TrustForge: Flexible Access Control for VehicleForge.mil Collaborative Environment

Oleg Sokolsky
PRECISE Center
University of Pennsylvania

Overview

- Concentrate on the access control and credentialing scheme for repository access
- Build upon prior expertise in quantitative trust management, provenance queries, and secure network provenance
- Close collaboration with other VehicleForge teams is planned

Team Composition

Faculty

- Oleg Sokolsky (PI)
  - Quantitative trust management, formal methods
- Insup Lee
  - Quantitative trust management, real-time and cyber-physical systems, run-time monitoring
- Zach Ives
  - Databases, distributed systems
- Andreas Haeberlen
  - Distributed systems, networking, and security

Research Associate

- Krishna Venkatasubramanian
  - Networking, security, trust management

Ph.D. Students

- Andrew West
- Jian Chang

Programmer

- Anders Miltner

Requirements and assumptions

- Repository hosts a collection of projects with different criticality levels
  - Experimental to mission-critical
- Non-critical projects should have the lowest barrier of entry
  - Participants should be good citizens
- Highly critical projects should enjoy strong security guarantees
- Users should gain additional capabilities by
  - Diligent participation in projects
  - Obtaining additional credentials

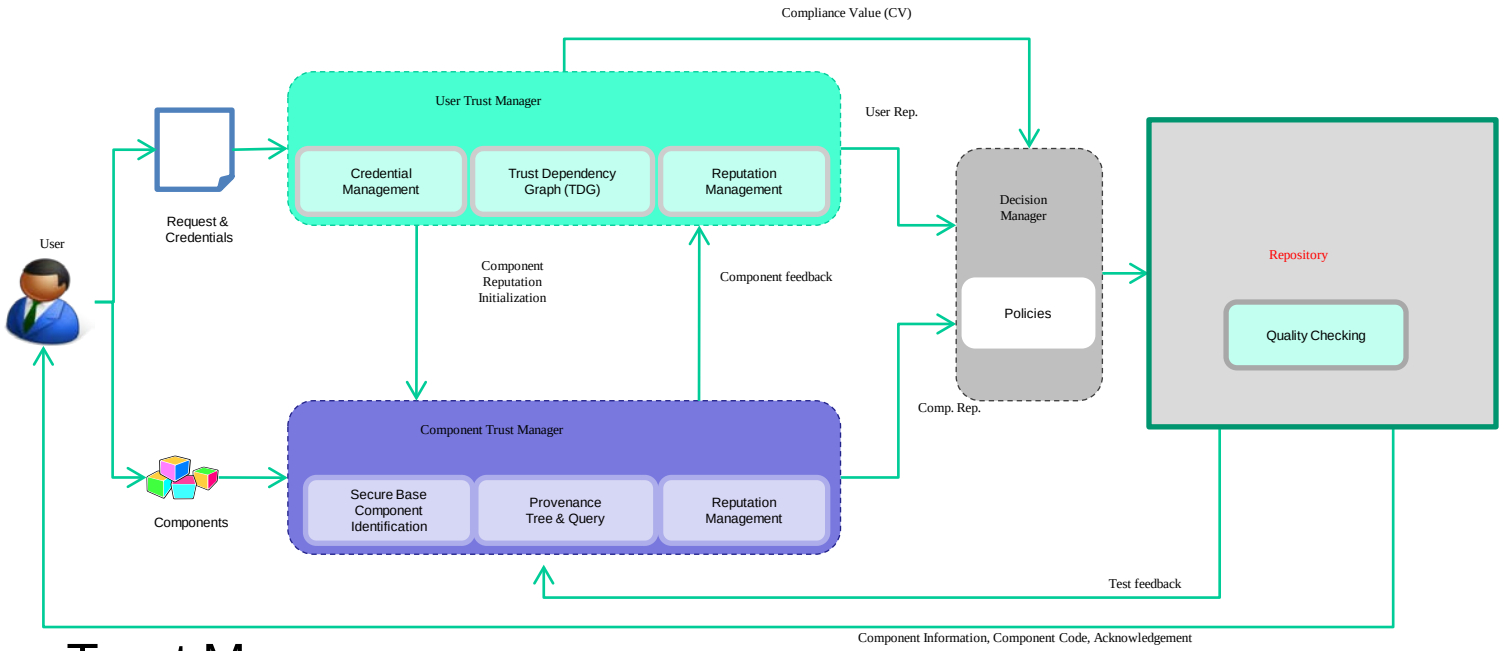
More assumptions

- Repository keeps revision history and identities of users making changes
- Repository stores hierarchically organized components
  - Composite components are built from simpler components
- Repository has means of checking quality of components
  - Based on component semantics
    - black box

