Tryst: The Case for Confidential Service Discovery

Jeffrey Pang, CMU
Ben Greenstein, Intel Research
Damon McCoy, University of Colorado
Srinivasan Seshan, CMU
David Wetherall, University of Washington, Intel Research

Local service discovery (SD)

Used to find:
- 802.11 networks
- consumer electronics
- local services
- other applications

How SD is done today

iTunes "Bob" is here — Connect to "Bob"

Is iTunes "Bob" here? — "Bob" is here

Services send **announcements** and/or clients send **probes**
(typically via unencrypted broadcast)

Important properties:
- *Plug-and-play networking*
  - Can proceed automatically without user input
- *Disconnected operation*
  - Requires only communication medium between client and service

Problem 1: SD reveals inventory

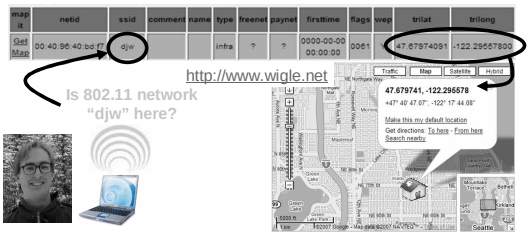
- The devices I have
  Problem: "Phone pirates in seek and steal mission" [Cambridge Evening News 2005]
- The applications I am running
  Problem: "Apple Mac OS X mDNSResponder buffer overflow vulnerability" [CERT 2007]

Problem 2: SD reveals identity

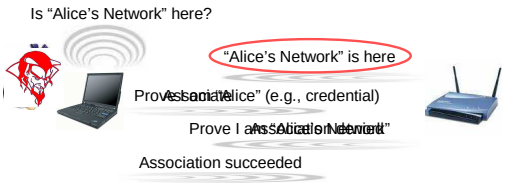
- Announcements expose explicit identifiers
  Problem: Some services are private and want to be hidden
  Problem: Mobile "services" vulnerable to tracking

Problem 3: SD reveals history

- Probes can reveal services you have used
  Problem: Network names can be correlated with location (e.g., using a wardriving database)



Problem 4: SD is not authenticated



- Authentication occurs only after SD
  - **Problem**: Anyone can elicit a response, even if they are not trusted to access the service

Solution requirements

Desired properties:

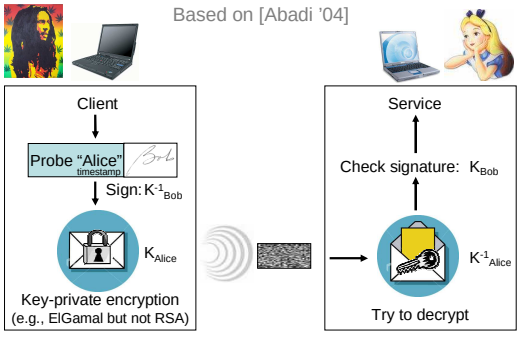
- *Confidentiality*
  - Hide contents that reveals type
  - Hide sender
  - Hide intended recipients
- *Authenticity*
  - Offer proof of identity

Challenges:

- Exposing information *only* to trusted clients/services
- Clients *and* services may want to hide from third parties
- Plug-and-play networking, disconnected operation

A public key protocol

Based on [Abadi ’04]



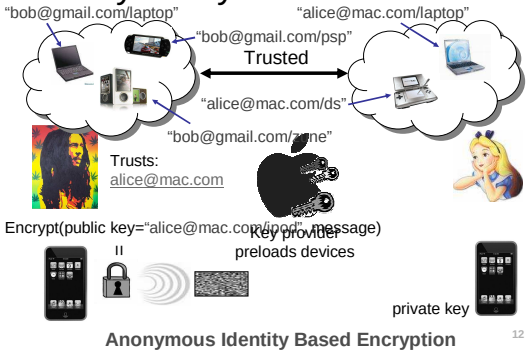
Problems

- Must obtain public keys for new services/clients
  - May be disconnected during discovery
  - Don’t want to involve extra user action
- Must try to decrypt *every* message
  - Public key decryption is slow (>100ms on typical AP)
    ⇒ adds jitter to relaying of other messages
  - 168x slower than 802.11 line-rate even on laptops
    ⇒ susceptible to low-rate denial-of-service attacks

Observations

- Must obtain public keys for new services/clients
  - Will have some relationship with those we trust
  - Can trust new services/clients in trusted domains
- Must try to decrypt *every* message

Tryst: key establishment



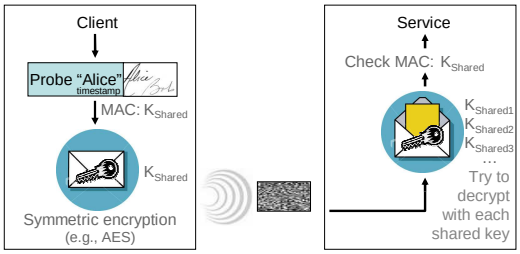
Anonymous Identity Based Encryption

Observations

- Must obtain public keys for new services/clients
  - Will have some relationship with those we trust
  - Can trust new services/clients in trusted domains
- Must try to decrypt *every* message
  - Common case is to *re*discover known services
  - Can negotiate a secret symmetric key the first time

