Privacy-preserving Policies, Protocols and Architectures

Sotiris Ioannidis
sotiris@ics.forth.gr

The Problem

- We operate in highly-connected, multi-application, multi-device environments
- We want to control flow of information in a consistent fashion
- We want to agree on what information we want to exchange and how
- We want to do all this without loss of privacy
- There is a semantic gap between policy and mechanism

This Talk

1. How multiple devices, applications, etc. can enforce the same privacy policy
2. How we can agree on policies and data exchanges in a privacy-preserving fashion

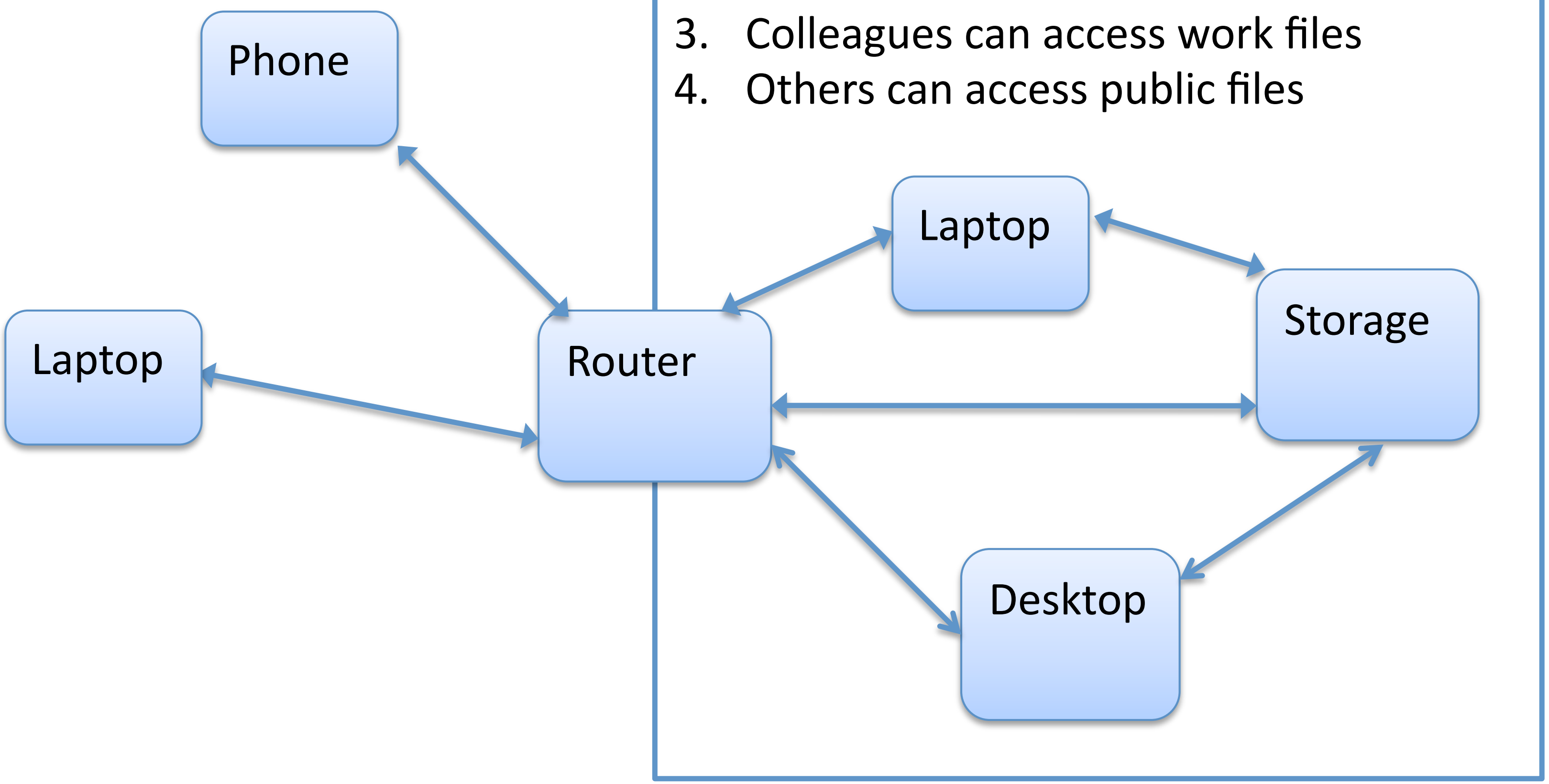
Definitions/Framework

- Policy [Lam71, Slo94, SV00]
  - A set of rules that express what actions are allowed/not allowed to happen in a system - *e.g.* foo is allowed to see bar
- Policy specification [WL93, Slo94]
  - Expression of the rules which is independent of where or how they are evaluated - *e.g.* doctors are permitted to access my_medical_data
- Policy evaluation [DDLS01, IBS01]
  - Actual interpretation of a rule when it gets triggered by an action
- Mechanism [DDLS01, IBS01]
  - The element that interprets rules when they get triggered by user actions
- Equivalence [I05]
  - The same policy rule evaluates to the same result on different mechanisms
- Consistency [I05]
  - There are no cases of policy rules not being equivalent