Key insights

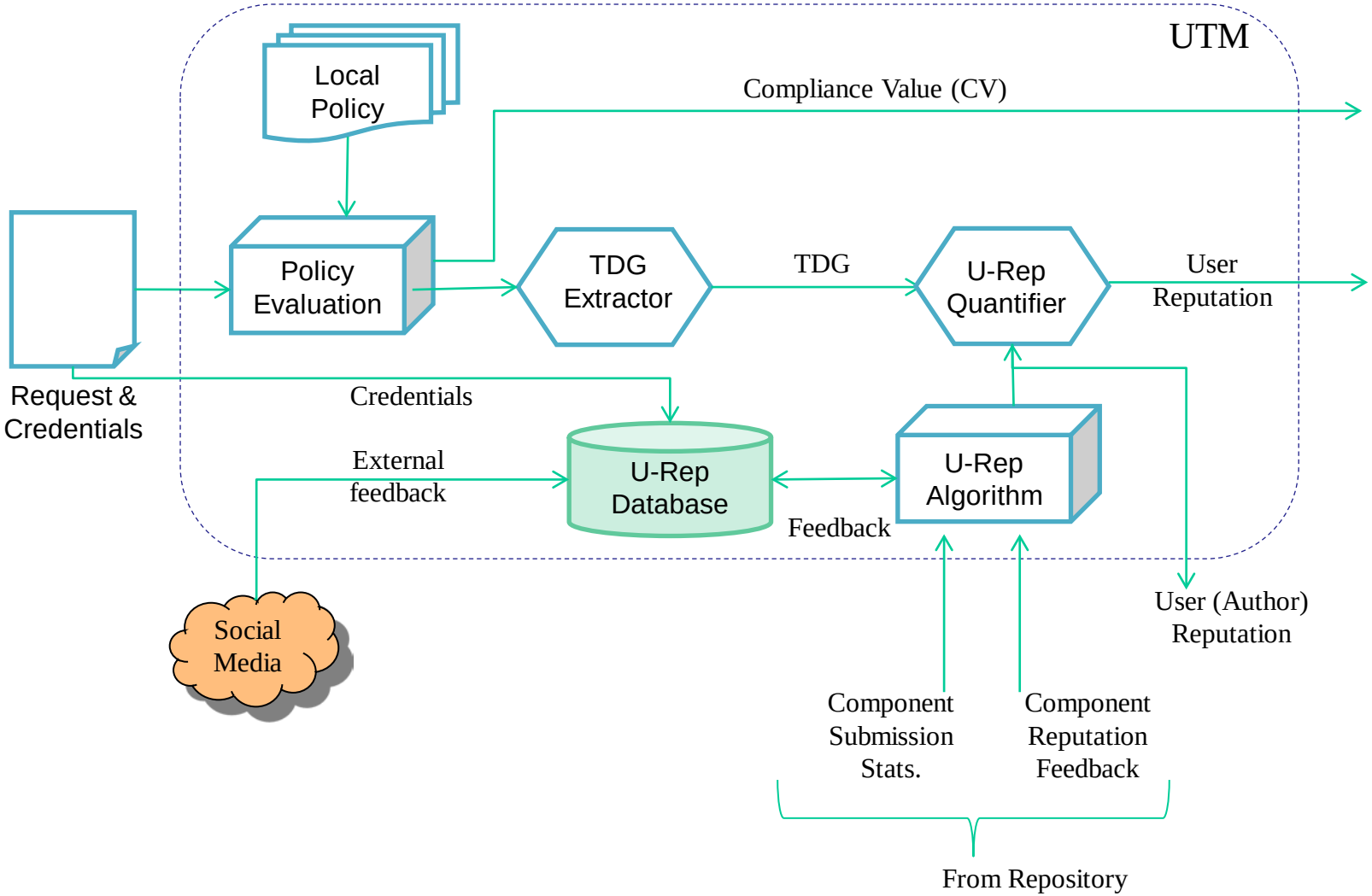
- Combine policy-based and reputation-based access control approaches
  - Cryptographic credentials provide guarantees
  - Reputations provide flexibility
- Reputations for both users and components
- User and component reputations are related
  - Good users make good components
  - Problem-free components make users look good
- Component reputation depends on its use
  - Components built from trusted parts are better
  - Heavily used components are trusted more

TrustForge Architecture



- User Trust Manager
  - Evaluate user credentials using an access control policy, and compute user reputation
- Component Trust Manager
  - Evaluate provenance of components and compute their reputation
- Decision Manager
  - Permit or deny repository access based on reputations and credentials

User Trust Management (UTM)



