SecureBlox: Customizable Secure Data Processing

William R. Marczak* , Shan Shan Huang[†], Martin Bravenboer[†],
Micah Sherr[‡], Boon Thau Loo[‡], Molham Aref[†]

*University of California, Berkeley [†]LogicBlox, Inc [‡]University of Pennsylvania

Introduction

- Many large-scale networked information systems
- Security is hard because:
  - Depends on execution environment
    - Administrative boundaries
    - Assumptions of attacker's capabilities
    - Computation/bandwidth constraints
  - Need reconfigurability
  - Security is cross-cutting
- No *one-size-fits-all* set of constructs

Our Solution: SecureBlox

- SecureBlox: an extensible distributed query processor built on top of a Datalog engine
  - Applicable to any Datalog engine
  - We use the commercial LogicBlox engine
- Security community uses Datalog (Abadi et al, DeTreville et al, etc.)
- High-level declarative recursive query languages: a promising new framework for distributed systems
  - Declarative Networking (Loo et al)
  - Cloud Computing (Alvaro et al)
  - etc.

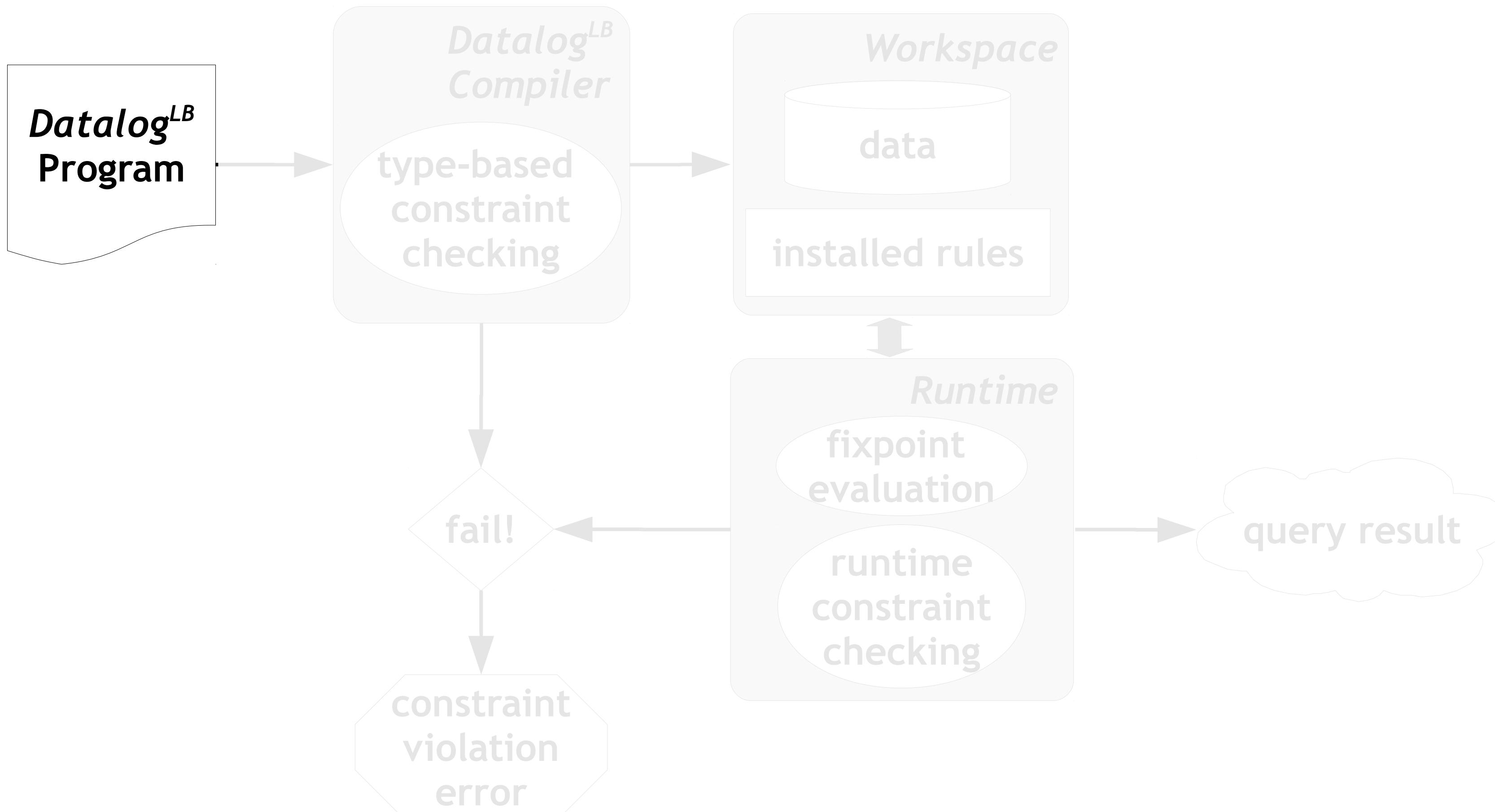
Our Solution: SecureBlox

- Keep application & security languages unified
  - Improves program understanding
  - Formal reasoning
- Elegant decoupling of security logic from application logic
  - Facilitates highly reconfigurable security
  - Programmers focus on “what” properties to enforce, rather than “how” to instrument their code
  - Security specified as automatic rewrites of application logic
  - Integrity constraints express security invariants

Outline

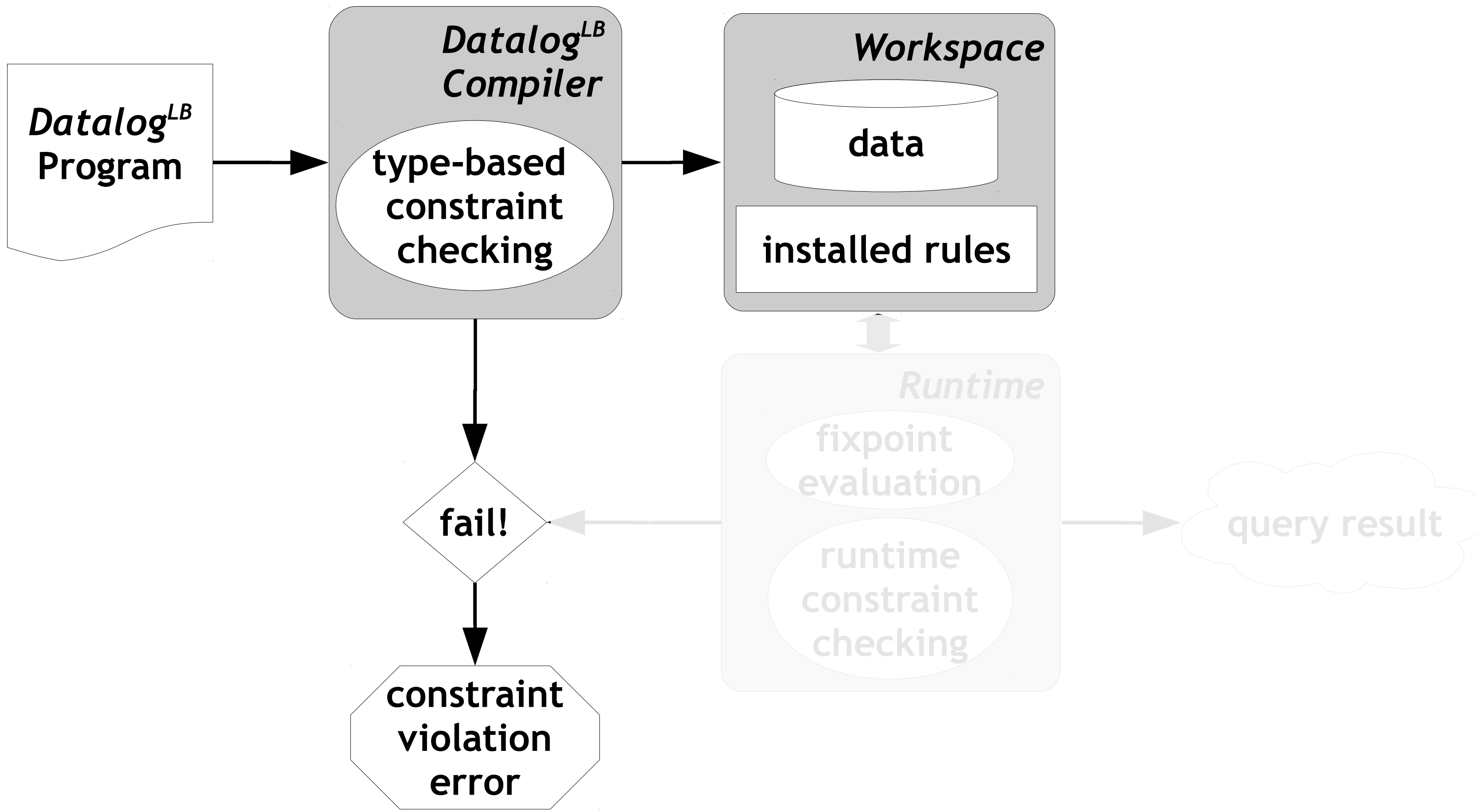
- Background: LogicBlox
- SecureBlox Architecture
- A taste of SecureBlox
  - Constraints
  - Meta-Programming
- Evaluation
- Conclusion

