

רב הנסתר על הגלוי - עולם ההצפנה ב-Linux

מאת יהונתן אלקבס

הקדמה

אחסון הוא אבן בניין בכל מערכות המחשוב של ימינו, בין אם מדובר במחשב אישי, שרת או טלפון חכם - כולם מכילים רכיבי אחסון. המונח data at rest ("מידע במנוחה" בתרגום לעברית שבורה) מתייחס אל נתונים המאוחסנים פיזית בפתרון אחסון דיגיטלי בכל תצורה דיגיטלית - ענן, מסדי נתונים, data warehouses, ארכיונים, גיבויי off-site וכד'. בין אם זה תמונות מביכות מהטיוול המשפחתי או קוד המקור של טסלה - הכל חייב לשבת איפשהו.

משפחת האימונים של מידע במנוחה הם רבים, החל מקבלת גישה לנתונים באופן דיגיטלי מתוכם או על ידי גניבה פיזית של אמצעי האחסון עצמם. בשנייה שמישהו נוסף נחשף למידע שלכם אתם מפסיקים להיות הבעלים הבלעדיים שלו.

ובכן, בסל אפשרויות ההגנה מפני גישה, שינוי או גניבה של מידע ישנם 2 אמצעי אבטחה בסיסיים - שימוש באמצעי הזדהות אקטיבי והצפנה קריפטוגרפית של הנתונים עצמם; כאשר במקרה הסביר ביותר אנחנו נראה שילוב של השניים. אפשרויות אבטחה אלו מכונות באופן כללי Data At Rest Protection - DARP. הצפנת הכונן עליו יושבת מערכת ההפעלה מספקת בדיוק את זה.

בסביבת Windows הפתרון המוכר מאז ומתמיד היה BitLocker. לאור העובדה שהוא מגיע built in עם מערכת ההפעלה, נתמך חומרתית ומצפין בצורה שקופה למשתמש את כל הכונן עליו יושבת מערכת ההפעלה - אין צורך בשפגאט מחשבתי בשביל להבין מדוע הוא הפתרון המוביל כבר מעל 15 שנה.

לעומת זאת, כשמחפשים פתרון הצפנת כונן לסביבת Linux נתקלים במספר פתרונות רב ולא פעם עולות השאלות "אז מה הפתרון הטוב ביותר?" וכמובן - "אבל רגע רגע אם אני צריך רק --הכנס שם של פיצ'ר בודד-- מה הכי עדיף לבחור?"



המאמר הקרוב הוא הראשון מבין סדרה של 2 מאמרים אשר מתמקדים בפתרונות ההצפנה בסביבת Linux. המאמר הקרוב יתעסק בסקירה של מגוון רכיבי הקרנל אשר מהווים את אבני הביניים למגוון פתרונות ההצפנה ב-Linux ולאחר מכן יסקור את אותם הפתרונות (8 במספר) והשוני ביניהם כאשר המאמר השני יכיל deep dive להעמקה ומימוש פתרון אחד - LUKS.

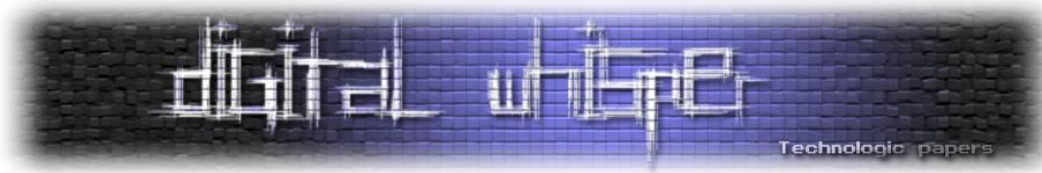
חשוב לי לציין כי המאמרים מניחים שלקורא ידע בסיסי מקדים בסביבת Linux אבל עם זאת מסבירים ברמה מפורטת ככול הניתן על המושגים השונים ולאורכם מצורפים קישורים ומקורות מידע נוספים להעמקה למי שיחפוץ בכך. כמו כן, לאורך המאמר לא יהיו שום נוסחאות מפוצצות ומסובכות של הצפנה, המאמר מתמקד בנושאים ברמה הפרקטית.

טוב אז מה יהיה לנו פה היום?

במסגרת המאמר הקרוב נסקור מספר נושאים, החל מתשתית **Device Mapper** אשר מהווה שכבת אבסטרקציה למיפוי התקני אחסון פיזיים, הצגת מודולי **Crypto-API**, **dm-crypt**, **Keyring**, **dm-verity** ותפקידם בקרנל, שימוש ב-**cryptsetup** בסביבת ה-**userspace** וכלה בהבדל בין הצפנה ברמת מערכת הקבצים לרמת הכונן.

לאחר שנבין כיצד כל אלו עובדים ומנגנים ביחד נעבור להציג את הפתרונות המובילים לביצוע ההצפנה עצמה. נדבר על הצפנת **TrueCrypt** אשר הייתה חלוצת הדרך ועל **VeraCrypt** שהיא fork עדכני ממנה, לאחר מכן נציג את **CryFS** אשר מהווה פתרון להצפנה בענן תוך השוואה אל **gocryptfs** בתור פתרון אלטרנטיבי, נציג את **Fscrypt** שהיא הפתרון של גוגל לסביבת אנדרואיד אשר באה להחליף את **eCryptfs**, נראה את הצפנת מערכת הקבצים של **OpenZFS** ולבסוף נציג גם את כוכבת המאמר הבא - **LUKS**. בסיום המאמר נסכם במספר טבלאות את החוזקות, החולשות והשוני בין כל הפתרונות.

אם כל אלו נשמעים לכם יותר כשמות של פוקימונים מאשר שמות של כלים והצפנות מוכרות - מעולה! יש לכם הרבה מה להפיק מהמאמר. הצעד הראשון מתחיל בפרק הבא.



הכנה למזג

כשזה מגיע למערכות הפעלה ישנם 2 סוגי אבסטרקציות של אחסון - volume & partition:

Volume - מקום אחסון יחיד אשר מנוהל על ידי מערכת קבצים אחת.

Partition - חלוקה לוגית של מקום אחסון פיזי. יכול להכיל או לא להכיל מערכת קבצים והפורמט לא מחויב להיות אחיד לכל אורכו. כלומר, אנחנו מדברים ב"סה"כ על חלק בדיסק שעבר אֶלּוּקְצִיָה. בעברית צחה נהוג לקשר לכך את המילה "מחיצה".

באמצעות partitioning ניתן לחלק כונן (SSD, HD) למספר volumes. מערכת ההפעלה תתייחס אל כל volume כאילו היו כוננים שונים ותיתן להם שמות (C:, D:, Z:) עם מערכת קבצים משלהם. עם זאת חשוב לציין כי הימצאות של volume לא מחייבת מחיצה. מצד שני, ישנם volumes שמכילים נתונים שמקורם ביותר ממחיצה אחת.

ללא ספק המונחים קצת מבלבלים. אם זה עוזר, גם Microsoft בעצמה טועה לא פעם במונחים ב-advisor-ים שהיא מפרסמת.

במערכות מבוססות Linux, ה-volumes מנוהלים על ידי ה-LVM (Logical Volume Manager) וניתנים לתמרון באמצעות פעולת mount. בצד הנגדי, במערכות Windows מבוססי NT ה-volumes מנוהלים על ידי הקרנל באמצעות ה-LDM (Logical Disk Manager).

על רגל אחת - Linux Storage Internals

כשזה מגיע לעולם הלינוקסי יש מונחים שחייבים להכיר לפני שנכנסים לעומקם של הדברים. סביבת Unix מכילה device files (לרוב יהיה ניתן לראות אותם תחת תיקיית `/dev`) שמהווים ממשק לדרייבר של התקן כזה או אחר (מקלדת, עכבר, כונן וכו'). בצורה כזו אפליקציות יכולות לבצע אינטראקציה עם התקנים על ידי שימוש בדרייברים שלהם באמצעות קריאות מערכת קלט\פלט רגילות. סטנדרטיזציה של קריאות מערכת אלו מפשט משימות תכנות רבות ומוביל למנגנוני קלט\פלט עקביים במרחב ה-user-space ללא קשר לפונקציות בנחלת ההתקן.

במילים פשוטות - מדובר בשכבת אבסטרקציה שהמפתחים של הקרנל יישמו על מנת שלאפליקציות המשתמש יהיה נוח להתממשק לפעולות ליבה של כתיבה וקריאה מכונני אחסון חיצוניים.

הקרנל הלינוקסי מכילה תשתית תוכנה שמטרתה לספק דרך גִּנְרִית ליצירת שכבות אחסון וירטואליות. אותה תשתית בעצם אחראית על מיפוי התקני אחסון פיזיים אל 'התקני אחסון וירטואליים' אותם היא מנהלת ברמתה. לתשתית הזו נתנו את השם (המאוד מקורי) **device mapper**.

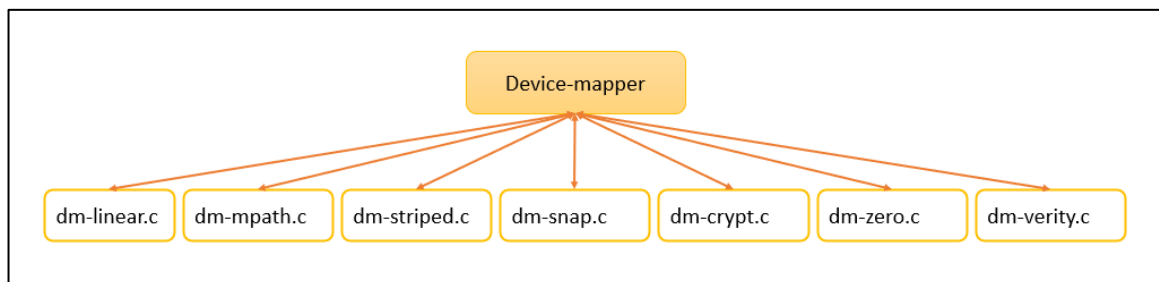
דרך טובה להגדיר את ה-device mapper (או dm בקצרה) היא על ידי כך שהיא מאפשרת יצירת פונקציונאליות ב-storage stack בכך שהיא יוצרת שכבה וירטואלית ופועלת עליה (על ידי מיפוי כלל פעולות ה-I/O מול אותה שכבה לוגית) אל מול ההתקנים הפיזיים. המיפוי הנ"ל משתמש במבנה נתונים בשם device mapper table אשר פשוט אומר איך כל סקטור (512 בתים) של השכבה הוירטואלית ממופה לסקטור בכונן הפיזי (פחות או פחות דומה לפעולה של הקצאת RAM על ידי ה-page table שמתרחשת בזיכרון וירטואלי).

כאמור, device mapper פועלת על ידי העברת מידע הלוך חזור מבלוקים וירטואליים אל התקני האחסון. היתרון המאוד בולט בכך הוא ניהול החלוקה והעובדה של **שלקרנל שליטה מלאה על התהליך**. לדוגמה, היא יכולה לשנות את המידע שעובר בזמן אמת - פיצ'ר שמאפשר לה לבצע הצפנת כונן בזמן אמת. כלומר, מ"ה קוראת בלוק בלוק מהזיכרון ומפענחת\ מצפינה אותו "on the fly", בצורה זו לא נדרש (אם כי אפשר לצרכי טיוב) לטעון קטעי נתונים גדולים לזיכרון.

לכן, זה כנראה לא יפגיע אתכם לדעת ש-Device mapper מהווה אבן בניין משמעותית בקרנל הלינוקסי. כראייה אפשר להסתכל על שלושה פרויקטים מוכרים בעולם הלינוקס שעושים בה שימוש:

- **Docker** - שימוש בוירטואליזציה ברמת מערכת ההפעלה ליצירת פלטפורמה להרצת יישומים עם כל התוכנות הפריפריאליות (ספריות, מסדי נתונים, קבצי קונפיגורציה וכו') בחבילה מאורגנת אחת (קונטיינר בלעז) תוך הפרדה של סביבת העבודה. ובכן, על פי [הדוקומנטציה](#), device mapper הוא אחד מהמודלים בקרנל שמנהל אספקטים חשובים בתשתית של docker בשביל כך.
 - **LVM2** - Logical Volume Manager, מנהל את ה-volumes-ים הלוגיים למטרות הקרנל. באמצעותו ניתן למפות מספר כוננים פיזיים ל-volume וירטואלי בודד ולשלוט על גודלו למטרות שירותים שנזקקים לשטח אחסון משתנה כמו חוות אחסון. בצורה דומה, ניתן להשתמש ב-LVM לביצוע גיבויים על ידי לקיחת snapshots של ווליום לוגי.
 - **RAIDs אפליקטיביים** - שימוש בכוח העיבוד של השרת ומערכת ההפעלה לניהול חלוקת המידע ופקודות \O על פני מספר כוננים פיזיים (במקום בחומרה ייעודית כמו שנעשה ב-RAIDs חומרתיים). גם הם עושים שימוש בתשתית של device mapper.
- כמו כן, בכל פעם שאתם רואים מודל בקרנל הלינוקסי שימוש ב-"dm", תדאו שהתוכנה שלו מתחברת ישירות אל התשתית של device mapper. חלק מהמודלים יושבים ישירות על device mapper וחלקם נכתבו במטרה לשפר ולהרחיב את הביצועים שלו.

מודלים מוכרים לדוגמה: *dm-crypt*, *dm-cache*, *dm-integrity*, *dm-verity* ועוד מלא מלא. ניתן לראות את הרשימה המלאה [בדוקומנטציה](#) של הקרנל או פשוט [בקוד המקור](#) עצמו.



האופן בו נכתבה תשתית המודלים מאפשר יישום של מספר שכבות לוגיות זו על גבי זו; כל שכבה כזו נוצרת על ידי הגדרת "a device mapper target" עבור אותה שכבה. ה-*target* מכיל את הקוד של הטמעת הפונקציונליות שהשכבה הווירטואלית מתכוונת לבצע ולפיו שולט בפעולות I/O שמעביר לה ה-*device-manager*.

