



# Secrecy in Concurrent Version Control Systems

Jerônimo Pellegrini<sup>1</sup>

<sup>1</sup>Instituto de Computação  
Unicamp

SBSeg 2006



# Outline

## Version Control Systems

- Centralized VCS

- Distributed VCS

- No secrecy

## The protocols

- Overview

- Access Control

- Secrecy for centralized VCS

- Secrecy for distributed VCS

## Apso: a prototype

- Apso

## Future and ongoing work

## Summary



# Centralized VCS

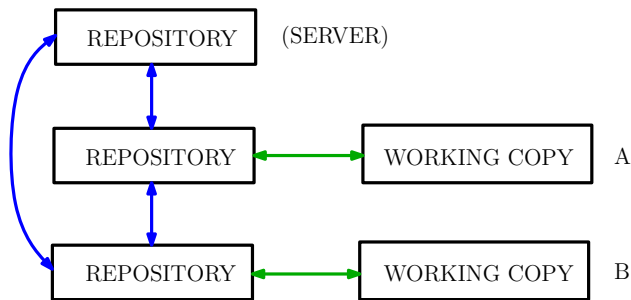
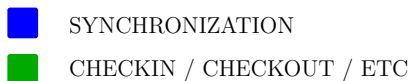
- ▶ One server controls everything
- ▶ Repository stays in a secured server
- ▶ Examples:
  - ▶ CVS
  - ▶ Subversion
  - ▶ ClearCase
  - ▶ Perforce



# Distributed VCS

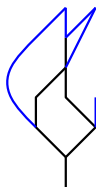
- ▶ No mandated centralization (but still possible)
- ▶ Users can commit without connectivity
- ▶ Multiple copies of partial repositories exist
- ▶ Examples:
  - ▶ Git
  - ▶ Mercurial
  - ▶ Monotone
  - ▶ Arch/Bazaar

# Distributed VCS

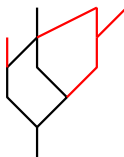


# Distributed VCS

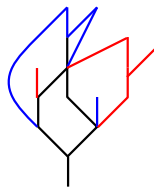
- ▶ Vertices are revisions



A



B



FULL REPOSITORY

## Secrecy in VCS

- ▶ No good degree of secrecy in current systems
- ▶ Why secrecy is important for VCS:
  - ▶ VCS server in colocation
  - ▶ Untrustworthy (but competent) sysadmin
  - ▶ Stolen hosts
  - ▶ Possible break-in



## The protocols

- ▶ Deltas are encrypted (w/symmetric cipher) before going to a hostile host
- ▶ Symmetric cipher key shared among users
  - ▶ But encrypted with asymmetric cipher in hostile host
- ▶ Access control uses PGP-like PKI and the VCS itself
  - ▶ Same for centralized and decentralized systems

## Access control

- ▶ Group of users  $\mathcal{G}$
- ▶ Group of administrators  $\mathcal{G}^*$
- ▶ List of symmetric keys  $\mathcal{K} = \{k_1, k_2, \dots, k_n\}$
- ▶ Mapping from keys onto revisions  $\mathcal{M} : \mathcal{K} \mapsto \mathcal{R}$
- ▶  $\mathcal{K}$  and  $\mathcal{M}$  are kept in the VCS



## Access control – grant & revoke

- ▶ Grant
  - ▶ Adds new user  $A$  to  $\mathcal{G}$  (in VCS)
  - ▶ Send  $\{\mathcal{K}\}_{K_A}$  to repository
- ▶ Revoke
  - ▶ Remove  $A$  from  $\mathcal{G}$
  - ▶ Put new key in list  $\mathcal{K}$
  - ▶ Send new mapping  $\mathcal{M}'$  to repository
  - ▶ For each user  $A$ , send new  $\{\mathcal{K}'\}_A$  to repository

## Key compromise

- ▶ Pick new symmetric key  $k'_0$
- ▶ Reencrypt all deltas using  $k'_0$
- ▶ Delete  $\mathcal{K}$  and  $\mathcal{M}$
- ▶ Send new  $\mathcal{M}'$  to repository
- ▶ For each user  $A$ , send new  $\{\mathcal{K}'\}_A$  to repository



