Max
Planck
Institute
for
Software Systems

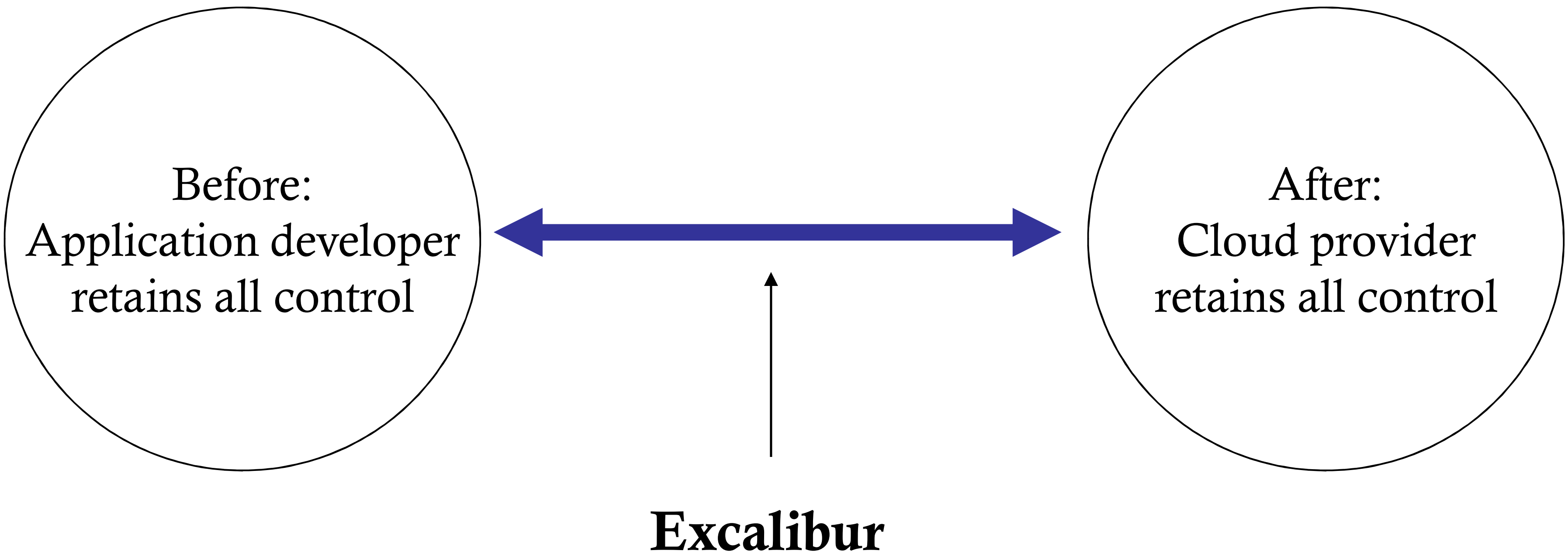
Gaining Customer Trust in Cloud Services with Excalibur

