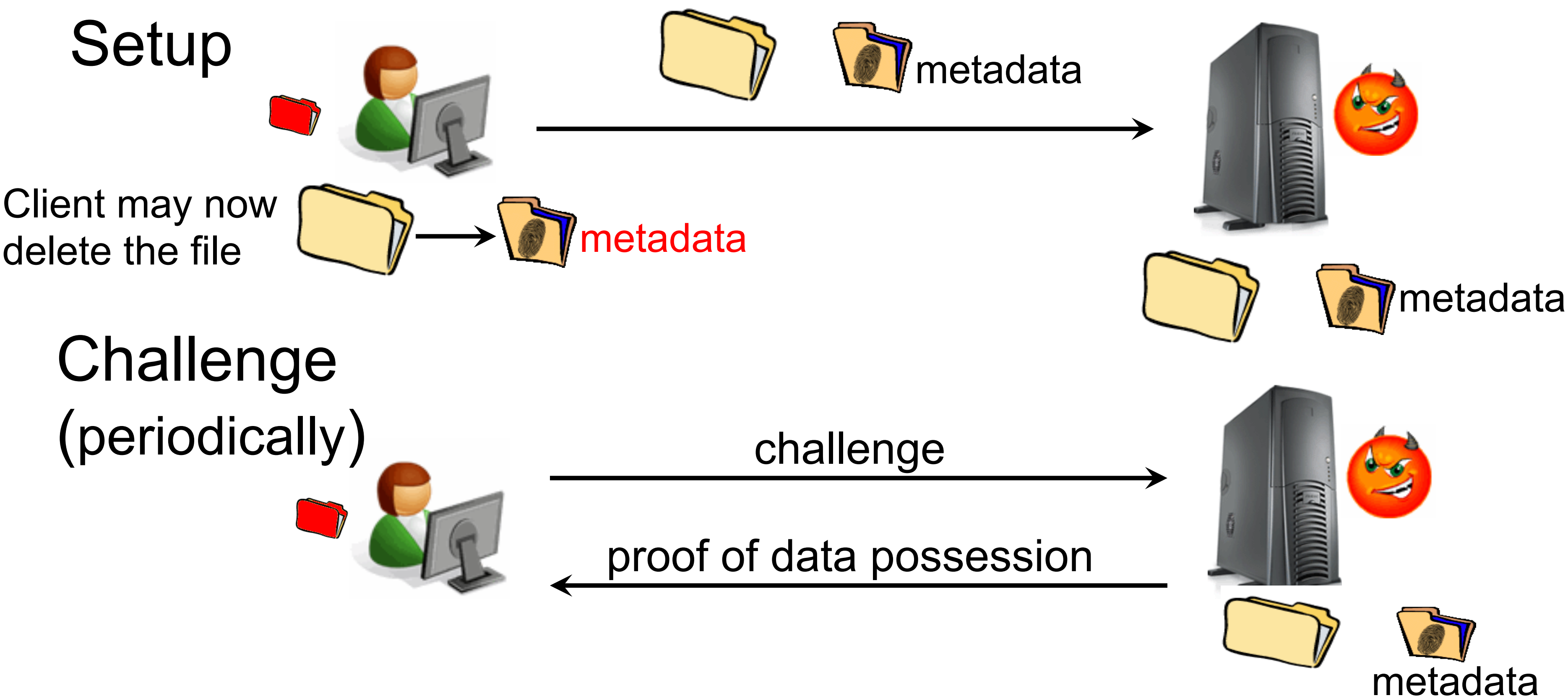
Auditable Version Control Systems

Bo Chen, Reza Curtmola

Department of Computer Science
New Jersey Institute of Technology

Remote Data Checking (RDC)

- Remote Data Checking (RDC) allows the data owner to check the integrity of data stored at an untrusted third party



  - Without retrieving the data
  - Without having the server access all the data (spot-checking)

Version Control Systems (VCS)

- A Version Control System automates the process of version control
  - Record all changes to the data into a data store called repository
  - Any version of the data can be retrieved at any time in the future

- Providers of VCS services are not necessarily trusted
  - May rely on a public cloud storage platform
  - Vulnerable to various outside or even inside attacks
  - Rely on complex distributed systems, which are vulnerable to various failures caused by hardware, software, or even administrative faults
  - Unexpected accidental events may lead to the failure of services

