Nigori

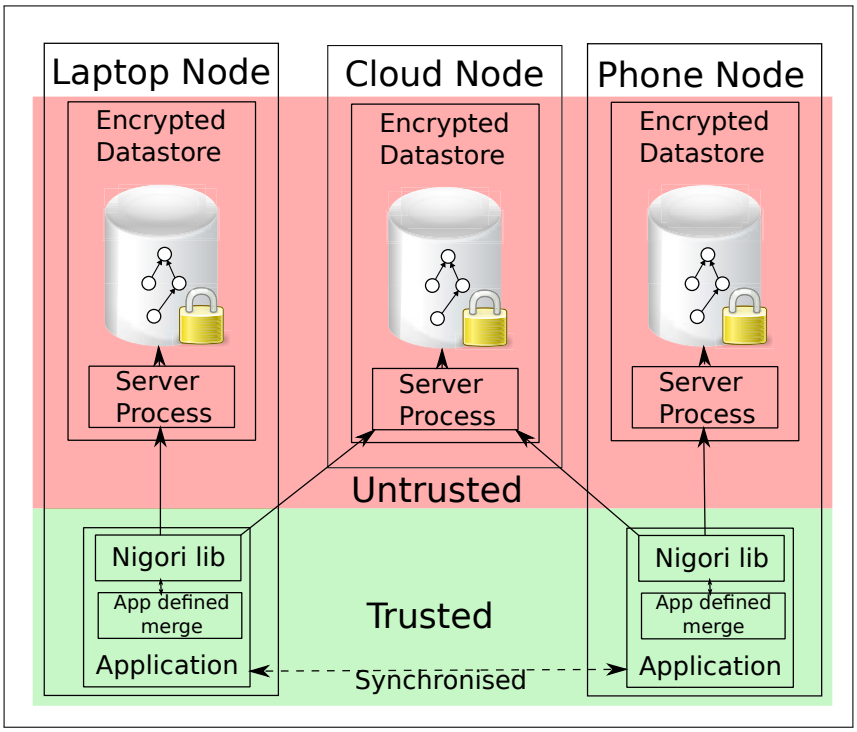
Secrets in the Cloud

Daniel Thomas, Alastair Beresford and Ben Laurie

UNIVERSITY OF CAMBRIDGE
Computer Laboratory
Digital Technology Group

Computer users today have a smartphone, a tablet, a laptop and a desktop machine. Consequently, many new computer applications seamlessly synchronise user data between devices using cloud storage as a highly-available intermediary. Whilst the communication link between the user device and cloud storage is often encrypted, user data is typically stored in a form which is readable by the cloud provider and the application developer.

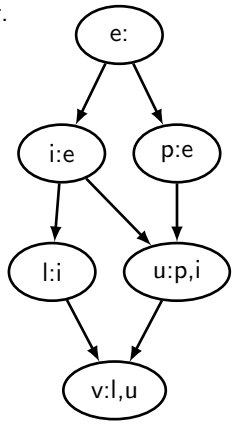
The aim of the Nigori project is to develop a practical, application neutral, mechanism for storing sensitive user data in the cloud in such a way that the cloud provider and application developer cannot read any of the stored information. We have an initial specification, and an implementation of Nigori for Java and Android. Work is underway on a Dart/JavaScript version suitable for use as a plug-in for Web browsers.



Nigori consists of two components: a datastore and a client library. A Nigori datastore is a service, either run locally on the device alongside the application, or run remotely in the cloud. The client library forms part of the application and runs on a user's device, encrypts data, and manages the user's datastores. A typical application deployment will contain one datastore on each user device and one datastore in the cloud; the application can then use Nigori to keep datastores, and therefore user data, synchronised across all their devices.

In Nigori the key material is derived from the user's unique username and secret password. Then authentication to the servers can be done using DSA and the data can be encrypted on the client using AES.

A Nigori datastore stores a mapping of users to indicies to revisions to values. All of these are opaque byte arrays to the datastore. On the client side the revisions are interpreted as revision objects consiging of an id and a list of parent revisions. As in git the ids are generated by taking hashes of the value and the parent revision information. This means that the client can verify that history has not been removed by the server.



When clients asynchronously make multiple conflicting changes to a index then different revisions are generated which allows the clients to do reconcilation on read. Since different applications will have different ways of automatically or semi-automatically resolving conflicts, Nigori allows the developer to specify how conflicts should be resolved.