13

Tryst: a symmetric key protocol



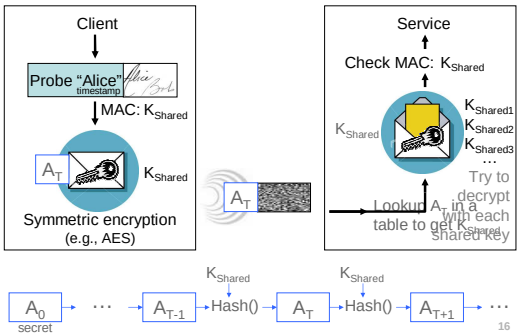
14

Observations

- Must obtain public keys for new services/clients
  - Will have some relationship with those we trust
  - Can trust new services/clients in trusted domains
- Must try to decrypt *every* message
  - Common case is to *re*discover known services
  - Can negotiate a secret symmetric key the first time
  - Linkability at short timescales is usually OK
  - Can use temporary *unlinkable* addresses

15

Tryst: a symmetric key protocol



16

Conclusions

- Ubiquitous clients *and* services may want privacy
- Service discovery exposes inventory, identity, and history
- Tryst helps enable *confidential* service discovery
- Outstanding challenges
  - Hiding other side channels (physical layer, message timing, size)
  - Other mechanisms to automatically establish keys
  - More efficient broadcast probes

17

Backup Slides

18

Other key establishment methods

- Certificates (e.g., secure websites)
  - Neither client nor service can offer certificate first!
- Pairing (e.g., Bluetooth peripherals)
  - Can not always physically identify service
  - *User* must perform discovery before *device* does!
- Discovery is also used to find *unknown* services
  - We want to automatically expand the trust horizon
  - Tryst mechanisms:
    - New services in trusted domains
    - New services trusted transitively

Anonymous identity based encryption

[Boneh & Franklin '01]

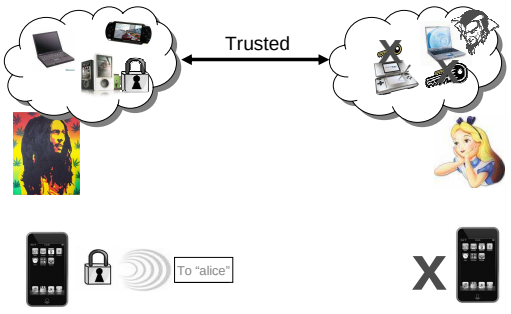
PublicParams, MasterSecret = **Setup** ()
(publicly published; known only by key provider)

ciphertext = **Encrypt** ($K_{AliceiPod}$, PublicParams, plaintext)
||
"alice@mac.com/ipod"

plaintext = **Decrypt** ($K^{-1}_{AliceiPod}$, PublicParams, ciphertext)
||
Extract("alice@mac.com/ipod", MasterSecret)

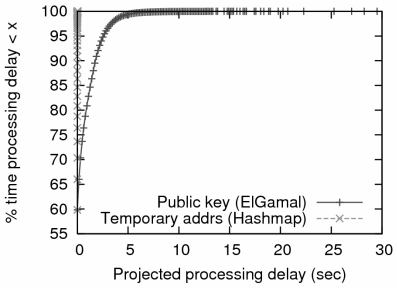
- Some assumptions over traditional public key crypto
  - *Alice and Bob trust key provider not to reveal secret keys to third parties*
    - Can instead trust that no t of n providers collude (use threshold crypto)
    - May also be able act as their own key providers (anonymity unproven)
  - *Revoking my public key implies changing my identity since identity = key*
    - Can instead use temporary identities ("alice@mac.com/ipod.nov.2007")
- Only need to use protocol until first discovery

Tryst: key establishment



Strawman Solution

Projected probe processing delay



Rough simulation based on:

- SIGCOMM 2004 (wifi probes)
- Soekris "AP" (265Mhz Geode)
- GnuPG crypto (~150ms/pkt)

Protocol message volume

- Each message encrypted for one recipient

⇒ As many messages as intended recipients
- Typically OK: E.G., 90% of WiFi clients probe for fewer than 12 unique network names [OSDI 2006 WiFi trace]

- Future work:
  - Efficient protocol to broadcast probes to many recipients

Key problem: messages can be linked

- Consistent naming enable correlation of SD messages
- Opaque names prevent some problems…
  but not all:
  - Example: location can be correlated with other databases

Related work

- **SmokeScreen** [Cox '07] – access control for discovering friends
  - Similar to symmetric key protocol
  - Uses online social network to exchange secret keys
- **SSDS** [Czerwinski '00] – secure service discovery architecture
  - Relies on trusted infrastructure
  - Not necessarily confidential
- **Broadcast Encryption** [Fiat '93] – encrypt message to many users
  - Making this private is an open problem
- **JFK** [Aiello '93] – efficient Internet key exchange
  - No service privacy …
  - … *or* not resilient to man-in-the-middle attacks