RDC can be used to address these concerns about the untrusted nature of a third party that hosts the VCS repository

On The Importance of Auditing VCS Systems

- Popular hosting services have a huge number of repositories
  - 2013: GitHub (> 6 million repositories), SourceForge (> 324,000 projects), Google Code (> 250,000 projects)

- Hosting providers that offer version control functionality rely on untrusted cloud storage services as the back-end storage
  - Dropbox uses Amazon S3 as the back-end storage

- VCS-es support many types of data (other than source code)
  - Subversion (SVN) supports both small text files and large binary files
  - Ongoing efforts to add support for large media binary files into VCS-es like Git

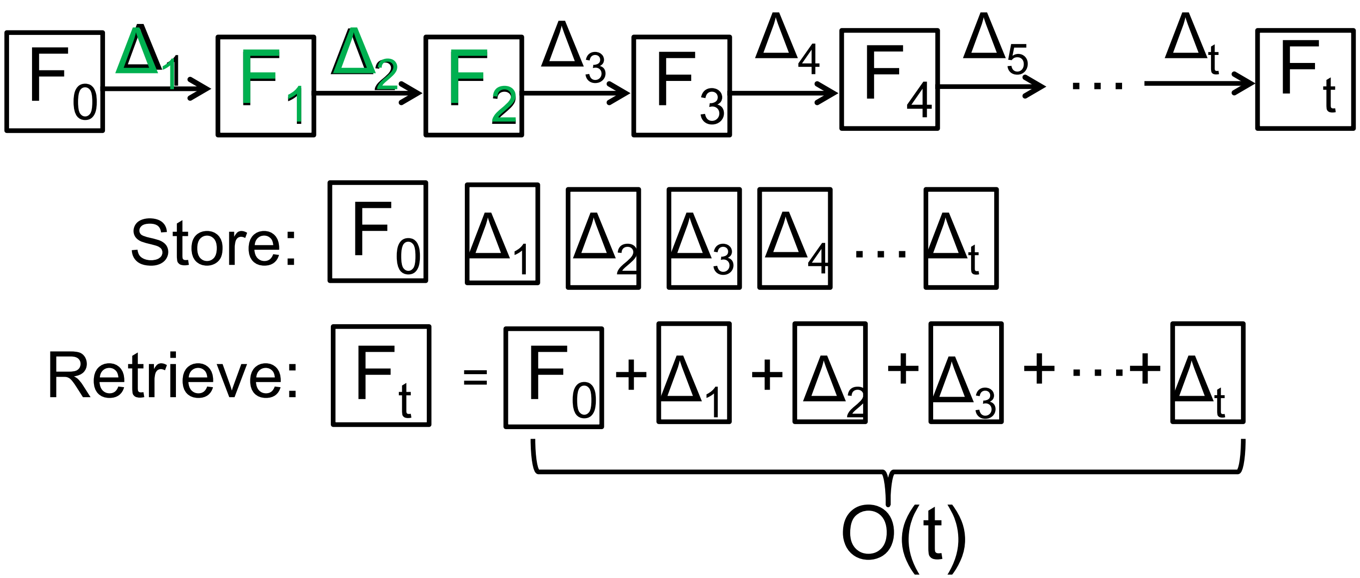
Data Organization in Version Control Systems

- A basic version control system
  - The VCS simply stores each file version
  - Very large storage overhead (e.g., the source code for GCC compiler has over 200,000 versions)

Store: F_0 F_1 F_2 $\cdots$ F_t

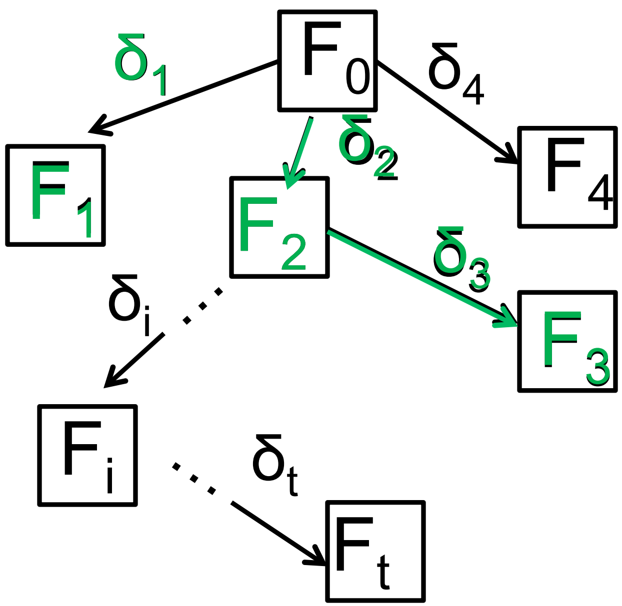
Data Organization in VCS-es (cont.)