Example

Policy for home network:

1. Owner can access all files
2. Friends can access private files
3. Colleagues can access work files
4. Others can access public files



Enforcement of the same Policy

- Under certain (common) conditions (discussed later), policy consistency is possible *without* solving the halting problem
- Approach:
  - We must find inconsistencies that occur between the *policy specification* level and the *mechanism* level!

Why should we care?

- Maintaining *correct* policy in decentralized environments is an important, complex, and challenging task
  - It's what most of today's environments are like
  - Increased number of hosts, applications, users, *etc*.
  - Diverse hosts, applications, users, *etc*.
  - Configuration changes over time
    - Even if initially correct it might progressively get out of sync
- *Correct* can mean a number of things:
  - Reflects policy maker's intention, is conflict free, is consistent (what we are interested in), *etc*.

Policy Consistency

- What it is:
  - Gap between policy specification level and mechanism level
    - Rules refer to abstract policy objects and are enforced on application specific objects
  - Possible discrepancies
    - Mismappings between policy objects and managed objects
  - *The same policy gives the same result on every mechanism*
    - A subject forbidden to perform an action on some node, should not be allowed to perform it on any other node
    - It is actually not necessary to solve the halting problem!!
- What it is not:
  - Not trying to prove program equivalence

Previous Work

- Piecemeal configuration (FW, compartmented FS, *etc*.)
- Single security policy language, decentralized mechanism (Ponder, SPL, KeyNote, *etc*.)
- Policy conflicts ([SPH88, LPGSF90, JSS97, LS99, *etc*.])
  - Hierarchy, narrowness, priority, modality, *etc*.
  - Not what we are interested in

Consistency, Assumptions:

- Single security policy that governs the systems
  - Abstractly defines rules about resources
- *M* mechanisms responsible for policy evaluation
  - Heterogeneous and distributed
  - Each mechanism uses its own representation of resources
  - Each mechanism is implemented correctly (no bugs)
- Resource identities exist uniquely
- These are realistic assumptions, that’s how today's systems are architected

Consistency Checking, Basic Idea:

- Policy rules define whether a subject is allowed to perform an action on a target
  - *e.g.* a tuple <someuser, someaction, someresource>
- Policy language refers to high-level object abstractions
  - *e.g*. TrustedUsers, PrivateFiles, *etc.*
- Abstractions map to application/OS/*etc.* specific objects
  - *e.g.* root, medicalrecords.pdf, *etc*.
- Application/OS/*etc.* objects map to real objects
  - *e.g.* some person, some file on disk, *etc.*
- Given such rules and mappings, for every mechanism, exhaustively explore the state space for inconsistencies

Overview of Consistency Algorithm:

Step 1:

```
for (all mechanisms)
    for (all rules)
        for (all mappings between
             policy and mechanism)
            expand and create tuple
```

Step 2:

```
for (every pair of mechanisms)
    for (every pair of tuples)
        compare
```

- Correctness theorem:
  - Finds inconsistencies in policy by identifying all rules for which their evaluation on different mechanisms gives different results
- Proof:
  - Exhaustive search
- Complexity:
  - O((|mechanisms|*|tuples|)^2) where |tuples| is a function of the |rules| and |objects|

Consistency Example 1

Policy:

Clinic heads are allowed complete access to the patient files
<Clinic Head, Complete Access, Patient Files>

- Clinic 1 server:
  - Clinic Head : root -> Alice
  - Complete Access: r/w -> Read/Write
  - Patient Files: /remote/recs -> Records

Consistency Example 2

Policy:

Doctors are allowed partial access to the patient files

<Doctor, Partial Access, Patient Files>

Clinic heads are allowed complete access to the patient files

<Clinic Head, Complete Access, Patient Files>

- Clinic 2 server:
  - Doctor: alice -> Alice
  - Clinic Head: root -> Bob
  - Partial Access: r -> Read
  - Complete Access: r/w -> Read/Write
  - Patient Files: recs-> Records

Consistency Example 3

Policy:

Doctors are allowed partial access to the patient files
<Doctor, Partial Access, Patient Files>
Clinic heads are allowed complete access to the patient files
<Clinic Head, Complete Access, Patient Files>

- Clinic 1 server:
  - Clinic Head: root -> Alice
  - Complete Access: r/w -> Read/Write
  - Patient Files: /remote/recs -> Records
- Clinic 2 server:
  - Doctor: alice -> Alice
  - Clinic Head: root -> Bob
  - Partial Access: r -> Read
  - Complete Access: r/w -> Read/Write
  - Patient Files: recs-> Records

Consistency Example 3, cont.

