# Apache Teaclave: Fearless Confidential Computing

If it compiles, it works, and it's trustworthy

Yu Ding, The Apache Teaclave Project (dingelish@gmail.com)

## Introduction

Confidential computing is born to be security critical. It aims at processing confidential data and guarantee data security. Many solutions, including homorphic encryption based solution, secure multiparty computation based solution, claim to have solved the problem but we rarely see them in practice. Observed challenges:

1. The size of trust computing base is huge
2. Lack of quality assurance on the entire dependency tree
3. Does not support fine grained auditing
4. Not flexible enough for updates

Apache Teaclave (incubating) is an open source universal secure computing platform, making computation on privacy-sensitive data safe and simple. It aims at solving all of the above challenges and helps user to fearlessly design and implement their own confidential computing software.

Apache Teaclave leverages trusted hardware to guarantee in-memory data isolation or encryption, and guarantees memory safety by using Rust programming language and a self-maintained pypy, which makes the software extensible and easty to use. Meanwhile, Apache Teaclave stays on "least privilege principle" with the finest granularity by redefining the standard library. Overall, though Apache Teaclave incorporates millions lines of codes in its codebase, one can easily verify the entire codebase and all dependencies and conduct compile time auditing or execution time auditing. Ultimate goal of Apache Teaclave is to guarantee "If it compiles, it works, and it's trustworthy" because the software cannot pay the price of any confidential data leakage.

## Details of Apache Teaclave's Solution

**Memory Safety Guarantees** First, the most critical problem is to reduce the number of memory safety issues. Apache Teaclave solves this by using Rust programming language. Though Rust itself has unsound issues and provides "unsafe superpower", we can easily create strong compiler plugins with the help of the well-organized Rust compiler to use state-of-art pattern based bug finding algorithm.

**Hybrid Memory Safety** Second, Apache Teaclave guarantees hybrid memory safety which includes not only Rust, also C/C++/Python by following the "memory safety rules-of-thumb":

1. Unsafe components should be appropriately isolated and modularized, and the size should be small (or minimized).

2. Unsafe components should not weaken the safe, especially, public APIs and data structures.
3. Unsafe components should be clearly identified and easily upgraded.

**Non-bypassable Security Check** By defining annotations and attributes with the help of Rust compiler, we can confirm that the execution time dataflow satisfies required rules such as "all outputs must be processed by func A".

**Redefines the Standard Library** Apache Teaclave redefines the "standard library" to conduct "least privilege principle" on the entire software stack. Unallowed libc usages are easy to locate and remove.
We observed that the POSIX libc is pretty dangerous and is always one of the critical part in exploits. We successfully identified a number of security vulnerabilities in multiple popular confidential computing frameworks, and we believe the root cause is that these frameworks aim at supporting unmodified or least modified regular software built on top of POSIX libc.
Instead, Apache Teaclave redefines the standard library of Rust and creates manifest to control every I/O related function. Different from the default standard library (like libc) which allows I/O, the modified standard library (sgx_tstd) does not allow any I/O. Violation of I/O causes compiler error. The consequence is that Apache Teaclave's standard library forces developers to manually audit each I/O statement in their software and removes unnecessary I/O operations.

**Dependency Modification Management** Different from modern software engineering habits, we stay on a weird and counterintuitive dependency modification policy: we keep the modifications stay on top of git history. It brings trouble when syncing with upstream (requires rebase and makes git confuses about lost commit id), but it makes code auditing more straightforward -- auditors can only look at the topmost commit and review the changes.

**Automated Security Auditing** Apache Teaclave maintains a set of well-organized dependencies and audits them daily. We can easily draw the world map with one shell command. And we use `cargo-audit` which is well maintained to check if the world of forked dependencies contains 1-day security vulnerabilities automatically. We've successfully identified lots of bad practices by using cargo audit and manual inspection on the world map.

**Formal Verification** Apache Teaclave leverages state-of-art algorithm to make sure its access control implementation is bug-free.

## Conclusion

Apache Teaclave is designed to process confidential data with best performance. It requires more human efforts on software development but allows fearless secret data processing. We successfully helped 10+ projects/start-ups (including almost half of the CCC projects) to onboard with the Apache Teaclave project and help them secured their computing infrastructure by design.