Background: LogicBlox (LB) Architecture



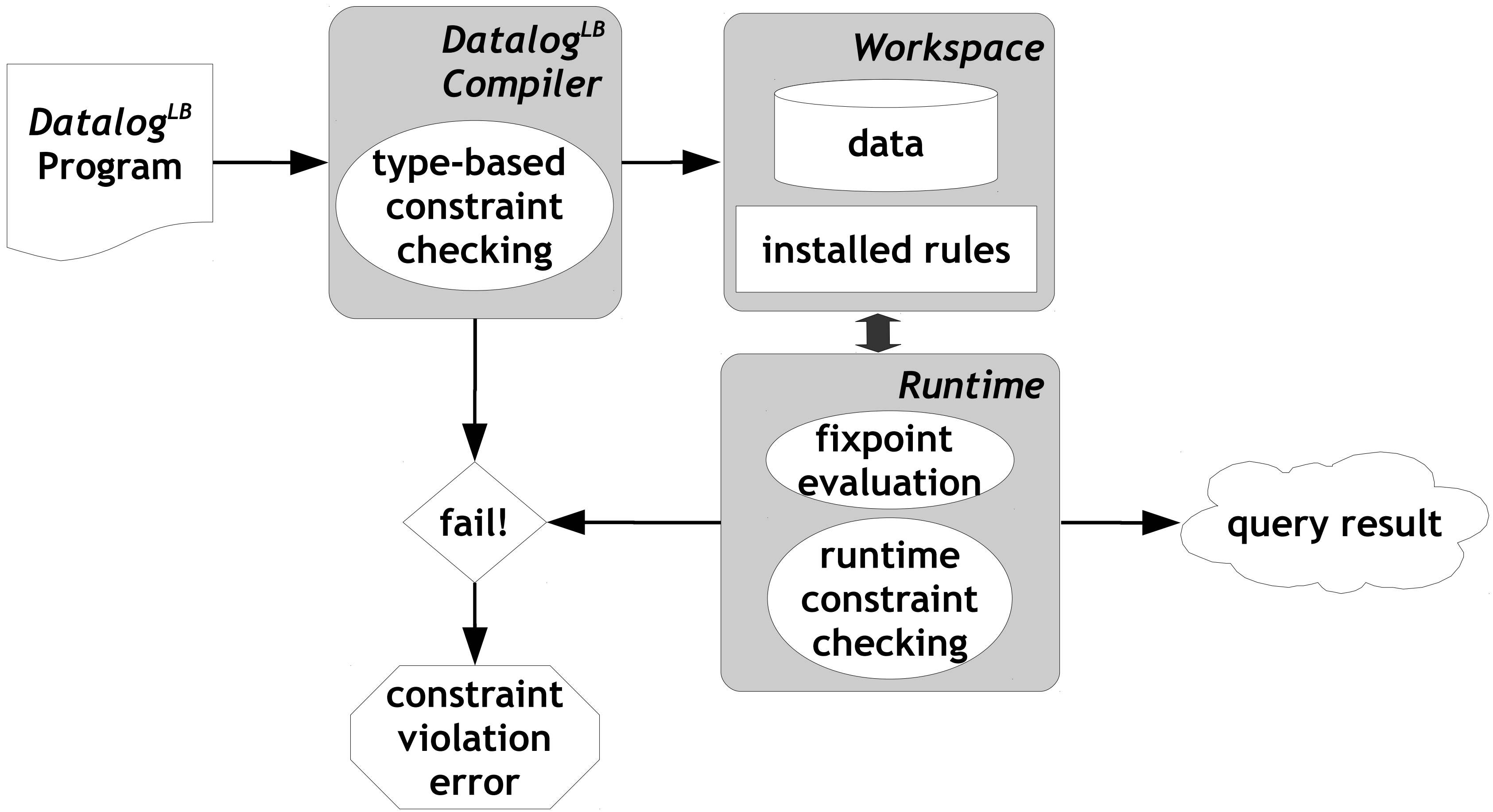
Datalog^{LB}: Datalog + integrity constraints + static type system + user-defined functions.

Background: LogicBlox (LB) Architecture



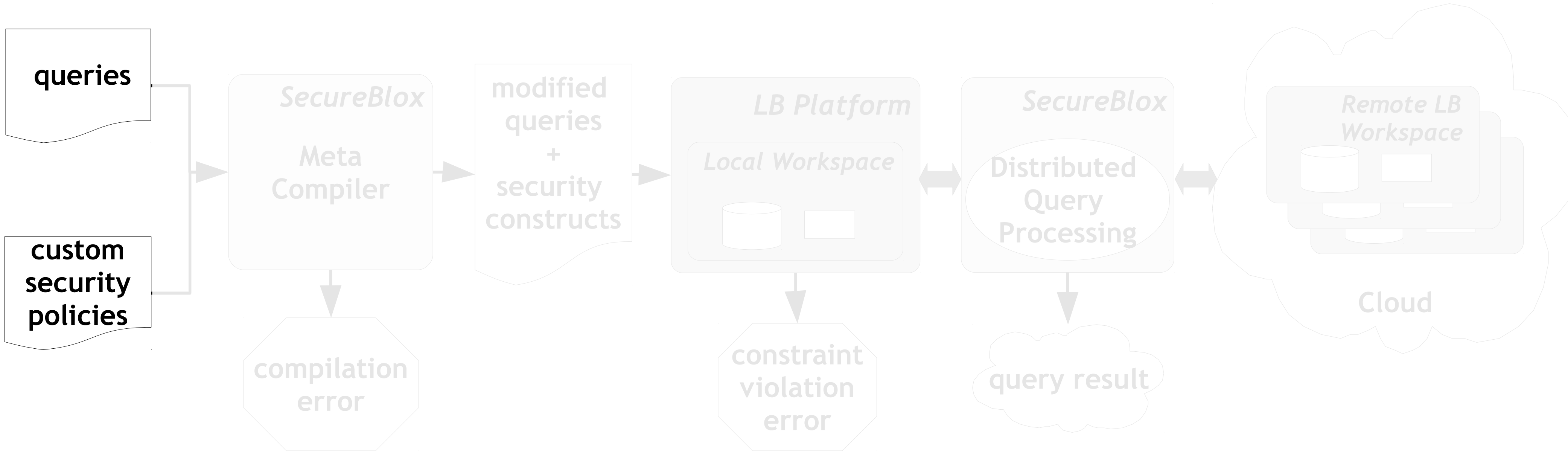
Workspace: Local DB instance with table definitions, installed rules (continuous queries), and constraints.