Policy:

Doctors are allowed partial access to the patient files

<Doctor, Partial Access, Patient Files>

Clinic heads are allowed complete access to the patient files

<Clinic Head, Complete Access, Patient Files>

- Clinic 1 server:
  - <root, r/w, /remote/recs>
    - Alice, Read/Write, Records
- Clinic 2 server:
  - <alice, r, recs>
    - Alice, Read, Records
  - <root, r/w, pass>
    - Bob, Read/Write, Records

Consistency Example 3, cont.

Policy:

Doctors are allowed partial access to the patient files
<Doctor, Partial Access, Private Files>
Clinic heads are allowed complete access to the patient files
<Clinic Head, Complete Access, Private Files>

- Clinic 1 server:
  - <root, r/w, /remote/recs>
    - Alice, **Read/Write**, Records
- Clinic 2 server:
  - <alice, r, recs>
    - Alice, **Read**, Records
  - <bob, r/w, recs>
    - Bob, Read/Write, Records

Policy inconsistency

How “expensive” is it to use?

- O((|mechanisms|*|tuples|)^2) where |tuples| is a function of the |rules| and |objects|
- It's best to run incrementally
  - Adding a new mechanism: O(|mechanisms|*|tuples|^2)
  - Adding/modifying a new rule: only compare the newly generated tuples
  - Adding/modifying an object/mapping: only compare the newly generated tuples

How “practical” is it to use?

- Our tool can potentially generate a large number of potential inconsistencies
- However:
  - It's better to have some idea of what problems your system may have
  - If the list is not “gigantic” then we can fix the inconsistencies
  - It's a first step towards automation – use resolution heuristics
  - As we pointed out before, it's best run incrementally

Summary so far

- Results and Contributions
  - Framing of the problem of policy consistency
    - Bridging the semantic gap between policy and mechanism
    - Departure from the notion of policy conflicts
  - Procedure for determining policy consistency on heterogeneous systems
    - Methods for assisting policy writers to debug policies

Open Directions

- Multiple administrative domains
- Error reporting and recovery
- Preservation of intent
- Solve the halting problem

Privacy-preserving Policy Reconciliation

- Reconcile policies between multiple parties

Motivation: Mobile Communications

- Network provider
  - Protect their network
  - Support legacy devices
- User
  - Maximize battery life
  - Use the network
- (partially) Conflicting preferences

Motivation: Corporate Policy

- Organizations
  - List of clients
  - Types of data
  - Types of users
  - Protocols
- Secret attributes
- Don't want to disclose policy unless both parties have common preferences

Problems in Policy Reconciliation

- Unsolvable in the general case
  - Parties must have common representation
  - Efficient solutions do exist for some representations O(nlogn)
- Participants release their complete policy
  - Disclose policy preferences
  - Disclose policy attributes
- Exposes too much about the participants
- What can we do?

Privacy in Policy Reconciliation

- *It is possible to guarantee privacy in policy reconciliation*

Policy Representation

- Assume participants use the same format
- Represent policy rules as bit-strings
- They represent which attributes are defined and which are not
- Policy is a set of rules; forms a matrix
- Policy rules can be ordered to express preference

Schedule Policy Example

	Applicant					HR				
	Mo	Tue	Wed	Thu	Fri	Mo	Tue	Wed	Thu	Fri
Rule1	1	0	0	0	0	0	0	1	0	0
Rule2	0	1	0	0	0	0	1	0	0	0
Rule3	0	0	1	0	0	1	0	0	0	0
Rule4	0	0	0	1	0	0	0	0	1	0
Rule5	0	0	0	0	1	0	0	0	0	1

Threat Model

- Semi-honest - Participants play nicely
  - Follow the protocol
    - They don't gain info about each others private data
    - They only see the output of the protocol
- Malicious - Participants don't play nicely
  - Can behave arbitrarily:
    - refuse to participate, give bogus data, abort protocol
    - They don't gain info about each others private data
    - They only see the output of the protocol