Rodrigo Rodrigues
MPI-SWS

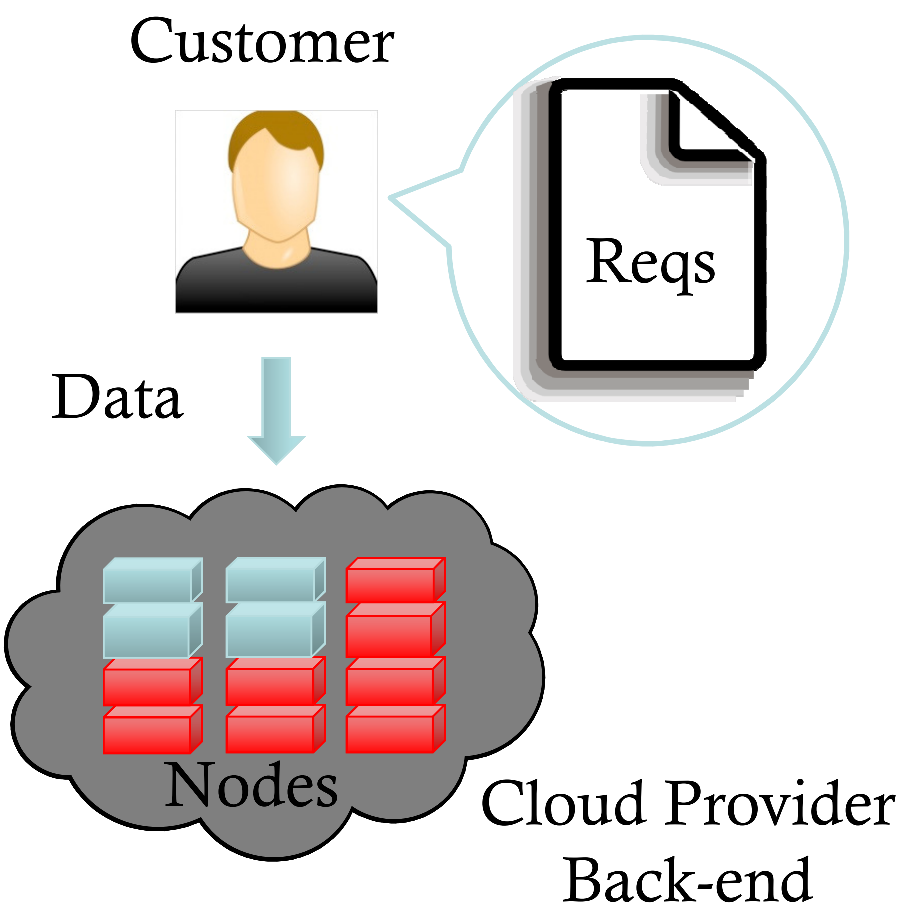
Joint work with Nuno Santos,
Krishna Gummadi, and Stefan Saroiu

Problem: Customers Surrender Full Control of their Data

BC versus AC


Before:
Application developer
retains all control
After:
Cloud provider
retains all control
Excalibur

Proposal: Share Control between Customers and Providers



Costumer specifies requirements for how data can be handled

Node Configurations

- Provider: publishes
- Customer: chooses
- Provider: enforces
- **Attribute-value pairs** for cloud nodes:
  - Software
  - Location
  - Hardware
- Customers define **policies** over configurations

Node configuration

service : “EC2”
version : “4.0.1”
country : “Germany”
zone : “z1”
type : “small”

Policy

service = “EC2”
and
version > “4.0”
and
(country = “Germany”
or
country = “UK”)

Trust Model

- Software administrator → adversarial role
- Platform developer → mitigation role
- Cloud provider → trusted to deploy defenses (but no control over all administrators)

- Threats against integrity and confidentiality of customer data

Trusted Platforms

- Configuration that implements expected behavior (integrity, confidentiality)
- Enforces this despite adversarial sysadmin behavior

- E.g., modified Xen to cripple admin control
- Still allows for migration, suspension, resumption of VM

Challenges

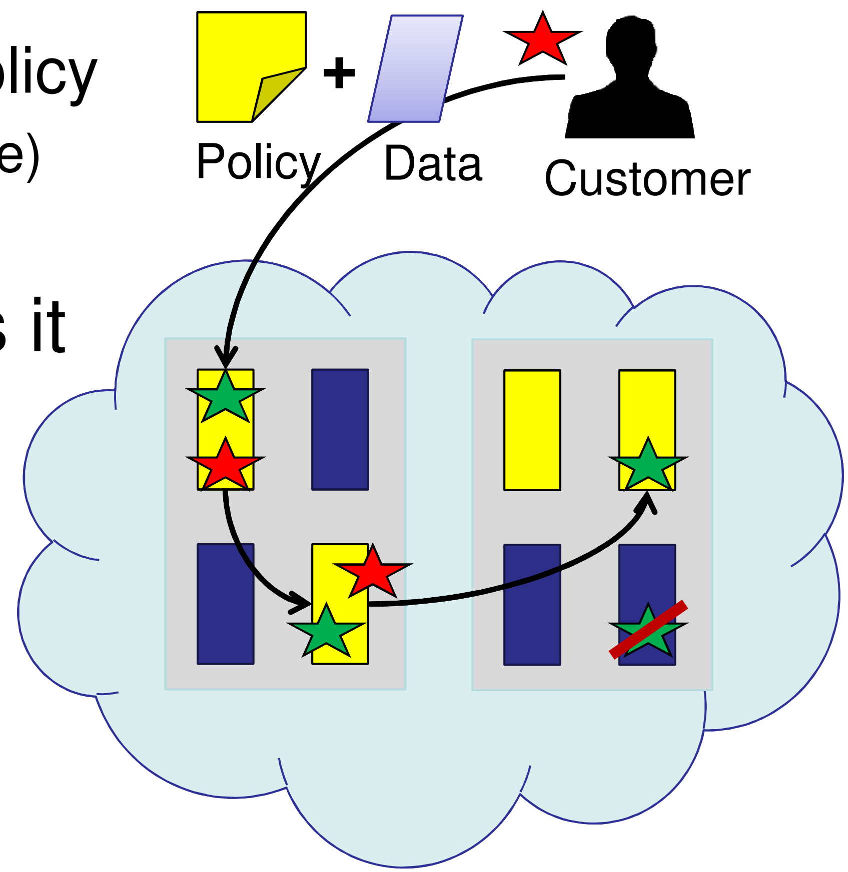
- Untrusted admin can:
  - Misuse interface that is provided
  - Reboot node into different platform
  - Listen network traffic
- How to convey guarantees to the user?