- Delta-based version control systems (e.g., CVS, Git)
  - Only the first file version is stored in its entirety
  - Each subsequent file version is stored as the difference from the immediate previous version
  - Reduce storage overhead significantly
  - Expensive retrieval: To retrieve version t, the VCS server starts from the initial version and applies t subsequent deltas



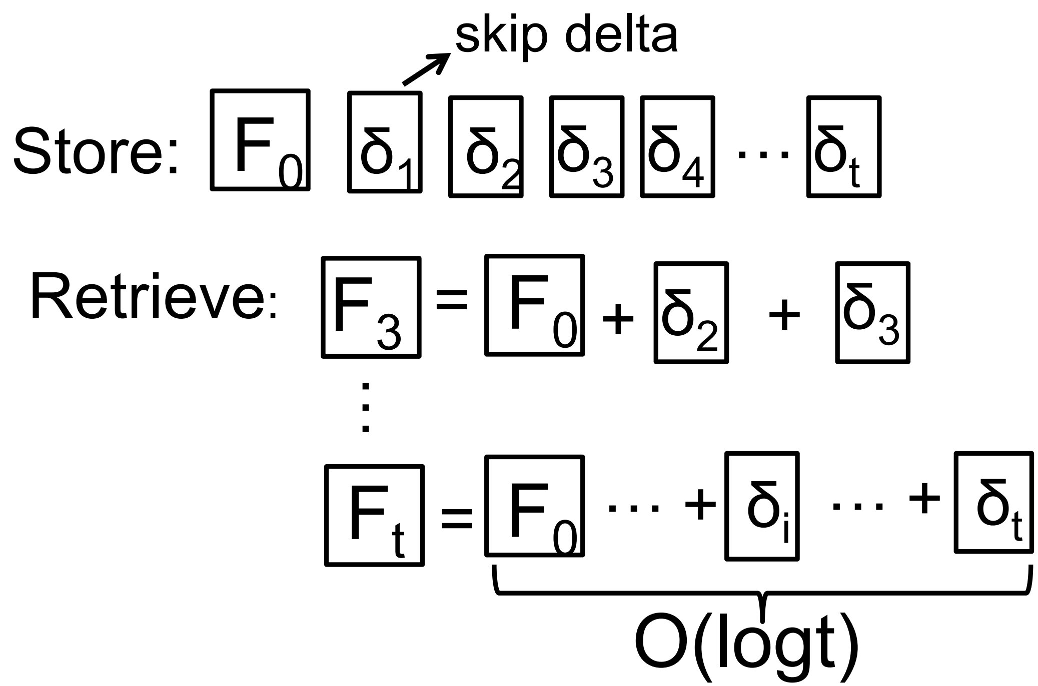
Data Organization in VCS-es (cont.)

- Skip delta-based version control systems (e.g., Subversion)
  - Further optimizes towards reducing the cost of retrieval
  - A new file version is stored as the difference from a previous file version
  - This difference is relative to another previous version (skip version)
  - Retrieval of any file version requires log(t) applications of skip deltas



$F_{skip(1)} = F_{skip(2)} = F_{skip(4)} = F_0$

$F_{skip(3)} = F_2$



Contributions

- The first to take a pragmatic approach for auditing real-world VCS-es
  - Previous solutions that rely on dynamic RDC are overkill
- Introduce the definition of Auditable Version Control Systems (AVCS)
  - Delta-based VCS-es designed to function under an adversarial setting

- Propose RDC–AVCS, an AVCS scheme for skip delta-based VCS-es
  - Rely on RDC mechanisms to ensure all the versions of a file are retrievable from the untrusted VCS server over time

- Build SSVN, a prototype for RDC–AVCS on top of Subversion (SVN)
  - Experimentally show that SSVN incurs only a modest decrease in performance compared to a regular (non-secure) SVN system
  - Build a tool which facilitates the migration of non-secure SVN repos into auditable SVN repos