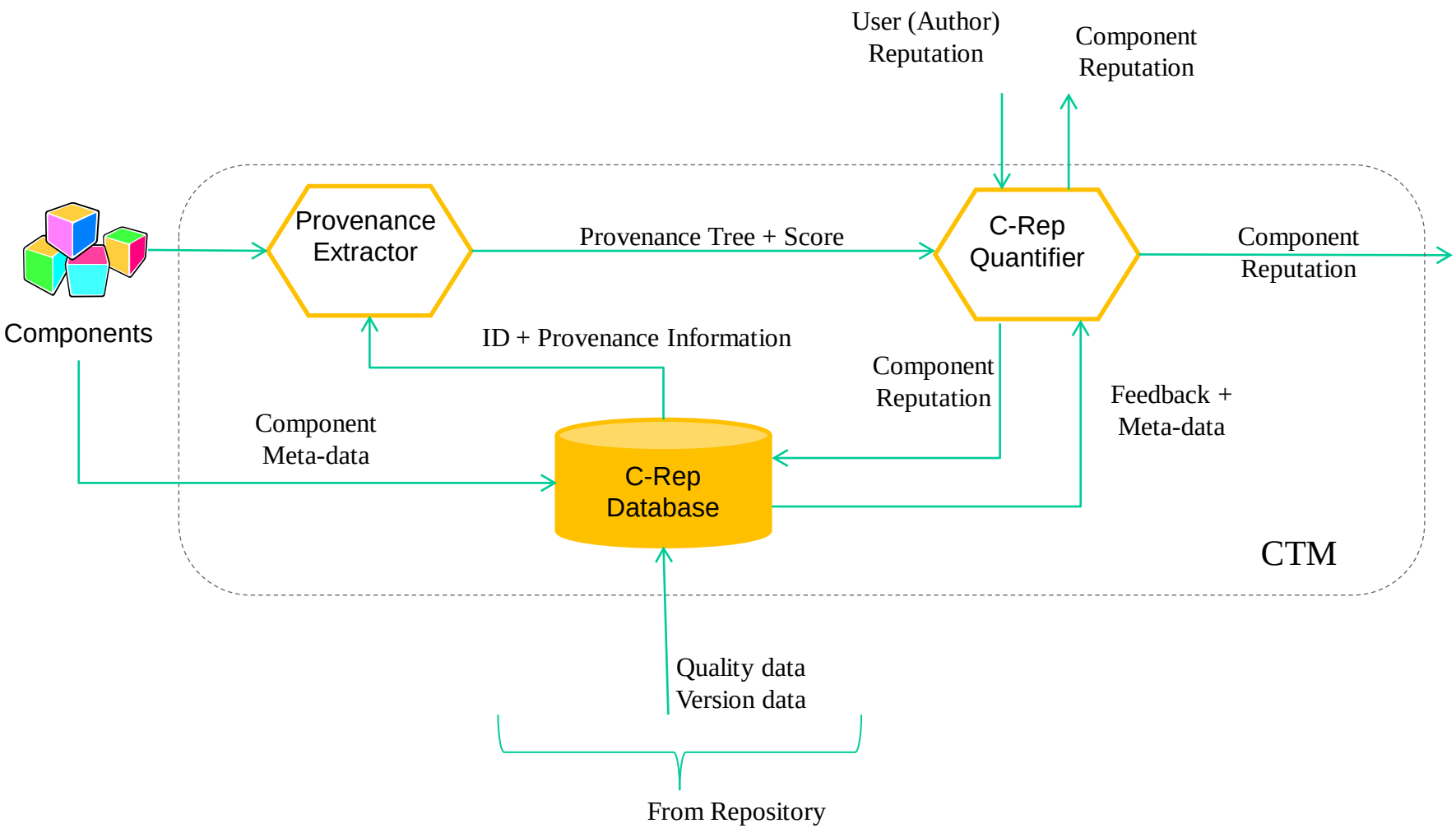
UTM Workflow

- Use local policies as initial barrier for entry
  - Can be tightened to meet project requirements
  - KeyNote language used for policy specification
  - A compliance value generated as output
- A trust dependency graph is extracted from the credentials
  - Reputation used to populate its edges
- Reputation is kept for each user in the system
  - Computed based on feedback such as:
    - Quality and utility of components
    - Direct feedback from other users
  - Reputation decays with time

Research Questions

- Policies:
  - How to specify policies?
    - Delegation and component contribution/access
- Feedback:
  - What feedback structure to use ?
  - How to collect feedback ?
  - How to avoid/detect slandering, collusion and other forms of attacks on feedback provided?
- Reputation:
  - What are the reputation semantics ?
  - What reputation function to use to model it ?

Component Trust Management (CTM)



CTM Workflow

- User makes a request to access the repository for adding a component
- Provenance extractor builds a provenance tree of the component and computes score
- Reputation database stores component reputations and feedback
  - Component quality checks
  - Past revisions
- Reputation quantifier updates the reputation function
- Reputation value is delivered to UTM and DM

Component Reputation

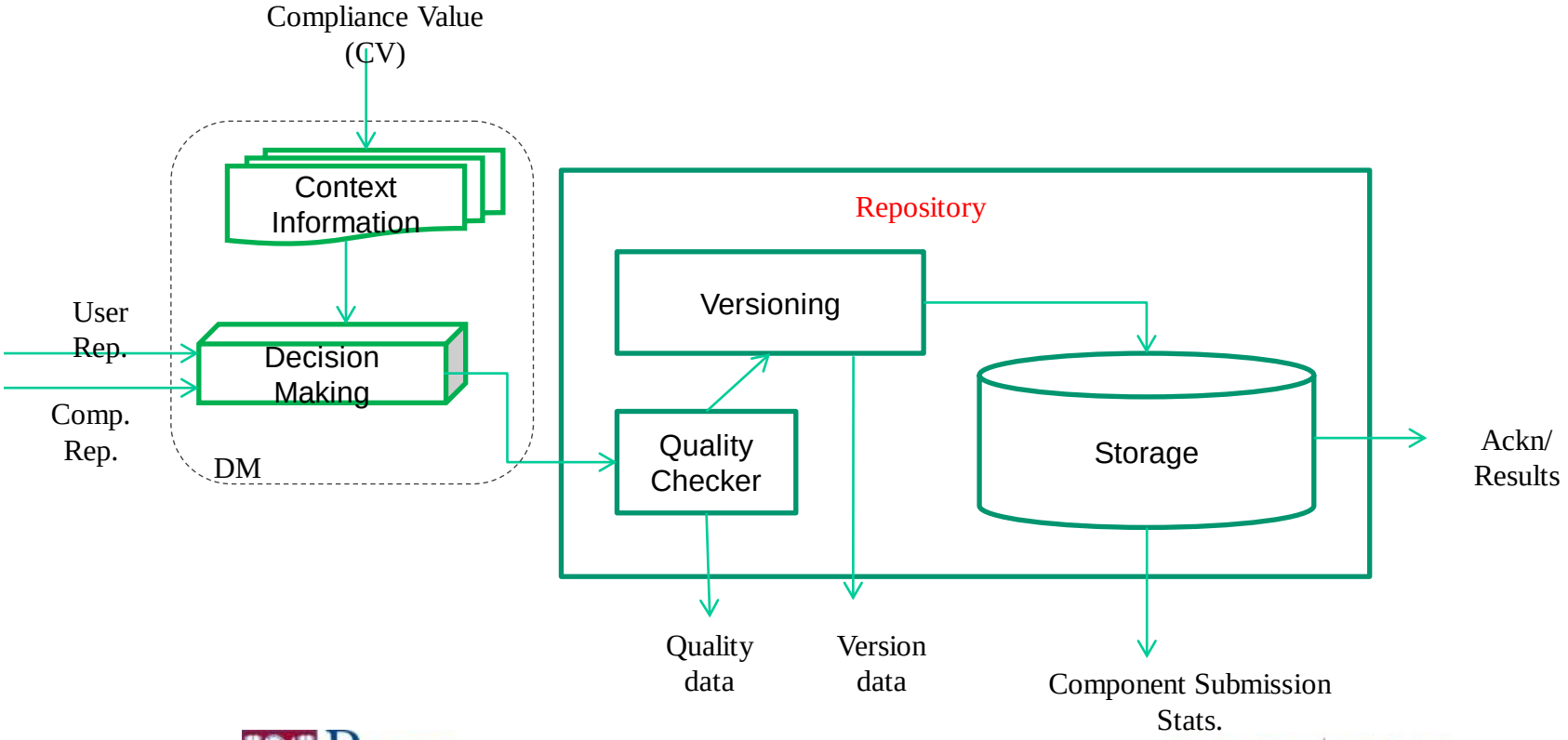
- Reputation function
  - Seeded by component author reputation
  - Reputation of composite components is a function of the provenance score
    - Changes to component reputations are dynamically propagated to all affected components
  - Reputation of any component is a function of feedback
  - Reputation does not decay with time

Research Questions

- Reputation semantics?
- What reputation algorithm to use?
- How to select reputation functions that
  - Meet the constraints of provenance querying and
  - Satisfy the requirements of component reputation?
- How to incorporate feedback?
  - How does feedback on a composite component affect its subcomponents?
  - Can new feedback types be dynamically incorporated?
- Evaluation of reputation effectiveness?