Policy-Sealed Data

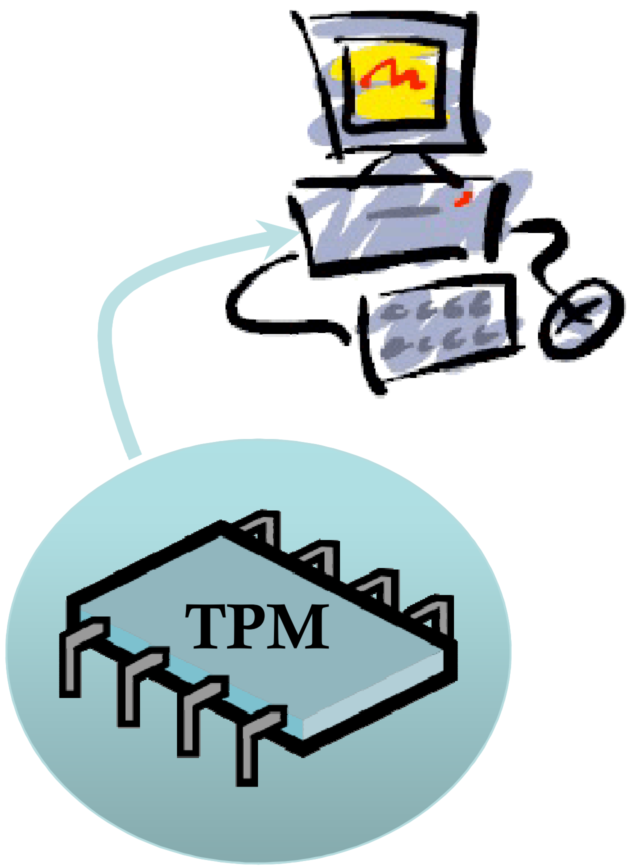
- Allow both customers and cloud apps to cryptographically **seal** data to policy
- Can only be **unsealed** by nodes obeying policy

Policy-Sealed Data Usage

- Data must be sealed (★) upon:
  - Costumer upload
  - Leaving node that obeys policy
    - Network transmission (migrate)
    - Disk (suspend)
- Must unseal (★) to access it



Implementing Policy-Sealed Data



- Leverage TPMs
  - Widely available
  - Enable remote query about node characteristics
- Adopted in real systems
  - E.g., BitLocker

TPM Characteristics

- Strong identities
  - Per-node identity key
- Trusted boot
  - Chain of measurements that computes the platform fingerprint upon boot.
  - Fingerprint cannot be forged or overwritten

TPM Characteristics

- Remote attestation
  - Allows a remote party (challenger) to authenticate the node and platform.
- Sealed storage
  - Allows sensitive data to be securely stored across reboots.
  - Seal/Unseal primitives: encrypt the data, can only be decrypted on the same node running the same platform

