Protocols for Secure Cloud Computing
Where is my data?

1986

2011
Who runs my computation?

1986

2011
Overview

1. Cloud computing and its security
2. Intercloud storage - Replication across clouds
3. Storage integrity
4. Cryptographic protocols
   - Proof of storage
   - Fully homomorphic cryptosystems
5. Conclusion
Cloud computing security
Cloud computing

- On-demand network access to a shared pool of configurable computing resources
  - networks, servers, storage, applications, and services
  - rapidly provisioned with minimal management effort and provider interaction

- Key features
  - On-demand self-service
  - Accessible over the network "from everywhere"
  - Resource pooling → provider distributes cost over many customers
  - Rapid elasticity → quickly scale to large sizes
  - Measured service → pay only for actually consumed resources

[NIST, 2009]
Cloud service models

Customer manages

Provider manages

SAAS
Collaboration
CRM/ERP/HR
Software as a Service

PAAS
Web 2.0 Application Runtime
Java Runtime
Platform as a Service

IAAS
Servers
Data Center Fabric
Networking
Storage
Infrastructure as a Service

Provider manages

Yahoo!, GMail, Google Docs, LotusLive, salesforce.com, sourceforge.net

SQL Azure, Google Apps, IBM test&devel., IBM SONAS

Amazon (S3, EC2), IBM, Rackspace, Windows Azure
Cloud deployment

- **Public clouds**
  - Provider offers service to the public
  - Standardized, large-scale, flexible, cheap
  - Resources shared by multiple tenants
  - Concerns about security

- **Private clouds**
  - Service limited to one enterprise
  - Customized, dedicated functions, local control, expensive
  - One or few tenants that trust each other
  - Increased security

- **Hybrid clouds**
  - Private infrastructure integrated with public cloud
  - Combines trust in private cloud with scalability of public cloud
Cloud storage - secure?

- Red Hat's servers were corrupted in Aug. '08
  - Package-signing key potentially exposed
  - Was source or binary content modified?
  - Red Hat stated in RHSA-2008:0855-6:
    ... we remain highly confident that our systems and processes prevented the intrusion from compromising RHN or the content distributed via RHN and accordingly believe that customers who keep their systems updated using Red Hat Network are not at risk.
Cloud computing - dependable?

- A problem in Amazon's cloud services disabled computing and storage services for one day
- Clients were affected, including websites that use Amazon's resources
- Problem affected multiple, supposedly independent "zones" in Amazon's infrastructure
Key concerns about cloud computing

Less Control
Many companies and governments are *uncomfortable* with the idea of their information located on *systems they do not control*. Providers must offer a high degree of security transparency to help put customers at ease.

Data Security
Migrating workloads to a *shared* network and compute *infrastructure* increases the potential for *unauthorized exposure*. Authentication and access technologies become increasingly important.

Reliability
High availability will be a key concern. IT departments will worry about a *loss of service* should outages occur. Mission critical applications may not run in the cloud without strong availability guarantees.

Compliance
Complying with SOX, HIPPA and other *regulations may prohibit* the use of clouds for some applications. Comprehensive auditing capabilities are essential.

Security Management
Providers must supply easy, visual controls to *manage firewall and security settings* for applications and runtime environments in the cloud.
How to address the concerns

- **Less control**
  - Providers become transparent and offer sophisticated controls

- **Data security**
  - Isolate multiple tenants from each other
  - Implement secure authentication, authorization, and identity management
  - Critical data must be encrypted and integrity-protected by client

- **Reliability**
  - Today, mission-critical applications do not run in the cloud
  - Highly available clouds in the future

- **Compliance**
  - Offer auditing capabilities to clients
  - Provider runs audited service (for a premium)

- **Security Management**
  - Client-controlled features important
Cloud security

- **Security implemented by provider** - **NOT A PROTOCOL SOLUTION!**
  - Isolation among resources of different tenants
    - Hypervisor
    - Storage
    - Network (VLAN)
  - Restrict administrator privileges on hosting systems
  - Strong authentication, authorization, and identity management
  - Interfaces for direct client-controlled audits
  - Engage third-party auditors

- **Mechanisms implemented by clients**
  - Cryptographically protect the data
    - Encryption
    - Integrity protection
  - Remote auditing