Background: LogicBlox (LB) Architecture



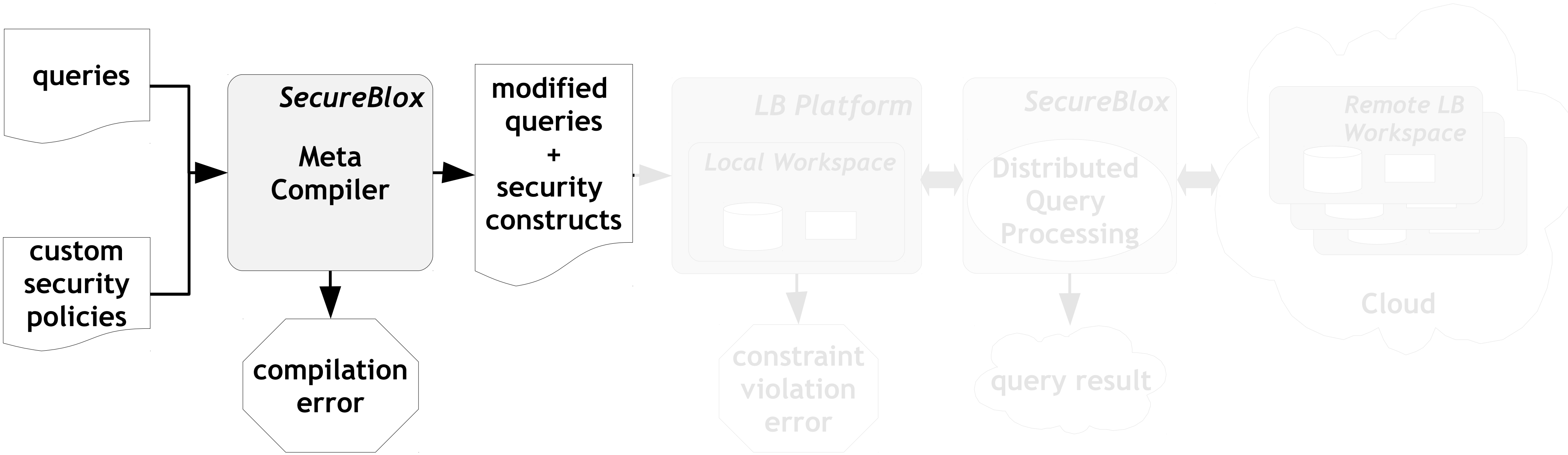
Queries and updates executed to a *fixpoint* in an ACID transaction. Constraint failure leads to abort.

SecureBlox Architecture



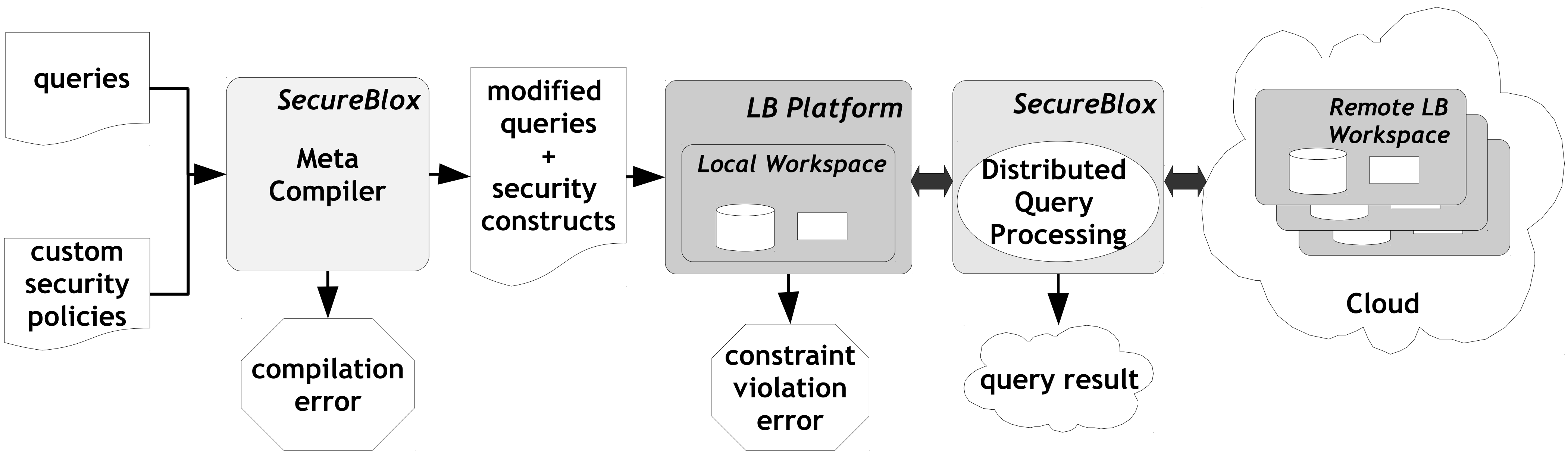
- Queries: Datalog^{LB} program that represents the distributed system/protocol
- Custom security policies: Datalog^{LB} program that operates on the queries at compile-time; called a "*meta*-program"
- System/protocol and security in same declarative language

SecureBlox Architecture



- Meta compiler runs fixpoint to transform queries based on security policies
- Rewritten queries and security policies disseminated to all *principals* at compile-time
- Principal: entity in the distributed computation

SecureBlox Architecture

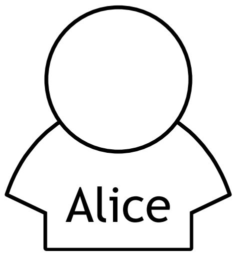


- Each *principal* has his/her own LB workspace – each principal may reside in any part of the network
- Updates from the principal's workspace, and other workspaces, trigger transactions; constraint violation implies (local) abort
- Transactions send updates to other workspaces

Outline

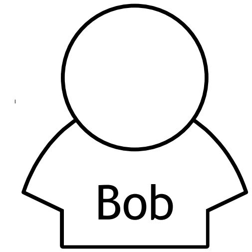
- Background: LogicBlox
- SecureBlox Architecture
- A taste of SecureBlox
  - Constraints
  - Meta-Programming
- Evaluation
- Conclusion

Exporting Paths with Authorization



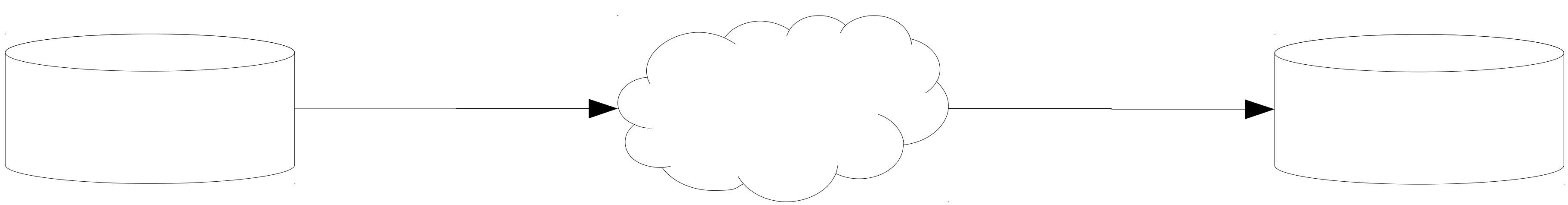
"Export paths by appending links to imported paths"

"Sign all path advertisements using digital signatures"