Using TPMs in the Cloud

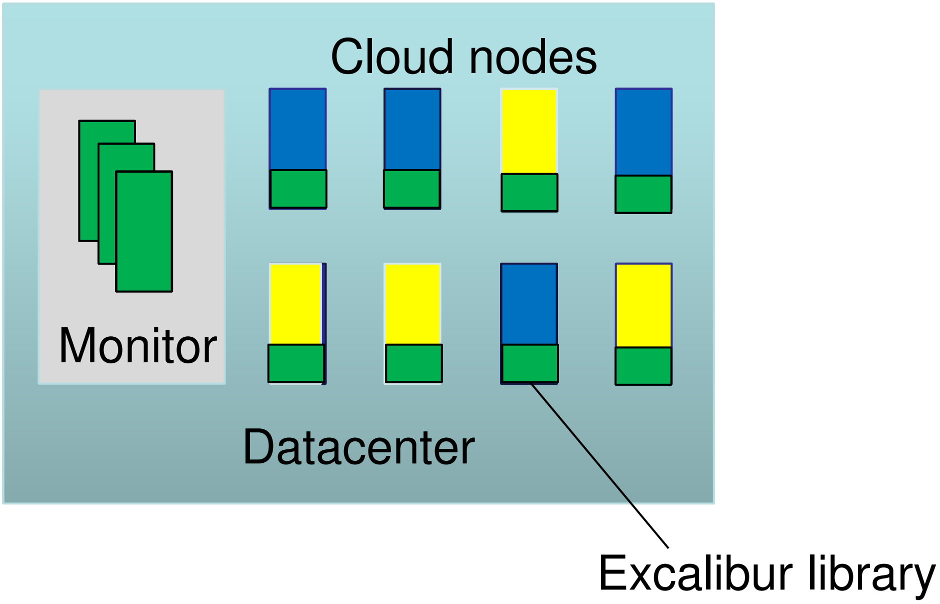
- Challenge 1: Who attests nodes in cloud back-end?
  - If customers, then exposes provider’s infrastructure
  - If cloud services, need to modify all services to attest before data leaves nodes.
    - Complex
    - Inefficient

Using TPMs in the Cloud

- Challenge 2: Avoid performance penalty
  - Attestation is painfully slow
  - How to handle membership changes efficiently?

- Challenge 3: Scalability
  - Work efficiently despite 1k..1M nodes

System Architecture



System Interface

- `ciphertext = sheathe(data, policy)`
- `data = unsheathe(ciphertext)`

- Example policy:

  service = “Xen” and version>4.0

  and country = “DE”

Dynamic attrs.

Static attrs.

Monitor Main Tasks

1. Keep track of mappings:
   - static attributes to hosts
   - dynamic attributes to low-level TPM measurements
2. Attest cloud nodes upon booting
3. Generate and distribute special credentials
4. Attest to clients that cloud service is trustworthy

Attribute-Based Encryption (CP-ABE)

1. Setup: Generate <public,master> keys
2. Create decryption keys:

   Master key + Attributes (string,int) → Decryption key
3. Encrypt data:

   Public key + Attributes + Plaintext → Cyphertext
4. Decrypt data:

   Cyphertext + Decryption key → Plaintext

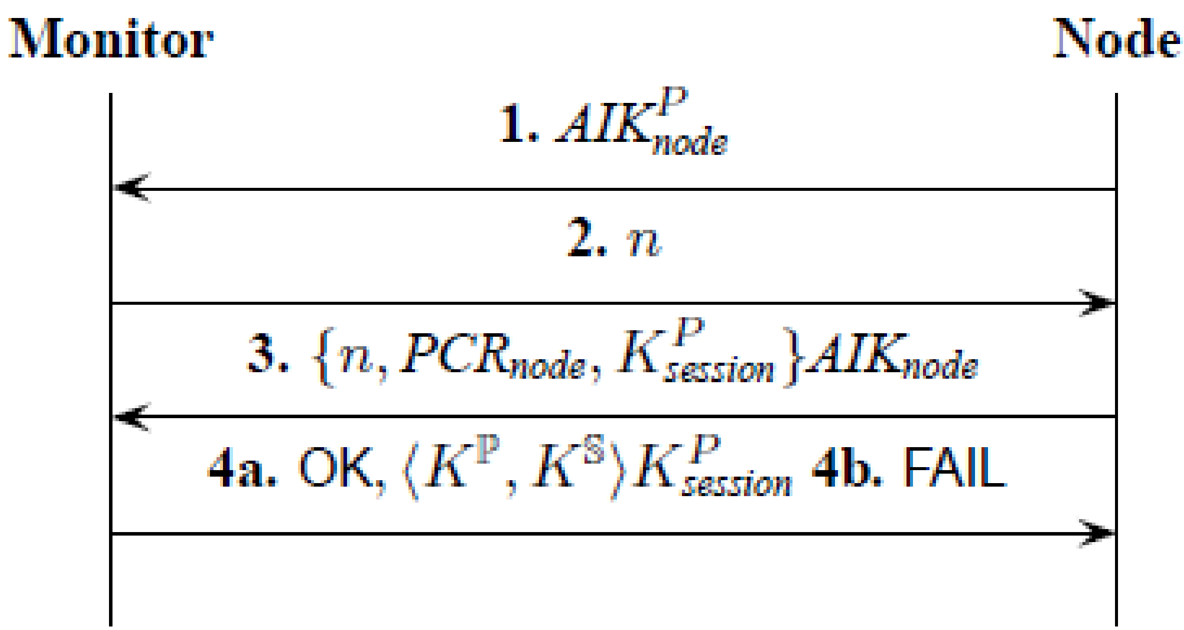
Attribute-Based Encryption (CP-ABE)

