Design

- The basic design of the driver is very simple. It loads the identified PCIe device the usual way and uses MMIO to access the device's registers.
- The driver uses a char device to expose device features to the users. The user library uses this char device interface to communicate with the driver and provide encryption/decryption and other functionalities to the end user.
- Device has a config struct used to store the device's configuration. The device keeps the last used config in this struct and any new call to `open()` will use this config by default. This default config can also be changed by the user using the exposed *sysfs* interface.
- Each `struct file*` has it's own config in it's private data too which the user can modify.
- Any encrption/decrption which exceeds the device's buffer size for MMIO or DMA will be split into multiple chunks will be carried out one-by-one.
- The driver uses a device specific mutex to prevent races.

Implementation

Driver (*cryptocard.c*)

- The following is the *driver struct* for the **cryptocard** device. `id_table` and custom handlers are discussed below.

```
static struct pci_driver driver = {
    .name = DRIVER,
    .id_table = id_table,
    .probe = on_device_load,
    .remove = on_device_unload,
};
```

- `id_table` : Used to match the device to the driver.

```
static const struct pci_device_id id_table[2] =
{
    {PCI_DEVICE(0x1234, 0xdeba)}
};
```

- `on_device_load()` : Called when the device is loaded. Enables the device and create mapping to access device IO using *MMIO*. Configures *DMA* for the device. Creates chardevice and sysfs variables for the device. Initializes *wiatqueue* and *mutex* to be used by other functions working on the device. Setup interrupt handler for the device.
- `on_device_unload()` : Called when the device is unloaded. Disables the device and removes the mapping.
- `cryptdev_config` : Used to store the global configuration for the device and handler specific configuration (stored in `struct file*`).

```
struct cryptdev_config {
    /* use DMA or MMIO */
    uint8_t use_dma;
    /* use interrupts or not */
    uint8_t use_intr;
    /* a and b values
    (in same format as in the device memory) */
    uint32_t key;
};
```

- `driver_pvt` : Device private data for the driver (unique for each device).

```
struct driver_pvt {
    /* Kernel VA for MMIO acess */
    u8 __iomem *hwmem;
    /* physical MMIO address, length and flags */
    unsigned long mmio_start, mmio_len, mmio_flags;
    /* config for the device */
    struct cryptdev_config config;
    /* pointer back to the "struct pci_dev" containing the data */
    struct pci_dev *pdev;
    /* DMA working or not */
    u8 dma_enabled;

    /* flag to store wether interrupt happened or not */
    u8 int_rec;
    /* for process waiting for interrupt */
    wait_queue_head_t wq;

    /* mutex for device */
    struct mutex dev_mutex;
};
```

Char Device (*chardev.c*)

- file operations for the char device.

```
static const struct file_operations cryptdev_fops = {
    .owner          = THIS_MODULE,
    .open           = cryptdev_open,
    .release        = cryptdev_release,
    /* for enctyption */
    .read           = cryptdev_read,
    /* for decryption */
    .write          = cryptdev_write,
    /* for config modifications */
    .unlocked_ioctl = cryptdev_ioctl,
    /* for mapping card buffer to userspace */
    .mmap           = cryptdev_mmap,
};
```

- `crypt_device_private` : Private data for the char device (unique for each `file*`).

```
struct crypt_device_private {
    uint8_t chnum;
    /* device specific data */
    struct driver_pvt* drv_pvt;
    /* config for handler */
    struct cryptdev_config config;
};
```

- `cryptdev_open()` : Called when the device is opened. Setups private data (`struct crypt_device_private`) for the handler and store it in `file->private_data` .
- `cryptdev_release()` : Called when the device is closed. Frees the private data (`struct crypt_device_private`) for the handler.
- `cryptdev_read()` : Called when `read()` is called on fd for the device. Take the data from the user buffer and use one of MMIO/DMA/mapped mode to encrypt the data and store it back in the buffer. NOTE: Pass `buffer` as `NULL` to encrypt in mapped mode.
- `cryptdev_write()` : Called when `write()` is called on fd for the device. Same as `cryptdev_read()` but for decryption.
- `cryptdev_ioctl()` : Called when `ioctl()` is called on fd for the device. Used to set/get the configuration for the device - KEY, DMA, and interrupts.
- `cryptdev_mmap()` : Called when `mmap()` is called on fd for the device. Used to map the device buffer to userspace. Uses `io_remap_pfn_range` to do the mapping.

Core (*core.c*)

Contains lowest level functions to do MMIO/DMA encryption/decryption and the interrupt handler.

- `mimo_op()` : Encrypt/decrypt data using MMIO.

- `dma_op()` : Encrypt/decrypt data using DMA.
- `irq_handler()` : Interrupt handler for the device.

Other important files

- `cryptocard_regs.h` : Contains register offsets and other information about the device.
- `cryptcard_user.h` : Contains information (offsets, sizes, etc) which is important to both driver and user libraries.

Testing

- Test cases provided in `eval-test/` directory were used to verify the correctness of the driver for each configuration.
- Each test case was modified to have a variable length input. This was done to test the driver for cases when input is larger than the buffer size.
- Each benchmark was run 10 times with 10 MB input file. No major vairations in CPU usage were observed for each case.

Benchmark Results

- Average benchmark CPU usage for each bechmarks:

S. No.	Config	%CPU (user)	%CPU (system)	%CPU (total)
1	MMIO	72.00	3.67	75.67
2	MMIO + Interrupt	71.00	2.67	73.67
3	DMA	0.01	8.54	8.55
4	DMA + Interrupt	0.01	0.14	0.15
5	Mapped	70.33	5.33	75.67
6	Mapped + Interrupt	68.25	3.50	71.75