- **Client must trust provider ...**
Intercloud storage
The Intercloud

- Cloud of clouds
- Client accesses it like a single cloud
- Service composed from multiple clouds
- Analogous to the "Internet"
  - Local-area networks → Internet (TCP/IP)
  - Isolated clouds → Intercloud (open protocols for interoperation, WS-* ...)
Intercloud storage

- Storage on the Intercloud
  - Limits trust in single provider

- Features
  - **Confidentiality**
    - Transparent encryption, keys managed by the Intercloud
  - **Integrity**
    - Data authenticity from cryptographic protection
  - **Resilience**
    - Replication tolerates data loss/corruption in a fraction of clouds
Key-value stores

- Popular storage interface using unstructured objects ("blobs")
  - Every object identified by a unique key
  - Objects grouped into containers

- Available in Amazon's Simple Storage Service (Amazon S3) and many, many others
  - Accessed via REST web interface

- Main operations
  - put(key, obj)
  - get(key) → obj
  - list() → {objs ...}
  - remove(key)
Intercloud storage stack

- Modular structure
- Layers are configurable and switchable  
  - E.g., encryption-only with single cloud
- Transparent to client
- No modification to clouds
- Multiple clients  
  - No client-to-client communication  
  - Clients may fail (crash)
- Client locally stores credentials
Intercloud storage - Confidentiality with encryption
Encryption

- Encryption with standard block cipher
  - AES
  - Secret key needed

- Objects may become slightly larger

- Software encryption
Key management

- Two options
  - Standardized key-management service
  - Keys managed in the Intercloud
Key management as a service

- Key management is becoming a service
  - Centralized control
  - Lifecycle management
  - Automated and policy driven

- Very important for storage encryption

- **OASIS Key Management Interoperability Protocol (KMIP)**
  - Vendor-neutral format for accessing key server in enterprise
  - Finalized Oct. 2010, available already in multiple products
  - Contributions from IBM Research [BCHHKP10]
Secret sharing for managing keys in the intercloud

- Share secret $s$ among parties $p_1, \ldots, p_n$ such that
  - Any $t < n/2$ parties have no information about $s$
  - Any group of $t+1$ parties can recover the secret $s$

- Trusted dealer picks random polynomial $a(X)$
  - $a(X) \in \mathbb{F}_q[X]$, degree $t$ and $a(0) = s$

- Share for $p_i$ is $s_i = a(i)$

- Given set $U$ of $t+1$ shares, recover secret: $s = a(0) = \sum_{j \in U} \lambda_j a_j$
  - $\lambda_i$ are Lagrange coefficients w.r.t. $U$

[S79]
Intercloud storage - Integrity with hashing and signatures
Integrity protection for one client

- Storage consists of $n$ data items $x_1, \ldots, x_n$ (objects in the same container)

- Client accesses storage via integrity layer
  - Uses small trusted memory to store short reference hash value $v$ (together with encryption keys)

- Integrity layer operations
  - Read item and verify w.r.t. $v$
  - Write item and update $v$ accordingly
Implementing the integrity layer

- Use hash function $H$ to compute $v$?  
  $v = H(x_1 || ... || x_n)$
  - Infeasible for many objects
  - No random access to one object
- Use a secret key with a MAC?  
  - Suffers from replay attacks
- Well-known solution: Hash tree [M79]
  - Overhead of read/verify and write/update is logarithmic (in $n$)
- Recent alternatives
  - Dynamic accumulators [CL02]
    - Overhead of read is constant, but write is linear in $n$
  - Incremental hashing [BM97]
    - Overhead of write/update is constant, but read is linear in $n$
Hash trees for integrity checking (Merkle trees)

- Parent node is hash of its children
- Root hash value commits all data blocks
  - Root hash in trusted memory
  - Tree is on extra untrusted storage
- To verify $x_i$, recompute path from $x_i$ to root with sibling nodes and compare to trusted root hash
- To update $x_i$, recompute new root hash and nodes along path from $x_i$ to root

Read & write operations need work $O(\log n)$
- Hash operations
- Extra storage accesses
Multi-client integrity protection with digital signatures

- **Single-client solution**
  - Relies on hash value \( v \)
  - Stored locally
  - Changes after every update operation

- **Multiple clients?**
  - Every client associated with a public/private key pair
  - Write operation produces signature \( \sigma \) on hash \( v \)
  - Client stores signature and hash \((\sigma, v)\) on cloud
  - Allows replay attacks ...

