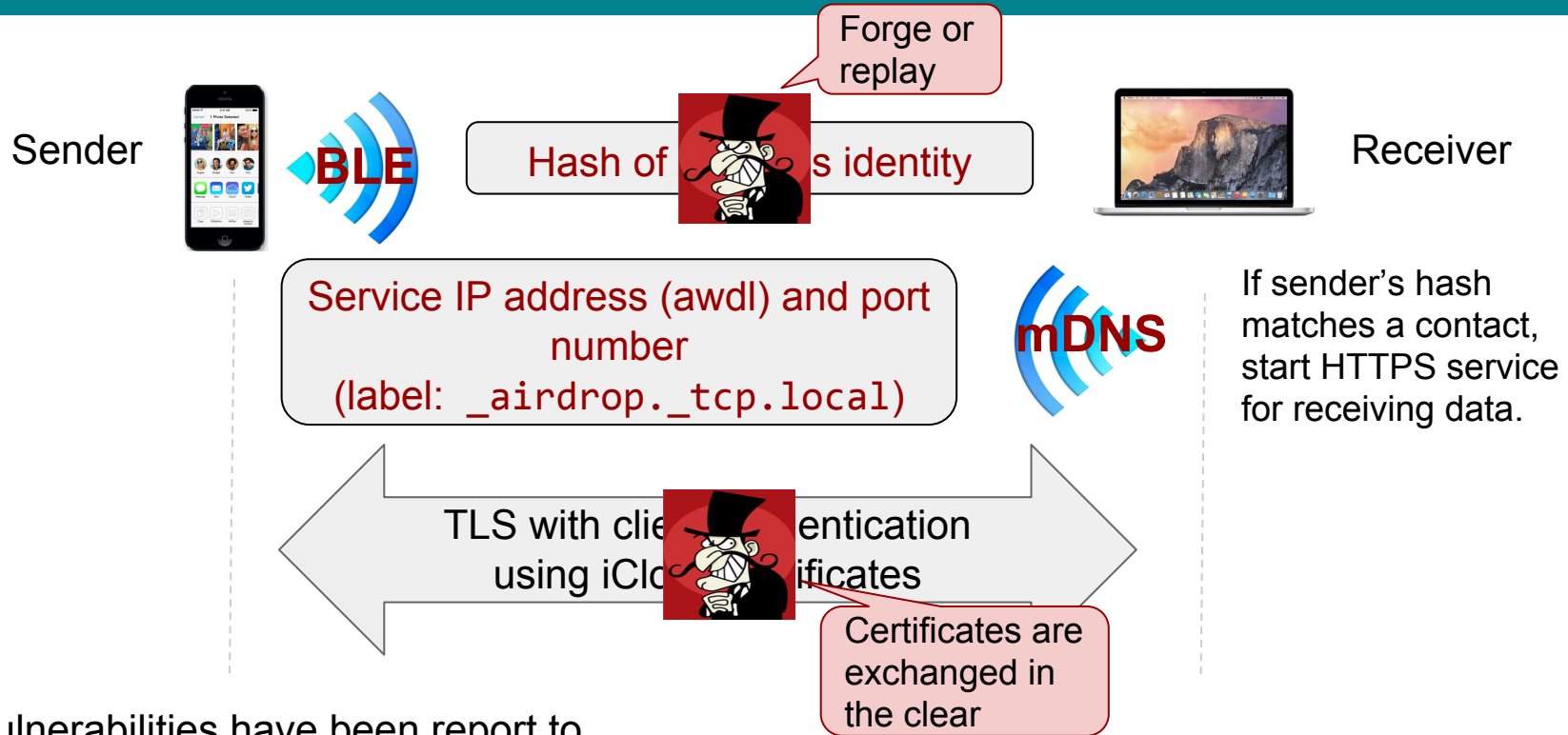
But, mDNS packet limit is 1300 bytes and BLE is 30 bytes

- Start an auxiliary service to host the advertisement
- Advertise the endpoint of this auxiliary service over BLE or Wifi