Related Work

- Previous work (DPDP [EK+ 09], DR-DPDP [EK 13]) uses full-fledged dynamic RDC to support all types of updates (insert, delete, modify)
  - Real-world VCS systems require only the append operation
  - Support for all types of updates is overkill (unnecessary overhead)
  - Higher complexity makes schemes more prone to security and implementation flaws
  - Built on top of delta-based version control systems

	DPDP [EK+09]	DR-DPDP [EK09]	Our scheme
Communication (Challenge phase)	O(logn+log(t))	O(1+logn)	O(1)
Server computation (Challenge phase)	O(logn+log(t))	O(1+logn)	O(1)
Client computation (Challenge phase)	O(logn+log(t))	O(1+logn)	O(1)
Server computation (Retrieve phase)	O(tn + log(t))	O(tn+1)	O(nlog(t)+1)

Comparison of different RDC schemes for version control systems

Model and Guarantees

- In AVCS, just like in a regular VCS, one or more clients store data at a server
  - The server maintains the main repository, storing all the file versions
  - Each AVCS client has a local repository, which stores the working copy

- Threat model: all clients are trusted, the server is not trusted
  - The untrusted server is rational and economically motivated
  - Cheating is meaningful only if it cannot be detected and if it achieves some economic benefit

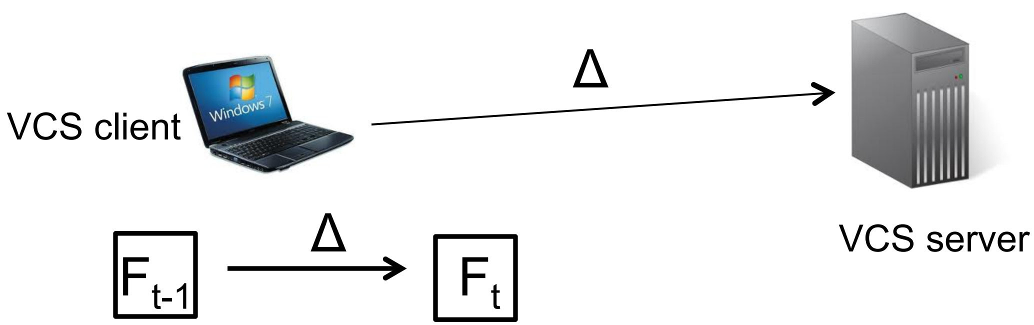
- Security Guarantees
  - Data possession: check integrity of all file versions (without retrieving the data)
  - Version correctness: verify correctness of a file version (upon retrieval)

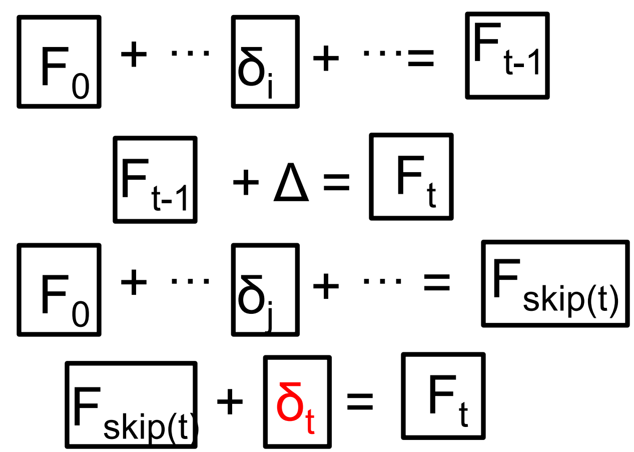
A Skip Delta-based VCS in a Benign Setting

- Existing version control systems (e.g., Subversion) , which use skip delta encoding , have been designed for a benign setting
  - The VCS server is assumed to be fully trusted
- The main operations of such VCS systems fall under three phases: Setup, Commit, and Retrieve
  - Setup: The client (data owner) contacts the server to create a new project in the main repository (e.g., svnadmin create, svn import, etc.)