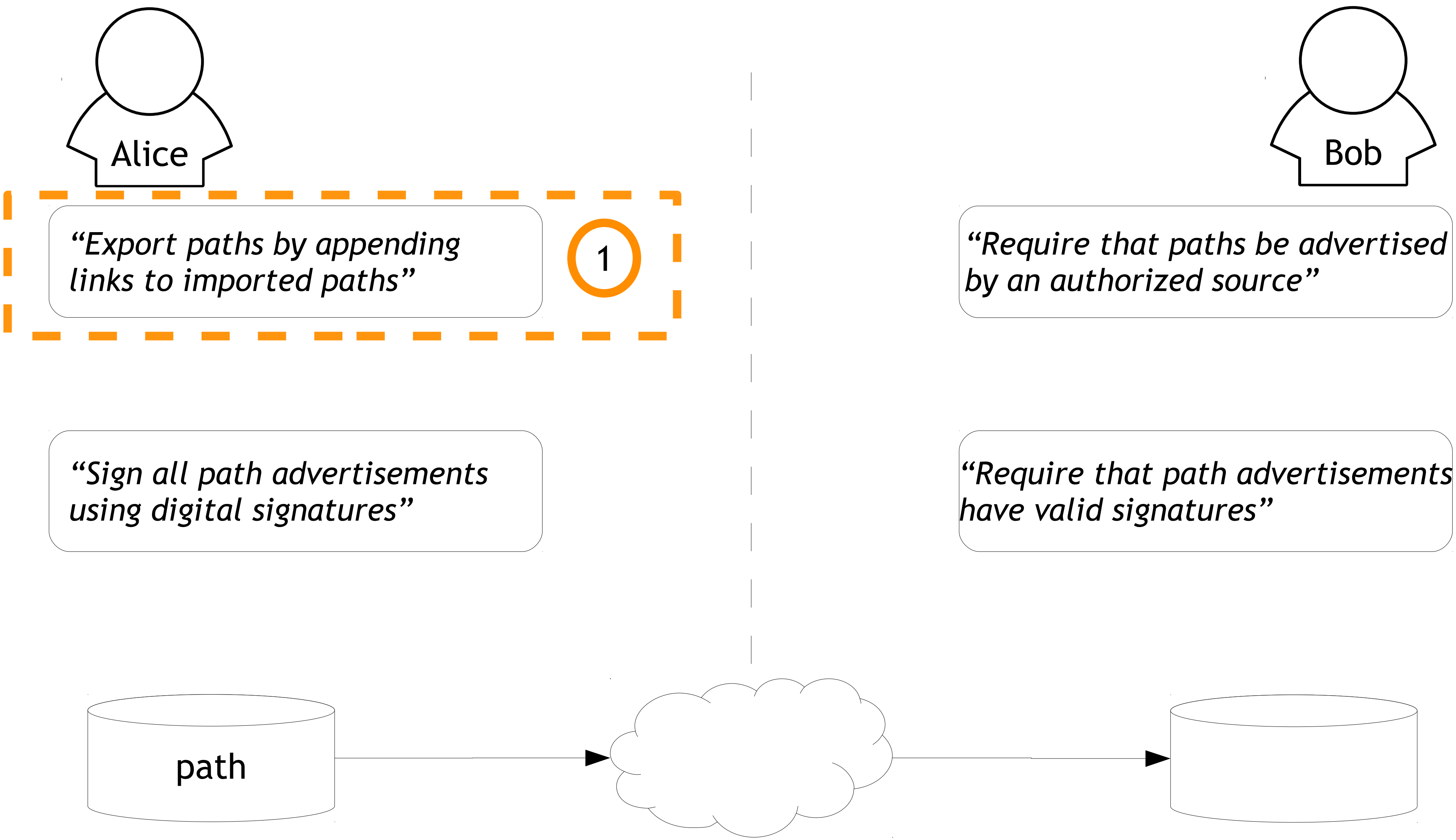


"Require that paths be advertised by an authorized source"

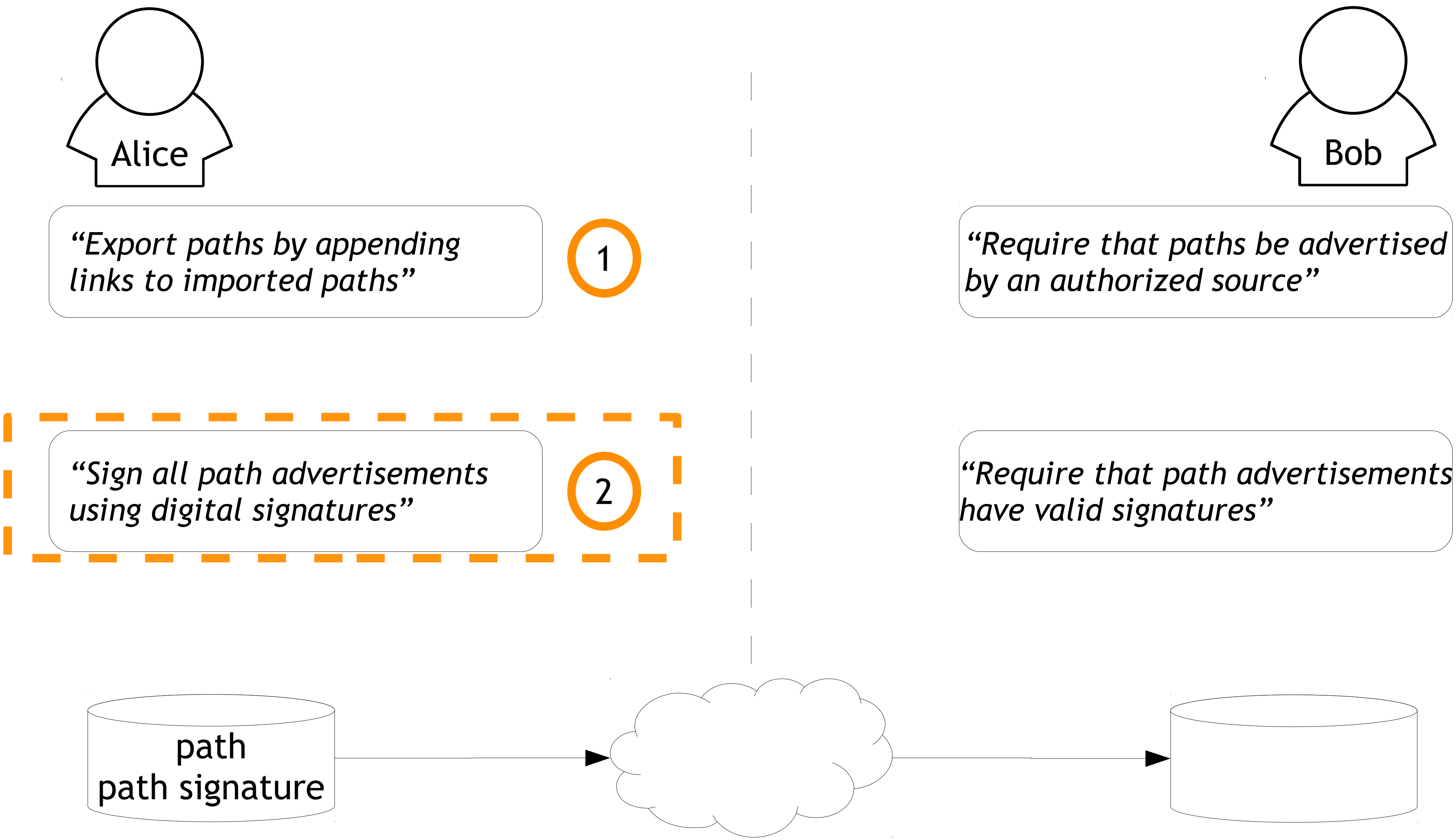
"Require that path advertisements have valid signatures"