Decision Management (DM)

- Decides whether to grant a request based on
  - Policy compliance value
  - User and component reputations



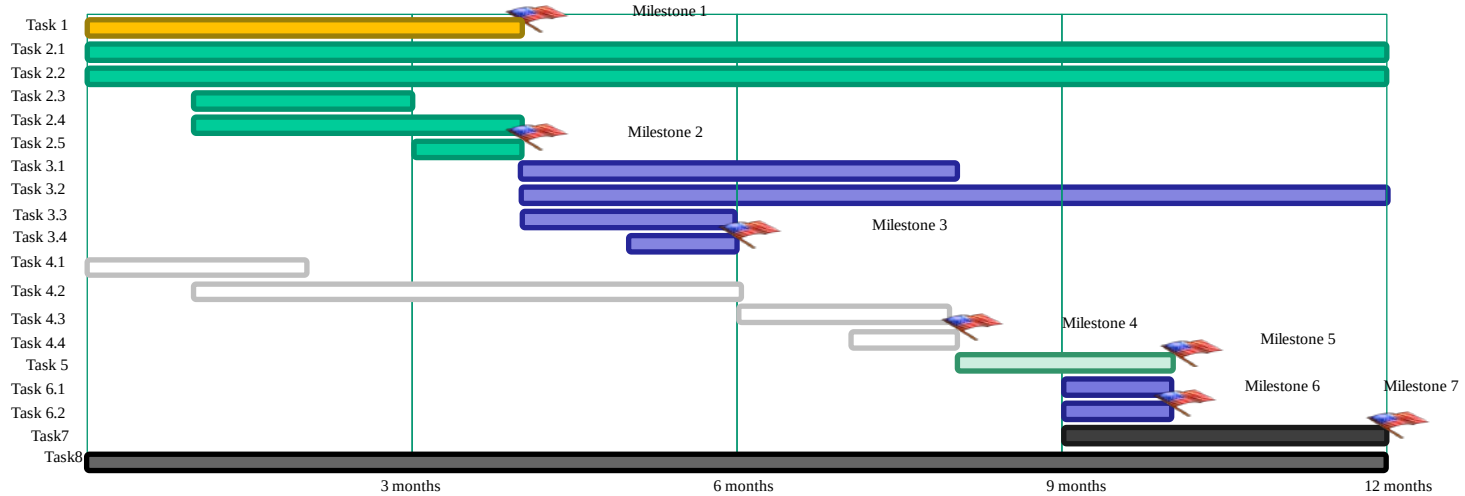
Research Questions

- Separation between local policies and decision policies based on compliance value
  - Is CV the right level of abstraction?
- How to specify decision policies for the decision manager?
  - What context information should be taken into account?
  - How to obtain cutoff values?
- Risk management strategies?
- Is reputation useful in the repository?

Implementation Strategy

- Plan for integration with other VehicleForge teams, but provide for independent development
  - Define interfaces first
    - Requires intensive collaboration with other teams
  - Build a simple repository stand-in for independent testing, evaluation, and demo
- Be ready for integration testing and pilot case study at 9 months mark
  - Fine tuning of reputation functions continues throughout the project

Implementation Timeline



Task 1: Interface definition
(**Milestone1**: Interface definition document)

Task 2.1: User reputation design
Task 2.2. Comp. reputation design
Task 2.3: Reputation DB design and Feedback structure
Task 2.4: Reputation implementation
Task 2.5: Integration & evaluation
(**Milestone2**: reputation infrastructure)

Task 3.1: KN implementation
Task 3.2: Meta-policy design
Task 3.3. Meta-policy implementation
Task 3.4: Admin & User interface implementation
(**Milestone3**: Credential system and policy evaluation)

Task 4.1: Interface module design & implementation
Task 4.2: Repository design & implementation
Task 4.3. Integration & evaluation of UTM, CTM, DM and repository
Task 4.4: User interface integration
(**Milestone4**: Standalone implementation)

Task 5 (**Milestone5**): Simulation Evaluation

Task 6.1/6.2: System-wide integration of interfaces &feedback
(**Milestone6**: Integration with VehicleForge.mil)

Task 7 (**Milestone7**): Pilot study

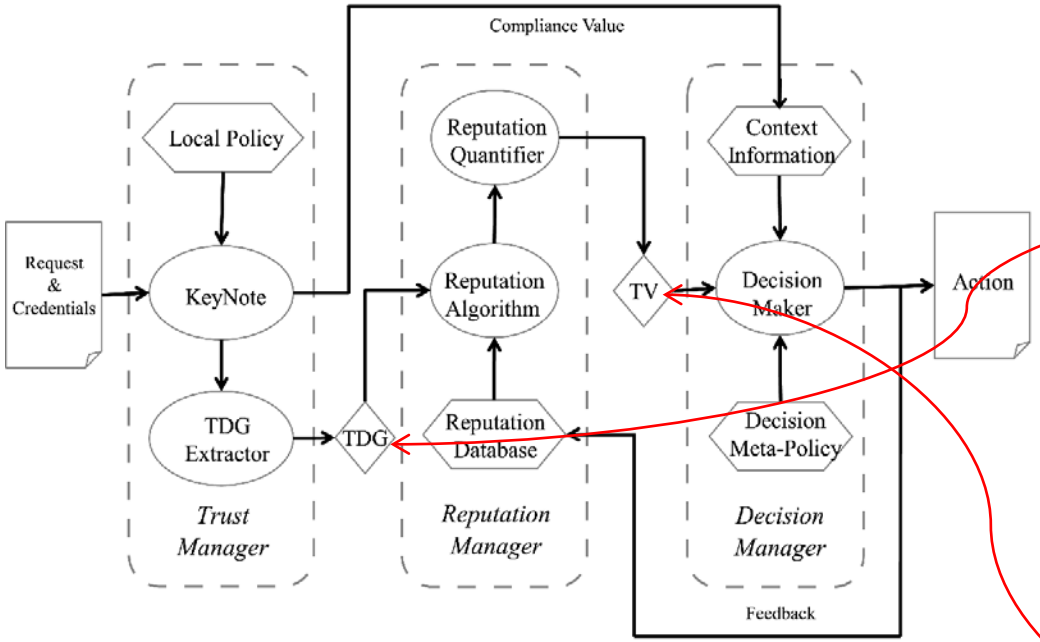
Task 8 (**Milestone8**): Reporting & meetings