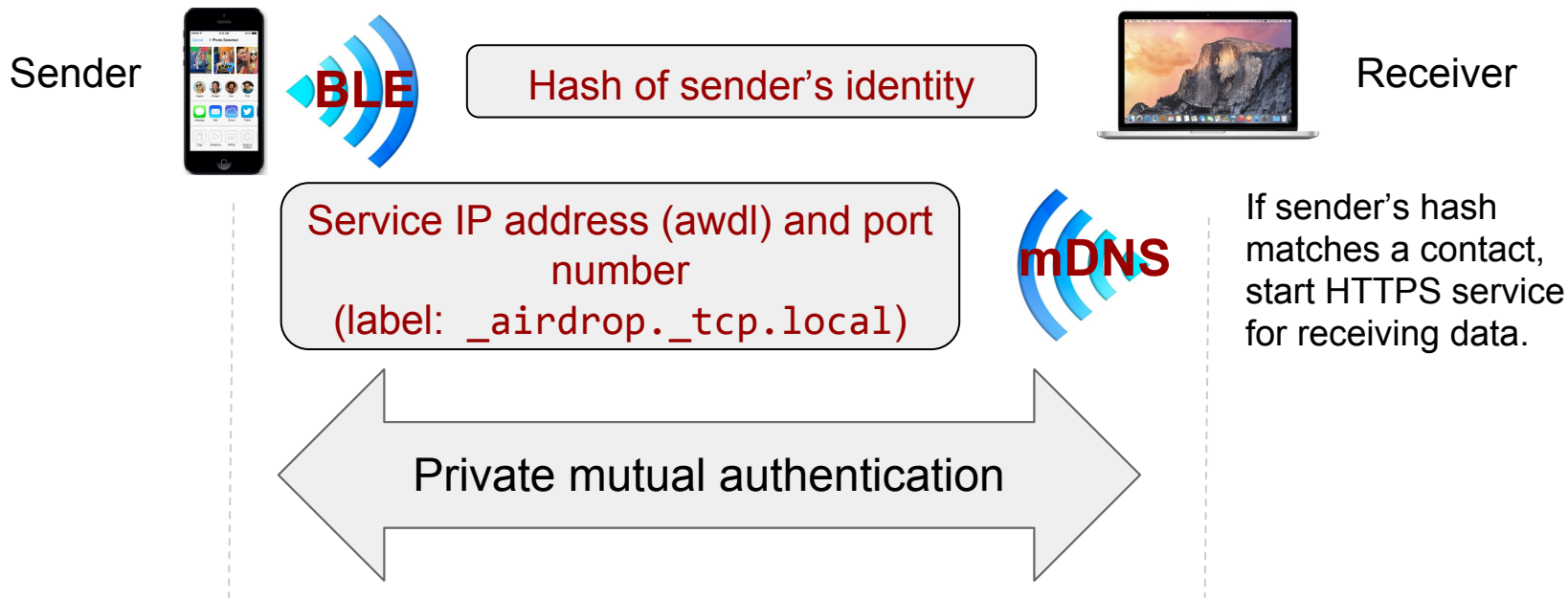
Fixing Apple AirDrop

AirDrop (contacts-only mode) vulnerabilities



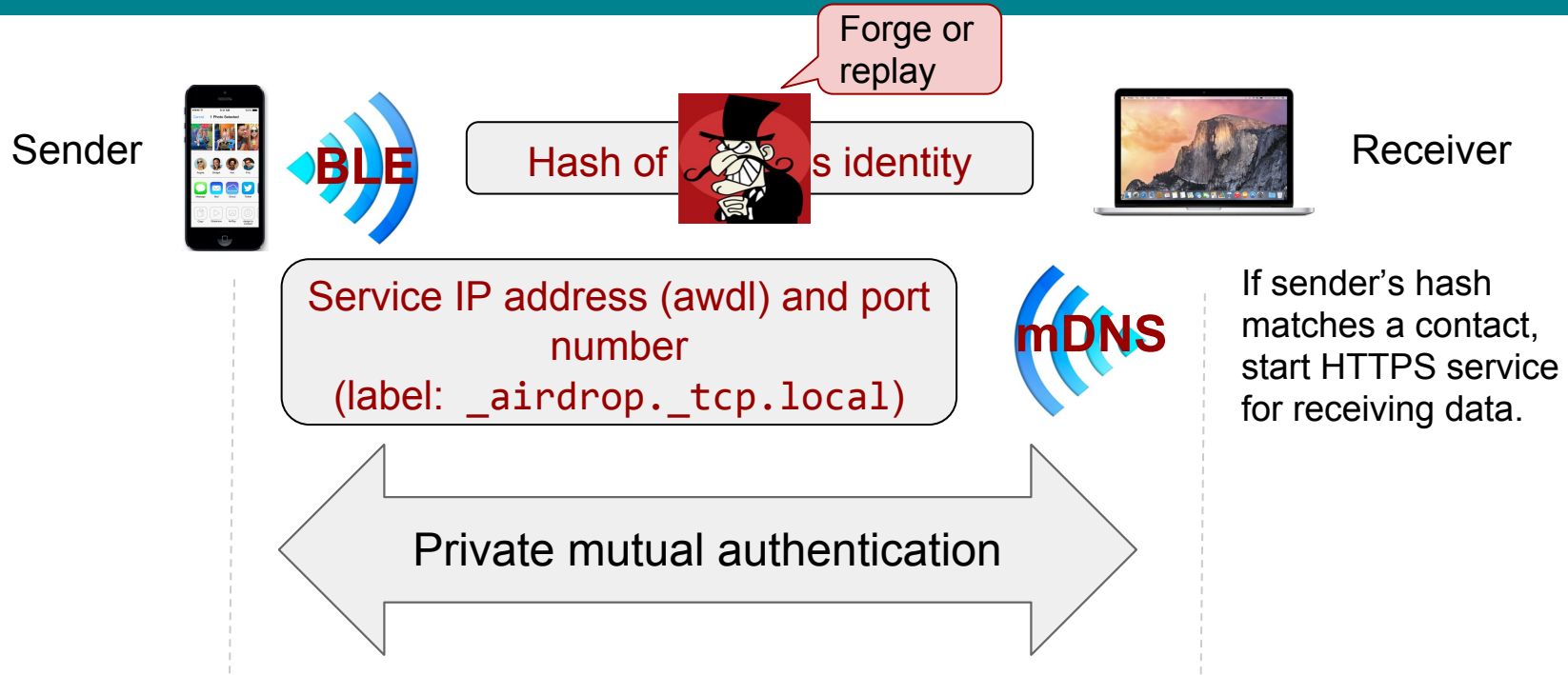
Vulnerabilities have been reported to Apple, awaiting response

Fixing AirDrop (contacts-only mode)



Fix reported to Apple, awaiting response

Fixing AirDrop (contacts-only mode)



Fix reported to Apple, awaiting response



Practicalities

Revocation

How do we revoke Bob's prefix-encryption key?

Idea: Include a timestamp in the prefix pattern

- e.g., encrypt under *Alice/Family/<2 Sept 2016>*
- Bob will exchange his blessing for a new private key every day
- IBE root will stop issuing keys once Bob is revoked

Revocation

But even after revocation, old ciphertexts can be decrypted by Bob!

Idea: Include a unique nonce in the prefix pattern for each ciphertext

- e.g., encrypt under **Alice/Family/<nonce>**
- Bob will have to obtain a new private key for each ciphertext
- IBE root will stop issuing keys once Bob is revoked
- After revocation, Bob cannot decrypt ciphertexts he hasn't seen before

Revocation

What if the IBE master private key needs to be revoked ??



No easy solution :-)

- Rotate master private key and associated public params
- Inform all clients about the new params

Private key escrow

IBE Root can generate private keys for any identity and therefore decrypt all ciphertexts

Approaches

- Double encryption [Gentry, Eurocrypt 2003]
- Each user is an IBE Root
 - Need key servers to distribute params :-)
 - Can still use IBEExtract locally to delegate keys

Perspective

- Privacy is a growing concern for the Internet of Things
- Privacy issues in authentication and discovery are still largely ignored
- IBE is far more practical than before, thanks to improved crypto algorithms and better hardware
- Still several trade-offs in the design, lots of opportunities for research

Privacy, Discovery, and Authentication for the Internet of Things,
David Wu, Ankur Taly, Asim Shankar, Dan Boneh [ESORICS 16]

Further reading

Private Authentication --- Abadi and Fournet, 2004

Secret handshakes from pairing-based key agreements --- Balfanz et al., 2003

SIGMA: The “SIGn-and-MAC” approach to authenticated Diffie-Hellman and its use in the IKE-protocols --- Krawczyk, 2003

Questions



email: ataly@google.com



Thank You!