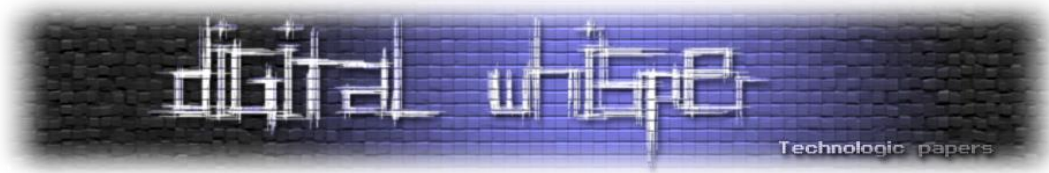
בצורה זו למשל נוכל לעשות שימוש ב-*dm-crypt* בשביל להצפין קובץ\כונן ולאחר מכן לעשות שימוש ב-*dm-integrity* על מנת לוודא את אמינות המידע שהוא מכיל לאורך זמן באמצעות פונ' *checksum* ואימותים קריפטוגרפיים אחרים.

אם נסתכל בקוד המקור של *device-mapper* נוכל לראות מהם חלק מהתכונות שכל *target* יכול להגדיר:

```

struct target_type {
    uint64_t features;
    const char *name;
    struct module *module;
    unsigned version[3];
    dm_ctr_fn ctr;
    dm_dtr_fn dtr;
    dm_map_fn map;
    dm_clone_and_map_request_fn clone_and_map_rq;
    dm_release_clone_request_fn release_clone_rq;
    dm_endio_fn end_io;
    dm_request_endio_fn rq_end_io;
    dm_presuspend_fn presuspend;
    dm_presuspend_undo_fn presuspend_undo;
    dm_postsuspend_fn postsuspend;
    dm_preresume_fn preresume;
    dm_resume_fn resume;
    dm_status_fn status;
    dm_message_fn message;
    dm_prepare_ioctl_fn prepare_ioctl;
    dm_report_zones_fn report_zones;
    dm_busy_fn busy;
    dm_iterate_devices_fn iterate_devices;
    dm_io_hints_fn io_hints;
    dm_dax_direct_access_fn direct_access;
    dm_dax_zero_page_range_fn dax_zero_page_range;
    dm_dax_recovery_write_fn dax_recovery_write;
    /* For internal device-mapper use. */
    struct list_head list;
};
  
```

רב הנסתר על הגלוי - עולם ההצפנה ב-Linux-



בצורה כזו נוצר ה-crypt_target שמיצג את dm-crypt (כאמור, נתעמק ב-dm-crypt בהמשך):

```
static struct target_type crypt_target = {
    .name = "crypt",
    .version = {1, 24, 0},
    .module = THIS_MODULE,
    .ctr = crypt_ctr,
    .dtr = crypt_dtr,
    .features = DM_TARGET_ZONED_HM,
    .report_zones = crypt_report_zones,
    .map = crypt_map,
    .status = crypt_status,
    .postsuspend = crypt_postsuspend,
    .preresume = crypt_preresume,
    .resume = crypt_resume,
    .message = crypt_message,
    .iterate_devices = crypt_iterate_devices,
    .io_hints = crypt_io_hints,
};
```

מכיוון שליבת המאמר היא הצפנה בעולם הלינוקסי ולא איך עובדים מודלים בקרנל ברמת ה-low level, ניקח צעד אחורה ונימנע מלהסביר את השדות הנ"ל (למרות שזה מקום מעולה להתחיל ממנו למי שמתעניין ב-"איך דברים עובדים באמת" ב-Linux).

זה פחות או יותר הזמן הנכון להזכיר עוד מודל חשוב בקרנל והוא Crypto-API. הממשק של Crypto-API מספק מגוון רחב של צפנים קריפטוגרפיים ושיטות הצפנה למודלים בקרנל שמתעסקים בקריפטוגרפיה (למשל IPsec, dm-crypt וכד'). הממשק מכיל את כל פונקציות ה-hash המוכרות וחלק נרחב מהצפנים. כך למשל אפשר למצוא בתוכו את הדרייבר של CRYPT (אחראי על block cipher), הדרייבר של HASH ואת CRC (מנגנון למציאת שגיאות במידע דיגיטלי).

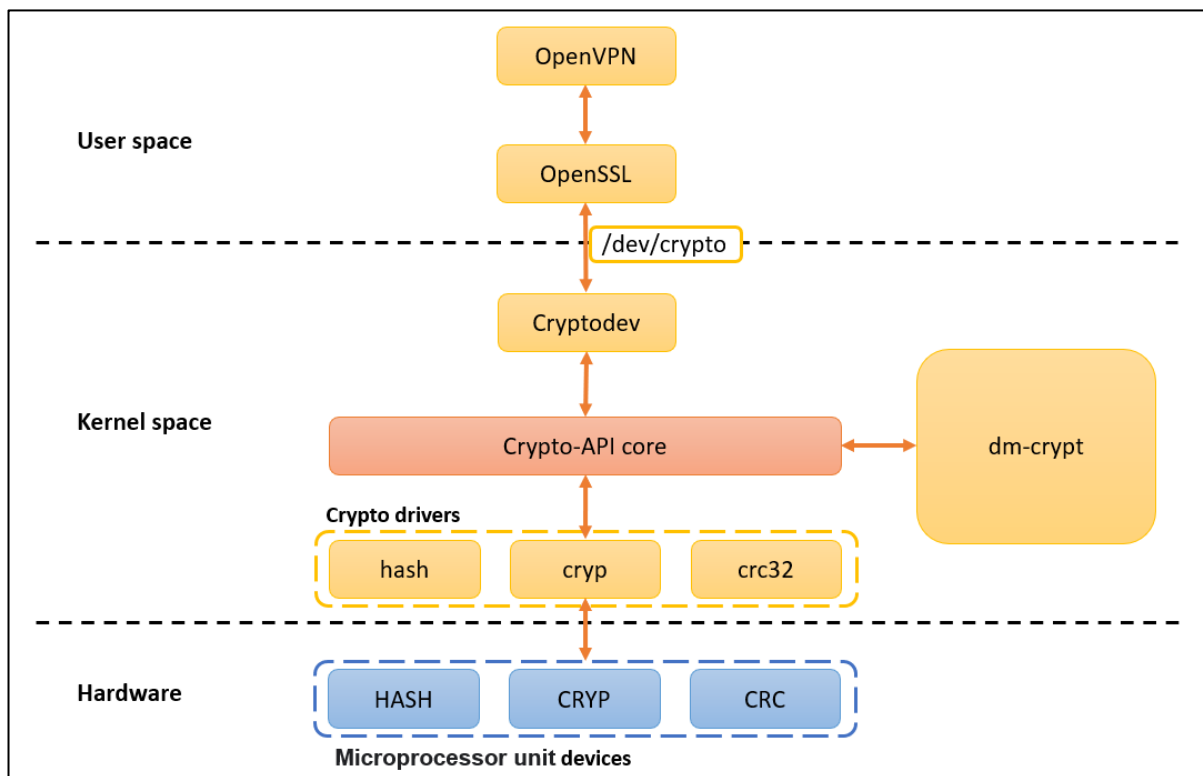
חלק נוסף ממנו הוא Cryptodev אשר מהווה ממשק לניהול [ioctl](#) דרך `/dev/crypto` עבור אפליקציות ב-userland.

על מנת לראות אילו צפנים זמינים לנו במערכת ניתן להריץ את הפקודה:

```
$ cat /proc/crypto
```

ניקח דוגמה מחיי היום-יום על מנת להמחיש את הארכיטקטורה הכללית של Crypto-API - **חיבור לתוכנת VPN** (תבחרו את תוכנת ה-client המועדף עליכם ל-VPN, גם ככה כולם רק מימשו wrapper יפה יפה ל-OpenVPN ועכשיו מוכרים אותו תחת המותג שלהם).

הארכיטקטורה הכללית של הדוגמה תראה בצורה הבאה:



ניתן לראות כיצד הרכיבים במרחב המשתמש מתקשרים עם הרכיבים במרחב הקרנל ואלו בתורם מתממשקים אל החומרה עצמה. בסופו של יום כשמערכת ההפעלה באה לעשות שימוש בפרוטוקול VPN כזה או אחר (IPsec, L2TP, PPTP, TLS, OpenVPN, SSH, MPLS) היא משתמשת בצפנים מוכרים (DES, AES, MD, SHA, RSA) בשביל להקים את התעלה בינה לבין השרת ובכך להעביר תעבורה מוצפנת. במידה ואתם מתחברים מהפצת Linux - מזל טוב אתם עוברים דרך crypto-API ☺

Keyrings

המנגנון הבא שנדבר עליו הוא **Keyrings**. בסה"כ מדובר בקונספט שמוכר לנו מהעולם האמיתי - צרור מפתחות: שמירה של כל המפתחות שיש לנו (אלו שפותחים את הבית, אלו שפותחים את הרכב, את המחשב ואת המשרד בעבודה) במקום אחד כדי שיהיה לנו נוח לסחוב אותם, להשתמש בהם ולהיות קבוע בבקרה אם אחד מהם לא נוכח.



רוב סביבות העבודה הלינוקסיות (GNOME, XDE, Xfce וכד') עושות שימוש כזה או אחר ב-gnome-keyring בשביל הפיצ'ר הנ"ל. באמצעותו ניתן לשמור את המפתחות ssh, GPG וכל המפתחות שנשמרים על ידי אפליקציות אחרות כגון הדפדפן של Chromium (יש אפליקציות שלא משתמשות בו מבחירה). ניתן גם לשמור סיסמאות ותעודות.

דיפולטיבית keyrings שמור באמצעות "master password" (לרוב זאת תהיה סיסמת ה-login של המשתמש אך לא בהכרח) כאשר לכל יוזר על המכונה תהיה את הסיסמה שלו. חלק מההפצות של לינוקס (כמו Ubuntu) מגיעות דיפולטיבית עם GUI ל-keyring בשם [seahorse](#) וחלק (כמו Kali) מגיעים בלי. בכל מקרה, ניתן למצוא את המפתחות של כל משתמש בתיקיית הבית תחת `~/.local/share/keyrings`

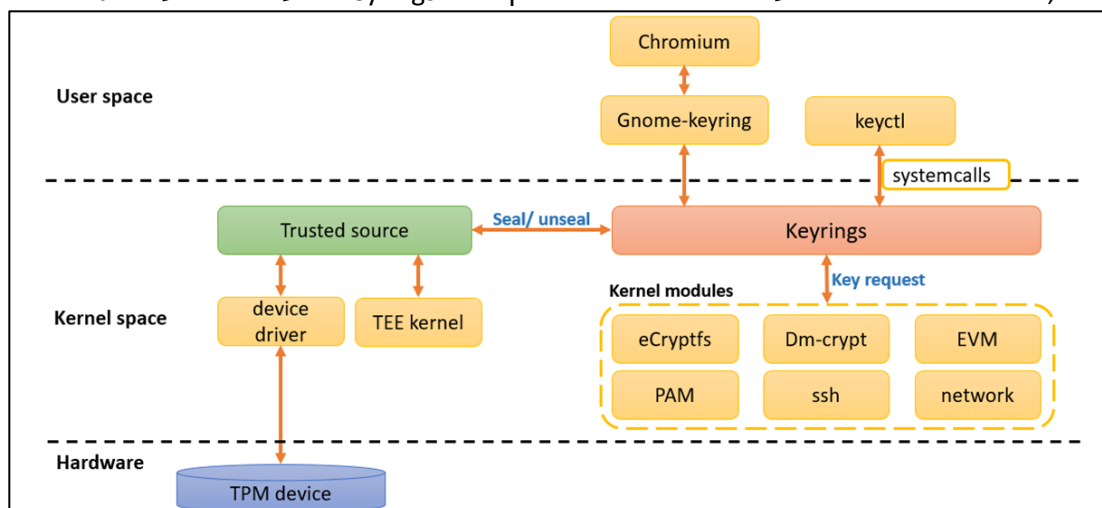
```
(jon@Jonikamoni)-[~/tools/PEASS-ng/winPEAS/winPEASbat]
└─$ ls -l ~/.local/share/keyrings
total 16
-rw-r--r-- 1 jon jon 15 Nov 16 2021 default
-rw----- 1 jon jon 1214 Jul 23 08:53 Default_keyring.keyring
-rw----- 1 jon jon 328 Nov 19 2021 login.keyring
-rw----- 1 jon jon 207 Nov 19 2021 user.keystore
```

ממשק ה-userspace של keyrings נקרא (בצורה מאוד מפתיעה) `keyctl` ודרכו המשתמש יכול להוסיף\לעדכן\למחוק לקרוא את מפתחות האישיים שלו. כל בקשה כזו עוברת באמצעות `syscall` למודל בקרנל. כמו כן, מודלים נוספים בקרנל (כגון `dm-crypt`, תשתיות רשת וכו') יכולים לפנות אל keyrings ולבקש ממנו מפתח ספציפי.

כמו כן, מונח נוסף שחשוב להכיר כשמדברים על keyrings ב-Linux הוא `Trusted keys`. הרעיון מאוד פשוט - ניקח אזור ש-"מנותק" מה-userspace ואפילו מהקרנל לחלוטין ונשמור בו את הסודות (מפתחות) שהכי חשובים לנו. בצורה כזו, גם אם כל המכונה נפרצה התוקף לא יוכל להשיג את המפתחות. לרוב, הפתרון של הסוגיה הנ"ל הוא `HSM` (Hardware Security Module) ובפרט `TPM` (Trusted Platform Module) - סטנדרט של חומרה ייעודית (מיקרו-קונטרולר שמלחם ללוח העם ליתר דיוק) שכל ייעודה זה להחזיק סודות ובאמצעות אינטגרציה וממשק ייעודי עם הקרנל לשלוף אותם לפי הצורך.

באנגלית צחה קוראים לרעיון "hardware root of trust". לאחרונה יש דיבור על אלטרנטיבה ל-TPM בסביבה לינוקסית והיא `TEE` (Trusted Execution Environment) אשר מספקת פתרון בזמן ריצה. לא ניכנס לפרקטיקה כי זה לא נושא המאמר אבל אני מצרף [מקור מעולה](#) לצעדים ראשונים למי שמתעניין.