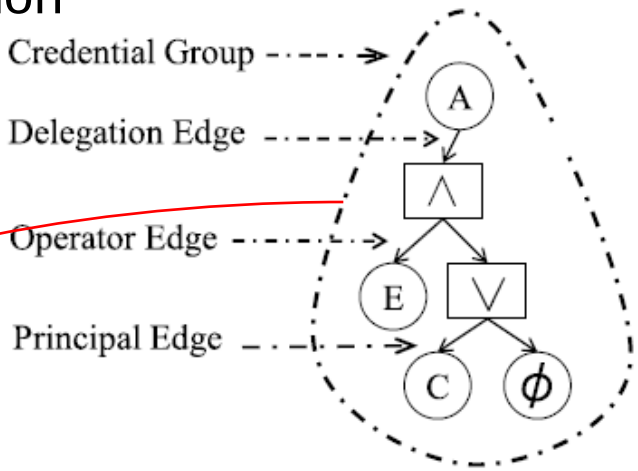
Background

- Quantitative Trust Management (Lee, Sokolsky)
  - QuanTM architecture
  - AS-CRED
  - STiki
- Data Provenance Queries (Ives)
  - Orchestra
- Secure Network Provenance (Haeberlen)

Quantitative Trust Management (QTM)

- QTM provides a dynamic interpretation of authorization policies for access control decisions using evolving reputations of parties
- QuanTM is a QTM system that combines elements from PTM and RTM to create a novel method for trust evaluation





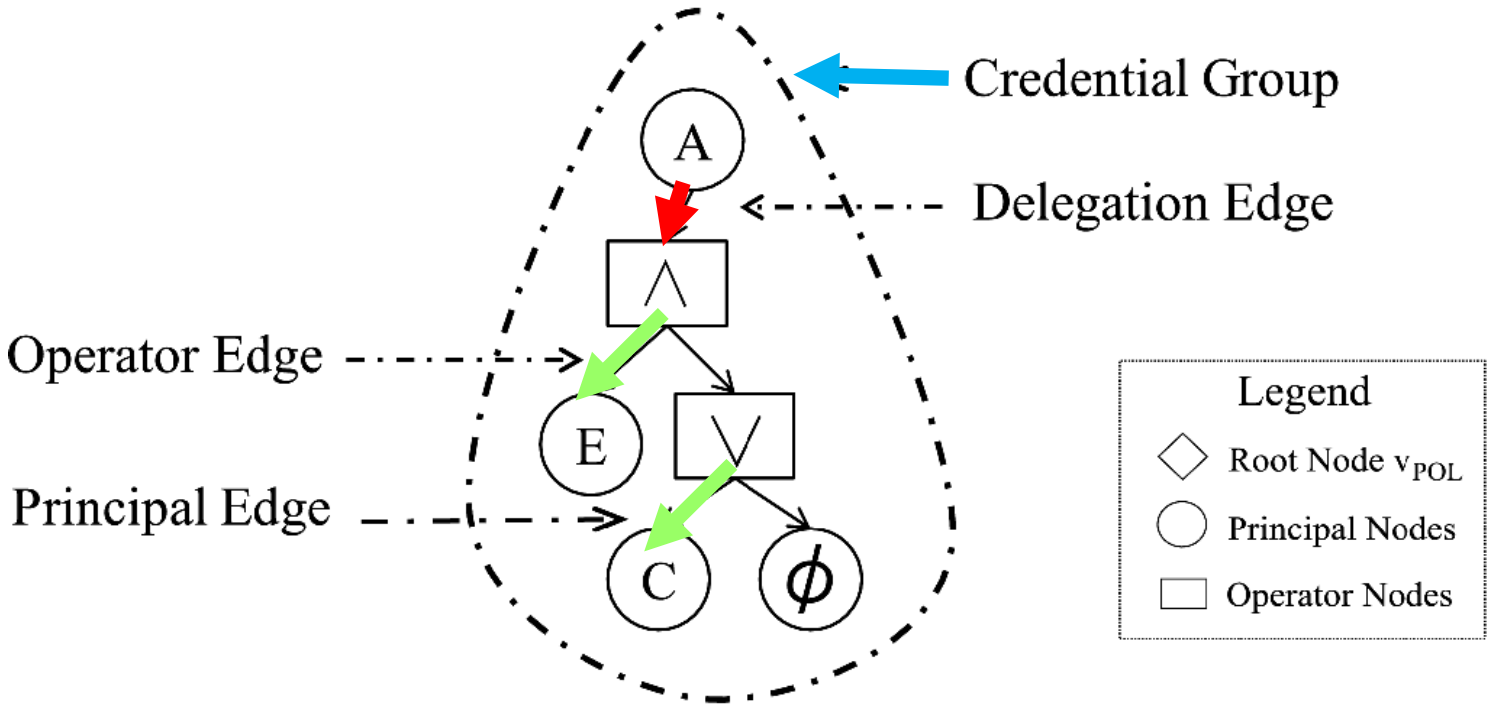
Trust Dependency Graph (TDG), encoding PTM relationships useful for RTM

Reputations of PRINCIPALS, DELEGATIONS and CREDENTIALS are aggregated

The QuanTM Architecture

Types of Reputation

- Reputation of a principal
- Reputation of a delegation by a principal
- Reputation of a credential