- Goal: Run sheathe/unsheathe locally
- Leverage CP-ABE:
  - Embed attributes in decryption keys
  - Generate and distribute one per cloud node boot
  - Sheathe uses public key and embeds policy

But, CP-ABE is not perfect…

- CP-ABE is slow
  - Both generation of CP-ABE decryption keys, and
  - Encryption and decryption (slower than RSA)
- To prevent node authentication bottleneck
  - Pre-generate the decryption keys
  - Reuse capability for same configuration
- To reduce performance impact to seal/unseal
  - Encrypt a symmetric key rather than the data

Attesting cloud nodes upon boot

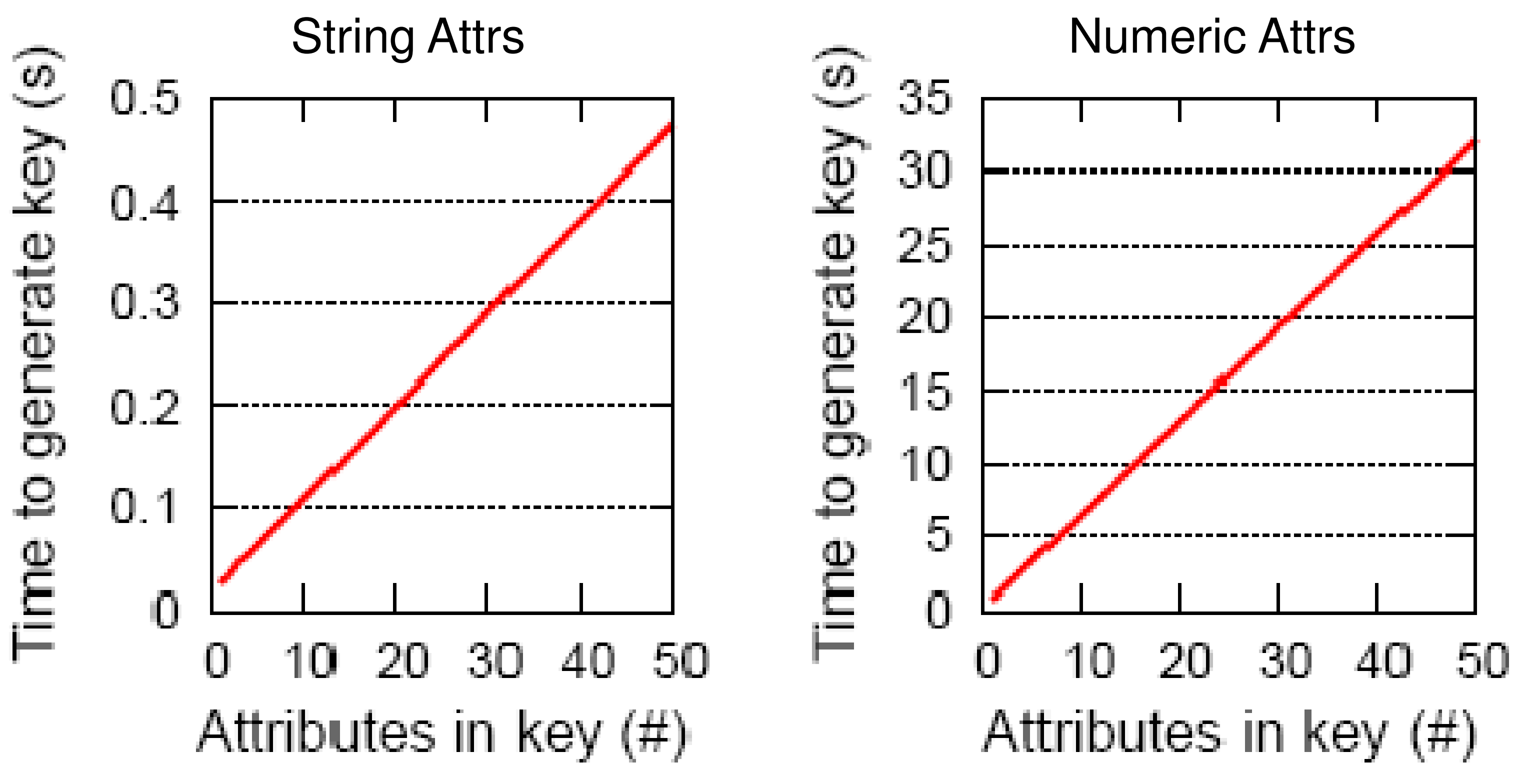


1. Attest to identity + dynamic attributes
2. Deliver special keys that encode attributes

Example App: VM Rental

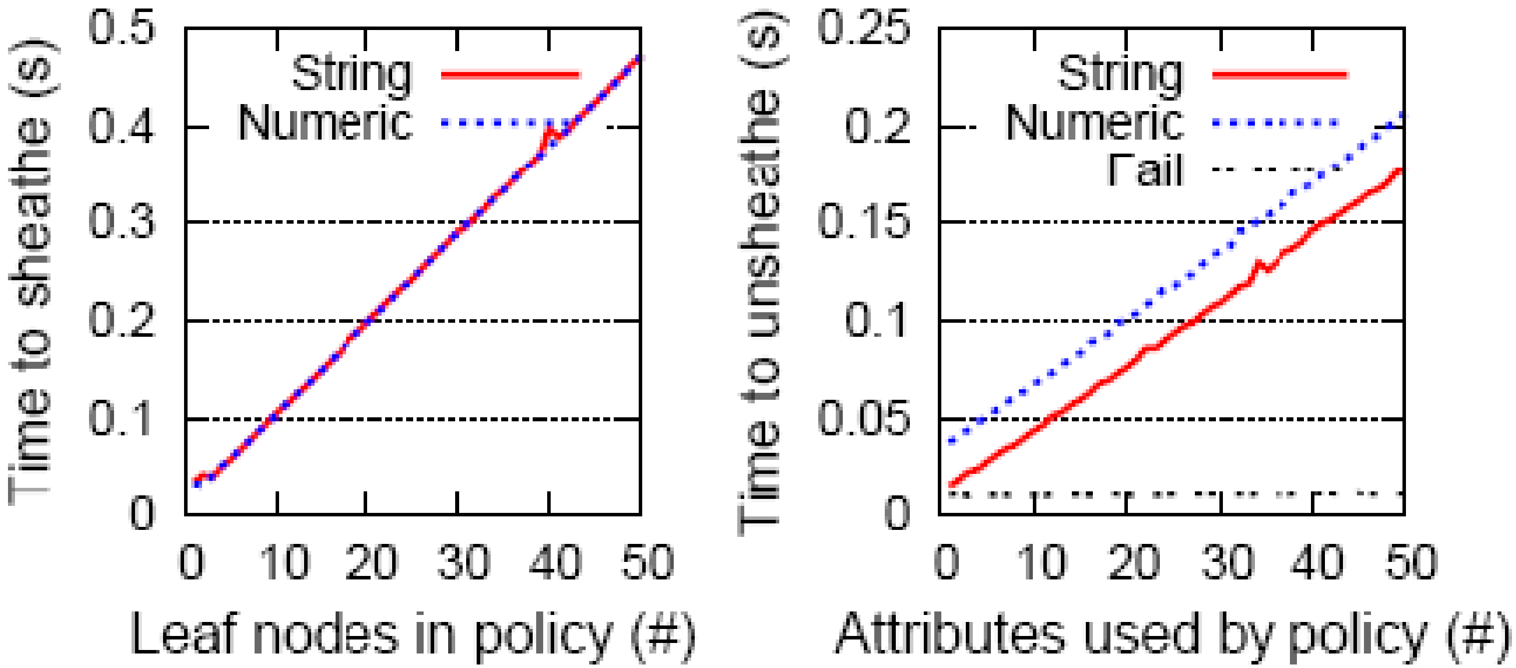
- EC2-like service based on Eucalyptus/Xen
- Extra assurance: VMs must be confined to certain locations
- Simple changes to Eucalyptus/Xen
  - Added/modified 52 lines of code in create, save, restore, migrate
- Other possible features: prevent curious sysadmin from accessing VM image

