# MPK-Based In-Process Crypto Key Vault with WRPKRU Instrumentation

Teaching Assistant
E0-256 - Computer Systems Security
Indian Institute of Science

Autumn 2024

## 1 Problem Description

Memory Protection Keys (MPK) is a hardware feature in modern Intel processors that enables efficient memory isolation within a single address space. Unlike traditional mechanisms requiring kernel-mode transitions or page table modifications, MPK allows threads to rapidly change access permissions to designated memory regions using a dedicated PKRU register. This capability is particularly valuable for security-critical applications where both performance and isolation are paramount. Just as discussed in the ERIM paper, MPK can enforce isolation between trusted and untrusted components within the same process.

In this project, we consider a binary that dynamically links two shared libraries: one trusted and one malicious. A cryptographic key will be stored in the process address space. The goal is to ensure that only the trusted library can directly access and use this key. Neither the main binary nor the untrusted library should be able to read or write the key region directly. The main function can perform cryptographic operations on the key only by invoking the API provided by the trusted library. This setup enforces strict in-process isolation, ensuring that cryptographic operations on the sensitive key material are confined to the trusted component while preventing any leakage or misuse by untrusted code.

## 2 Phase 1 : MPK Setup and Basic Isolation

This phase involves becoming familiar with Intel MPK and storing a cryptographic key inside the process address space. A trusted shared library will be created to provide cryptographic functions that can operate on this key.

The following components must be implemented:

- A function defined directly in the `main` binary that attempts to access the key.

- A function in the trusted library that legitimately accesses the key for cryptographic operations.
- A function in the untrusted library that attempts to access the key.

A function in the `main` binary that attempts to access the key directly, and a function in the trusted library that accesses the key legitimately, will be tested under different MPK configurations. When calling from `main`, the PKRU permissions will be set to disallow both read and write access to the key region. When invoking the trusted library, the PKRU will be configured to allow read access during the cryptographic operation.

Additionally, an untrusted library will be added to the system. While calling functions from this untrusted library, PKRU will again be configured to disallow access to the key region. This ensures that neither the `main` binary nor the untrusted library can directly access the key, and only the trusted library is permitted to use it for cryptographic operations.

### 2.1 Deliverables

In this phase, you should have a proper setup of Intel MPK along with the shared libraries and the cryptographic key mapped into the process address space. By configuring the `wrpkru` instructions appropriately, it should not be possible to access the key when invoking functions defined in the `main` binary or in the untrusted library. Only the trusted library should be able to perform cryptographic operations on the key through its API.

## 3 Phase 2 : Binary Instrumentation of Untrusted Components

As we already know, the `wrpkru` instruction or the APIs of `libmpk` can be invoked from user space. This means that the `main` function or the untrusted library itself could potentially modify the PKRU settings that were configured in the previous phase. For example, even if you set PKRU permissions to disallow access and then invoke a function in the untrusted library, that function could itself execute a `wrpkru` instruction to reconfigure the register and gain access to the key.

The goal in this phase is to prevent such arbitrary permission changes. Before running the binary, both the main binary and the untrusted library must be instrumented to remove all occurrences of the `wrpkru` instruction or any direct `libmpk` API calls. Instead, only legitimate `libmpk` calls should be inserted through binary instrumentation. For example:

- When a call instruction targets a function defined in the `main` binary or untrusted library, instrumentation should insert the appropriate `wrpkru` configuration that disallows access to the key region.

- When a call instruction targets a function in the trusted library, instrumentation should insert `libmpk` APIs that configure the permissions to allow read-only access for the cryptographic operation.

Tools such as `Dyninst` or similar binary instrumentation frameworks can be used for this task. Alternative approaches are also acceptable, as long as the outcome ensures that untrusted components cannot arbitrarily set PKRU permissions. The critical requirement is to eliminate all untrusted uses of `wrpkru` and enforce a controlled use of MPK via trusted instrumentation.

## 3.1 Deliverables

In this phase, you should be able to perform appropriate binary instrumentation. For simplicity, focus only on a subset of `libmpk` APIs that are critical for changing permission bits, rather than handling every possible API. The objective is to remove or replace these sensitive API calls to prevent untrusted components from modifying PKRU settings arbitrarily.

During evaluation, we will test whether these APIs are successfully removed or replaced. Additionally, the disassembly of both the binary and the untrusted library after instrumentation will be examined to verify that the appropriate checks and transformations have been applied correctly.