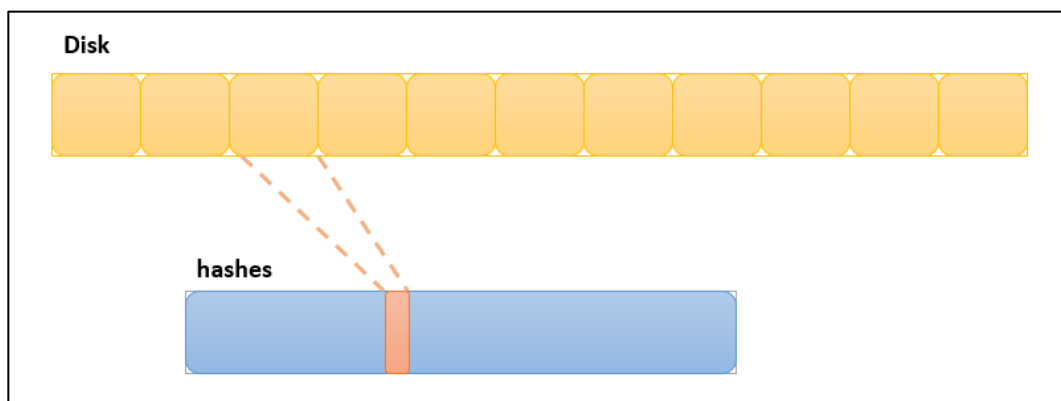
אם ככה, אנחנו בשלים להסתכל על התמונה הכוללת והמקום של keyrings במערכת ההפעלה Linux:



dm-verity

מנגנון נוסף ששווה להכיר בקצרה הוא **dm-verity** אשר מספקת שלמות כונן (disk integrity בלעז) עם תקורה מינימלית ושיקוף ליישומים - אפליקציות ב-userspace אפילו לא מודעים בכלל לעובדה ש-dm-verity קיימת.

הרעיון מאחורי dm-verity הוא למעשה די פשוט. חלקו את הדיסק לבלוקים ועבור כל בלוק, רשמו את ה-hash של הנתונים המאוחסנים בתוכו. כשמגיע הזמן לקרוא נתונים מהדיסק, צריך לאמת כל בלוק שנקרא מול ה-hash שחושב קודם לכן. מכיוון שכפי שכבר אמרנו, קיימים מיקרוקונטרולים ייעודיים שכל מטרתם זה להאיץ את פעולת המעבד בעת חישוב hash, אין זה מפתיע כי התקורה הביצועית מינימלית. כמו כן, hashes הם בסיס string אז הם לא תופסים הרבה מקום נוסף.



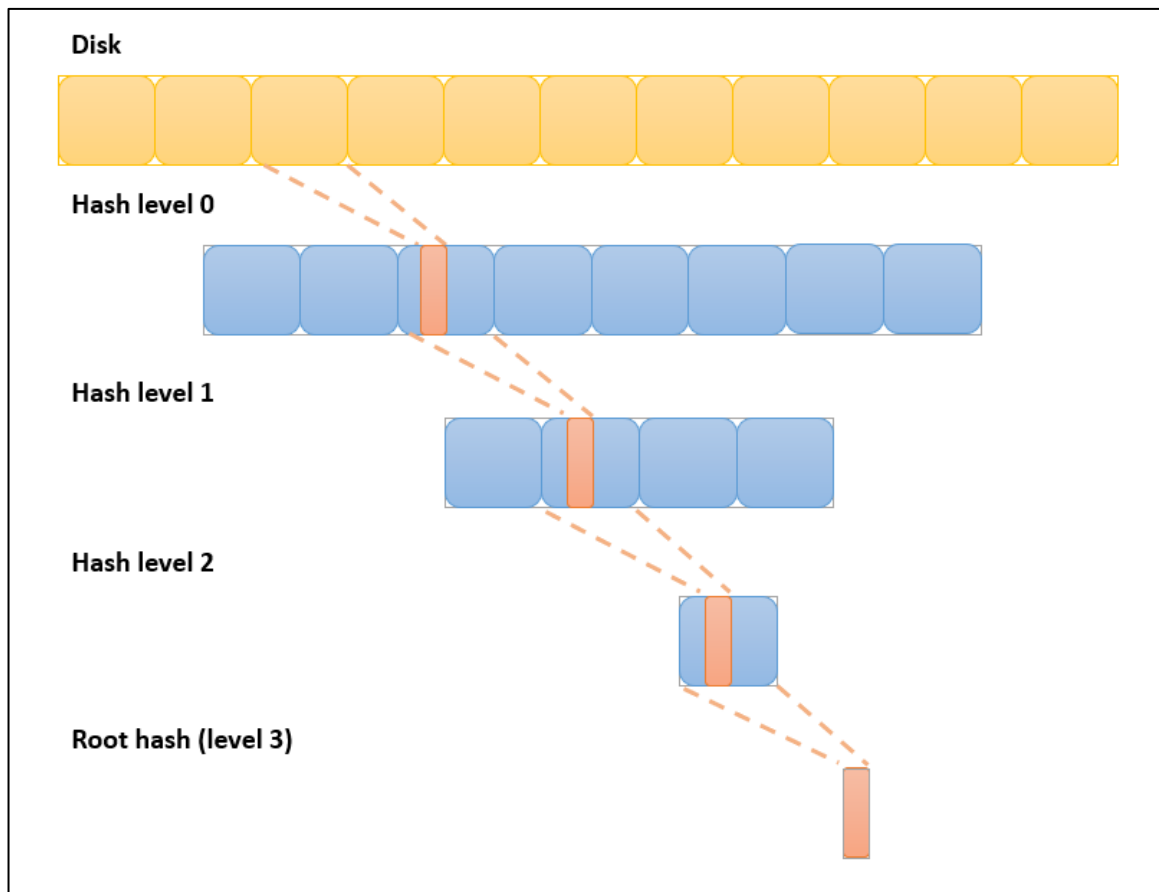
במידה וקראתם את הפסקה הקודמת בזהירות, שאלות שאמרות לצוץ לכם הן: היכן מאוחסנים כל ה-hashes כשהמחשב כבוי, ואיך נדע שאנחנו יכולים לסמוך על הגיבובים האלה ושלא בוצע בהם איזשהו שינוי כשאנחנו טוענים אותם בחזרה?

כאשר dm-verity מגן על דיסק, הקרנל מונע מכולם לכתוב לאותו התקן. במילים אחרות, הכוננים של dm-verity הם **תמיד במצב קריאה בלבד**. זה אחד הפיצורים הבולטים ביותר של התוכנה להבדיל מפונקציות שלמות אחרות שמבוצעות ברמת מערכת הקבצים.

ובכן, אמרנו ש-dm-verity מייצרת מלא מלא hash-ים כדי לשמור על אמינות המידע. אבל, **מי שומר על שלמות המידע שלהם?** קטע מצחיק, למעשה dm-verity מגנה על עצמה באמצעות שימוש בפונקציונאליות שלה בעצמה בצורה רקורסיבית, כיצד?

אם נתייחס אל מערך הגיבובים הענק ש-dm-verity יוצרת בתור דיסק ונחלק אותו לפי בלוקים (128 גיבובים בכל בלוק ליתר דיוק) ועל כל בלוק נריץ את פונקציית הגיבוב נקבל hash שמוודא את אמינות המידע כמו מקודם. תריצו את הפעולה הזו ברקורסיה (כל פעם בלוק הגיבובים יקטן) כאשר תנאי העצירה הוא שנותר רק hash אחד בלבד... זהו! יש לנו root hash שמגלם בתוכו את כלל האמינות של כלל ה-hash-ים.

כעת, שינוי של סיבית אחת במידע המקורי תיצור גל של שינויים אשר יקאו בצורה ישירה ב-root hash. כלומר, אם אנחנו יודעים מה צריך להיות ה-root hash, נוכל לוודא שכל המידע מהימן על ידי השוואה של string אחד פשוט מבלי לעבור על כל חלקי הכונן. תמונה שווה 7 פסקאות:



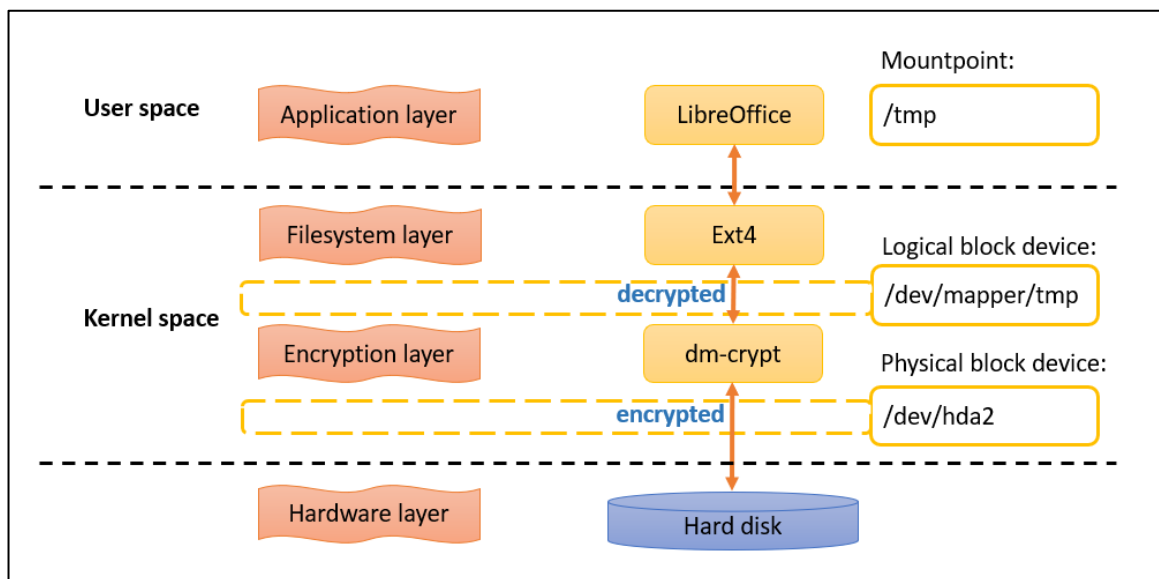
dm-crypt - הצפנה מעולם אחר

טוב אז לאחר שהבנו והכרנו מספר אבני בניין בעולם ההצפנה בלינוקס זה הזמן לכנס הכל למקום אחד ולהכיר את החלק האחרון שחסר לנו בפאזל.

dm-crypt subsystem בקרנל אשר באמצעות שימוש ב-crypto API מהווה שכבת אבסטרקציה ברמת ההצפנה מול התקני הזיכרון. הרעיון שלה מאוד straightforward - תן לי מפתח, צופן סימטרי, mode הצפנה ומקום ואני אצור לך כונן וירטואלי מוצפן. מעתה והילך כל הכתיבות לכונן הנ"ל יהיו מוצפנות וכל הקריאות יפוענחו. כל זאת בעוד שמערכת ההפעלה מתייחסת לאותו כונן כאילו היה כונן 'רגיל'. כלומר, עדיין יהיה אפשר לבצע mount למערכת הקבצים אליו ו\או להשתמש בכוננים שהוצפנו בצורה זהה לשם יצירת RAID volume או שמנוהל על ידי LVM. בסופו של יום יש להם את אותה תשתית, זוכרים?

אז מה ניתן להצפין באמצעות dm-crypt? ובכן, כמעט כל דבר עם שטח אחסון החל מקבצים רגילים ועד כוננים פיזיים שלמים - מדיה נתיקה (דיסק-און-קי, כונן חיצוני וכד'), מחיצות, volume-ים מסוגים שונים, אמצעי אחסון לוגיים, קבצים ועוד.

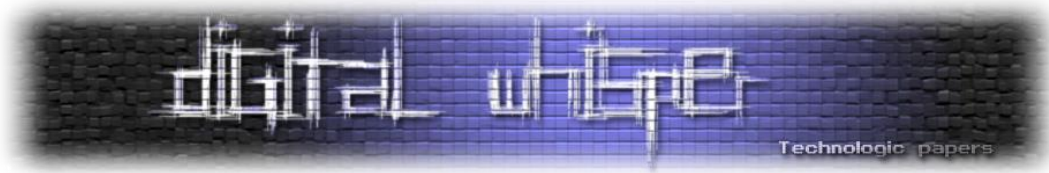
לשם ההבנה, ניקח דוגמה פשוטה מחיי היומיום: נגיד אתם רוצים לפתוח קובץ בתוכנת LibreOffice (המקבילה ל- Office בסביבת Linux וללא תג מחיר מוגזם) שנמצא בתיקיית /tmp. התהליך של ההצפנה / פתיחת ההצפנה יראה בצורה די כללית כך:



ניתן לקנפג את dm-crypt שתדרוש הזדהות (באמצעות סיסמה או מדיה נתיקה) בשלב ה-pre-boot וכך להצפין את כל המידע שעל המחשב (חוץ מהקרנל, ה-bootloader ו-initrd שאחראית על טעינת מערכת קבצים קטנה ורגעית לזיכרון בעת עליית המערכת).

כאמור, dm-crypt הוא מודל בקרנל ואין למשתמש גישה אליו, לכן בסביבת ה-userspace נעשה שימוש ב-Cryptsetup אשר מהווה כלי לממשק ה-CLI של dm-crypt ליצירה, גישה וניהול כוננים מוצפנים. יצירת כלי CLI למודל בקרנל היא מתודולוגיה לא מקורית במיוחד ב-Linux.

בנוסף, קיים כלי בשם dmsetup שבאמצעותו ניתן לקנפג את כל ה-targets של device mapper (ובניהם את dm-crypt כמובן). למרות שנראה שהוא מספק יכולות גרנדיוזיות יותר מ-Cryptsetup, הוא מסובך בהרבה לשימוש. על פי הדוקומנטציה נראה אפוא שהוא נכתב למפתחי הקרנל ופחות למשתמשים של מ"ה. לכן, על מנת לקבוע את מצבי ההצפנה של dm-crypt יהיה מוטב לנו לעבוד עם Cryptsetup ובמידה ויהיה צורך לשנות משהו ב-low level מבחינת הדרייבר של device mapper (מה שקשה לי להאמין שתצטרכו כל עוד אתם לא מפתחים פתרון הצפנה משלכם בלינוקס) יהיה נדרש לעשות שימוש ב-dmsetup.



כמו כן, ש-Cryptsetup לא תומכת במנגנון salt פחות מומלץ להשתמש בה as is, אולי חוץ ממקרים מיוחדים בהם מנגנון האימות שונה - כמו שימוש באמצעי אימות פיזיים כגון smart cards. פורמט זה מוגדר בדוקומנטציה כ-"Plain".

