Computer Science Project Proposal

Path ORAM on MirageOS

R. Horlick, Homerton College

Originator: Dr N. Sultana

14 October 2015

Project Supervisors: Dr N. Sultana & Dr R. M. Mortier

Director of Studies: Dr B. Roman

Project Overseers: Dr M. G. Kuhn & Prof P. E. Sewell

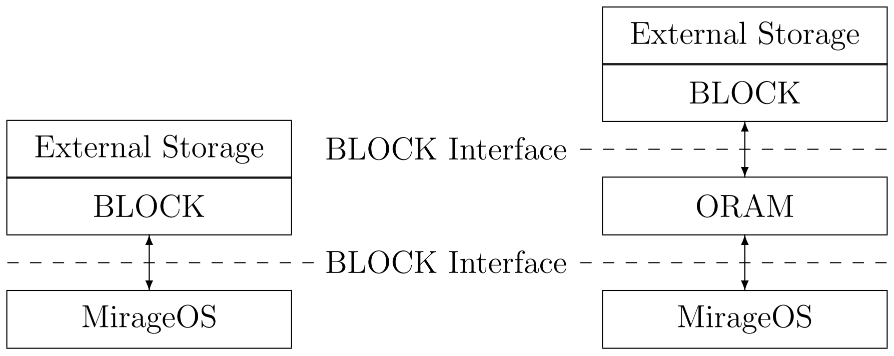

Figure 1: The MirageOS stack with and without ORAM

Introduction and Description of the Work

As the cost of large-scale cloud storage decreases and the rate of data production grows, individuals and small businesses will find themselves increasing reliant on the trust of cloud providers. We can, of course, use encryption and be safe in the knowledge that no adversary will be able to view the plaintext of our data, but this is not enough. It turns out that the pattern of access to the data can leak large amounts of information. In a study on an encrypted email repository, up to 80% of search queries could be inferred from the access pattern alone! So clearly this a leak worth plugging, but how can we do it?

The solution to our problem is Oblivious Random Access Memory (ORAM), a cryptographic primitive that ensures that an adversary has negligible probability of learning anything about the logical access pattern, even with full access to the physical one. We normally talk about an ORAM as a block device, with N blocks of B bits each, giving an ORAM of size $N \cdot B$. We are most interested in the asymptotic bandwidth cost of the ORAM protocol, because we are operating over the internet.

A trivial ORAM algorithm operates by scanning over the whole ORAM and reading/updating only the relevant block, but this has $O(N)$ bandwidth cost, which is highly impractical for large-scale storage. Luckily, much better algorithms have been proposed. We choose to focus on Path ORAM, because it has only $O(\log N)$ bandwidth cost in the worst case, as well as being incredibly simple conceptually.

ORAM protocols generally operate in a client-server model. For our purposes the server is the cloud storage provider and the client will be a MirageOS cloud instance. MirageOS is a unikernel operating system designed to run on the Xen hypervisor. I have chosen to use Mirage for a number of reasons. Firstly, it is lightweight and designed to be run in the cloud, meaning that simple cloud services can be built on top of it that fully leverage the ORAM. Secondly, it is written in OCaml, meaning that I can take full advantage of the rich module system. This will allow me to write my implementation as a functor that takes an implementation of Mirage's BLOCK interface and creates a new BLOCK implementation that uses ORAM, as shown in Figure 1. This means that the ORAM could be used with any underlying implementation of the BLOCK interface and that it would plug seamlessly into any existing program.

Starting Point

Description of Mirage's structure

Also using Jane Street's Core Library for their data structures

Substance and Structure of the Project

Substance

The Path ORAM protocol has three main components: a binary tree, a stash and a position map. The binary tree is the main storage space. Every node in the tree is a bucket, which can contain up to Z blocks. The tree has height L, where the tree of height 0 consists only of the root node, and the leaves are at level L. The stash is temporary client-side storage, consisting only of a set of blocks waiting to be put back into the tree. The position map associates, with each block ID, an integer between 0 and $2^L - 1$. The invariant that the Path ORAM algorithm maintains is that if the position of a block x is p, then x is either in some bucket along the path from the root node to the p^{th} leaf, or in the stash. On every access to the tree, a whole path is read into, the accessed block is assigned a new random position and then as many blocks as possible are written back into the same path. This means that in any two access to the same block, the paths that are read are statistically independent.

Talk about recursive ORAM and other optimisations on top of it, also note that we are moving optimisations from the realm of secure processors to the cloud computing section and so we have to justify this

Talk about evaluation phase (statistical tests on data to show that security properties are there as well as complexity and performance metrics between optimisations)

Structure

The project breaks down into the following sub-projects:

1. Familiarising myself with OCaml, MirageOS and LWT
2. Implementing the basic Path ORAM functor and testing that it works in place of existing BLOCK device implementations
3. Implementing the three main optimisations to Path ORAM as extensions to the same functor, allowing for the use of different combinations of optimisations
4. Creation of a suite of tests and experiments to evaluate the performance and latency of the implementation
5. Writing the dissertation

Success Criterion for the Main Result

Possible Extensions

If I achieve my main result early I shall try the following alternative experiment or method of evaluation . . .

Timetable: Workplan and Milestones

Planned starting date is 16/10/2011.

1. **Michaelmas weeks 2–4** Learn to use X. Read book Y. Read papers Z.
2. **Michaelmas weeks 5–6** Do preliminary test of Q.
3. **Michaelmas weeks 7–8** Start implementation of main task A.
4. **Michaelmas vacation** Finish A and start main task B.
5. **Lent weeks 0–2** Write progress report. Generate corpus of test examples. Finish task B.
6. **Lent weeks 3–5** Run main experiments and achieve working project.
7. **Lent weeks 6–8** Second main deliverable here.
8. **Easter vacation:** Extensions and writing dissertation main chapters.
9. **Easter term 0–2:** Further evaluation and complete dissertation.
10. **Easter term 3:** Proof reading and then an early submission so as to concentrate on examination revision.

Resources Required