Evaluation: Generating CP-ABE Keys



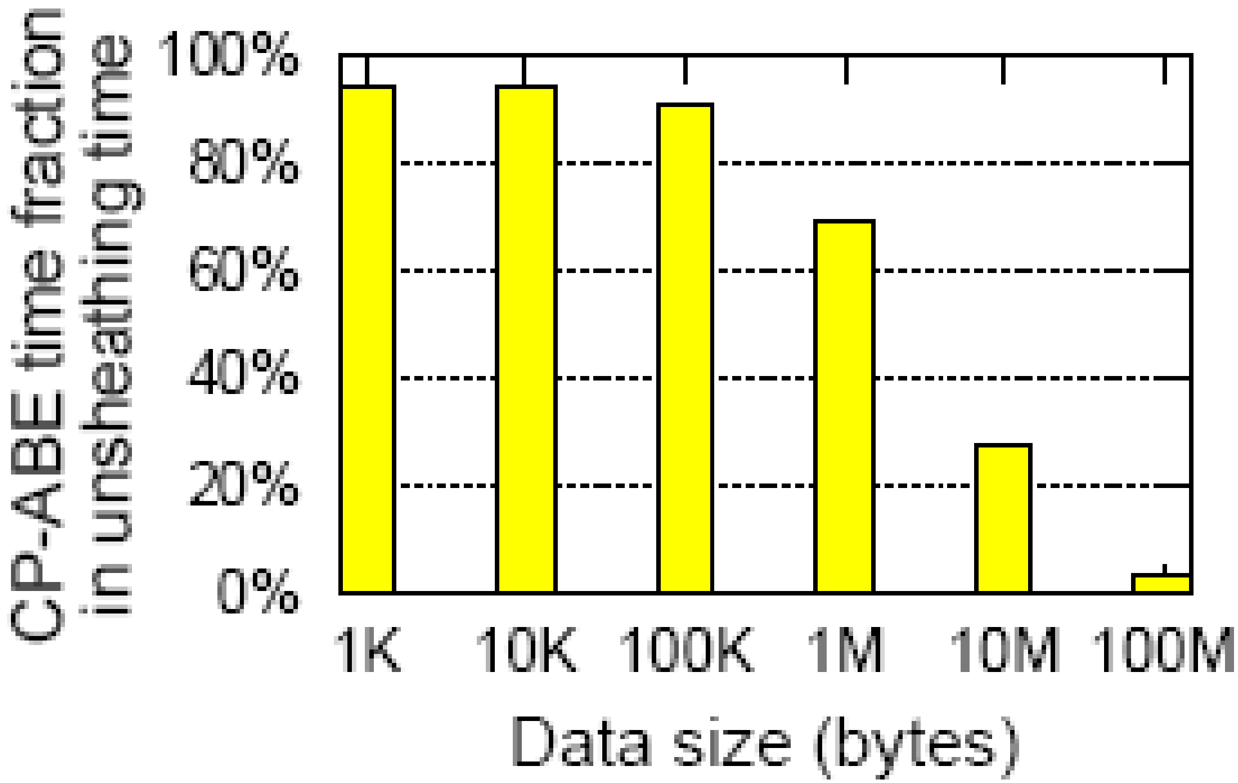
- Only need to generate one per class of hosts with same attributes