בשלב מסוים בחיי הכלי, התבצעה הרחבת תוכנה ולאחר refactor לייעוד הכלי, נכנס מצב נוסף לשימוש - "LUKS" אשר מכיל אל כל ה-setup ואופרציית ניהול המפתחות עבור dm-crypt. בכך הרחיבו את סט האפשרויות וגם הקלו משמעותית את השימוש. עם זאת, ישנם כמה סוגים של LUKS וחשוב לעשות את ההבחנה ביניהם (נרחיב סט הפתרונות של LUKS משמעותית במאמר השני בסדרה).

מצבי הצפנה ש-Cryptsetup תומכת בהם לשימוש עם dm-crypt (באמצעות דגל --type)

- Plain: שימוש ב-dm-crypt במצב רגיל
 - Luks: שימוש במצב הדיפולטיבי שמוגדר ל-LUKS
 - Luks1: שימוש ב-LUKS1 - הגרסה הכי נפוצה בינתיים
 - Luks2: שימוש ב-LUKS2 - הגרסה הכי עדכנית של LUKS ויישום הרחבות נוספות
 - Loopeas: שימוש במצב LoopAES (legacy)
 - Tcrypt: שימוש ב-TrueCrypt (legacy) לצרכי תאימות. נדבר עליו בהמשך.
 - Bitlk: שימוש בהרחבה של BitLocket לצרכי תאימות עם Windows. כרגע המצב מוגדר כניסיוני בלבד.
- ניתן להשתמש בכלי בצורה הבאה:

```
# cryptsetup OPTIONS action action-specific-options device dmname
```

הערה: סימון "#" בתחילת פקודה מסמל כי היא רצה בהרשאות גבוהות - root, בעוד ששימוש ב-"\$" מסמל כי הפקודה רצה בהרשאות נמוכות - User.

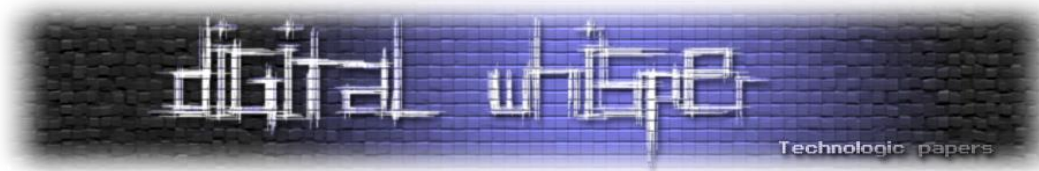
Cryptsetup מכילה סט דיפולטיבי של מצבי הצפנה מקומפלים מראש - מה שמקל על החיים משמעותית. להצגת הסט ניתן להריץ:

```
$ cryptsetup --help
```

לא תהיו מופתעים אם אגיד כי מהירות ההצפנה והפינוח חשובה מאוד בעת ההצפנה. מכיוון שמסורבל לשנות צופן הצפנה של בלוק \ התקן לאחר ההגדרה הראשונית שלו, חשוב לבדוק מראש את ביצועי dm-crypt עבור הפרמטרים הבודדים. ניתן לבצע זאת באמצעות:

```
$ sudo cryptsetup benchmark
```

אלו יכולים לתת אינדיקציה טובה לגבי החלטה על אלגוריתם וגודל מפתח לפני ביצוע ההצפנה. אם צפני AES מסוימים מצטיינים עם תפוקה גבוהה משמעותית, כנראה שמדובר באלו עם תמיכה בחומרה של המעבד ולכן נבחר בהם.



למשל במקרה של המחשב שלי:

```

jon@ubuntu22:~$ sudo cryptsetup benchmark
# Tests are approximate using memory only (no storage IO).
PBKDF2-sha1      1060238 iterations per second for 256-bit key
PBKDF2-sha256   1605782 iterations per second for 256-bit key
PBKDF2-sha512   1038194 iterations per second for 256-bit key
PBKDF2-ripemd160 686240 iterations per second for 256-bit key
PBKDF2-whirlpool 504123 iterations per second for 256-bit key
argon2i         4 iterations, 1048576 memory, 4 parallel threads (CPUs) for 256-bit key
argon2id        4 iterations, 1048576 memory, 4 parallel threads (CPUs) for 256-bit key
# Algorithm | Key | Encryption | Decryption
aes-cbc     128b  842.2 MiB/s  2346.9 MiB/s
serpent-cbc 128b   84.3 MiB/s   560.4 MiB/s
twofish-cbc 128b  198.5 MiB/s  304.8 MiB/s
aes-cbc     256b  704.8 MiB/s  1881.6 MiB/s
serpent-cbc 256b   85.4 MiB/s   564.4 MiB/s
twofish-cbc 256b  189.2 MiB/s  311.3 MiB/s
aes-xts     256b  2163.4 MiB/s 2238.2 MiB/s
serpent-xts 256b   517.9 MiB/s  430.5 MiB/s
twofish-xts 256b   268.2 MiB/s  273.8 MiB/s
aes-xts     512b  1680.8 MiB/s 1715.3 MiB/s
serpent-xts 512b   482.8 MiB/s  523.6 MiB/s
twofish-xts 512b   287.2 MiB/s  306.3 MiB/s

```

ניתן לראות שמבחינת אלגוריתם הצפנה הכי משתלם לבחור ב-AES-XTS מכיוון שממוצע מהירות ההצפנה והפיענוח שלו הכי גבוה. נקודה לא אינטואיטיבית שחשוב לחשוב עליה היא האם אנחנו רוצים לבחור בפונקציית ה-hash עם המדדים הכי גבוהים?

כאשר אנחנו משתמשים בשירות של cryptsetup על מנת להצפין בלוק של מידע אנחנו משתמשים בסיסמה (או keyfile); התוכנה עושה שימוש בסיסמה הזו בשביל ליצור ולהצפין את המפתח (master key) שישמש לכל פעולות ההצפנה ופיענוח של הבלוק. על הסיסמה המקורית היא מבצעת את פונקציית ה-hash ולאחר מכן מאחסנת את ה-hash בלבד (כדי למנוע שליפה של הסיסמה ב-clear text). לאחר מכן, בכל פעם שהמשתמש ירצה לפתוח את ההצפנה הוא יצטרך להזין את הסיסמה שלו, התוכנה תבצע עליה hash ותשווה אותה למחזורת ה-hash שיש לה, במידה והמחרוזות זהות - תאפשר פתיחת ההצפנה.

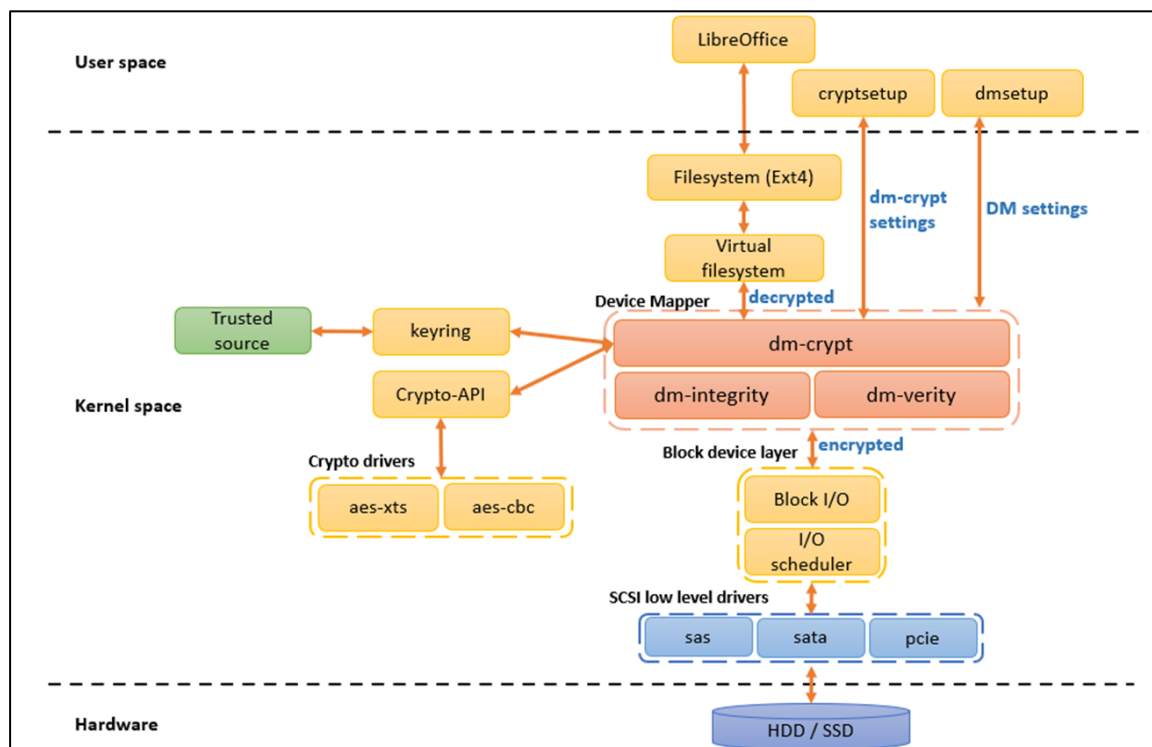
אם עקבתם אחרי כל הבלגן כנראה שהתחלתם להבין מדוע זה לא אינטואיטיבי לבחור בפונקציה המהירה ביותר - הדבר יאפשר לתוקף לבצע Brute force על הסיסמה שלנו במהירות גבוהה יותר. לכן, אולי הכי מומלץ יהיה לבחור דווקא ב-PBKDF2-whirlpool מכיוון שהיא האיטית ביותר?

ובכן, Cryptsetup חשבו על זה גם ולכן ניתן באמצעות הדגל --iter-time לקבוע את זמן בדיקת הסיסמה (ובכך את זמן קבלת מפתח ההצפנה) לאינטרוול ספציפי (למשל 5 שניות) - מה שיכול להאט משמעותית מתקפות BF.

אם נסכם את הכל, לפי תוצאות ה-benchmark כנראה שהכי מומלץ עבורי לבחור ב-AES-XTS עם מפתח 256 ביט, SHA512 עם מפתח 256 ביט וזמן אטרקציה של 2-5 שניות.

אז איך כל מה שראינו עד כה מנגן ביחד?

מכיוון ש-dm-crypt היא הליבה של פתרון ההצפנה ב-Linux (לפחות ברמה הכי low level) שווה לעשות ריכוז של כל מה שעברנו עד כה ולהבין איך פתרון העושה שימוש ב-dm-crypt יראה פחות או יותר מבחינת ארכיטקטורה:



באירור ניתן לראות כיצד כלל המודלים שהצגנו עובדים ביחד על מנת לבצע את ההצפנה:

- באמצעות Cryptsetup ניתן לקנפג את dm-crypt אשר מהווה מודל בתת-מערכת רוחבית יותר בשם Device mapper השוכנת במרחב הקרנל.
- Device mapper מהווה שכבת אבסטרקציה בין ההתקנים החומרתיים אל שכבת אחסון וירטואלית שהיא מנהלת ברמתה. על ידי מיפוי כלל פעולות ה-I/O אל מול אותה שכבה לוגית ושימוש במבנה נתונים בשם device mapper table מערכת DM שולטת על כיצד כל סקטור של השכבה הוירטואלית ממופה לסקטור בכונן הפיזי.
- בצורה הזו, dm-crypt יכולה לשלוט במידע שעובר ולהצפין אותו לפני פעולת הכתיבה לדיסק.
- מכיוון שתשתית DM מאפשרת ערימת מספר device targets זה על זה, ניתן לעשות שימוש במודלים נוספים בתשתית כמו dm-verity על גבי dm-crypt בשביל לוודא את אמינות המידע.
- dm-crypt עושה שימוש במודל ה-keyring על מנת להשתמש בסיסמאות ומפתחות ההצפנה.
- על מנת לבצע את הפעולות הקריפטוגרפיים שמעורבות בהצפנה, dm-crypt עושה שימוש במודל ה-crypto-API.
- בעת קריאת הקובץ, תוכנת LibreOffice למעשה פונה ישירות למערכת הקבצים מבלי לדעת שהיא עוברת דרך האבסטרקציה של device mapper אשר מיוצגת באמצעות מערכת קבצים וירטואלית.

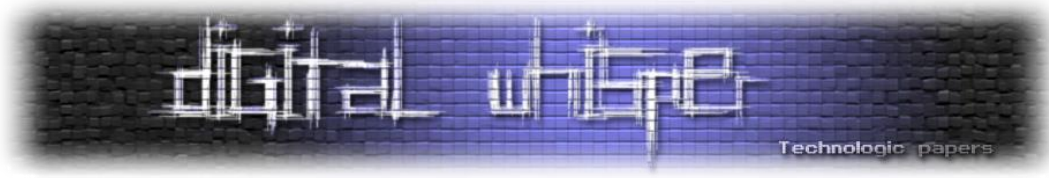
עצירה אחרונה לפני שמגיעים

במסגרת המאמר אנחנו הולכים לפגוש מספר פתרונות הצפנה בסביבת Linux. כלל הפתרונות שנוציג ניתנים לחלוקה ל-2 קבוצות עיקריות:

- הצפנה ברמת מערכת הקבצים
- הצפנה ברמת הכונן (Block device)

כלומר, סוגם נקבע על פי החלק אותו הם מצפינים. ניתן לדבר רבות על השוני ביניהם ועל עולם המאפיינים שכל שיטה מציעה אבל הנקודה תתבהר בצורה משמעותית לאחר הקריאה של כמה מהפתרונות ושיח על השוני ביניהם. כרגע נסתפק בטבלה שמרכזת את ההבדלים בגסות:

מה מוצפן	הצפנה ברמת הכונן	הצפנה ברמת מערכת הקבצים
המידע המוצפן יכול להיות	כל הכונן	קבצים
יחס עם מערכת הקבצים	דיסק, מחיצה מדיסק, קובץ (כ-loop device)	תיקייה במערכת קבצים
מצפין את metadata של הקבצים? (מספר קבצים, מבנה ספריות, גדלי קבצים, הרשאות, זמני שינויים וכו')	פועל מתחת לשכבת מערכת הקבצים: לא משנה מה יש מעליו. תוכן הכונן המוצפן יכול להיות מערכת קבצים, טבלת מחיצות, נפח מנוהל על ידי LVM או למעשה כל דבר אחר	מוסיף שכבה נוספת למערכת קבצים קיימת, על מנת שפעולות ההצפנה\ פענוח של הקבצים יקרו בצורה אוטומטית בכל פעם שמתבצעת פעולת קריאה\ כתיבה
האם ניתן להשתמש להצפנת כוננים קשיחים שלמים בהתאמה אישית (כולל טבלאות מחיצות)?	כן	חלקי בלבד (רק שמות הקבצים מוצפן, כל שאר ה-metadata גלוי)
יכול להצפין נפחי swap?	כן	לא
האם ניתן להשתמש מבלי להקצות מראש כמות קבועה של מקום עבור נפח הנתונים המוצפן?	לא	כן
האם ניתן להשתמש בו על מנת להגן על מערכות קבצים קיימות ללא חסימת גישה למכשיר- למשל שיתופי NFS או Samba, אחסון בענן וכו'.	לא	כן
האם מתפשר גיבוי offline של התוכן המוצפן בקבצים?	לא	כן



פתרונות מוכרים להצפנה בסביבת Linux

כלל פתרונות ההצפנה שנדון עליהם במסגרת המאמר נוגעים בצורה כזו או אחרת ברכיבי ההצפנה שהצגנו עד כה אך עם ייעוד שונה. בגסות ניתן לחלק את הצורך בהצפנה בסביבת Linux-ית ל-5 צרכים עיקריים.

