Obliviate: portable, efficient, and crash-consistent secure deletion enforced using the Rust compiler

Speaker: Eugene Chou (euchou@ucsc.edu)

Users have a legal right to securely delete their data stored on services that they use. Secure deletion requires that deleted data is irrecoverable, even if an attacker has physical access to the underlying device. Overwrite erasure is a traditional secure deletion technique; it erases data by overwriting in-place with random patterns. Unfortunately, overwrite erasure requires the ability to perform in-place updates, which is not supported by flash media, which is ubiquitous, and WORM media. This makes securely deleting data on storage solutions such as cloud incredibly difficult, due to the lack of knowledge of where data is placed and the storage media it is placed on, as well as the use of replication and versioning.

We designed Obliviate to be a portable and efficient secure delete system. Obliviate is implemented as an interposition layer for POSIX-compliant systems that uses the technique of *large erasable storage* to enable arbitrary applications to securely delete any written data. Large erasable storage is simply described as the recursive application of cryptographic erasure in a tree of encryption keys. The leaves of the tree encrypt data blocks, and internal keys encrypt their children. Secure deletion of a data block is carried out by updating the path to the data block with fresh keys from the root and erasing the old root key. This technique is used by state-of-the-art secure delete systems in the form of *key management schemes*: data structures that manage the accessibility and revocation of encryption keys.

Prior work on Obliviate focused on crash consistency. Crash consistency for a secure delete system boils down to ensuring that data is written with the updated key management scheme that provides its associated key atomically. State-of-the-art secure delete systems are currently either not crash consistent (e.g., allow data corruption), or resort to expensive crash consistency mechanisms (e.g., data journaling). We previously proposed the usage of the *stable key principle* along with atomic in-place sector writes to simplify and accelerate Obliviate's crash consistency mechanism. Unfortunately, the use of the stable key principle would require us to intermittently re-encrypt all data in the system to uphold secure delete invariants for overwritten data.

Obliviate's new design aims to provide the same portability and crash consistency guarantees as the old design, but without the need to rewrite data for security. To achieve this, we use the *single-use key principle*. This principle ensures that overwritten data is never encrypted using the same key as the data that it overwrites, thereby mitigating the need for re-encryption. Obliviate enforces this principle using the typestate programming pattern in Rust. The typestate

pattern, while efficient due to Rust's promise of zero-cost abstractions, also drastically reduces the amount of effort needed to establish confidence in the correctness of Obliviate's secure delete key management.