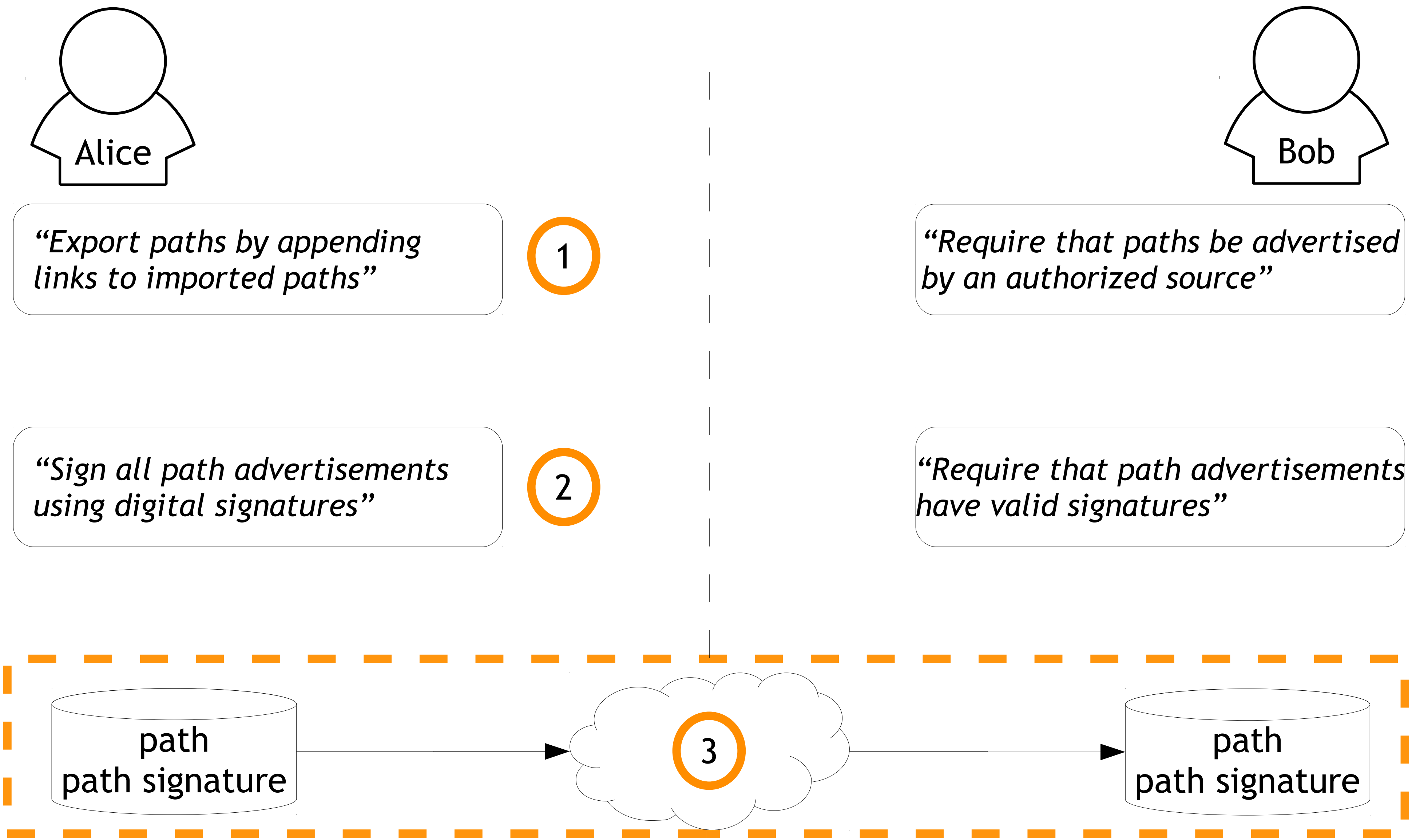
Exporting Paths with Authorization



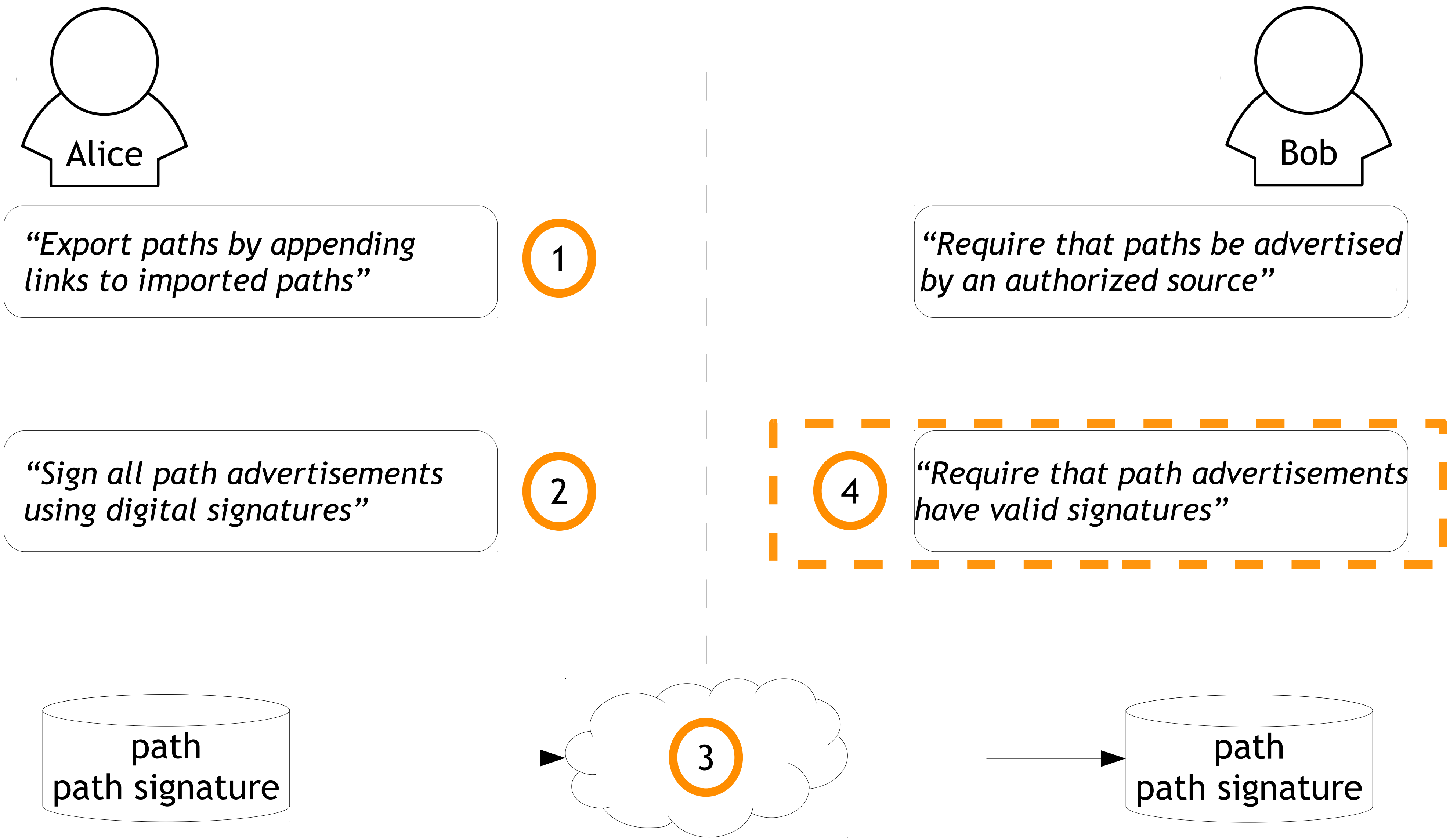
Exporting Paths with Authorization



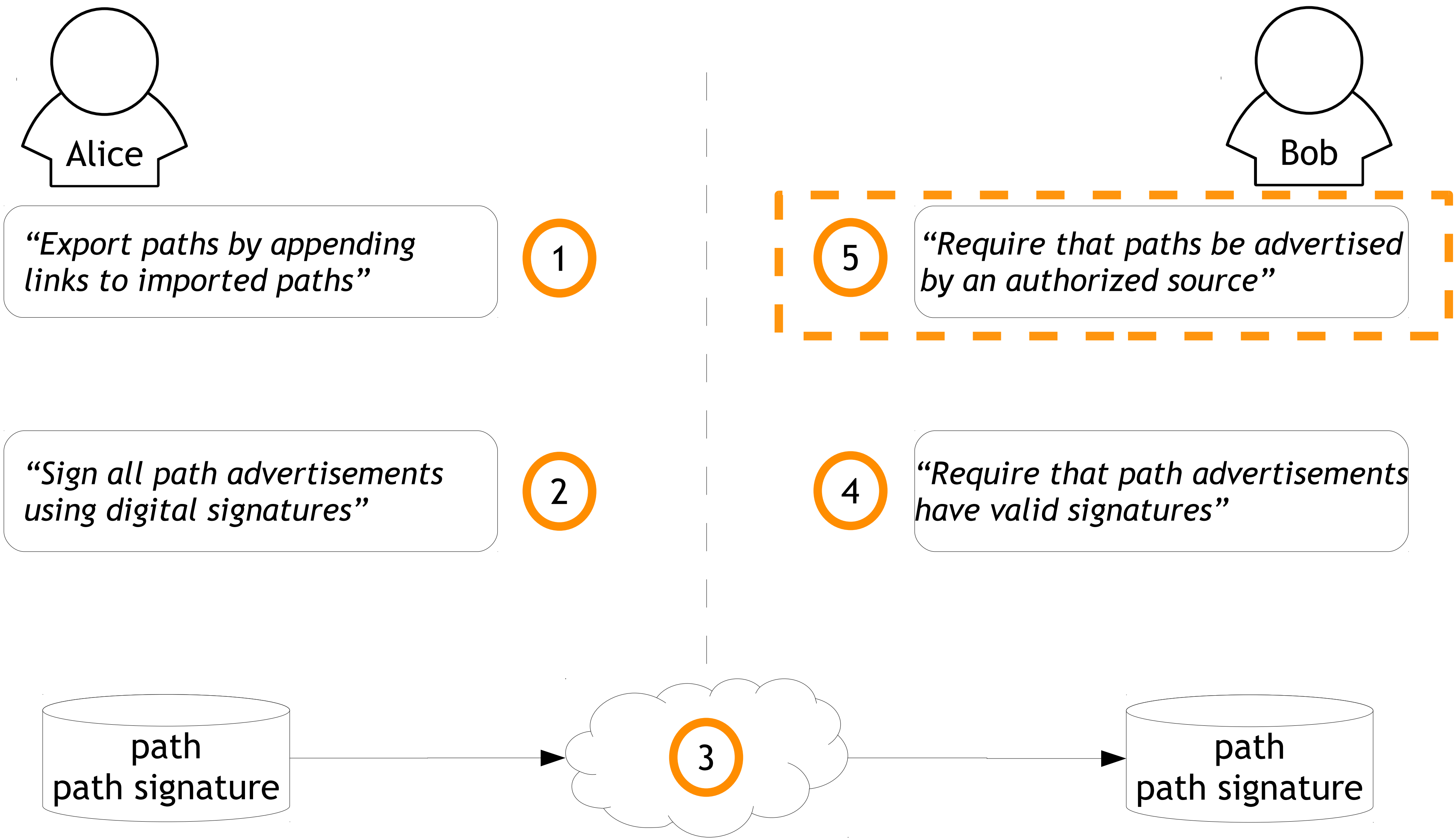
Exporting Paths with Authorization



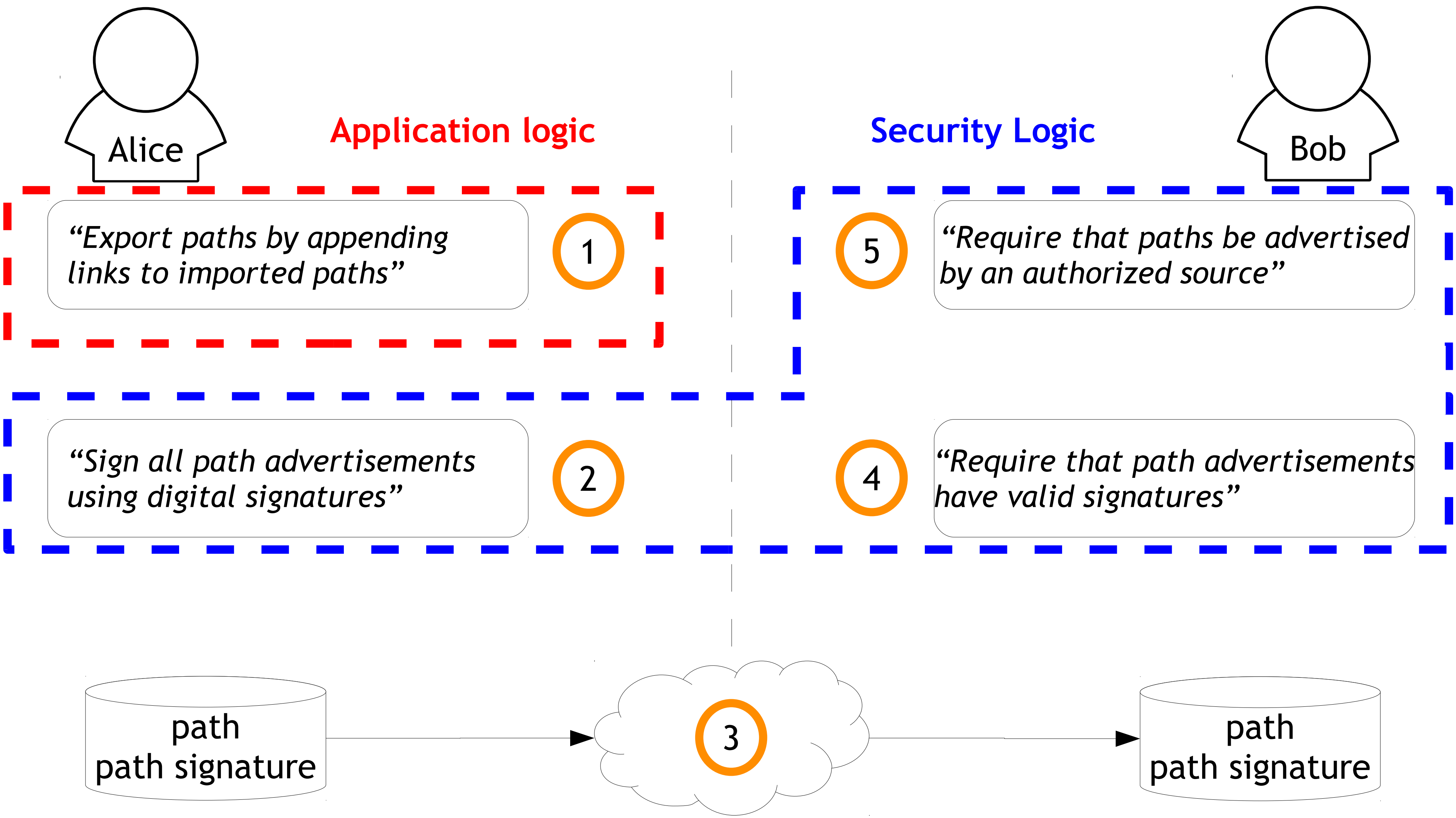
Exporting Paths with Authorization



Exporting Paths with Authorization



Exporting Paths with Authorization



Exporting Paths (Application Logic)

```
path(X,Y) <- link(self,X), path(self,Y).
```

*“whenever there is a link from **self** to X, and a path from **self** to Y, export a path to **X** from X to Y”*

Exporting Paths (Application Logic)

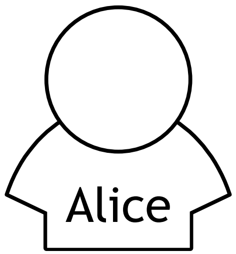
```
path(X,Y,self) <- link(self,X), path(self,Y,A), authorized(A).
```

*"whenever there is a link from **self** to X, and a path from **self** to Y **imported from A, and A is authorized**, export a path to **X** from X to Y"*

```
path_sig(X,Y,Sig) <- path(X,Y,self), rsa_sign...
```

*"whenever I export a path to **X**, generate an RSA digital signature for the path and export to **X**"*

Exporting Paths with Authorization




```
path(X,Y,self) ←
  link(Z,X), path(Z,Y,A),
  authorized(A).
```

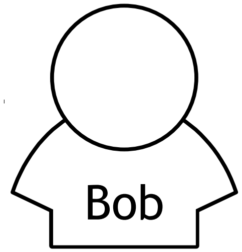
1

“Export paths by appending links to imported paths”

```
path_sig(X,Y,Sig) ←
  path(X,Y,self), rsa_sign...
```

2

“Sign all path advertisements using digital signatures”