- עבודה עם ספקיות ענן
- הצפנת קבצים\ תיקיות בודדות במערכת ההפעלה
- הצפנת כלל הכונן ומערכת ההפעלה
- תאימות cross platform
- היכולת ליצור volume נסתר - [Plausible deniability](#)

כל פתרון שנציג במסגרת המאמר עונה על מספר מהצרכים. עקב המורכבות התפעוליות של כל סביבה, לא קיים מצב של "one size fits all" ולכן לפני כל בחירה בפתרון כזה או אחר נדרש לתת את הדעת על מספר גורמים והצרכים שהם מביאים עימם.

ציטוט נחמד שראיתי בנוגע לקריפטוגרפיה לפני הרבה שנים ומלווה אותי מאז:

"Open-source for cryptography is a requirement, not just a bonus."

בקווי מתאר דומים, המסמך יתעסק במציאת פתרון קוד-פתוח ולא יסקור פתרונות שמוגנים מאחורי קובץ הרצה בתשלום (קצת אירוני לומר זאת בהתחשב בעובדה שמספר העיניים שנחשפו לקוד המקור של BitLocker קטן מ-4 ספרות ולא נראה שזה פוגע לו בפופולאריות).

כמו כן, היות וקוד פתוח עסקינן, פרויקטים ש"ננטשו" ולא בוצע אליהם pull request כבר שנים לא יכללו בסקופ של המסמך (למרות שעדיין קיימת אהדה ענפה בקהילה לגביהם). פרויקטים הנופלים תחת קטגוריה זו:

- TrueCrypt (גרסה אחרונה שוחררה ביולי 2014)
- eCryptfs (גרסה אחרונה שוחררה במאי 2016)
- EncFS (גרסה אחרונה שוחררה באפריל 2018)

TrueCrypt

לא מעט בלוגים ופורומים באינטרנט מדברים על TrueCrypt עבור הצפנת הכונן מכיוון שהכלי כולל הרבה פיצ'רים נוחים (כמו הצפנה ב-real time, תמיכה ב-Linux ו-windows, שקיפות למשתמש הקצה ותמיכה במאיצים חומרתיים במעבדים) אך האמת היא כי הפרויקט כבר לא נתמך משנת 2014. מה שמעלה דרסטית את הסיכוי כי התוכנה כוללת פרצות אבטחה ועל כן **אין לעשות שימוש ב-TrueCrypt**.



מדובר למעשה ב-fork לפרויקט של TrueCrypt שנעשה ב-2013 אשר נתמך עד היום ותומך בהצפנת כונן. הספרייה מבוססת [קוד פתוח](#) נותנת מענה לנגזרות של Debian, Ubuntu, CentOS/ Fedora, openSUSE וגם למערכות Windows ו-macOS. קיצר cross platform לפנים. בדומה ל-TrueCrypt גם VeraCrypt היא open-source (אם כי חלות מגבלות על השימוש).

אחד היתרונות של VeraCrypt הוא העובדה שהיא מאפשר הצפנה ב-real time. כלומר, כל כתיבה\קריאה לדיסק מערבת הצפנה\פיענוח ישיר. כמו כן, VeraCrypt מאפשרת הצפנה של כל מערכת הקבצים כולל שמות התיקיות והקבצים, כלל התוכן של הקבצים וה-metadata שלהם, סך הזיכרון הפנוי וכו'. הדרך בה VeraCrypt עובדת זה באמצעות יצירת כונן וירטואלי מוצפן בתוך קובץ ואז לבצע mount לכונן הוירטואלי כאילו היה כונן אמיתי.

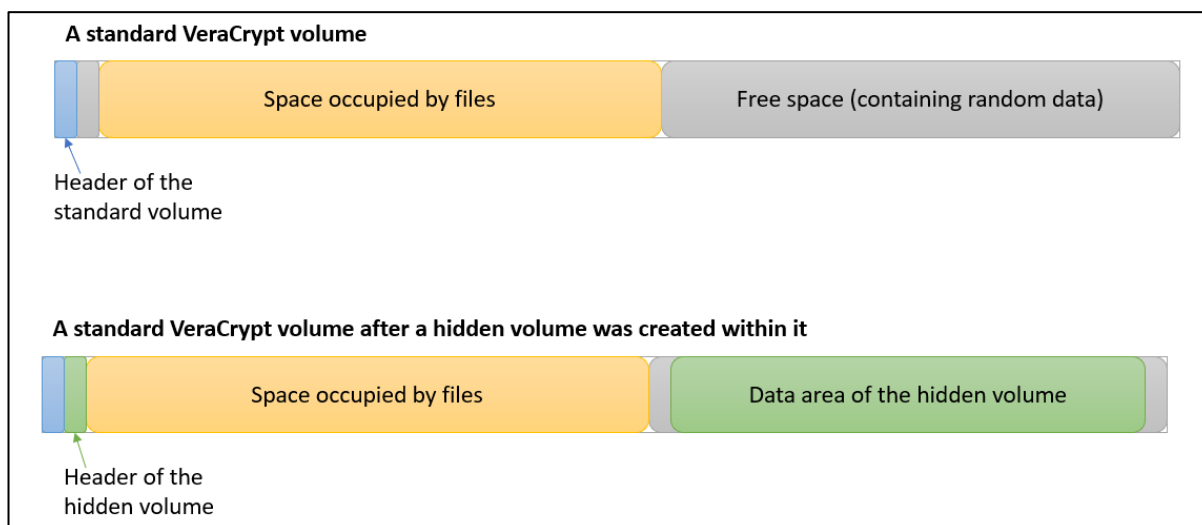
יש לכך חסרונות רבים מכיוון שמשתמע כי **VeraCrypt שומרת את מפתח ההצפנה בזיכרון ה-RAM**. כחלק מפעילותו התקינה, הדרייבר של VeraCrypt מוחק את המפתחות שב-RAM ברגע ש-volume שלה עובר unmount. עם זאת, במידה והספקת החשמל למחשב נפסקה בפתאומיות, הדרייבר שאחראי על ניהול האפליקציה (ומחיקת המפתחות) יפסיק לעבוד וחלק מרכיבי ה-DRAM ישמרו את הזיכרון שבתוכם מספר שניות לאחר שהמתח אליהם נפסק והמחשב נכבה (ואפילו למספר שעות אם הטמפרטורה נמוכה). משמע, במידה ולתקוף יש גישה פיזית למחשב שעובד הוא יכול לבצע [cold boot attack](#) ולחלץ את המפתחות מהזיכרון. למעשה, PoC למתקפה נעשה על TrueCrypt עוד ב-2008.

בשביל לענות על ליקוי האבטחה הנ"ל, בגרסה 1.24 נוסף פיצ'ר שמאפשר הצפנה למפתחות ב-RAM (אשר גורר overhead של בין 5%-15% מכוח ה-CPU ולא מאפשר מצב hibernation יותר) ואף מחיקה כוללת של המפתחות מהזיכרון במידה ומכשיר נוסף חובר למחשב. עם זאת, על פי [הדוקומנטציה](#) של VeraCrypt הם עדיין ממליצים לכבות את המחשב לאחר כל שימוש בכונן שהוצפן באמצעות התוכנה.

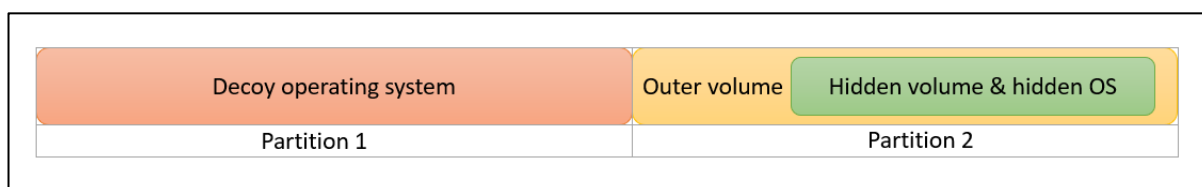
חיסרון נוסף של VeraCrypt הינו העובדה כי התוכנה **אינה מנצלת את רכיב ה-TPM**. למעשה, המפתחים (האמיצים?) של VeraCrypt ו-TrueCrypt אף טוענים כי TPM הוא לא יותר מ-bloatware וכל מטרתו להוות 'security theatre'. טענה זו אולי מחזיקה מים בנוגע למצב בו תוקף השיג הרשאות Administrator ו\או גישה פיזית למחשב בזמן ריצה (למרות שאלו מעולם לא היו חלק מסל [מטרות השימוש](#) ב-TPM) אבל רכיב חומרה לאחסנת מפתחות בהחלט יכול לעזור בכל הקשור למניעת מתקפת ה-cold boot שהוזכרה מקודם.

יתרון ייחודי, אם כי פחות רלוונטי לצרכים שלנו ויותר לצרכי פרטיות, הוא העובדה ש-VeraCrypt מאפשרת יצירת volume "נסתר" כך שבמידה ואתם חייבים למסור את הסיסמה ו\או לפתוח את הצפנת הכונן מול גורמי חוק \ לא גורמי חוק (מעולם לא היה מדובר טוב יותר ממחבט בייסבול לברך) עדיין יתאפשר לשמור

על מידע שרגיש לכם באמצעות השימוש ב-volume נסתר. פיצ'ר נשגב זה מקבל את מטבע הלשון [Plausible deniability](#) בלעז. כך זה נראה מבחינת הזיכרון:



ובכן, כמו שחלקכם בוודאי כבר הסיקו, אם אנחנו יכולים ליצור volume נסתר, מה מונע מאיתנו לשים בו מערכת הפעלה ולעלות ב-boot דרכו? ובכן, שום דבר. זה בדיוק מה שמוגדר בדוקומנטציה של VeraCrypt בפרק [decoy operating system](#). השמת 2 מערכות הפעלה על אותו כונן. לא יודע מה איתכם אבל בתור מישהו שמפעיל Tails OS מידי פעם, העובדה ש-VeraCrypt יכולה לספק פיצ'ר כזה קוסם במיוחד. [מדריך](#) קצרצר לכיצד ניתן לבצע את זה למי שמתעניין:



אנקדוטה לסיום: חלק טוענים כי VeraCrypt היא פתרון הצפנת דיסק עדיפה אפילו על BitLocker בסביבת Windows מכיוון שבניגוד אליו היא מבוססת קוד פתוח וניתנת להרצה גם על מערכות הפעלה שאינן בגרסת Windows Pro (גרסת Home לא מאפשרת דברים בסיסיים כמו התחברות לדומיין, חיבור RDP מרחוק והצפנת כונן באמצעות BitLocker).

כנראה שאם אתם "לקוחות פשוטים" לא אכפת ל-Microsoft מהאבטחה שלכם). למרות שאני חובב לא קטן של פרויקטי קוד פתוח, בשביל לצודד באמירה כזו ידרש מצדי ידע והיכרות מעמיקה יותר עם שני הפתרונות.

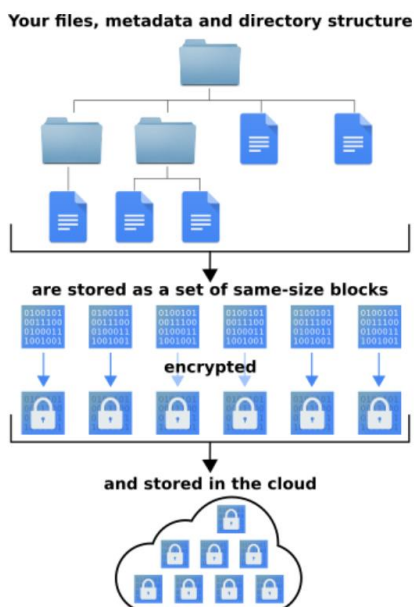
פתרון להצפנת קבצים מבוסס [קוד פתוח](#) בסביבת Linux שמדובר עליו לא מעט היא חבילת CryFS.

אמל"ק: אם אתם עובדים עם אחסון בענן מכל תצורה שהיא (Dropbox, Azure Blob, S3 bucket) וסודיות המידע רגישה לליבכם (כולל מספקיות הענן) כנראה ש-CryFS היא פתרון ששווה לכם לשקול. במידה ולא, כנראה שקיימים פתרונות טובים יותר.

מדובר בפרייקט קוד פתוח די חדש (2016) לעומת החלופות. CryFS מצפינה לא רק תוכן הקבצים אלה גם את ה-**metadata** שלהם ואת מבנה התיקיות (directory structure) בהם הקבצים יושבים. כלומר, בדומה לשאר הפתרונות שקיימים, היא יוצרת מערכת קבצים וירטואלית. היא עושה את זה בצורה מאוד פשוטה - חלוקת כלל המידע לבלוקים שווים בגודלם והצפנתם. CryFS 'זוכרת' את ההיררכיה של מבנה הקבצים (וכיצד החלוקה בוצעה) על ידי מבנה נתונים מסוג עץ שגם אותו היא שומרת בבלוק של ההצפנה.