A Skip Delta-based VCS in a Benign Setting

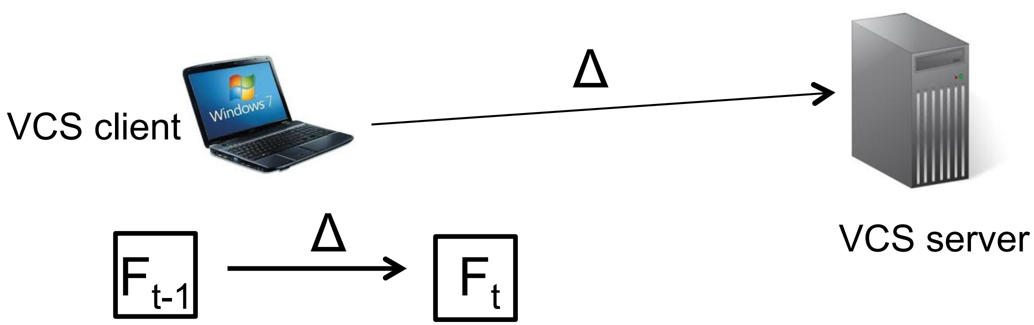
- Existing version control systems (e.g., Subversion) , which use skip delta encoding , have been designed for a benign setting
  - The VCS server is assumed to be fully trusted
- The main operations of such VCS systems fall under three phases: Setup, Commit, and Retrieve
  - Setup: The client (data owner) contacts the server to create a new project in the main VCS repository (e.g., svnadmin create, svn import, etc.)
  - Commit: The client commits changes in its local working copy into the main repository (e.g., svn commit)





A Skip Delta-based VCS in a Benign Setting

- Existing version control systems (e.g., Subversion) , which use skip delta encoding , have been designed for a benign setting
  - The VCS server is assumed to be fully trusted
- The main operations of such VCS systems fall under three phases: Setup, Commit, and Retrieve
  - Setup: The client (data owner) contacts the server to create a new project in the main VCS repository (e.g., svnadmin create, svn import, etc.)
  - Commit: The client commits changes in its local working copy into the main repository (e.g., svn commit)



$$F_0 + \cdots \delta_i + \cdots = F_{t-1}$$

$$F_{t-1} + \Delta = F_t$$

$$F_0 + \cdots \delta_j + \cdots = F_{skip(t)}$$

$$F_{skip(t)} + \delta_t = F_t$$

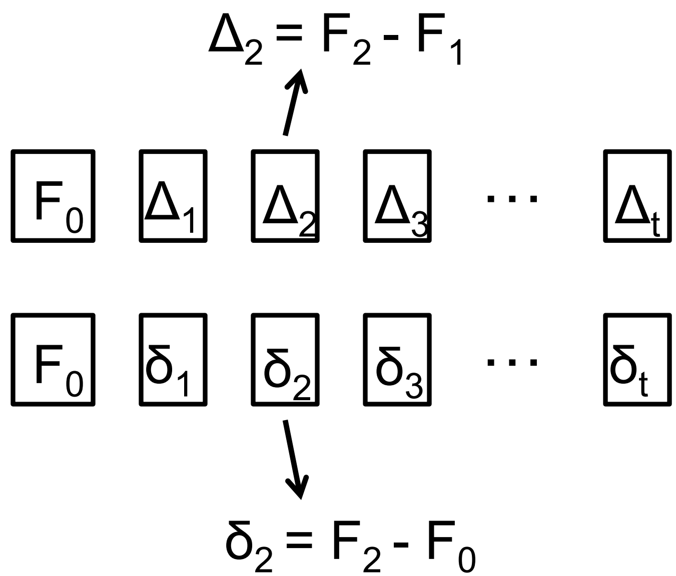
  - Retrieve: The client retrieves an arbitrary file version (e.g., svn co)

Moving to an adversarial setting: Challenges

- The gap between the server's and the client's view of the repository

client's view

server's view

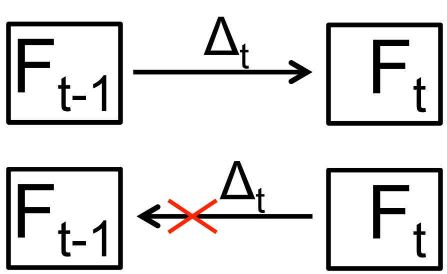


Δ – delta against the immediate previous version

δ – skip delta

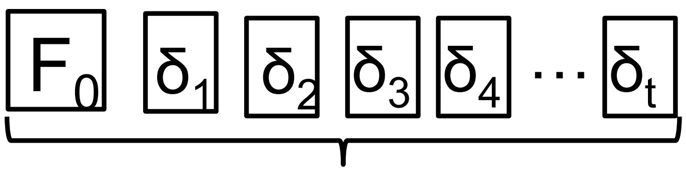
- Delta encoding (and skip delta encoding) is not reversible