- **See next part of presentation!**
Intercloud storage - Resilience with replication
Replication

Clients read and write object values
- Do not communicate
- No clock synchronization

Storage nodes replicate data
- Faulty nodes may erase or modify data
- Do not communicate with each other
Replication algorithm

- Clients read and write objects (values)
- Client operations take time and may execute concurrently
  - No locks!
  - No single point of failure!
  - Clients may fail!

- **Algorithm ensures a consistent view of single storage object**
  - If no operation is concurrent, then every read returns the most recently written value
  - Otherwise, read may return old value (written before) or new value (written concurrently)

- **Algorithm emulates a shared memory**
  - Many other consistency conditions exist
  - Most strict ensures that all operations appear atomic

- Implementation based on logical timestamps (sequence numbers)
Quorum algorithm

- Here nodes may only crash

- A quorum is a majority of the storage nodes
  - More generally: every two quorums overlap in one node
  - With \( n \) nodes, every set of \( > n/2 \) nodes is a quorum

- Data structure
  - A node stores a timestamp/value pair \((ts, v)\)

- Read
  - Send \([READ]\) message to all nodes
  - Receive \([VALUE, ts, v]\) msgs from nodes in a quorum and return \(v\) with the highest \(ts\)

- Write\((v)\)
  - Determine highest timestamp \(ts\) used so far, let \(ts' ← ts+1\)
  - Send \([WRITE, ts', v]\) to all nodes; nodes reply with \([ACK]\)
  - Receive \([ACK]\) msgs from nodes in a quorum and return
Quorum algorithm example

Alice
write(b) with ts=4

Bob
write(c) with ts=5
write(y) with ts=6

Charlie
read() → y

Time

No reply from faulty nodes
Quorum algorithms with malicious nodes

- Nodes may behave arbitrarily: more difficult
  - See presentation by Marko Vukolic (Friday)
Intercloud storage - Summary

- **Cloud computing, the computing services of the future**
  - Storage is perhaps the most prominent example

- **Security problems of cloud computing**
  - Provider not trusted
  - Multiple tenants share infrastructure

- **Intercloud storage protects data stored in cloud**
  - Confidentiality through encryption
  - Integrity through cryptographic hashing and signatures
  - Resilience through replication
    - Modular, layered architecture
    - Prototype being developed
Storage integrity
System model

- **Server** $S$
  - Normally correct
  - Sometimes faulty (untrusted, Byzantine)

- **Clients** $C_1 \ldots C_n$
  - Correct, may crash
  - Run operations on server
  - Disconnected
  - Small trusted memory

- Asynchronous
Storage model

- Functionality MEM
  - Array of registers $x_1 \ldots x_n$
  - Two operations
    - $\text{read}(i) \rightarrow x_i$ returns $x_i$
    - $\text{write}(i,x) \rightarrow \text{ok}$ updates $x_i$ to new value $x$

- Operations should be atomic

- Abstraction of shared memory

- Most work on forking consistency conditions considers MEM
  [MS02] [LKMS04] [CSS07] [CKS09] ...
Storage integrity protection

- Clients interact with service through operations to read/write data

- Clients may *digitally sign* their write requests
  - Server cannot forge read values
  - But answer with outdated values ("replay attack")
  - But send different values to different clients (violates consistency)
Background - Semantics of concurrent operations

**Safe** - Every \textit{read} not concurrent with a \textit{write} returns the most recently \textit{written} value.

**Regular** - \textit{Safe} & any \textit{read} concurrent with a \textit{write} returns either the most recently \textit{written} value or the concurrently \textit{written} value: \(C_3\) may read \(x\) or \(u\).

**Linearizable (atomic)** - All \textit{read} and \textit{write} operations appear to execute atomically at one point in time: \(C_3\) must read \(u\).
Linearizability illustrated

- Every operation appears to execute **atomically** at its linearization point.