כל בלוק מוצפן מאוחסן בתור קובץ עם ID רנדומאלי כך שכאשר אדם לא מורשה נכנס לתיקייה בה מאוחסן המידע המוצפן כל מה שהוא יוכל לראות זה בעצם אותם בלוקים בגודל זהה עם ID מוזר. מפתח הצפנה רנדומאלי מג'ונרט ומאוחסן בקובץ (ביחד עם פרמטרי קונפיגורציה נוספים) שמוגן עם סיסמה וכך כאשר המשתמש מזין את הסיסמה לאפליקציה קורה תהליך הפוך בו התוכנה ממפה את הבלוקים לפי ה-entry point של העץ שרשום בקובץ.

אלמנט נחמד של הארכיטקטורה הנ"ל הוא העובדה שניתן לנייד את כולה לשרתי קבצים ו\או **לתשתית בענן** תוך כדי שמירה על ההצפנה ואופן הפיענוח. ואכן, היתרון הבהוק של השימוש ב-CryFS הוא הסנכרון האינטואיטיבי עם ספקיות הענן תוך שמירה על zero-knowledge מצד הספקיות. מקרה קלאסי של תמונה שווה אלף מילים:



[מקור: <https://www.cryfs.org/howitworks>]

החסרון העיקרי של CryFS הוא ה-overhead שמערכת הקבצים הוירטואלית שלה יוצרת (לעומת חלופות); מהירות הקריאה מ-SSD עומדת על 170MB/s ומהירות הכתיבה סביב ה-80MB/s. זה בהחלט לא איטי אבל רחוק מהמהירויות שתוכנות אחרות הציגו. בנוסף, CryFS לא תומכת בהצפנת דיסק מלאה - פיצ'ר מרכזי שנדרש רבות.

זה לא יהיה הוגן לדבר על פתרון להצפנת קבצים בסביבת Linux אל מול הענן מבלי להזכיר את **gocryptfs**. מדובר בספרייה פופולארית נוספת מבוססת [קוד פתוח](#) אשר מייצרת מערכת קבצים וירטואלית כך שההצפנה מתרחשת ברקע ואינה מפריעה לעבודה השוטפת. פיצ'ר משמעותי בה הוא הסנכרון בין עמדות שונות המחוברת לענן. לשם הדוגמה אל ה-Dropbox: כאשר משתמשים עורכים את אותו הקובץ במחשבים שונים בו-זמנית, הם למעשה משנים קבצים מוצפנים שונים ב-Dropbox ותוכנת ה-Client של Dropbox מסוגלת להבחין בשינוי הנ"ל ולבצע את הסנכרון. עם זאת, אופן הפעולה הזה מהווה גם חיסרון. מכיוון שלכל קובץ של המשתמש נוצר קובץ מוצפן, ניתן לראות בדיוק כמה קבצים יש לך, מה הגדול כל קובץ ואיך הם בנויים בעץ התיקיות. קיימים אספקטים נוספים אבל כאן נעוץ ההבדל המהותי בין gocryptfs אל CryFS.

נקודה נוספת ששווה לדבר עליה בנושא הצפנה בענן למטרת zero-knowledge היא שחלק משירותי אחסון בענן מציעים זאת ישירות דרך הפלטפורמה שלהם ולכן שימוש בתוכנות הצפנה כגון gocryptfs ו-CryFS על פני חלופות של [יישומי client](#) של ספקי ענן הוא לא אינטואיטיבי.

למרות כל זאת, העובדה שניתן לשמור לוקאלית על קובץ הקונפיגורציה של מערכת הקבצים תוך כדי ניהול הקבצים בענן (גם במידה וקיימת גישה לקבצים מכמה מקומות שונים) ושמירה על סודיות המידע (וה-metadata של הקבצים) הופכים את CryFS לפתרון אטרקטיבי.

ext4, F2FS, UBIFS - Fscrypt

fscrypt היא פתרון הצפנת קבצים מבוסס [קוד פתוח](#) מעת **Google**. כן כן, לא אחרת מאשר גוגל עצמה. למה שגוגל יפתח פתרון הצפנה עבור לינוקס? פשוט מאוד, הוא בא לתת מענה לפרויקטים רחבים יותר שלהם - Chrome OS ו-Android.

fscrypt היא ספרייה שמערכות קבצים יכולות להתחבר אליה כדי לתמוך בהצפנה שקופה של קבצים וספריות. כאמור, פתרונות הצפנת אחסון ברמת מערכת ההפעלה פועלים ב-2 רמות: רמת מערכת הקבצים או ברמת ההתקן (block device). Fscrypt מאפשרת שימוש [בהצפנת ה-native](#) של 3 מערכות קבצים פופולאריות - ext4, F2FS, UBIFS. כנראה שהכי מוכרת מבין השלוש היא ext4 עם תמיכה רחבה מאוד בקהילה.

fscrypt פועלת בשכבת מערכת הקבצים - בשונה מ-dm-crypt אשר פועלת ברמת ההתקן. לאופן זה השפעה ענפה על פונקציונאליות הספרייה. זה מאפשר להצפין קבצים שונים עם מפתחות שונים ולהכיל קבצים לא מוצפנים באותה מערכת קבצים זה צמוד לזה. האופן הנ"ל שימושי במיוחד עבור מערכות מרובות משתמשים שבהן הנתונים at rest של כל משתמש צריכים להיות מבודדים קריפטוגרפית מהאחרים. עם זאת, למעט שמות הקבצים, **fscrypt אינה מצפינה מטא-נתונים** של מערכת הקבצים. כלומר, ניתן יהיה לראות את עץ התיקיות, הרשאות, וגודל הקבצים.

יתרון משמעותי הוא בביצועים - אנחנו מצפינים את התוכן ולא את כל המסביב. מתאפשר לקרוא ולכתוב קבצים מוצפנים מבלי לאחסן ב-cache הן את העמודים המפוענחים והן את הדפים המוצפנים ב-pagecache, ובכך להפחית כמעט בחצי את הזיכרון בשימוש ולהביא אותו בקנה אחד עם קבצים לא מוצפנים.

תוסיפו לזה את העובדה ש-fscrypt מאפשרת גם למשתמשים חסרי הרשאות גבוהות להשתמש ב-API שהיא מחייצנת ללא צורך בביצוע mount לשום דבר וקיבלתם סיבה טובה מאוד למה היא כמעט תמיד הפתרון כאשר נדרש להצפין קבצים בלינוקס.

דודה רחוקה של fscrypt היא פרויקט [eCryptfs](#) אשר בעיני רבים נחשב בעבר לפתרון הטוב ביותר להצפנת קבצים ולכן סיכוי סביר שבמידה וחיפשתם קצת מידע באינטרנט על הצפנת קבצים בלינוקס נתקלתם בשם שלה לא מעט. eCryptfs היא **Stacked filesystem**, כלומר, היא יושבת על גבי מערכת קבצים אמיתית, במקום להיות משולבת ישירות בה. למערכות קבצים מוערמות יש כמה יתרונות (תאימות על כמעט כל מערכת קבצים אמיתית היא הדוגמה הטובה ביותר), אבל גם כמה חסרונות משמעותיים. כמו כן, העובדה ש-eCryptfs מובנת כבר שנים בקרנל מוסיפה לה לא מעט זוהר.

עם זאת, **מומלץ מאוד לא לעשות שימוש בכלל ב-eCryptfs**. הספרייה לא בתחזוקה כבר מספר שנים (גרסה אחרונה שוחררה במאי 2016) ואפילו המחבר המקורי של eCryptfs ממליץ להשתמש במקום זאת בהצפנת מערכת קבצים native של לינוקס במקום (למעשה, אפילו Michael Halcrow - המפתח שדחף והוביל את הוספת eCryptfs לקרנל, היגר לפרויקט אחר והוביל את הפיתוח של מודל ההצפנה ב-Ext4 ש-fscrypt עושה בו שימוש. **דברים משתנים ומתקדמים**) בנוסף, המשתמשים הגדולים ביותר של הספרייה (Ubuntu, Chrome OS) עברו להשתמש ב-dm-crypt במקום.

טוב אז הבנו ש-fscrypt זה כנראה הפתרון למקרה ואנחנו רוצים להצפין את הקבצים בסביבה שלנו. אילו מקרים אמורים להסב את תשומת ליבנו למקרה ונחוץ לעשות שימוש ב-fscrypt?

- **מערכות מרובות משתמשים**, שכן ניתן להצפין את הקבצים של כל משתמש באמצעות מפתח משלו שנפתח על ידי סיסמה משלו.

- מערכות למשתמש יחיד שבהן לא ניתן (או לא רוצים) לייחס לכל הקבצים את רמת ההגנה החזקה ביותר. לדוגמה, ייתכן שיהיה צורך שהמערכת תבצע boot ללא אינטראקציה של המשתמש. כל הקבצים הדרושים לשם כך יכולים להיות מוצפנים רק על ידי מפתח מוגן חומרה (למשל, מעוגן ב-TPM) במקרה הטוב אם בכלל. אם הקבצים האישיים של המשתמש ממוקמים באותה מערכת קבצים, אז באמצעות dm-crypt/LUKS הקבצים האישיים של המשתמש יהיו מוגבלים לרמת הגנה חלשה דומה. מצד שני, באמצעות fscrypt, הקבצים האישיים של המשתמש יכולים להיות מוגנים במלואם באמצעות סיסמה (שונה) של המשתמש.

מתי fscrypt הוא בוודאות לא הפתרון?

- בכל מצב שבו אתם נדרשים להצפין את כלל מערכת ההפעלה (ואת המחיצה עליה היא יושבת).
- בכל מצב שבו ניתן ליצור מערכת קבצים נפרדת עבור כל מפתח הצפנה שבו תרצו להשתמש.

ZFS

OpenZFS היא מערכת קבצים מבוססת [קוד פתוח](#) מ-fork שבוצע לפרויקט ZFS של OpenSolaris לאחר שחברת Oracle קנתה את כל חברת Solaris וסגרה את מערכת הקבצים תחת closed source license. פיצ'ר שזוכה ללא מעט אהדה בקרב משתמשי ZFS הוא היכולת להצפין את מערכת הקבצים כמעט seamlessly ומעט תקורות מצד המשתמש.

הערה: מאחר ובדוקומנטציה (וגם בלא מעט פורומים) מתייחסים אל OpenZFS כ-"ZFS" אמשיך עם קו דומה למרות השוני המהותי בין השתיים.

מכיוון ש-ZFS היא מערכת קבצים מתקדמת שיישמה פתרון הצפנה תפור במיוחד למידותיה, אני מקווה שלא יפתיע אתכם כי הביצועים שלה טובים וברמת יעילות מהגבוהים ביותר שקיימים. אבל, וזה "אבל" גדול אתם תהיו חייבים לעבוד עם מערכת קבצים הזו בשביל כך.

ניכנס טיפה לפרקטיקה; ההצפנה המקורית של ZFS פועלת על גבי שכבות האחסון הרגילות של מערכת הקבצים, ולכן אינה מפריעה לשלמות הנתונים של מערכת הקבצים עצמה. נקודה נוספת היא העובדה שההצפנה לא משפיעה על פעולת הדחיסה של ZFS - הנתונים נדחסים לפני שמירתם במערך נתונים מוצפן או ב-zvol (מערך נתונים המייצג התקן בודד תחת ZFS) - מה שמיעיל את המהירות התפעול הכללית וחוסך overhead.

בניגוד להצפנות "הכל או כלום", ההצפנה המקורית של ZFS מוחלת על בסיס מערך נתונים. מה שמציע את הגמישות לערבב מערכי נתונים מוצפנים ולא מוצפנים באותה pool. כלומר, אין צורך לפענח את כל מערכי הנתונים כשמבצעים mount או import על pool מסוימת.



כמו כן, בהצפנה מקורית של ZFS קיים פיצור שנקרא "raw send" שמאפשר לשכפל מערכי נתונים מוצפנים ו-zvols מבלי לחשוף כלל את המפתח למערכת מרוחקת. המשמעות היא שניתן להשתמש ZFS replication כדי לגבות את הנתונים למיקום בסיווג untrusted (מישהו אמר ענן?) מבלי לדאוג כי הנתונים הפרטיים ששלחנו יחשפו. כלומר, אנחנו מבצעים שכפול למידע מוצפן ועד שאנחנו נחליט לפתוח אותו הוא יישאר מוצפן. יותר מכך, ניתן להוריד אותו לסביבה לוקאלית (ברמת trust) ורק שם לפתוח את ההצפנה.

דבר שמייד את ZFS לעומת הפתרונות האחרים שהוצגו (ויוצגו) במסגרת המאמר הוא העובדה שהיא **מסוגלת לבצע הצפנת קבצים בודדים וגם הצפנת כונן.**

ZFS אינה משתמשת במפתח או בסיסמה כדי להצפין ישירות את מערך הנתונים. בדומה ל-LUKS ושיטות הצפנת דיסקים אחרות כגון BitLocker, גם הצפנת ZFS מצפינה את הנתונים עם מפתח ראשי. המפתח הראשי, בתורו, יוצפן (=עטוף) במפתח בינארי שסופק על ידי המשתמש ו\או תוצאה של N איטרציות של PBKDF2 שבוצעו על הסיסמה של המשתמש. בצורה כזו ניתן לשנות את סיסמת המשתמש לקובץ\ כונן מבלי לבצע פענוח והצפנה מחדש לכל בלוק המידע.

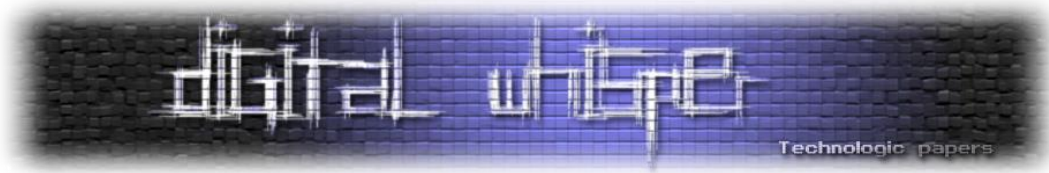
אחד החסרונות בעבודה עם ZFS הוא העובדה שהיא תחת רישיון CDDL הנחשב "חלש" ולא עולה בקנה מידה עם רישיון GPL של הקרנל שנחשב "חזק" ולכן **ZFS לא מופצת עם הקרנל "out of the box"** אלא יש צורך בהפצה של מודול קרנלי על ידי צד שלישי. מצב זה גורר לא פעם את נעילת תהליך העדכון הרגיל על ידי תלות לא מסופקת מכיוון שגרסת הקרנל החדשה (והמוצעת על ידי עדכון) אינה נתמכת על ידי פרויקט ZFSonLinux (Zol).