## Centralized – Checkin

- ▶ Update mapping and keylist
- ▶ Before sending content to server:
  - ▶ Decrypt latest key  $k$
  - ▶ Encrypt delta with  $k$
  - ▶ Encrypted delta is sent to server
- ▶ Import: pick first key, then checkin

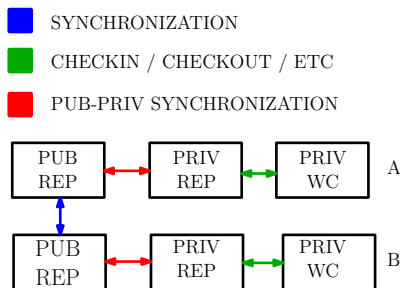


## Centralized – Checkout & update

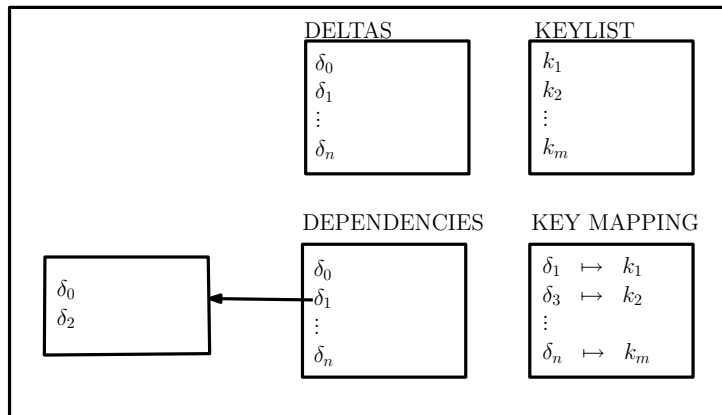
- ▶ Update mapping and keylist
- ▶ For each delta received, *in topological order*:
  - ▶ Check mapping and pick right key  $k_n$
  - ▶ Decrypt  $k_n$  using private key
  - ▶ Decrypt delta using  $k_n$
  - ▶ Apply delta

## Secrecy for distributed VCS

- ▶ Users operate on private repositories (as usual)
- ▶ Keep public (encrypted) copy of the private repository
- ▶ Public repositories are synchronized



## Distributed – The public repository



## Distributed – Repository synchronization

- ▶ For all delta  $d \in \Delta^R \setminus \Delta^U$ 
  - ▶ Get the latest key  $k$
  - ▶ Send the encrypted delta,  $\{d\}_k$ , to  $U$
  - ▶ Send a list of parents of  $d$  in  $U$
- ▶ For the set of deltas  $\{d' \in \Delta^U \setminus \Delta^R\}$ 
  - ▶ Use the list of parents to put the deltas in topological order
- ▶ For each delta  $d'$ :
  - ▶ Get the key  $k$  used to encrypt  $d'$
  - ▶ Apply the decrypted  $\{d'\}_k$  to  $R$

## Apso: a prototype

- ▶ Written in C++
- ▶ Implements protocol for distributed VCS
- ▶ Extensible:
  - ▶ Version control systems are plugins
  - ▶ Cryptographic engines are plugins
- ▶ Portable:
  - ▶ Uses BOOST for interacting with the filesystem
- ▶ Currently supports Monotone and Nettle
- ▶ Available at <http://aleph0.info/apso>

## Future work

- ▶ Better key distribution
- ▶ Currently relies on VCS for limiting access to the encrypted  $\{k\}_{K_A}$
- ▶ Refine grant and revoke:
  - ▶ Grant with expiry date
  - ▶ Better granularity
- ▶ Include support for more cryptographic libraries in Apso
  - ▶ (In particular, more asymmetric encryption algorithms)
- ▶ Add other VCS to Apso (Git and Mercurial, for example)



## Summary

- ▶ No real secrecy in VCS today
- ▶ Two protocols for VCS secrecy
  - ▶ Centralized and decentralized
- ▶ Implementation of prototype
  - ▶ Portable C++ code
  - ▶ Pluggable crypto engines and VCS
- ▶ Contact: `jeronimo@ic.unicamp.br`
- ▶ Questions?