הוספת רכיב אבטחה עבור RISC-V Multiprocessor

תכנון ממשק ראשוני

נכתב ע"י אורי בן צור בהנחיית פרופ' אבי מנדלסון  
08.05.2022

# הגדרת הרכיב

בשלב זה של הפרויקט, נרצה לתכנן רכיב פשוט, שיוכל לבצע הצפנת AES (להלן "הרכיב").  
הרכיב יוכל לייצר מפתחות קריפטוגרפים בטוחים, אך גם יוכל לקבל מפתחות מבחוץ.  
הרכיב יוכל לשרת מספר רק של תוכניות במהלך ריצת המערכת, אך יוכל לשרת רק תוכנית אחת בעת אחת.  
הרכיב יתחבר למעבד בעזרת bus כלשהו.  
הנחת העבודה בשלב זה היא שהרכיב יהיה off-core אבל on-chip.

# אתחול

לפני תחילת הפעולה של הרכיב, על המשתמש לאתחל את הרכיב לפי צרכיו. המשתמש יוכל לייצר מפתח קריפטוגרפי באמצעות הרכיב.  
מנקודת מבט המעבד – הרכיב ישלח פקודה לרכיב, והרכיב ישלח את המפתח ייצר למעבד.  
במידת הנדרש, ניתן להשתמש בפונקציונליות זו ליצירה IV אקראי עבור ההצפנה.  
  
אינני בקיא מספיק באופן שבו מחוברים רכיבי המעבד. שליחת המפתח יכולה להיות לתוך כתובת זיכרון שייקבע המשתמש והמעבד ישלח לרכיב או לתוך חוצץ ייעודי שנממש בחומרה.  
ניתן גם לאחסן את המפתח ישירות ברגיסטר/רגיסטרים של המעבד, אך אינני בטוח אם זה אפשרי/רצוי בגלל (1) טעמי נוחות ותאימות וגם (2) הדבר עלול להאט את פעולת המעבד, אך זה תלוי OOO.

# שליחת פקודה

עבור כל תצורת שימוש, המעבד יידרש לשלוח פקודה לרכיב ההצפנה. הפקודה תמיד תכיל מפתח הצפנה ו-IV (ייתכן 0).

# תצורת פעולה 1 – Dynamic Shared Memory

## תקציר

בתצורת פעולה זו, המשתמש יקצה שני חוצצים במרחב הזיכרון שלו – אחד למידע שברצונו להצפין והשני עבור המידה המוצפן. רכיב ההצפנה לכתוב המידע המוצפן לתוך החוצץ המתאים בזיכרון.  
המעבד ישלח לרכיב האבטחה את כתובות החוצצים ואת גודלם, יחד עם מפתח ההצפנה ו-IV.  
הרכיב ישלח למעבד Interrupt כאשר יסיים.  
עבור תהליך המשתמש, הוא יוכל לבחור האם הפעולה תהיה חוסמת.

## יתרונות:

* פשוט לשימוש
* לא תופס זמן מעבד

## חסרונות:

* פוטנציאל להתקפת בומרנג
* תקורה גבוהה במקרה של הצפנות קטנות
* חשוף להתקפות ערוץ-צד?

## סדר הפעולות:

1. Command
2. While(data\_to\_encrypt.size > 0)
   1. Read
   2. Write
3. ‘done’ interrupt

Dynamically allocated buffers

Read

Write

Memory

*‘done’ interrupt*

*command*

Encryptor

Core

# תצורת פעולה 2 – Static Shared Memory

## תקציר

בתצורת פעולה זו, נרצה שמערכת ההפעלה תקצה מספר מסגרות זיכרון פיזיות סטטיות עבור רכיב האבטחה שדרכן תהליכי המשתמש יוכלו להעביר אינפורמציה לרכיב. המסגרות ימופו למרחבי הזיכרון של המשתמשים לפי בקשתם ומדיניות מערכת ההפעלה. בתוך המסגרות הללו יוכל המשתמש להקצות חוצצים משלו, ואופן התקשורת עם הרכיב יהיה זהה לתצורה 1.   
כמות המסגרות הפיזיות הסטטיות תלויה במערכת ההפעלה.