נסכם, המאפיינים הכלליים של הצפנת ZFS:

- **משתמשת בהצפנת AES בלבד.** אין תמיכה באלגוריתמי הצפנה של צד שלישי או פונקציות גיבוב.
- ביצועים איכותיים (דומים לקריאה\כתיבה של קבצים לא מוצפנים) ללא תלות בגודל הקובץ.
- ההצפנה חושפת metadata מינימאלי אודות המידע המוצפן
- ביצוע blind backups ושכפול של snapshot ליעדים שמוגדרים untrusted נתמכים ללא שימוש במפתח ההצפנה.

בסופו של יום, למרות ש-ZFS אכן נראית כפתרון איכותי חשוב לזכור כי היא אינה חלק מהקרנל (וגם לא נראה כי תהיה בקרוב) והיא פחות נפוצה ממערכות קבצים אחרות, כך שפתרון זה בדרך כלל אינו אופציה. נקנח עם ציטוט שזכה למספר רב של upvotes:

"The most prominent drawback of ZFS is ZFS itself: one must use the ZFS file system in order to use native ZFS encryption."



LUKS

LUKS היא הצפנת כונן [קוד פתוח](#) הנפוצה ביותר אשר פועלת ברמה נמוכה בסביבת הקרנל.

בעוד שרוב תוכנות הצפנת כונן מיישמות פורמטים שונים, ללא תאימות ולפעמים גם ללא תיעוד מעמיק, LUKS מיישמת פורמט **סטנדרטי** ובלתי תלוי בפלטפורמה לשימוש בכלים שונים. זה לא רק מקל על תאימות ותפעול בין תוכניות שונות, אלא גם **מבטיח שכולן מיישמות את אספקט ניהול הסיסמאות בצורה מאובטחת ומתועדת**. רוצים להצפין Ubuntu Thumb drive ולהיות מסוגלים לעשות לו mount גם ב-CentOS REHL ix? זה הפתרון שלכם!

החיסרון העיקרי של LUKS הוא העובדה שכל פעולת שינוי קטנה בתוכן המוצפן מחייבת פתיחה מלאה של ההצפנה ולאחר מכן סגירתה. במידה ויש לכם blob ענק של מידע שצפוי לגדול אינקרמנטלית אין ספק ש-LUKS כנראה לא עבורכם.

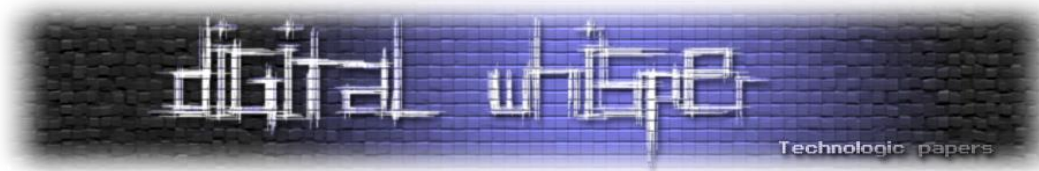
שימוש ב-dm-crypt/LUKS בצמוד להצפנה ברמת מערכת הקבצים (כמו fscrypt או ZFS) אינם סותרים זה את זה; ניתן להשתמש בהם יחד **כאשר הפגיעה בביצועים של הצפנה כפולה נסבלת**. עם זאת, זה הגיוני לביצוע רק כאשר המפתחות של כל שכבת הצפנה מוגנים בדרכים שונות, כך שכל שכבה משרתת מטרה אחרת. הגדרה סבירה תהיה להצפין את כל מערכת הקבצים באמצעות dm-crypt/LUKS באמצעות מפתח קשור ל-TPM שנפתח אוטומטית בזמן האתחול, ולהצפין את ספריות הבית של המשתמשים באמצעות fscrypt באמצעות הסיסמה שלהם בעת login.

מקבילה טובה להצפנה כפולה ברמת מערכת ההפעלה היא הצפנה כפולה ברמת הרשת - באותה צורה שאפשר לשים IPsec מעל MACsec אבל תעבורת הרשת תסבול פיזצים מכיוון שכעת לכל הפאקטות יצטרפו header-ים ענקיים ככה גם ניתן להבין את הרעיון של השמת הצפנה ברמת מערכת הקבצים (/ ZFS fscrypt) מעל dm-crypt/LUKS. **אם תרחיש איום הייחוס שלכם לא מצדיק את זה - אין סיבה לבצע זאת**.

הכל במקום אחד - סיכום פתרונות ההצפנה

כאמור ישנם מספר פתרונות רבים להצפנה בסביבת Linux. סה"כ הצגנו 8 פתרונות שונים כאשר 4 מתוכם מהווים פתרונות מוכרים ומתאימים למספר צרכים ו-4 אחרים מתאימים לפתרונות קצת יותר "אזוטריים". הדרך הקלה ביותר שאני מכיר להשוות בין הפתרונות (ואני חושב שכנראה תסכימו איתי) היא טבלה. בשביל למנוע הפצצה של תוכן שלא יהיה רלוונטי לרובכם, אני מצטרף טבלה עם סקירה רק על 4 הפתרונות השכיחים ביותר.

טוב לאחר ההקדמה הזו, בלי עיכובים נוספים - מספר טבלאות בחסות ArchWiki שמסכמות את כל היתרונות, החסרונות והשוני בין שיטות ההצפנה השונות שסקרנו.



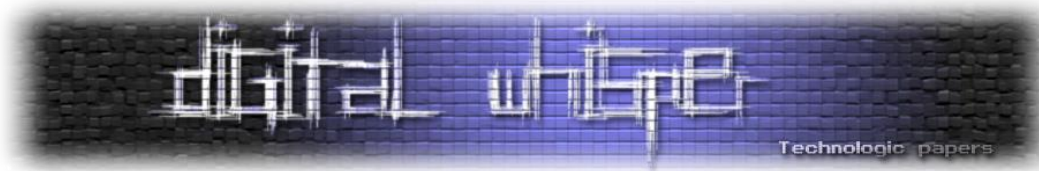
מידע כללי

Summary	dm-crypt +/- LUKS	VeraCrypt	ZFS	fscrypt
Encryption type	block device	block device	native filesystem or block device	native filesystem
Note	de-facto standard for block device encryption on Linux; very flexible	maintained fork of TrueCrypt, supporting TrueCrypt and VeraCrypt volumes	encryption feature relatively new (2019); encrypted block devices provided by ZVOL	default for Chrome OS and Android encryption
Encryption implemented in...	kernel space	kernel space	kernel space	kernel space
Cryptographic metadata & Wrapped encryption key stored in...	LUKS Header	begin/end of (decrypted) device (format spec)	DSL (dataset & snapshot layer; talk/slides)	.fscrypt directory at root of each filesystem

זכור השוני העיקרי בין פתרונות ההצפנה הוא העובדה ש-2 מתוכן מבוססות הצפנת מערכת קבצים ו-2 האחרות מבוססות הצפנת כונן. מלבד ההבדל המהותי הנ"ל, אפשר לראות כיצד כל פתרון בחר לשמור את נתוני ההצפנה.

תכונות שימושיות

Usability features	dm-crypt +/- LUKS	VeraCrypt	ZFS	fscrypt
Non-root users can create/destroy containers for encrypted data	No	No	Yes	Yes
Provides a GUI	No	Yes	No	No
Support for automounting on login	No	Yes with systemd and /etc/crypttab	No	Yes
Supports TPM usage	Yes	No	'Yes' not natively and hard to configure	No
Creates new volumes	Yes	No	Yes	Yes



אם ממשק גרפי הוא deal breaker בשבילכם אז אתם בבעיה כי גם שיש ל-VeraCrypt זה לא ממשק הגרפי הכי מתקדם (מזכיר מאוד את איך שהיה נראה פעם ממשק החלונות ב-XP). מצד שני, העובדה שניתן לפתוח את VeraCrypt ו-fscrypt תוך כדי תהליך ה-login של המשתמש מקל משמעותית על החיים ומונע הזנה כפולה של סיסמה. בהתחשב בעובדה שאנשים הם לא האוהדים הכי גדולים של סיסמאות מורכבות, העובדה שנדרש לזכור ולהזין 2 סיסמאות בכל עליית מחשב, ככל הנראה פוגעת ברמת האבטחה. עם זאת, חשוב לציין כי LUKS מאפשרת עלייה מ-TPM.

מאפייני אבטחה

Security features	dm-crypt +/- LUKS	VeraCrypt	ZFS	fscrypt
Supported ciphers	AES, Anubis, CAST5/6, Twofish, Serpent, Camellia, Blowfish,... (every cipher the kernel Crypto API offers)	AES, Twofish, Serpent, Camellia, Kuznyechik	AES	AES, ChaCha12
Integrity	optional in LUKS2	none	CCM, GCM	none
Support for salting	Yes (with LUKS)	Yes	Yes	Yes
Support for cascading multiple ciphers	Not in one device, but block devices can be cascaded	Yes AES-Twofish, AES-Twofish-Serpent, Serpent-AES, Serpent-Twofish-AES, Twofish-Serpent	No	No
Support for multiple (independently revocable) keys for the same encrypted data	Yes (with LUKS)	Yes	No	Yes
Encrypt partitions or entire disks	Yes	Yes	Yes	No
Plausible Deniability	No	Yes	No	No

כנראה הטבלה הכי משמעותית לאנשי אבטחת מידע. נתחיל מהפיצ'ר שסביר להניח תופס הכי מעט אנשים וזה היכולת ליצור מערכת הפעלה נסתרת על גבי הכונן הראשי ובכך להצדיק Plausible Deniability במקרה הצורך. כנראה שאם אתם corporates ענקי, אין לכם סיבה למצמצם לכיוון הנ"ל אבל במידה ואתם עיתונאים ו\או אקטיביסטים ו\או עוברים על החוק בצורה כזו או אחרת בשביל fun & profit אז שווה להתעמק קצת יותר בפתרון שמציעה VeraCrypt (במיוחד בשילוב עם Tails).

בהינתן שלכל לוח-אם יש על המעבד מרכיב האצה של AES זה לא מפתיע כי היא הצפנה סימטרית הנפוצה ביותר. עם זאת, אם אתם זוכרים כאשר דיברנו על אילו הגדרות צופן הכי שווה לבחור בהתאם ליכולות המכונה שלנו הראינו את התמונה הבאה:

```
jon@ubuntu22:~$ sudo cryptsetup benchmark
# Tests are approximate using memory only (no storage IO).
PBKDF2-sha1      1060238 iterations per second for 256-bit key
PBKDF2-sha256   1605782 iterations per second for 256-bit key
PBKDF2-sha512   1038194 iterations per second for 256-bit key
PBKDF2-ripemd160 686240 iterations per second for 256-bit key
PBKDF2-whirlpool 504123 iterations per second for 256-bit key
argon2i         4 iterations, 1048576 memory, 4 parallel threads (CPUs) for 256-bit key
argon2id        4 iterations, 1048576 memory, 4 parallel threads (CPUs) for 256-bit key
# Algorithm | Key | Encryption | Decryption
aes-cbc     128b  842.2 MiB/s  2346.9 MiB/s
serpent-cbc 128b   84.3 MiB/s   560.4 MiB/s
twofish-cbc 128b  198.5 MiB/s  304.8 MiB/s
aes-cbc     256b  704.8 MiB/s  1881.6 MiB/s
serpent-cbc 256b   85.4 MiB/s   564.4 MiB/s
twofish-cbc 256b  189.2 MiB/s  311.3 MiB/s
aes-xts     256b  2163.4 MiB/s 2238.2 MiB/s
serpent-xts 256b   517.9 MiB/s  430.5 MiB/s
twofish-xts 256b   268.2 MiB/s  273.8 MiB/s
aes-xts     512b  1680.8 MiB/s 1715.3 MiB/s
serpent-xts 512b   482.8 MiB/s  523.6 MiB/s
twofish-xts 512b   287.2 MiB/s  306.3 MiB/s
```

בצורה כזו ניתן לדעת אילו הצפנה תהיה הכי אפקטיבית עבור המכונה האישית שלנו. מהטבלה ניתן לראות למשל ש-ZFS תומכת רק במשפחת AES הבסיסית ולא בנגזרות שלה - מה שכול הנראה פוגע ישירות במהירות שלה. כמו כן, העובדה ש-VeraCrypt מאפשרת לשרשר מודולי הצפנה (אם כי ברור כי התקורה של פעולה כזו תהיה בשמיים) מוסיפה לה נקודות על אבטחה.

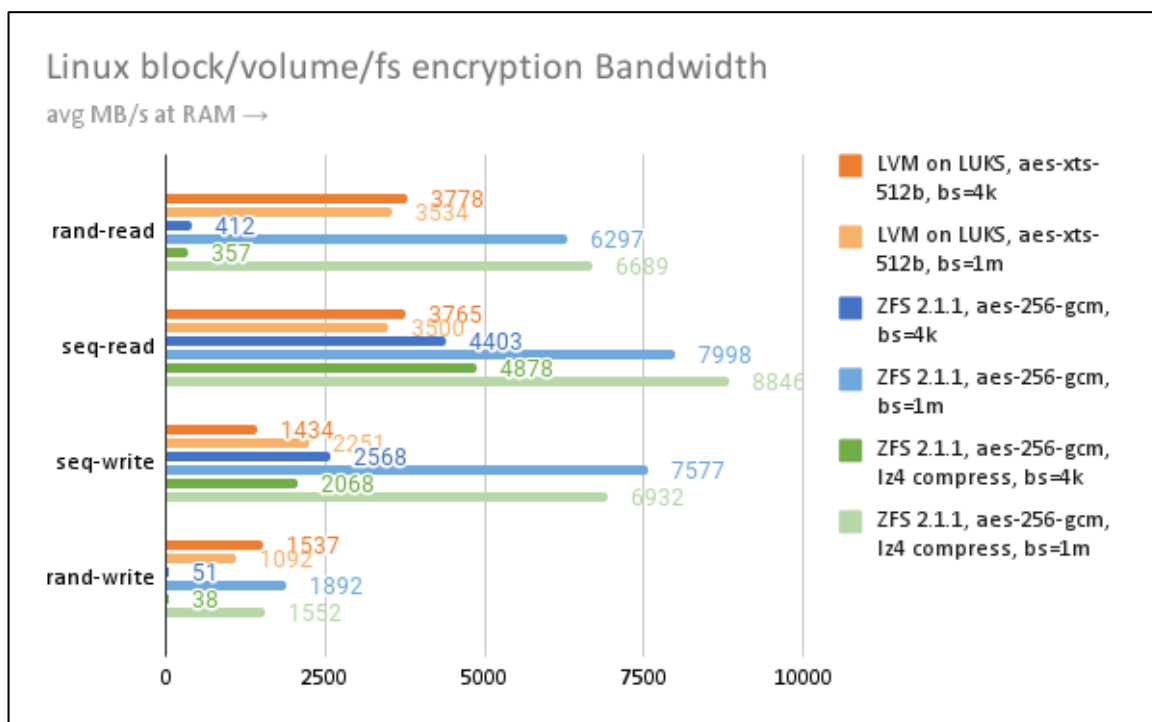
בנוסף, חסרון ברור של ZFS הוא העובדה שאינה תומכת בשימוש במספר מפתחות\ סיסמאות לניהול ה-volume המוצפן. עובדה זו כנראה מבטלת את היתרון פתרון רלוונטי להצפנת סביבת עבודה עבור מערכת עם מספר משתמשים שונים.