Tools

- Homomorphic cryptosystem
- Given E(a) and E(b) I can calculate E(a+b)
  - Efficiently – without breaking the cryptosystem!
- Given r and E(a) I can calculate E(r*a)
  - Efficiently – without breaking the cryptosystem!
- Which means that given the encrypted coefficients of a P(x), and y and z, I can efficiently calculate E(y*P(x) + z)

Privacy Goals

- Privacy-preserving policy without preference
  - Cardinality: Returns number of common policies
  - Common Policy: Returns the policies
- Privacy-preserving policy with preference
  - Sum of Ranks: maximizes joint preference order
  - Maximized Ranks: maximizes each ones ranks

Cardinality

$$(0) A, B : \gamma$$

$$(1) A : f_A(X) = (X - a_1)(X - a_2)....(X - a_k) = \sum_{i=0}^{k} \alpha_i X^i$$

$$(2) B : f_B(X) = (X - b_1)(X - b_2)....(X - b_k) = \sum_{i=0}^{k} \beta_i X^i$$

$$(3) A : E_A(\alpha_i) \rightarrow B$$

$$(4) B : E_B(\beta_i) \rightarrow A$$

$$(5) A : E_B(r_i^{'} f_B(\alpha_i) + \gamma) \rightarrow B$$

$$(6) B : E_A(r_i f_A(\beta_i) + \gamma) \rightarrow A$$

Common Policy

$$(1) A : f_A(X) = (X - a_1)(X - a_2)....(X - a_k) = \sum_{i=0}^{k} \alpha_i X^i$$

$$(2) B : f_B(X) = (X - b_1)(X - b_2)....(X - b_k) = \sum_{i=0}^{k} \beta_i X^i$$

$$(3) A : E_A(\alpha_i) \rightarrow B$$

$$(4) B : E_B(\beta_i) \rightarrow A$$

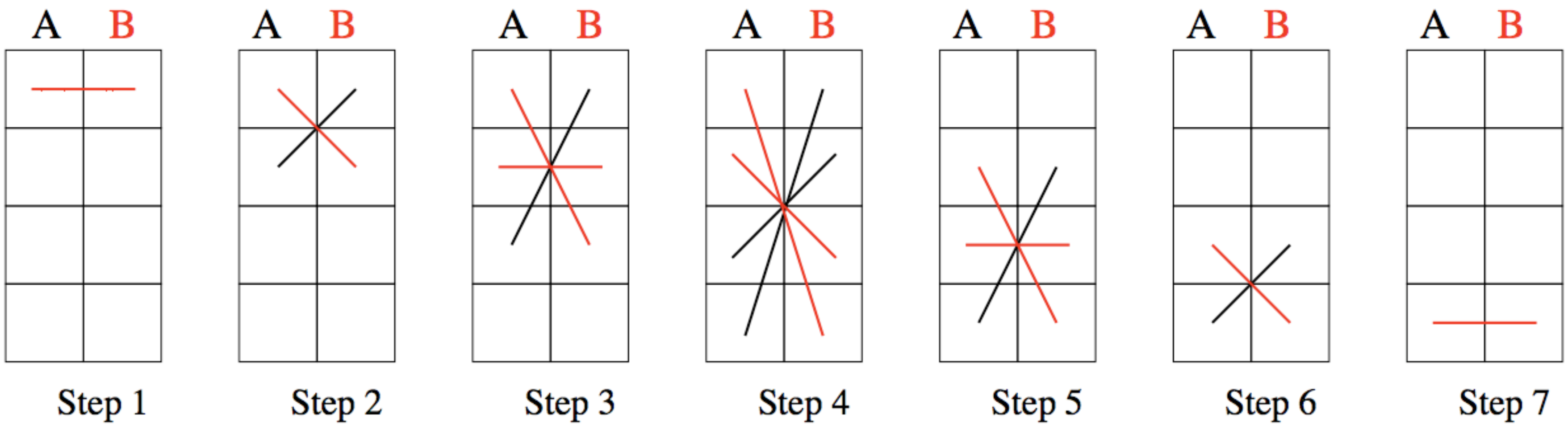
$$(5) A : E_B(r_i^{'} f_B(\alpha_i) + \alpha_i) \rightarrow B$$