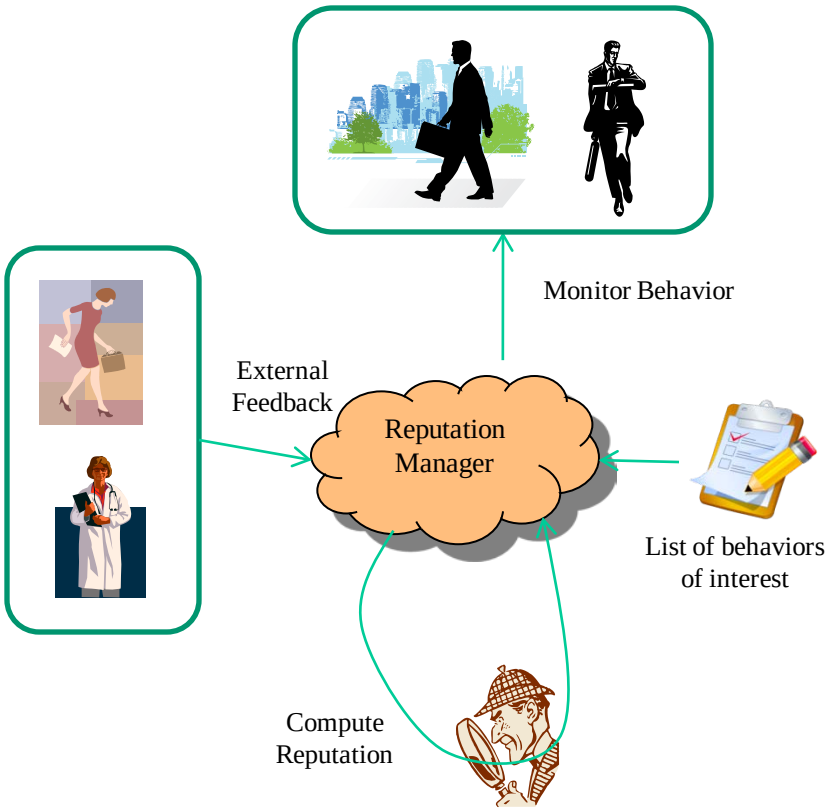


Propagating reputation

- TNA-SL
  - Trust network analysis with subjective logic [Josang 2006]
  - Compute a 4-tuple 'opinion' based on pos/neg feedback: (belief, disbelief, uncertainty, base-rate)
- Combine using discount and consensus operators
  - Consensus averages together two opinions
    - P and Q have an opinion of S
  - Discount is used along transitive chains
    - P has an opinion Q, and Q has an opinion in S
  - Export opinion to a numeric value

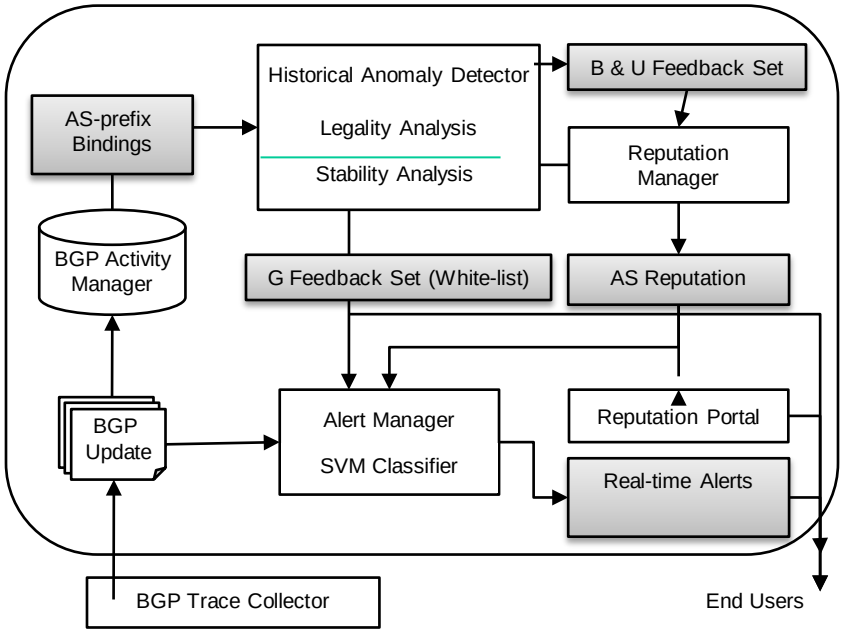
Reputation Computation

- Reputation is an effective tool for modeling consistent user behavior
- Prerequisites for using reputation successfully:
  - a strong user id system
  - identifying user behaviors of interest
  - monitoring behaviors
  - providing feedback



Example: AS-CRED

- Principal Question:
  - Are announced updates valid?
- Key Observation
  - ASes repeat their behaviors
  - Use past to predict future
- Our Approach
  - Reputation as a quantitative trust metric of AS behavior.
- Goals
  - Compute the reputation for Autonomous Systems
  - Provide accurate reputation-based anomaly alert
- Advantages
  - Improved alert accuracy
  - Avoid anomalies by tuning routing policy
  - Incentives for "Do No Evil"



Historical Anomaly Detection: Stability

- Observation
  - Invalid bindings usually last for a short period of time, i.e., they are unstable.
- Use historical data
  - 60 days analysis window
- Two complementary metrics
  - Prevalence: percentage of analysis window binding lasted
  - Persistence: average duration of a binding

Length of Learning Window

$$Pr(p, M) = \sum_i (Tw^i(p, M) - To^i(p, M))/T_{learn}$$

Time prefix (p) withdrawn by AS (M)

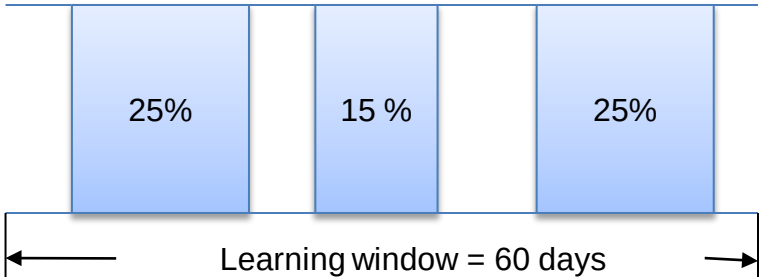
Time prefix (p) announced by AS (M)

$$Ps(p, M) = \sum_i^N (Tw^i(p, M) - To^i(p, M))/N$$

Index of each announcement and withdrawal

Total number of announcements and withdrawals

AS-prefix binding timeline



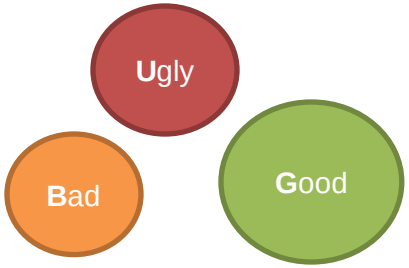
Pr = 65%; Ps = (0.25+0.15+0.25)*60/3 = 13 days

Feedback & Refinement

Initial Classification

Prevalence	Persistence	Feedback
Hi	Hi	Good
Hi	Lo	Bad (Vacillating)
Lo	Hi	Good
Lo	Lo	Ugly (Hijacked)