- This point lies between invocation and response in (imaginary) real time.
Problem - Integrity violation

C₁

write(1,x) write(1,u) write(1,t)

write(2,v) read(1)→x write(2,w)

read(1)→u read(2)→w

C₂

C₃
Solution - Fork-linearizability

- Server may present different views to clients
  - “Fork” their views of history
  - Clients cannot prevent this

- Fork linearizability [MS02]
  - If server forks the views of two clients once, then
    → their views are forked ever after
    → they never again see each others updates

- Every inconsistency results in a fork
  - Not possible to cover up

- Forks can be detected on separate channel
  - Best achievable guarantee with faulty server
Fork-linearizability graphically

$w(1,x)$ $w(1,u)$ $w(1,t)$

$w(2,v)$ $\text{read}(1) \rightarrow x$ $w(2,w)$

$\text{read}(1) \rightarrow u$ $\text{read}(2) \rightarrow w$

$\text{View of } C_1$

$\text{View of } C_2$

$\text{View of } C_3$
Linearizability formally

A history $\sigma$ is linearizable (with respect to $F$)

$\Leftrightarrow \exists$ permutation $\pi$ of $\sigma$ such that
- $\pi$ is sequential and follows specification (of $F$);
- $\forall i$ all operations of $C_i$ are in $\sigma$;
- $\pi$ preserves real-time order of $\sigma$. 
Fork-linearizability formally

A history $\sigma$ is fork-linearizable (w.r.t. $F$)

$\iff \forall i \exists$ subset $\sigma_i \subseteq \sigma$ and permutation $\pi_i$ of $\sigma_i$ such that
- All operations of $C_i$ are in $\sigma_i$;
- $\pi_i$ is sequential and follows specification (of $F$);
- If $o \in \pi_i \cap \pi_j$, then $\pi_i = \pi_j$ up to $o$;
- $\pi_i$ preserves real-time order of $\sigma_i$. 
Protocol $P$ emulates functionality $F$ on a Byzantine server $S$ with fork-linearizability, whenever

- If $S$ correct, then history of every (...) execution of $P$ is linearizable w.r.t. $F$;

- The history of every (...) execution of $P$ is fork-linearizable w.r.t. $F$.

[CSS07]
A trivial protocol

- Fork-linearizable Byzantine emulation of MEM

- Idea [MS02]: sign the complete history of read/write operations
  - Server sends history with all signatures
  - Client verifies all operations and signatures
  - Client adds its operation and signs new history

- Impractical since messages and history grow with system age
Fork-linearizable storage (1)

- Client $C_i$
  - Stores timestamp $t_i$ and
  - Version (vector of timestamps) $T$, where $T[i] = t_i$
  - Increments $t_i$ and updates $T$ at every operation

- Versions order operations
  - After every operation, client signs new timestamp, version, and data
    $$V = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

- Verification with version $T$ of last operation
  - Version $V$ of next operation must be $V \geq T$
  - Signatures must verify
Fork-linearizable storage (2)

Version $T = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$

$V \geq T$?
verify $V[i] = T[i]$?
verify($\sigma, V\ldots x_j$)?
if not then abort
$T := V; T[i] := T[i]+1$
$\phi := \text{sign}(T\ldots)$
return $x_j$

Version $V = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$

Memory $x_1 \ldots x_n$
Signature $\sigma$ from $C_c$

$[\text{SUBMIT, READ, } j]$}

$[\text{REPLY, } V, \ldots x_j, c, \sigma]$}

$[\text{COMMIT, } T, \phi]$}

$V := T$
$\sigma := \phi$
$c := i$
Fork-linearizable storage (3)

- If clients are forked, they sign and store incomparable versions:
  \[
  \begin{bmatrix}
  u \\
  v \\
  w+1
  \end{bmatrix}
  \quad ? 
  \begin{bmatrix}
  u \\
  v+1 \\
  w
  \end{bmatrix}
  \]

- Signatures prevent server from other manipulations:
  - Protocol uses $O(n)$ memory for emulating fork-linearizable shared memory (MEM) on Byzantine server.