Sheathing and Unsheathing



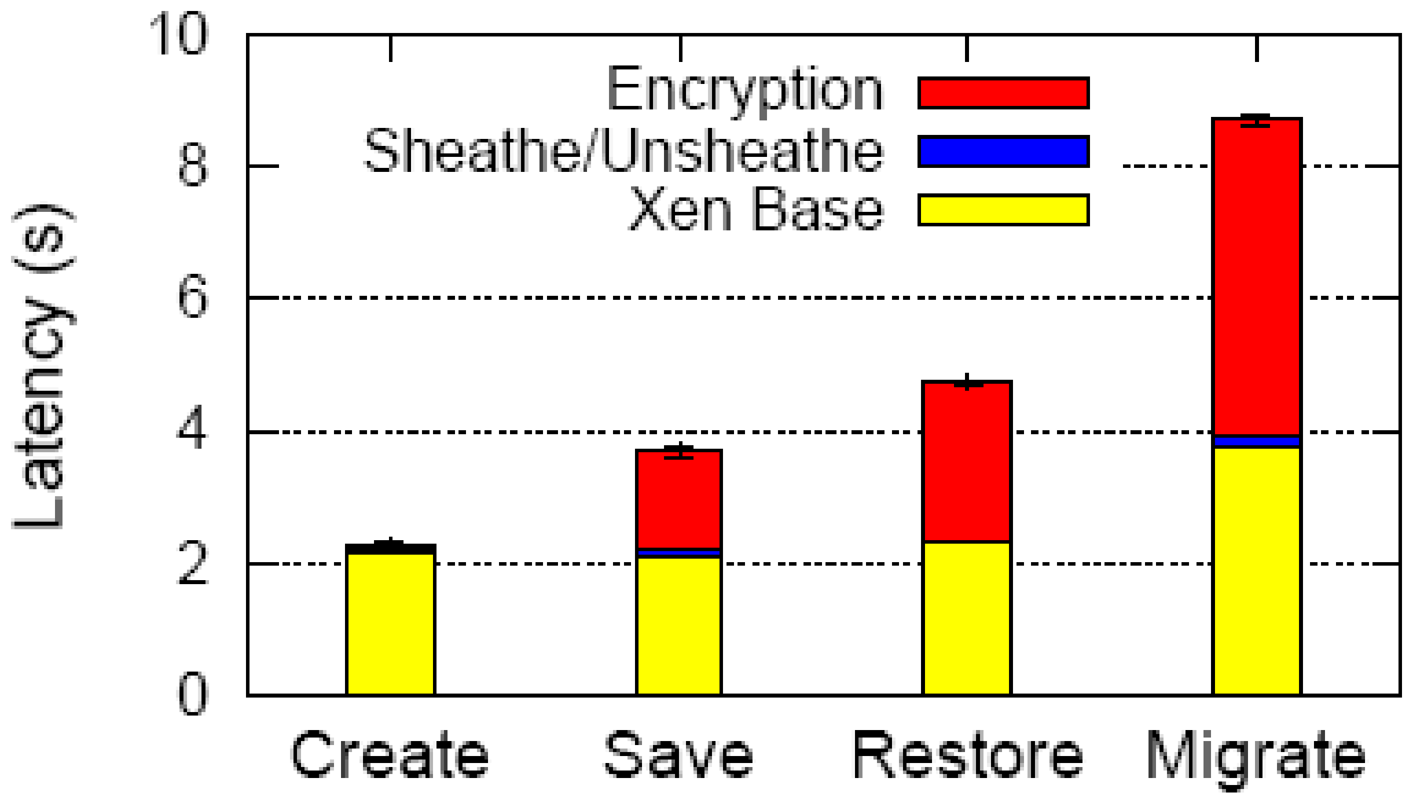
- 1 KB data object (CP-ABE dominates)

Overhead from ABE



- For large data, sheathe symmetric key and encrypt data

Performance of VM Rental



- Most delay is inevitable (encrypting)

Conclusion

- Excalibur implements policy-sealed data
  - Shifts some control over data from cloud providers to customers
- Leverages important technologies
  - TPMs
  - ABE
- Demonstrate usefulness by building EC2-like system with stronger guarantees
- Future: Build “real” trusted platforms