ביצועים

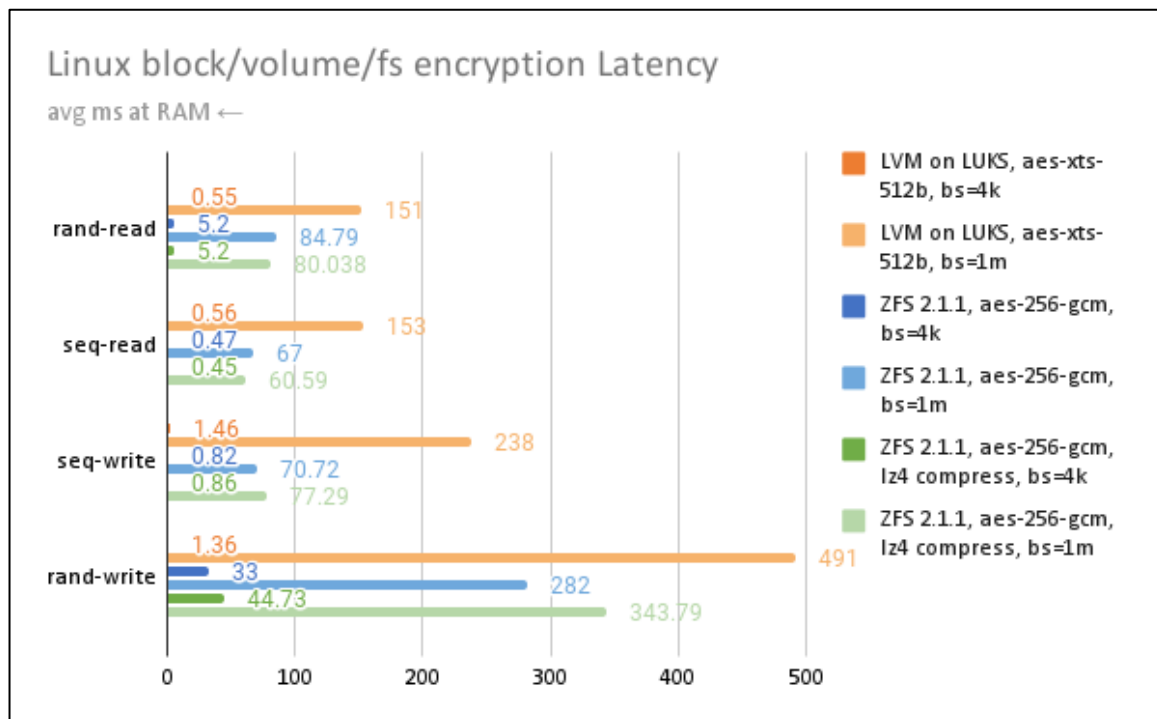
Performance features	dm-crypt +/- LUKS	VeraCrypt	ZFS	fscrypt
Multithreading support	Yes	Yes	Yes	Yes
Hardware-accelerated encryption support	Yes	Yes	Yes	Yes

טבלה חלקית בהחלט - לא ניתן לקבל תמונה מלאה על ביצועי הפתרונות השונים רק מ-2 הרובריקות הנ"ל. ביצועי מערכת הפעלה בשילוב יישומי הצפנה צריכים להימדד באספקט רחב של מדדים - החל ממהירות קריאה \ כתיבה לדיסק ועד האופן בו שימוש באפליקציות desktop כמו דפדפן מתעכבות עקב יישום ההצפנה.

במסגרת המחקר שוטטתי לא מעט בפורומים של הקהילה בנושאי אחסון והצפנה [ונתקלתי](#) בהשוואה הבאה שאחד המבקרים ביצע בין LUKS אל ZFS כאשר מדובר בהצפנת כונן:

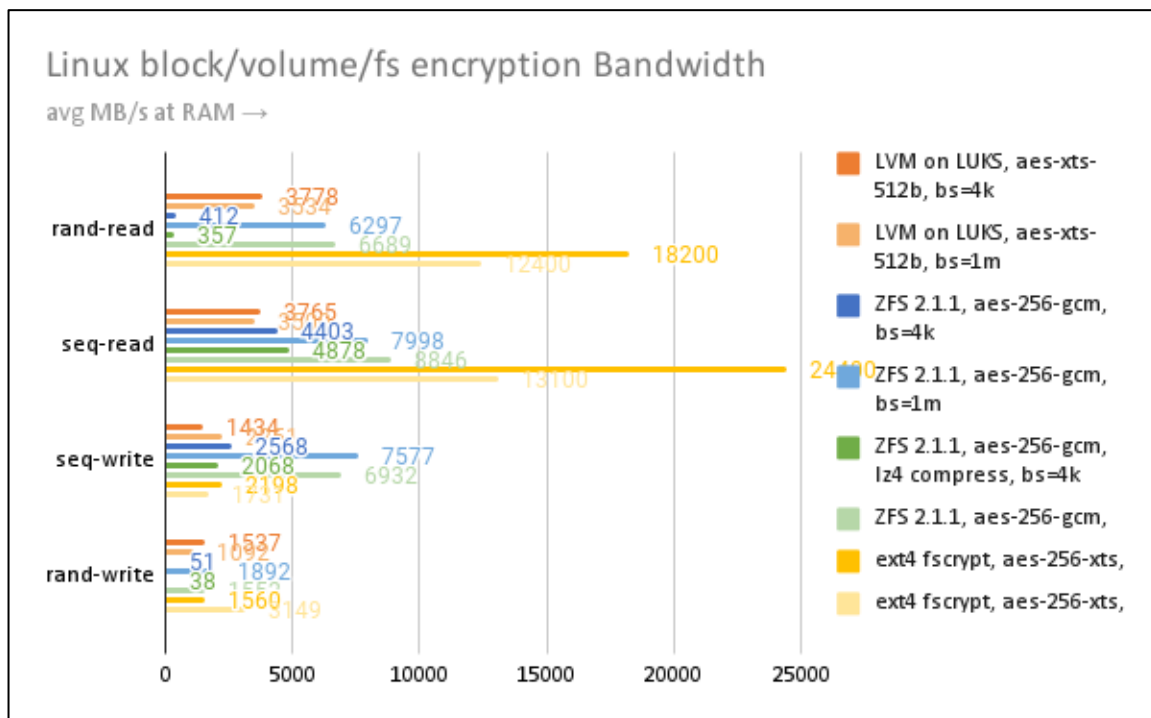


כאשר אם מסתכלים בנוסף על הדיליי שנוצר עקב latency ברור ש-LUKS נופל בביצועיו מ-ZFS:



כאמור התוצאות היו (ברמה מסוימת) צפויות מראש מכיוון ש-ZFS היא מערכת קבצים מתקדמת ובודדת שיישמה פתרון הצפנה במיוחד עבור עצמה אז מתבקש כי הוא יהיה היעיל ביותר לעומת LUKS אשר

מסוגלת להריץ תחתיה כמעט כל מערכת קבצים. עם זאת, כאשר מוסיפים את fscrypt על Ext4 התוצאות נראות שונה בהרבה:



העובדה ש-fscrypt מספקת הצפנת aes-256-xts עבור מידע ועושה שימוש ב-aes-256-cts עבור שמות הקבצים גורם לה להיות מהירה כמעט כמו פעולות מערכת הקבצים עצמה. עם זאת, fscrypt חושפת לא מעט מטא-נתונים על הקבצים עצמם ולא ניתן לעשות בה שימוש עבור הצפנת כונן אלא רק עבור מערכת הקבצים (וגם זה ללא כל תיקיית root : /). לכן, כנראה שתכל'ס אם תרחיש איום הייחוס שלכם מאפשר את הסיכונים הנ"ל (ואתם סבבה עם מערכת קבצים כמו ext4) כנראה שיהיה הכי משתלם לעשות שימוש ב-fscrypt.

לא התייחסנו אל VeraCrypt אבל חשוב לציין כי מבין כלל הפתרונות היא הכי איטית בצורה משמעותית.

מגבלות של הצפנת קבצים

Stacked filesystem encryption specific	ZFS	fscrypt
Supported file systems	ZFS	ext4, F2FS, UBIFS
Ability to encrypt filenames	Yes	Yes
Ability to not encrypt filenames	No	No

לא משהו שלא ידענו קודם. Fscrypt נועדה לרוץ על 3 מערכות קבצים כאשר המוכרת בין היא ext4 ו-ZFS נועדה להצפין את ה-ecosystem שהיא פועלת בתוכו.

תאימות ושכיחות

Compatibility & prevalence	dm-crypt +/- LUKS	VeraCrypt	ZFS	fscrypt
Supported Linux kernel versions	CBC-mode since 2.6.4, ESSIV 2.6.10, LRW 2.6.20, XTS 2.6.24	kernel 2.6 or compatible	2.6.32 or newer (as of 0.8.3)	4.1 or newer
Encrypted data can also be accessed from <u>Windows</u>	Yes Via WSL or LibreCrypt	Yes	Yes OpenZFS on Windows (repo)	No
Encrypted data can also be accessed from <u>Mac OS X</u>	No	Yes	Yes OpenZFS on OS X (repo)	No
Encrypted data can also be accessed from <u>FreeBSD</u>	No	Yes	Yes ZFS on FreeBSD (native; repo)	No
Used by	Debian/Ubuntu installer (system encryption) Fedora installer	?	?	Android, Chrome OS

טבלה חשובה וכנראה שבסופו של יום מבין כל הטבלאות היא עם ההשפעה הכי מכרעת לבחירה באחד מפתרונות ההצפנה.

אם יש לכם קרנל ישן\ לגסי - כנראה ש-fscrypt לא בתמונה עבורכם. בצורה דומה אם אתם צריכים פתרון נגיש מסביבת Windows אז כנראה שהפתרון הכי נוח עבורכם יהיה VeraCrypt למרות שעם קינפוג נכון ומשחק עם WSL גם LUKS יכול לעבוד (למרות שמאוד לא מומלץ מכיוון שרעיונית הוא מעולם לא נבנה לזה וכנראה צפויים לא מעט באגים). במידה ומכניסים סביבת cross platform אשר מרובה במחשבים\ שרתים מסוגי Linux/ Windows/ Mac/ FreeBSD אז כנראה שאין על מה להתאבק ולבחור מראש ב-VeraCrypt מכיוון שהיא מספקת פתרון native לכולם; לחילופין ניתן לשקול **מעבר לענן** (כאמור שם יש מספר פתרונות הצפנה נוספים).

מכיוון שענקיות טכנולוגיה עומדות מאחורי fscrypt וקיימת סטנדריזציה חזקה מאחורי LUKS כנראה שבמבחן המציאות הם הפתרונות המובילים שהולכים להישאר איתנו ולגדול אל נתח שוק גדול יותר בשנים הבאות.

סיכום

רוב מנגנוני האבטחה לתשתיות המחשוב **מתמקדים בהגנה על המערכת כשהיא פועלת** - לא מפתיע כלל בהתחשב בעובדה שכאשר המערכת רצה משטח התקיפה ופוטנציאל הנזק הם הגדולים ביותר אבל קיימת מחלקה שלמה של מתקפות כאשר המערכת כבויה. מטרת המאמר הייתה להראות אלו פתרונות הצפנה בסביבת Linux יכולים להוריד את משטח התקיפה הנ"ל.

הכרנו את רכיבי הקרנל אשר לוקחים חלק באותם תהליכי הצפנה, הכרנו את: dm-crypt, Device Mapper, dm-verity, Crypto-API ו-keyrings. לאחר מכן סקרנו את מרחב הפתרונות אשר מוצעים לביצוע ההצפנה ואת הושני ביניהם - הן ברמת מערכת הקבצים והן ברמת הכוון.

כמובן שיש לבטים נוספים שלא ניתן להציג בצורה אחידה לכולם - דרישות ממשתמשי קצה, עקומת למידה, תמיכה וכו'. מכיוון לרוב לאחר שנבחר פתרון הצפנה הוא יהווה **סטאטוס קוון**, שווה לתת את הדעת על מספר רב ככול הניתן של גורמים (עם עדיפות ל-PoC) לפני שמחליטים סופית. וזו למעשה הייתה המטרה הנוספת של המאמר - **התחככות ראשונית עם מספר פתרונות הצפנה מוכרים**.

בסופו של יום, כנראה שאם אתם מריצים נגזרת של Debian או Arch או REHL ואתם רוצים פתרון במינימום מאמץ שיתמוך רוחבית בכל התוכנות שיש לכם תוך הצפנת כוון מלאה - **LUKS הוא הפתרון** עבורכם. היותו סטנדרט מוביל את השימוש בו בצורה הטובה והרוחבית ביותר.

מסיבה זו, מאמר ההמשך יעסוק בדיוק בכך - הטמעת LUKS באופן ממשי ו-drill down לכיצד הוא עובד מאחורי הקלעים. יש למה לצפות 😊

על הכותב

[יהונתן אלקבץ](#), בן 27, חוקר אבטחת מידע בחברה לא קטנה ולא פרטית. חובב סוקולנטים, סודה וקפה.