- Increasing concurrency?
  - Here, clients proceed in lock-step mode.
  - Yes, but see papers...
Fork-linearizability benefits

- Client $C_i$ writes many values $u, v, w, x ...$

- Without protection, faulty $S$ may return any such value to a reader $C_i$

- With fork-linearizable emulation
  - $C_i$ writes $z$ and tells $C_j$ "out-of-band"
  - $C_j$ reads $r$ from location $i$
    - if $r = z$, then all values that $C_j$ read so far were correct
    - if $r \neq z$, then $S$ is faulty

  - The "out-of-band" communication might be only synchronized clocks
Storage systems providing fork-linearizability

**SUNDR [LKMS04]** Secure untrusted data repository
- NFS network file system API
- Extensions to NFS server and NFS client
- Hash tree over all files owned by every user

**CSVN [CG09]** Integrity-protecting Subversion revision-control system
- SVN operations are verified
- Hash tree over file repository
- Based on fork-linearizable storage protocol [CSS07]

**Venus [SCCKMS10]** Integrity-protecting cloud storage
- Protects Amazon S3 "key-value store"
- Prototype implementation
Storage integrity - Summary

- Remote checking for storage and applications in cloud

- Target is collaboration among group of mutually trusting clients

- Fork-linearizable storage protocol
  - In normal case, it is linearizable and sometimes “blocking”
  - In case of Byzantine server, it respects fork-linearizability

- Related work
  - Extension to verify integrity of arbitrary services (not only storage MEM, but any computation) [C11]
  - Extension with an out-of-band communication system to detect violations in Fail-Aware Untrusted STorage (FAUST) [CKS09]
  - SPORC, a practical system for group collaboration [FZFF10]
Cryptographic protocols
Fully homomorphic encryption [G09]

- **Public-key cryptosystem**
  - KG() → (pk, sk) - generate a public key/secret key pair
  - E(pk, m) → c - encrypt message m to ciphertext c
  - D(sk, c) → m - decrypt ciphertext c to message m
  - Ciphertext and public key alone reveal nothing about message

- **Permits algebraic operations on ciphertexts**
  - Two operations ⊗, ⊕ on ciphertexts c1=E(m1), c2=E(m2) such that
    - c1 ⊕ c2 = E(m1) ⊕ E(m2) = E(m1 + m2) → gives m1 + m2
    - c1 ⊗ c2 = E(m1) ⊗ E(m2) = E(m1 × m2) → gives m1 × m2

- **Use to secure outsourced computation in the cloud**
  - Model computation as a binary circuit
  - Client provides input
  - Provider evaluates circuit on encrypted data, learns nothing about input
  - Client decrypts output locally

- **Problems in practice**
  - Circuit model, external inputs and outputs, not efficient enough
Proofs of storage [ABCHKPS07, AK07]

- **Provider proves possession of data without sending it**
  - Client stores many data items \((x_1, \ldots, x_n)\) at server
  - Server may "forget" data
  - Client wants to know: is data still stored at server?
  - Trivial solution: send some data back to client

- **Cryptographic proof of storage**
  - Client initially generates \(pk/sk\), computes a tag \(t_i\) for every item \(x_i\) using \(sk\)
  - Server stores \((x_1, t_1), \ldots, (x_n, t_n)\)
  - Later, client periodically sends challenge \(c\) and \(pk\)
    - Server computes short response \(r = \text{prove}(pk, c, x_1, \ldots, x_n, t_1, \ldots, t_n)\)
    - Client verifies \(r\) with \(sk\) if \(r\) is valid

- **Secure proof with low communication overhead**
  - Client stores only key and small state (not proportional to \(n\))
  - Length of \(c\) and \(r\) independent of \(n\)
  - Infeasible for malicious server to forge a valid response without \(sk\)
Conclusion
Summary

- **Cloud computing, the computing services of the future**
  - Storage is perhaps the most prominent example

- **Security problems of cloud computing**
  - Provider not trusted
  - Multiple tenants share infrastructure

- **Intercloud storage**
  - Exploit replication and independent providers

- **Storage integrity**
  - Clients remotely verify actions of cloud provider

- **Cryptographic protocols**
  - Further security guarantees
  - "Total protection" of client's data from cloud service is not possible
Introduction to Reliable and Secure Distributed Programming

- C. Cachin, R. Guerraoui, L. Rodrigues
- 2nd ed. of Introduction to Reliable Distributed Programming
- Published by Springer, 2011
- More info on book website

http://ww.distributedprogramming.net
Another advertisement - Workshop on cloud computing security

- ACM Cloud Computing Security Workshop (CCSW)
- Co-located with ACM Computer and Communications Security Conference (CCS)
- October 21, 2011, Chicago (USA)
- More info on the web

http://crypto.cs.stonybrook.edu/ccsw11
Further reading


References (1)


References (2)