Feedback Type

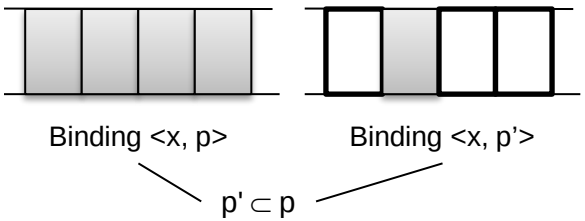


Entry format

AS	prefix	Timestamp of announcement

Refinement

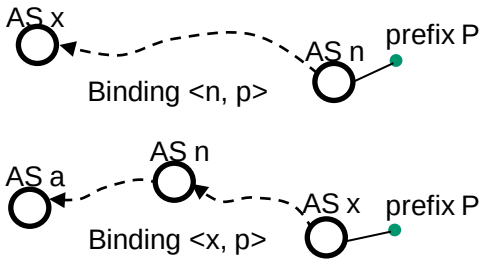
De-aggregation



Before Refinement
<x, p> in G
<x, p'> in U

After Refinement
<x, p> in G
<x, p'> in G

Stable Owner in the Path



Before Refinement
<n, p> in G
<x, p> in U

After Refinement
<n, p> in G
<x, p> ignored

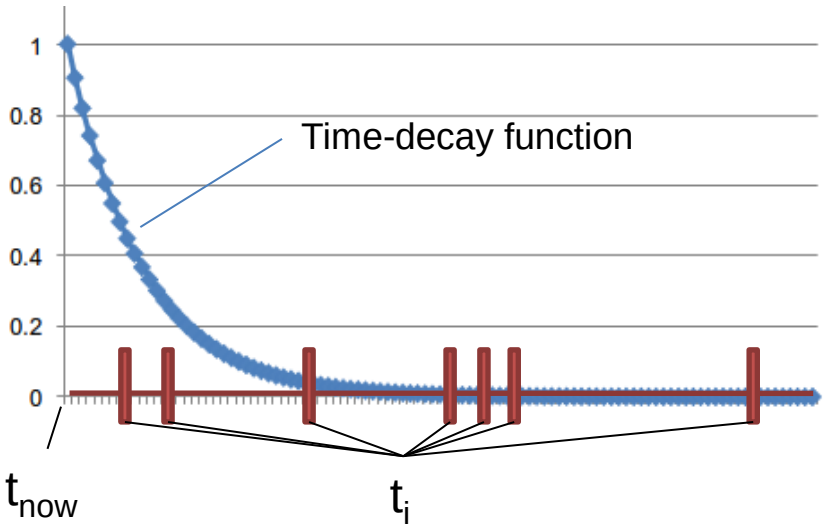
Reputation Computation

- Reputation semantics
  - Untrustworthiness of ASes in announcing valid updates
  - Reputation is computed based on Bad and Ugly feedback
- Time-decay function

$$Rep_X(a) = \sum_{t_i} 2^{-(t_{now}-t_i)/h_X}$$

  - X is either B or U
  - h_X is a half-life of behavior X
  - t_{now} is the current time
  - t_i is the feedback timestamp

- Half-life: set based on by when 75% of the ASes repeat their invalid updates
  - $h_U = 3$ days, $h_B = 6$ days



STiki: Wikipedia vandalism detection

- Based on revision meta-data only
  - Fast and efficient
- Spatio-temporal reputation
  - Use past event history
  - In the absence of history for a given entity, use reputation of similar entities
- Editor reputation is based on rollback history
  - Fully automatic feedback collection
  - Provided by trusted users
  - Does not require vandalism definition

Data Provenance

Whenever data is shared (science, Web, …), a common set of questions:

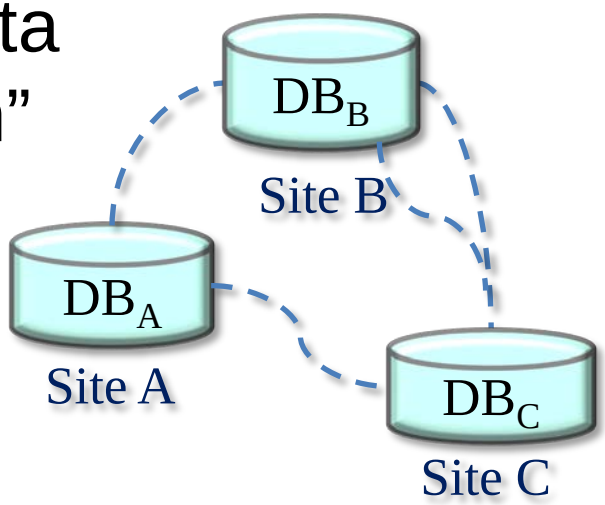
- How did I get this data?
- What operations were used to create the data?
- How much should I trust (believe) it?

Data provenance (lineage) captures the relationships between tuples in a set of data instances

- What is the "data model" of provenance?
- How do we query it? What operations should we support?

Orchestra: Collaborative Data Sharing

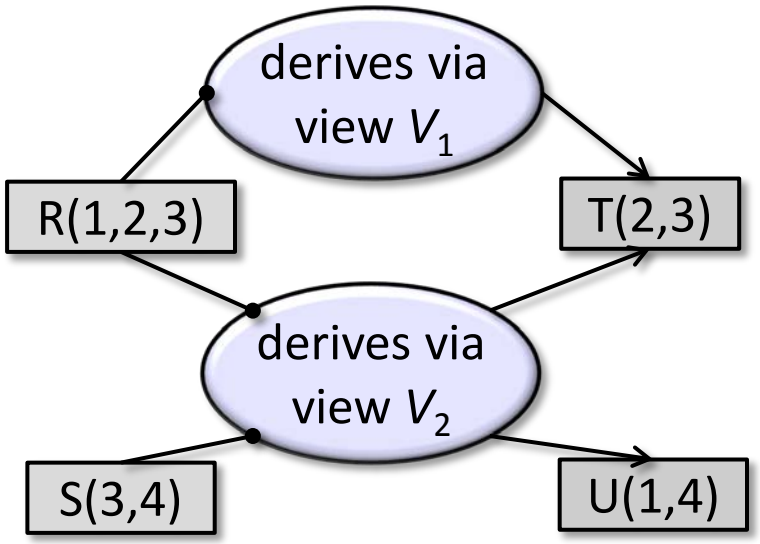
- Formalizes many data interchange settings
- Sites have different schemas, data versions and viewpoints on "truth"
  - Create new data
  - Import (map) data from other sites
    - Determine what data to trust
    - Modify, delete, replace data



- Provenance is critical throughout the system:
  - Enables users to understand their data
  - Distinguishes between different data versions / sources
    - Combined with policies, enables computation of trust
  - Incremental update propagation

What Is Data Provenance?