Perform a delete operation on F_{t-1}, Δ_t encodes only the position of the deleted portion from F_{t-1}, rather than the actual content being deleted



RDC-AVCS

- We propose RDC–AVCS, an AVCS scheme which uses RDC to ensure all the versions of a file can be retrieved from the untrusted server
- Basic idea:
  - View the repository as a virtual file, obtained by concatenating the initial file version and the subsequent skip deltas
    - Any RDC protocol that supports the append operation securely (e.g., PDP [AB+11]) can be used to audit the integrity of a VCS server
    - No need to support other dynamic updates except append



a virtual file with skip deltas appended

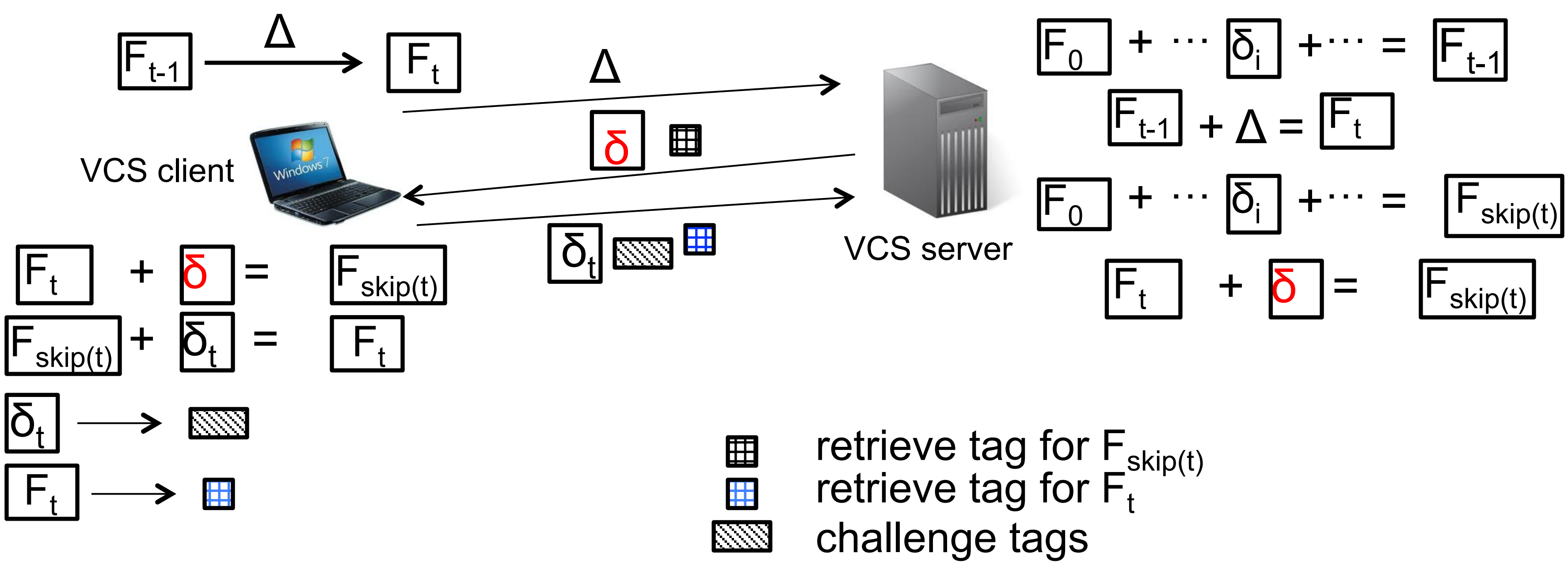
  - To bridge the gap between the server's and the client's view of the repository, the skip delta is computed by the client and not by the server

RDC-AVCS (cont.)

- We use two types of verification tags
  - Challenge tags: To check data possession of the whole repository
  - Retrieve tags: To check the integrity of individual file versions
- Four phases: Setup, Commit, Challenge, and Retrieve
  - Setup: The client initializes the VCS repository

RDC-AVCS (cont.)