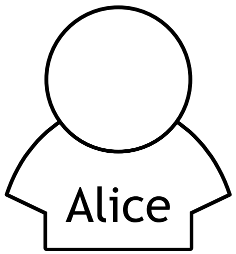
5

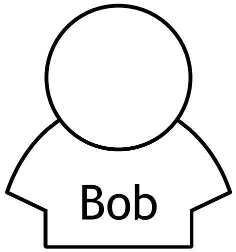
“Require that paths be advertised by an authorized source”

4

“Require that path advertisements have valid signatures”

Exporting Paths with Authorization






```
path(X,Y,self) ←
   link(Z,X), path(Z,Y,A),
   authorized(A).
```

1

"Export paths by appending links to imported paths"

```
path_sig(X,Y,Sig) ←
   path(X,Y,self), rsa_sign...
```

2

"Sign all path advertisements using digital signatures"

5

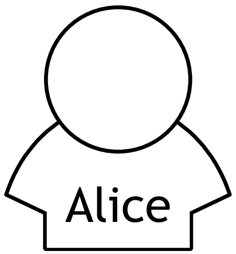
"Require that paths be advertised by an authorized source"

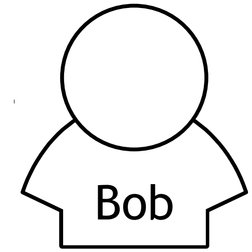
4

"Require that path advertisements have valid signatures"

Good: Security (2) written in same language as application (1)

Exporting Paths with Authorization






```
path(X,Y,self) ←
   link(Z,X), path(Z,Y,A),
   authorized(A).
```

1

"Export paths by appending links to imported paths"

```
path_sig(X,Y,Sig) ←
   path(X,Y,self), rsa_sign...
```

2

"Sign all path advertisements using digital signatures"

5

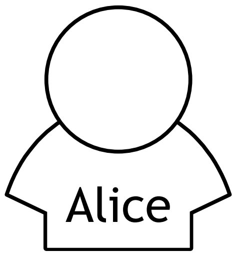
"Require that paths be advertised by an authorized source"

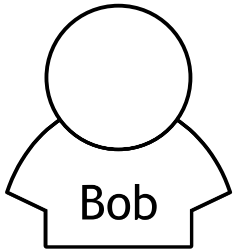
4

"Require that path advertisements have valid signatures"

Problem (Code duplication): Authenticate other predicates?
i.e., link_sig(X,Y,Sig) ← link(X,Y,self), rsa_sign...