## יתרונות:

* מונע את הפוטנציאל להתקפת בומרנג.
* יחסית פשוט לשימוש
* לא תופס זמן מעבד

## חסרונות:

* דורש הקצאה סטטית של מסגרות זיכרון פיזיות (מסגרות סטטיות זה משאב יקר)
* כמות הזיכרון שניתן להקצות עבור הצפנה נקבעת מראש ע"י מערכת ההפעלה
* תקורה גבוהה במקרה של הצפנות קטנות
* חשוף להתקפות ערוץ-צד?

## סדר הפעולות:

1. Command
2. While(data\_to\_encrypt.size > 0)
   1. Read
   2. Write
3. ‘done’ interrupt

Statically allocated buffers

Read

Write

Memory

*‘done’ interrupt*

*command*

Encryptor

Core

# תצורת פעולה 3 – Direct Encryption

## תקציר

בתצורת פעולה זו, המעבד ישלח באופן ישיר לרכיב חבילות להצפנה. למען פשטות, נוכל לדרוש שהחבילות יהיו בגודל קבוע מראש (עבור 128b-AES, נרצה חבילות בגודל 128 ביטים). המעבד ישלח לרכיב את החבילות (אלה ייכתבו לתוך חוצץ ייעודי ברכיב), וכאשר הרכיב יסיים, הוא ישלח את החבילה המוצפנת למעבד.  
כמובן שתצורת פעולה זו אפשרית אך ורק אם הוספנו למעבדים חוצצים ייחודיים עבור השימוש ברכיב (או שהרכיב יוכל לכתוב ישירות לרגיסטרים של המעבד – כאמור, לא אידיאלי). ייתכן שנוכל לממש תצורה זו ע"י MMIO.

## יתרונות:

* תקורה נמוכה במקרה של הצפנות קטנות
* פחות חשוף להתקפות ערוץ צד?
* לא דורש הקצאה של זיכרון

## חסרונות:

* תקורה גבוהה במקרה של הצפנות גדולות
* במידה והמעבד לא תומך ב-OOO, תצורת פעולה זו עלולה לפגוע בביצועי המעבד – על מנת להימנע מכך יהיה עלינו לממש רכיב הצפנה שמבצע פעולות הצפנה ב- מחזורי שעון.

*command*

*Encrypted data*

*data*

Encryptor

Core

# שתי ליבות ויותר

מאחר והגדרת הפרויקט המקורית היא עבור שיתוף רכיב ההצפנה בין שתיים או יותר ליבות, אסקור כאן מספר רעיונות שהיו לי בנוגע לחיבור הרכיב למספר ליבות:

## Ring Bus

חיבור הרכיב למספר ליבות גדול מ-1 יהיה ע"י שימוש במנגנון דמוי Ring bus.

### יתרונות:

* פשטות, פרוטוקול ידוע ומוכח

### חסרונות

* תקורה גבוהה במקרה של מספר גדול של ליבות

## Separate Busses

לרכיב יהיו כניסות כמספר הליבות במעבד, וכל ליבה תהיה מחוברת לכניסה אחרת ותוכל לפנות לרכיב בצורה בלתי תלויה. כאשר ליבה תפנה בבקשה לרכיב, היא תחכה עד אשר הוא יגיב לה באישור לבקשה.

# הערות:

## DMA

עבור תצורות 1+2 נציין שבעת כתיבת הרכיב לחוצץ הממוקם בזיכרון, נרצה שהמעבד יבצע invalidate לכניסות המתאימות ב-cache.

## On Core VS Off Core

בעת כתיבת המסמך יצאתי מנקודת הנחה שהרכיב ישב מחוץ לליבת המעבד. במידה ואנו מעוניינים לכלול את הרכיב כחלק מהליבות אך משותף להן, נשתמש ברכיב זה כ-Execution Unit לכל דבר, שתקבל ארבעה רגיסטרים כארגומנטים: 128 ביט להצפנה, מפתח הצפנה, IV, יעד. דרושים רגיסטרים בגודל 128-ביט. למען פשטות, ייתכן שנקבע 4 רגיסטרים קבועים שימשו לפעולה זו.  
מימוש עבור שיתוף הרכיב בין מספר ליבות:  
יחידת ה-Instruction dispatch של כל ליבה תשלח פקודות ל-buffer ייעודי של יחידת ההצפנה, שתבצע את הפקודה כאשר תהיה פנויה. נוכל גם להחזיק buffer לכל ליבה (מימוש יותר פשוט, פחות יעיל), ויחידת ההצפנה תעשה Round Robin בין החוצצים (ובכך, בין הליבות)