- A model of the relationships among tuples in source and derived tables
- Annotations on each tuple describing derivation:
  - An $\oplus\otimes$-algebraic expression in terms of "neighbor" tuples



T(2,3)	V_1(R(1,2,3)) $\oplus$ V_2(R(1,2,3) $\otimes$ S(3,4))
…	

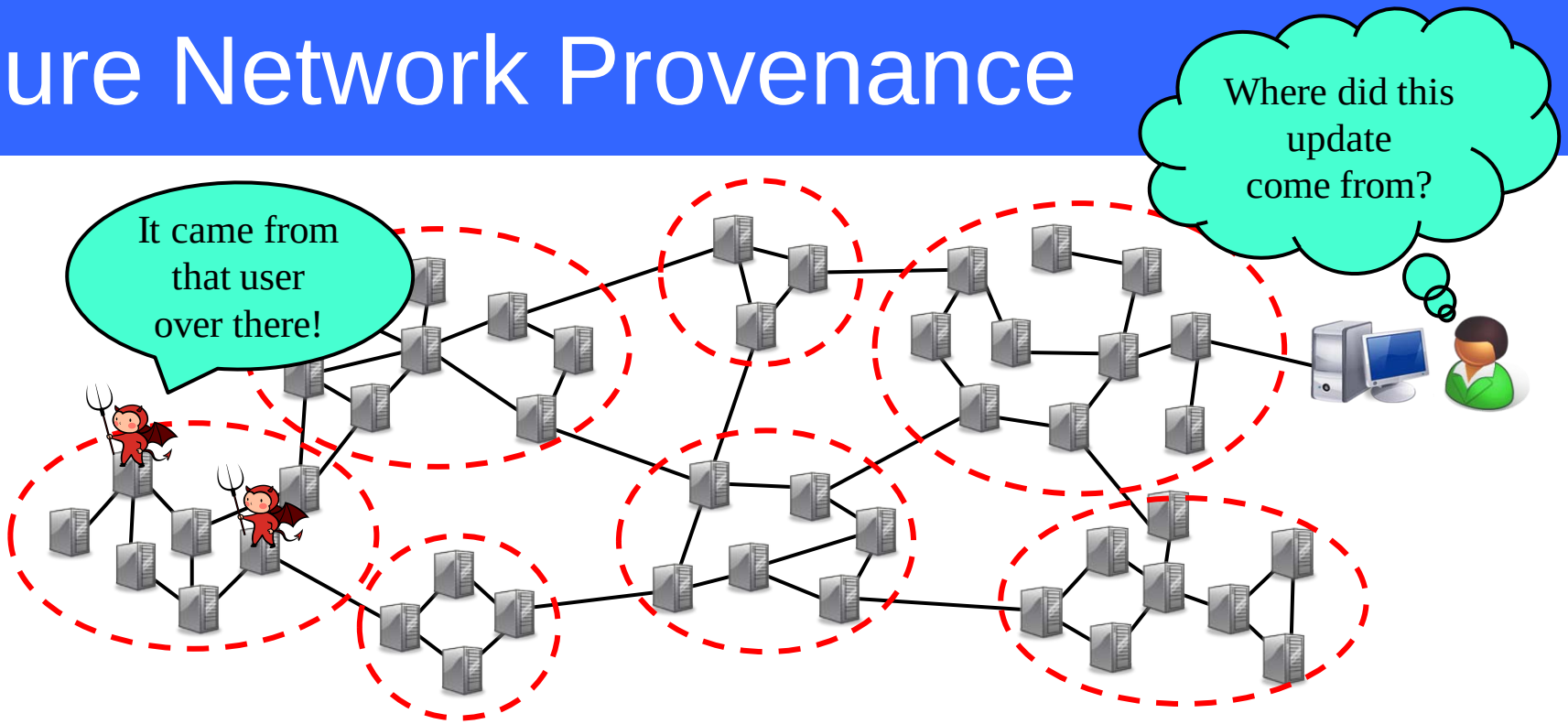
U(1,4)	V_2(R(1,2,3) $\otimes$ S(3,4))
…	

View V_1	View (Mapping) V_2
T(b,c) :- R(a,b,c)	T(b,c) :- R(a,b,c), S(c,d) U(a,d) :- R(a,b,c), S(c,d)

Provenance as Annotations

- We can annotate each tuple with an algebraic expression over the IDs of directly-related tuples
  - Abstract $\oplus$ operator for union or projection
  - Abstract $\otimes$ operator for join
  - Abstract function name for each mapping/view
- Operators should form a commutative semiring
  - $\otimes$ distributes over $\oplus$; exists 0, $a \oplus 0 = a$, $a \otimes 0 = 0$, etc.
- Compute tuple annotations:
  - Counts of the number of tuple derivations
  - Weights or trust levels
- ProQL: scalable distributed query language

Secure Network Provenance



- Provenance is key to evaluating the quality of information
- What if the network is attacked?
  - Attacker might change data or modify its provenance to make them appear more (or less) trustworthy
- Goal: We should detect such attacks
  - We want to avoid trusting any single node, or small set of nodes

Secure Provenance

- Idea: Make provenance data structures tamper-evident
  - Modifications can be kept in an append-only log, with entries connected by a hash chain
  - Nodes can exchange cryptographic commitments to detect modifications or inconsistencies
- When tampering is detected, the other nodes will be able to obtain evidence
  - I.e., a "proof" that a given node has tried to tamper with the provenance of data
  - It is provably impossible for an attacker to obtain valid evidence against a correct node

Summary: TrustForge Features

- Different levels of access control
  - Policy-based guarantees via cryptographic credentials
    - For highly critical projects
  - Reputation-based flexibility via user action feedback
    - For non-critical projects
  - Policy and reputation combinations (decision policies)
    - Everything in between
- Separate reputations for users and components
  - Mutually dependent
  - Different reputation functions
- Repository interface to facilitate reuse