Exporting Paths with Authorization





(1)

path(X,Y,self) ←
  link(Z,X), path(Z,Y,A),
  authorized(A).

"Export paths by appending links to imported paths"

(2)

path_sig(X,Y,Sig) ←
  path(X,Y,self), rsa_sign...

"Sign all path advertisements using digital signatures"

(5)

"Require that paths be advertised by an authorized source"

(4)

"Require that path advertisements have valid signatures"

Problem (Entanglement of security & application logic): New argument to path is part of security logic, but requires change to application logic (rule 1).

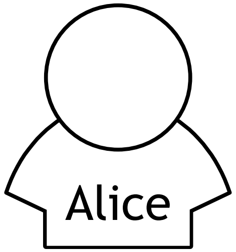
Exporting Paths with "says"

```
path(X,Y) <- link(Z,X), A says path(Z,Y), authorized(A).
```

Principal A supports the fact: path(Z,Y)

"says" authentication construct from formal security community

Exporting Paths with Authorization



1

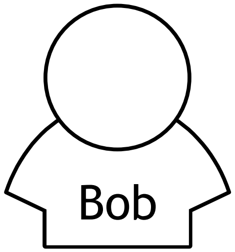
path(X,Y) ←
link(Z,X), A says path(Z,Y),
authorized(A).

“Export paths by appending links to imported paths”

2

[hard-coded into runtime]

“Sign all path advertisements using digital signatures”



5

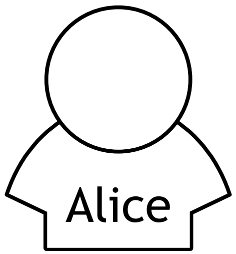
“Require that paths be advertised by an authorized source”

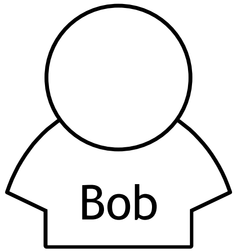
4

[hard-coded into runtime]

“Require that path advertisements have valid signatures”

Exporting Paths with Authorization





1

path(X,Y) ←
link(Z,X), A says path(Z,Y),
authorized(A).

"Export paths by appending links to imported paths"

2

[hard-coded into runtime]

"Sign all path advertisements using digital signatures"

5

"Require that paths be advertised by an authorized source"

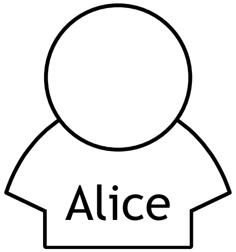
4

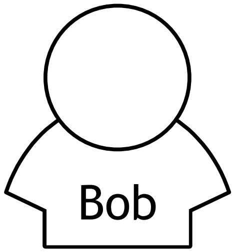
[hard-coded into runtime]

"Require that path advertisements have valid signatures"

Good: Avoids code duplication, entanglement

Exporting Paths with Authorization





path(X,Y) ←
link(Z,X), A says path(Z,Y),
authorized(A).

(1)

"Export paths by appending links to imported paths"

[hard-coded into runtime]

(2)

"Sign all path advertisements using digital signatures"

(5)

"Require that paths be advertised by an authorized source"

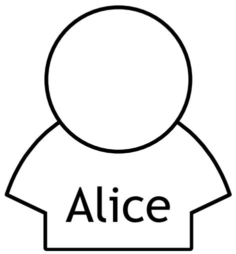
(4)

[hard-coded into runtime]

"Require that path advertisements have valid signatures"

Problem ("says" modeled outside of language): Can't reconfigure 2 and 4

Exporting Paths with Authorization



path($\underline{X}$,Y) ←
link($\underline{Z}$,X), A says path($\underline{Z}$,Y),
authorized(A).

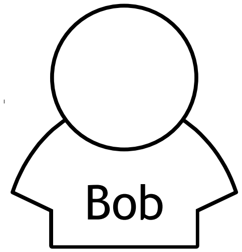
1

"Export paths by appending links to imported paths"

[hard-coded into runtime]

2

"Sign all path advertisements using digital signatures"



5

*"**Require** that paths be advertised by an authorized source"*

4

[hard-coded into runtime]

"Require that path advertisements have valid signatures"

What does "require" mean?

Exporting Paths with “says”

- “Require” == integrity constraints

- Problem (Entanglement): “says” abstraction solves this
- Problem (“says” modeled outside of language): model “says” abstraction in the language
- Problem (Code Duplication): write “says” logic “**for all authenticated predicates P**”

Integrity Constraints

- Logical implication that always holds in a consistent instance

```
link(X,Y) -> node(X), node(Y).
```

“whenever there is a link from X to Y, require that X and Y are both in the 'node' set”

Integrity Constraints

- Logical implication that always holds in a consistent instance

```
link(X,Y) -> node(X), node(Y).
```

“whenever there is a link from X to Y, require that X and Y are both in the 'node' set”

- “<-” generates new facts if RHS true; “->” causes abort if LHS true and RHS false

Meta-Model: Rules as Data

```
rule(rule_id).
head(rule_id, pred_id).
body(rule_id, pred_id).
pred(pred_id, name).
arg(pred_id, position, expr_id).
var(expr_id, name).
```

[VLDB09, CIDR09]

Meta-Model: Rules as Data

```
rule(rule_id).
head(rule_id, pred_id).
body(rule_id, pred_id).
pred(pred_id, name).
arg(pred_id, position, expr_id).
var(expr_id, name).
```

```
path(X,Y) <- link(Z,X), path(Z,Y).
```

```
                                              arg(1,1,1)  var(1,"X")
                                              arg(1,2,2)  var(2,"Y")
         head(1,1)  pred(1,"path")            arg(2,1,3)  var(3,"Z")
rule(1)  body(1,2)  pred(2,"link")            arg(2,2,4)  var(4,"X")
         body(1,3)  pred(3,"path")            arg(3,1,5)  var(5,"Z")
                                              arg(3,2,6)  var(6,"Y")
```

[VLDB09, CIDR09]

A Reconfigurable “Says”

```
‘{

  sig[P](self[],T,S,V*) <-
     says[P](self[],T,V*),
     privkey[] = K,
     rsa_sign[P](K,S,V*).

  says[P](T,self[],V*) ->
     sig[P](T,self[],S,V*),
     pubkey(T,K),
     rsa_verify[P](K,S,V*).

}

  <-- predicate(P),
      auth_pred(P).
```

A Reconfigurable "Says"

```
‘{

    sig[P](self[],T,S,V*) <-
       says[P](self[],T,V*),
       privkey[] = K,
       rsa_sign[P](K,S,V*).

    says[P](T,self[],V*) ->
       sig[P](T,self[],S,V*),
       pubkey(T,K),
       rsa_verify[P](K,S,V*).

}
```

"For all predicates that I want to be authenticated (auth_pred)"

```
<-- predicate(P),
    auth_pred(P).
```

(Universally quantify over predicates)

A Reconfigurable "Says"

```
'{
```

*"Whenever I say a predicate **P**, generate a signature."*

```
sig[P](self[],T,S,V*) <-
   says[P](self[],T,V*),
   privkey[] = K,
   rsa_sign[P](K,S,V*).

says[P](T,self[],V*) ->
   sig[P](T,self[],S,V*),
   pubkey(T,K),
   rsa_verify[P](K,S,V*).

}
```

"For all predicates that I want to be authenticated (auth_pred)"

(Universally quantify over predicates)

```
<-- predicate(P),
    auth_pred(P).
```

A Reconfigurable "Says"

```
‘{
```

*"Whenever I say a predicate **P**, generate a signature."*

```
sig[P](self[],T,S,V*) <-
    says[P](self[],T,V*),
    privkey[] = K,
    rsa_sign[P](K,S,V*).
```

*"Whenever I import a predicate **P** said by principal **T**, ensure the signature is valid for principal **T**"*

```
says[P](T,self[],V*) ->
    sig[P](T,self[],S,V*),
    pubkey(T,K),
    rsa_verify[P](K,S,V*).
```