25

SD is widely used

- **Example 1**: Application Protocols (OSDI 2006)

Protocol Name	Devices (%)
NetBIOS	530 (69.9)
mDNS	270 (35.6)
UPnP SSDP	260 (34.3)
SLP	106 (14.0)
Microsoft Office v.X	50 (6.6)
Internet Printing Protocol	8 (1.1)
TiVo Beacon Protocol	8 (1.1)
Dantz Retrospect	8 (1.1)
LiveTribe SLP	6 (0.8)
BakBone NetVault	4 (0.5)
Other	31 (4.1)
Any Protocol	685 (90.4)

- **Example 2**: **85%** devices send WiFi discovery probes (SIGCOMM 2004)

26

Problem 3: SD reveals history

- Probes can reveal services you have used

  Problem: Network names can be correlated with location (e.g., using a wardriving database)

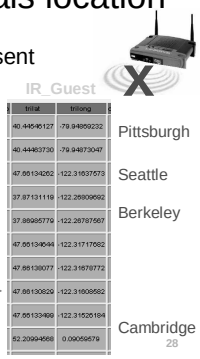
23% of devices at SIGCOMM 2004 probed for an name that WiGLE isolates to one city

All 4 known home networks located to within 2 blocks

27

Problem 5: SD reveals location

- The fact that my service is present
  - **Problem**: Common practice to disable WiFi annoucements to (try to) hide access points [O'Reilly 802.11 Guide]
- Where my service is located
  - **Problem**: Knowledge of network name at one site can tell you where other sites are [WiGLE Wardriving Database]



28

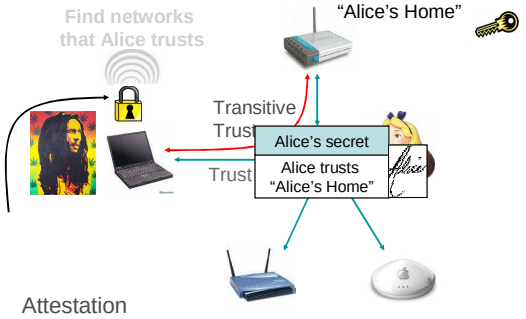
Problem 6: SD reveals social contacts

- Emerging social devices also offer "services"
  - **Microsoft Zune**: music sharing service
  - **PSP, Nintendo DS**: multiplayer gaming service

- Service discovery exposes social contacts

29

New services transitively trusted

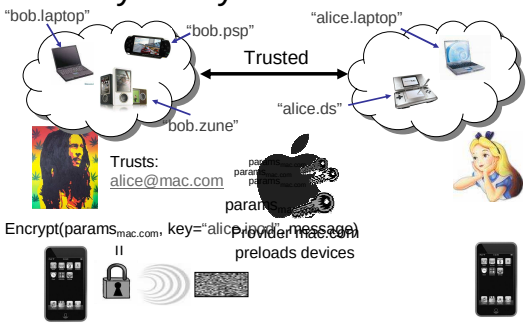


30

Old Slides

31

Tryst: key establishment

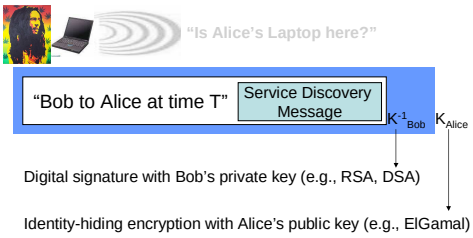


Anonymous Identity Based Encryption

32

Public Key Protocol

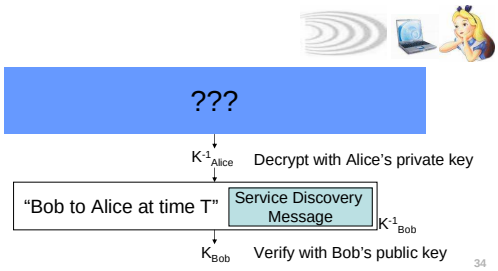
- Existing theoretical public key protocol [Abadi '04]



33

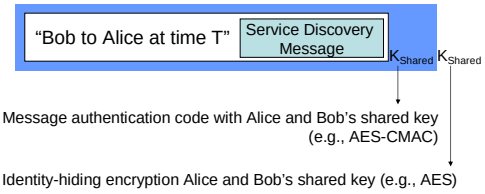
Public Key Protocol

- Existing theoretical public key protocol [Abadi '04]



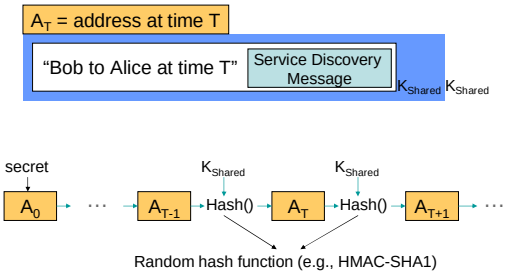
34

Symmetric Key Protocol



35

Symmetric Key Protocol



36