$$(6) B : E_A(r_i f_A(\beta_i) + \beta_i) \rightarrow A$$

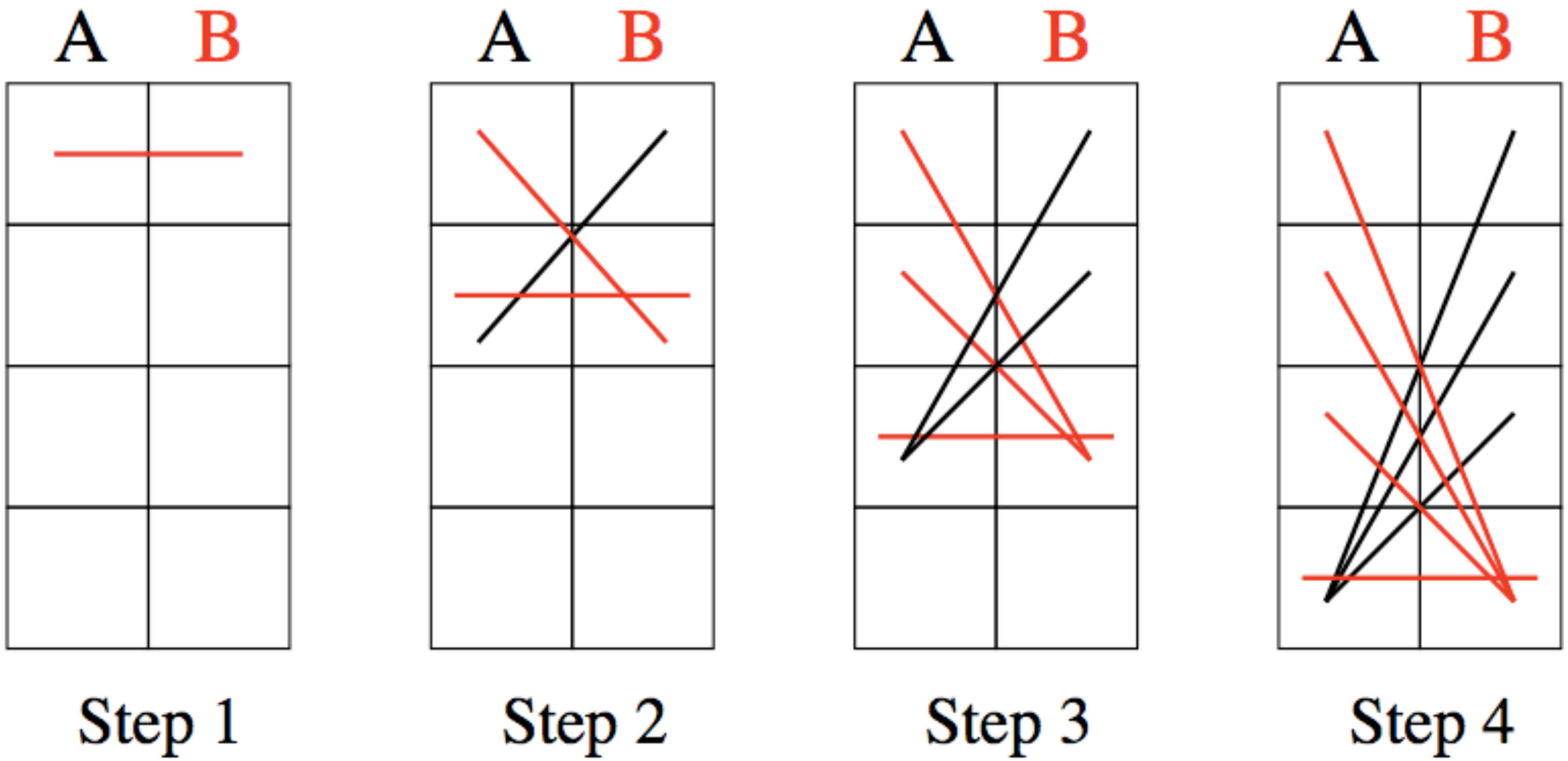
Schedule Policy (Again)

	Applicant					HR				
	Mo	Tue	Wed	Thu	Fri	Mo	Tue	Wed	Thu	Fri
Rule1	1	0	0	0	0	0	0	1	0	0
Rule2	0	1	0	0	0	0	1	0	0	0
Rule3	0	0	1	0	0	1	0	0	0	0
Rule4	0	0	0	1	0	0	0	0	1	0
Rule5	0	0	0	0	1	0	0	0	0	1

Sum of Ranks



Maximized Ranks



Summary so far

- It is possible to do privacy-preserving policy reconciliation
- Participants can privately:
  - Discover *if* they have common policies
  - Discover *only* the common policies
  - *Select* a policy according to their *preferences*

Open Directions

- Other privacy-preserving operations on policy
- Other types of ranking/preferences
- Different representation for efficiency