CS 294: Special Topic in Cryptography: Secure Computation Feb 25, 2016

Lecture 12: Oblivious RAM

Instructor: Sanjam Garg *Scribe: Peihan Miao*

1 Motivation

Roughly speaking, an ORAM enables executing a RAM program while hiding the access pattern to the memory. ORAM have several fundamental applications. For example, imagine a client has a huge memory/database D. He wants to (encrypt and) store it on the server in such a way that later he can request and get access to a specific location of the database $D[i]$ by communicating with the server without leaking any information of the location i to the server.

Definition 1 *An ORAM scheme* $\mathsf{O} = (\mathsf{DGen}, \mathsf{LGen})$ *consists of the following:*

- $\mathsf{DGen}\left(1^\kappa, D\right) \rightarrow \left(\tilde{D}, \mathsf{sk}\right)$ *given the security parameter and the initial database outputs an oblivious database and a secret key stored by the client, where* $|\mathsf{sk}| = \mathcal{O}\left(\mathsf{polylog}\left(|D|\right)\right)$*.*
- $\mathsf{LGen}\left(\mathsf{sk}, \ell_1, \cdots, \ell_T\right) \rightarrow \left(\ell_1, \cdots, \ell_{T'}\right)$ *given memory access locations of* D *outputs memory access locations of* $\tilde{D}$*. Note* $T' = \mathcal{O}\left(T \cdot \mathsf{polylog}\left(|D|\right)\right)$*.*

Provided $\mathsf{sk}, \tilde{D}\left[\ell_1\right], \cdots, \tilde{D}\left[\ell_{T'}\right]$ *the client is able to recover* $D\left[\ell_1\right], \cdots, D\left[\ell_T\right]$*. Furthermore, for any* $\left(\ell_1, \cdots, \ell_T\right)$ *and* $\left(\ell'_1, \cdots, \ell'_T\right)$ *it satisfies*

$$\mathsf{LGen}\left(\mathsf{sk}, \ell_1, \cdots, \ell_T\right) \approx_s \mathsf{LGen}\left(\mathsf{sk}, \ell'_1, \cdots, \ell'_T\right).$$

2 Construction

In this section we first describe an ORAM construction where the client has storage of $\frac{n}{\alpha}$ for some constant α, where n is the size of D. Then we will show this basic scheme suffices for constructing an ORAM scheme where the client has only $\mathsf{polylog}\left(n\right)$ storage.

2.1 A Basic Construction

Assume the client has storage of $\frac{n}{\alpha}$. First DGen splits the database D into $\frac{n}{\alpha}$ blocks, each of size α. Then it samples a "position" for each block uniformly at random from $\left[\frac{n}{\alpha}\right]$, as in Figure 1. The position map is stored at the client.

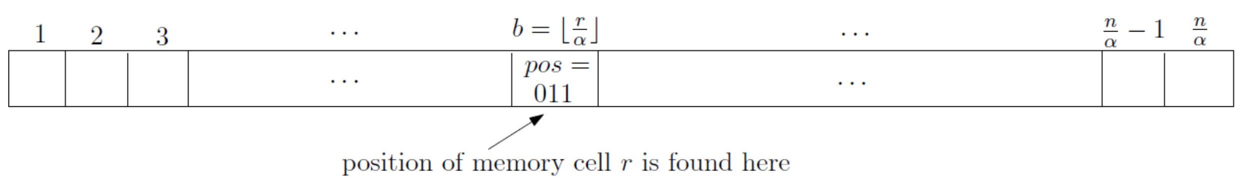

Figure 1: Position Map

An ORAM tree is then created as in Figure 2 (both figures are from [CP13].). It is a binary tree, each node in which is associated with a bucket which stores (at most) K tuples (b, pos, v) where v is the content of block b and pos is the leaf associated with the block b. K is a parameter that will determine the security of the ORAM.