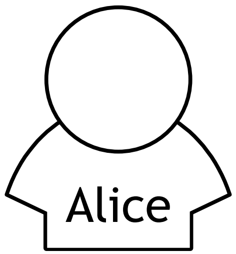
```
}
```

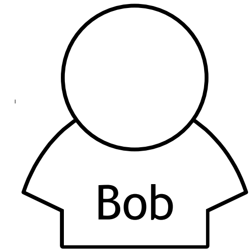
"For all predicates that I want to be authenticated (auth_pred)"

(Universally quantify over predicates)

```
<-- predicate(P),
    auth_pred(P).
```

Exporting Paths with Authorization





*export the **path** fact <-*
append ***link*** to imported ***path***

1

"Export paths by appending links to imported paths"

5

***T** says **Path** fact to me ->*
T is "authorized".

"Require that paths be advertised by an authorized source"

*signature for a **path** fact <-*
I say a ***path*** fact
sign fact with my private key

2

"Sign all path advertisements using digital signatures"

4

***T** says **Path** fact to me ->*
valid signature with ***T***'s pub key

"Require that path advertisements have valid signatures"

Generated by meta-rule on previous slide

A Reconfigurable “Says”

- See paper for:
  - Tunable authentication, encryption
  - Anonymity
- Not just “says!”
  - Authorization
  - “Delegation” of access rights

Outline

- Background: LogicBlox
- SecureBlox Architecture
- A taste of SecureBlox
  - Constraints
  - Meta-Programming
- Evaluation
- Conclusion

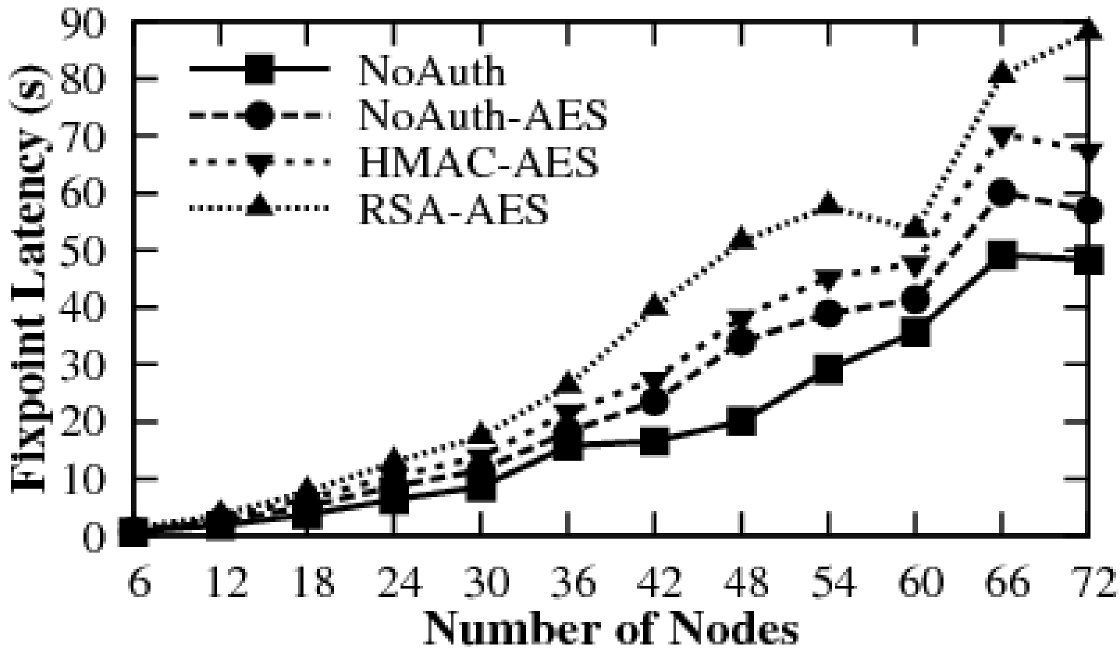
Secure Path Vector Protocol

```
pathvar(P) -> .
path[P,Src,Dst]=C -> pathvar(P), node(Src), node(Dst), int[32](C).
pathlink[P,H1]=H2 -> pathvar(P), node(H1), node(H2).
bestcost[Src,Dst]=C -> node(N1), node(N2), int[32](c).
link(N1,N2) -> node(N1), node(N2).

path[P,self[],U]=1, pathlink[P,Me]=N <-
   link(Me,N), prin_node[self[]]=Me,
   prin_node[U]=N.

says[`path](self[],U,P,N,N2,C+1),
says[`pathlink](self[],U,P,H1,H2),
says[`pathlink](self[],U,P,N1,Me) <-
   pathlink[P,H1]=H2, link(Me,N),
   path[P,Me,N2]=C, bestcost[Me,N2]=C,
   prin_node[U]=N, prin_node[self[]]=Me,
   N!=N2, !pathlink[P,N]=_.
```

- Based on declarative path vector protocol [SIGCOMM 05]
- Still using “says” abstraction. Any implementation of “says” can be used here

Evaluation: Performance Snapshot

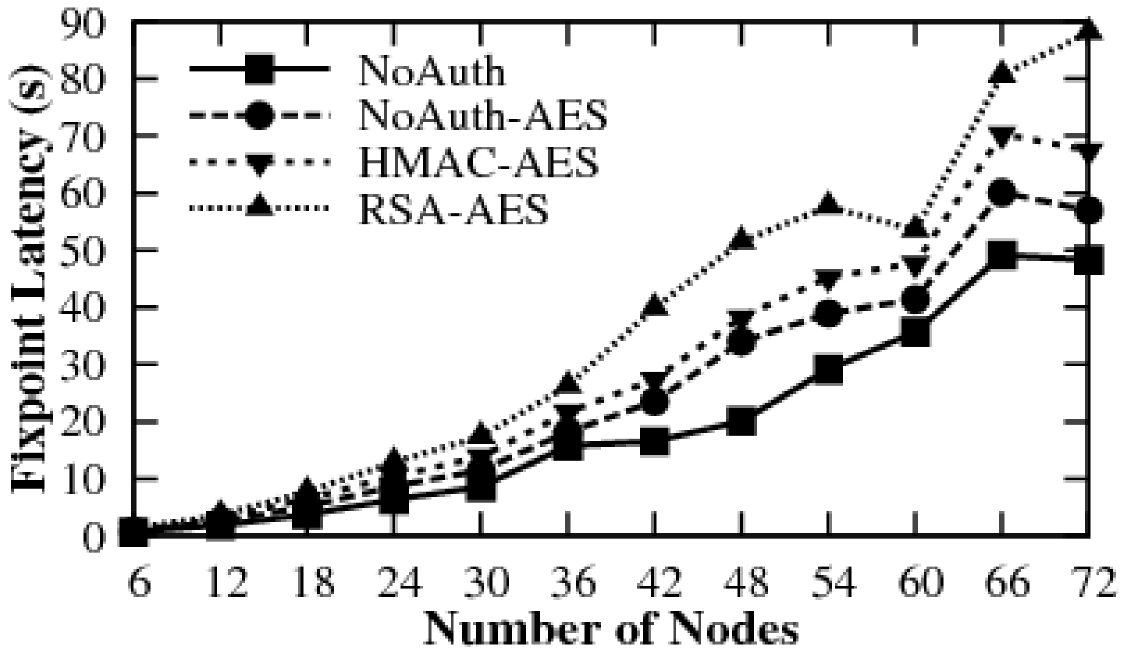
Evaluation of Path Vector Protocol running on 32 machines in a local cluster with various implementations of “says”



Fixpoint latency of a Declarative Networking [SIGCOMM 05] path-vector routing protocol.
Performance is comparable to Declarative Networking implementation.

Evaluation: Performance Snapshot

Evaluation of Path Vector Protocol running on 32 machines in a local cluster with various implementations of "says"



Fixpoint latency of a Declarative Networking [SIGCOMM 05] path-vector routing protocol.
Performance is comparable to Declarative Networking implementation.

See paper for **parallel hash join, anonymous join**

Conclusion and Future Work

- Contributions:
  - SecureBlox architecture: reconfigurable security
  - Static meta-programming framework
  - Case studies & eval: **parallel hash join, anonymous distributed join**
- Future work:
  - New programming model, i.e. **secure MapReduce**
  - Formally reason about security properties (theorem proving, model checking)

SecureBlox: Customizable Secure Data Processing

William R. Marczak* , Shan Shan Huang[†], Martin Bravenboer[†], Micah Sherr[‡], Boon Thau Loo[‡], Molham Aref[†]

*University of California, Berkeley [†]LogicBlox, Inc [‡]University of Pennsylvania