- We use two types of verification tags
  - Challenge tags: To check data possession of the whole repository
  - Retrieve tags: To check the integrity of individual file versions
- Four phases: Setup, Commit, Challenge, and Retrieve
  - Setup: The client initializes the VCS repository
  - Commit:



RDC-AVCS (cont.)

- RDC-AVCS has four phases: Setup, Commit, Challenge, and Retrieve
  - Challenge: Verifier uses RDC (based on spot checking) to check the whole repository (viewed as a virtual file)

RDC-AVCS (cont.)

- RDC-AVCS has four phases: Setup, Commit, Challenge, and Retrieve
  - Challenge: Verifier uses RDC (based on spot checking) to check the whole repository (viewed as a virtual file)
  - Retrieve: The client retrieves a file version (together with its retrieve tag), and uses the retrieve tag to check its correctness

Analysis and Discussion

- RDC–AVCS achieves both security guarantees
  - Data possession: check integrity of all file versions (without retrieving the data)
  - Version correctness: verify correctness of a file version (upon retrieval)

- RDC-AVCS is efficient
  - Challenge phase: The computation and communication complexity for checking the whole repository is O(1)
    - Regardless of the repository size or the version size
  - Retrieve phase: To retrieve an arbitrary version from the repository, the server only needs to go through at most log(t) skip deltas

Implementation and Experiments

- SSVN: a prototype for RDC–AVCS on top of Apache Subversion (SVN)
  - Added 4,000 lines of C code into the SVN code base (V1.7.8)
  - Implemented the most common VCS operations (e.g., commit, etc.)
  - Built a tool migrating non-secure SVN repos to secure SVN repos

- Experimental setup
  - Repository selection:
    - Small-size: < 5,000 files (FileZilla)
    - Medium-size: 5,000 - 50,000 files (Wireshark)
    - Large-size: > 50, 000 files (GCC)
  - Evaluated the overhead for the Commit and Retrieve phases

Implementation and Experiments (cont.)

- Commit Phase: Overhead for committing one file version

	FileZilla	Wireshark	GCC1	GCC2
SSVN (s)	0.427	0.416	0.417	10.776
non-secure SVN (s)	0.389	0.376	0.386	10.502

The average time (in seconds)

	FileZilla	Wireshark	GCC1	GCC2
SSVN (KB)	4.599	3.458	4.123	6
non-secure SVN (KB)	4.391	3.246	4.017	5.696

The average communication from the client to the server

	FileZilla	Wireshark	GCC1	GCC2
SSVN (KB)	1.559	1.437	1.047	3.244
non-secure SVN (KB)	0.574	0.58	0.574	0.571

The average communication from the server to the client

- Retrieve Phase: Overhead for retrieving one file version

	FileZilla	Wireshark	GCC1	GCC2
secure SVN (s)	0.0535	0.0453	0.0506	5.086
non-secure SVN (s)	0.0416	0.0376	0.0416	4.779

The average time (in seconds)

Conclusion

- We introduce Auditable Version Control Systems (AVCS), which are delta-based VCS systems designed to function in an adversarial setting

- We propose RDC–AVCS, an AVCS scheme for skip delta-based version control systems, which relies on RDC mechanisms to ensure all the versions of a file can be retrieved from the untrusted VCS server over time

- We build a prototype on top of Apache SVN which incurs a modest decrease in performance compared to a non-secure SVN system

References

- [AB+07] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song, “Provable data possession at untrusted stores,” in Proc. of ACM CCS, 2007.
- [AB+11] G. Ateniese, R. Burns, R. Curtmola, J. Herring, O. Khan, L. Kissner, Z. Peterson, and D. Song, “Remote Data Checking Using Provable Data Possession,” ACM Transactions on Information and System Security (TISSEC), May 2011.
- [JK07] A. Juels and B. S. Kaliski, “PORs: Proofs of retrievability for large files,” in Proc. of ACM CCS, 2007.
- [SW08] H. Shacham and B. Waters, “Compact proofs of retrievability,” in Proc. of Asiacrypt, 2008.
- [EK+09] C. Erway, A. Kupcu, C. Papamanthou, and R. Tamassia, “Dynamic Provable Data Possession,” in Proc. Of ACM CCS, 2009.
- [EK13] M. Etemad and A. Kupcu, “Transparent, distributed, and replicated dynamic provable data possession,” in Proc. of ACNS, 2013.

Thank you!

Questions?