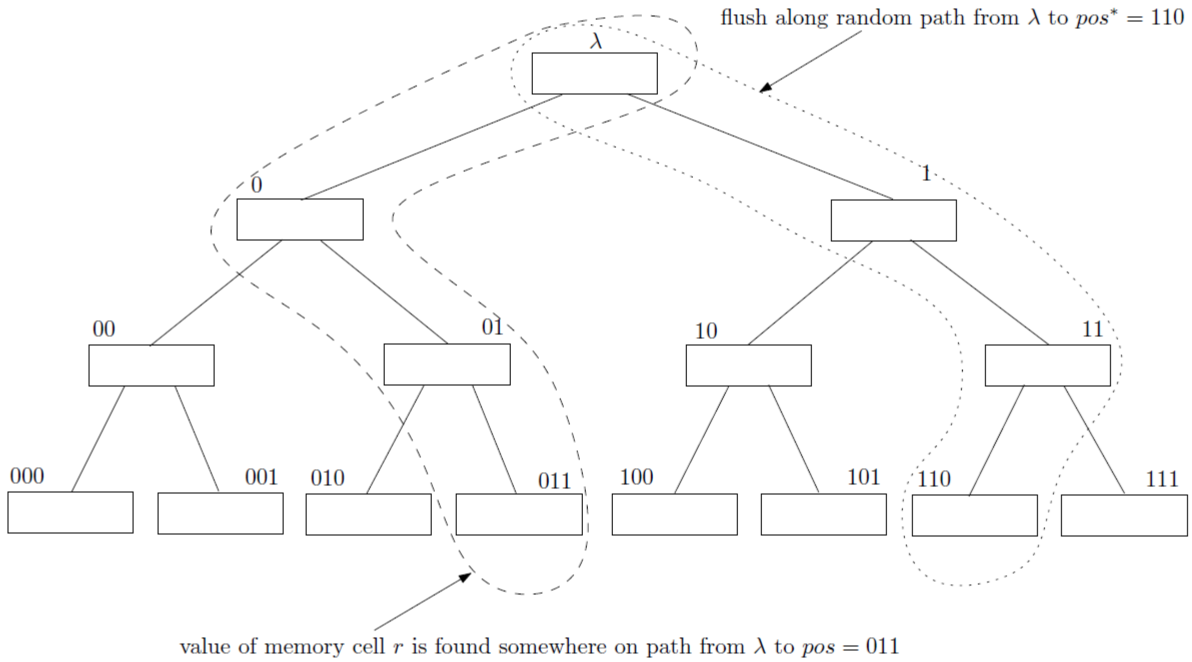

Figure 2: The ORAM Tree

When reading (or writing to) a memory block b, the client first requests the server for the entire path of pos in the ORAM tree, then generates a new random position pos' for b, deletes the old tuple (b, pos, v) from the path, adds to the root a new tuple $(b, pos', v(\text{or } v'))$, and sends the entire path back to the server. After this, there is a *flush* step, in which the client requests for a random path, and pushes each tuple in the path down as far as possible.

2.2 Security Proof

It is clear that the access patterns are hidden in the above construction, since every read/write/flush requests a uniformly random path. We only need to argue that the probability of *overflow* (meaning that at any time a node in the ORAM tree contains more than K tuples) is negligible.

Consider a dart game: you have an unbounded number of white and black darts. In each round of the game, you first throw a black dart, and then a white dart; each dart independently hits the bullseye with probability p. You continue the game until at least K darts have hit the bullseye. You "win" if none of darts that hit the bullseye are white. The winning probability is upper bounded by 2^{-K}.

Suppose there is a tree node γ containing more than K tuples at some point of time. Among the K tuples at γ, WLOG assume at least $K/2$ tuples has pos with prefix $\gamma||0$. Think of black darts hitting bullseye as assigning a memory block to a leaf pos with prefix $\gamma||0$, and white darts hitting bullseye as performing a flushing associated with a leaf *pos* with prefix $\gamma||0$. By the union bound the probability of overflow is upper bounded by $T2^{-K}$.

2.3 The ORAM Scheme

Given an ORAM scheme where the client has storage of size $\frac{n}{\alpha}$ for some constant α, the client can apply ORAM again on the smaller memory of size $\frac{n}{\alpha}$. After $\log(n)$ iterations, the client ends up needing storage of size $\mathsf{polylog}\,(n)$.

References

[CP13] Kai-Min Chung and Rafael Pass. A simple oram. Technical report, DTIC Document, 2013.