digital whisper Technologic Papers

LUKS Encryption Deep Dive

מאת יהונתן אלקבס

הקדמה

נכון להיום חלק גדול מהשרתים מריצים שירותי תוכנה המסתמכים על מערכת הפעלה **מבוססת Linux**. בין אם זה שרת reverse proxy ו\או Load balancer על גבי NGINX, שרת web מבוסס Apache, שרת שיתוף קבצים מבוסס FreeNAS ועוד. ההפצות השונות של Linux קלות משקל, מבוססות קוד פתוח (חינם ואבטחה תמיד היו פיצ'ר חשוב), נוחות מאוד למוֹדִיפִיקַצְיָה ומספקות מְהֵימָנוּת גבוהה. לכן, אין זה מפתיע מדוע אינדיבידואלים וחברות ענק בוחרים לעשות שימוש בסביבות מבוססות Linux.

בצורה דומה, אין זה מפתיע שפוטנציאל התקיפה על מערכות Linux נוכח. כשזה מגיע להגנה אקטיבית לא חסרים אמצעים, הן ברמה האפליקטיבית והן ברמה התשתיתית בשביל להגן על תעבורת השרת, אך מה בנוגע למידע עצמו שנמצא על השרת בתצורת Data at Rest? כיצד ניתן לאבטח את תכולת השרת?

להבדיל מהצפנה ברמת מערכת הקבצים אשר במסגרתה מצפינים מספר בודד של תיקיות וקבצים, בהצפנת כונן **מצפינים את הכל**. אם בהצפנה מערכת הקבצים קיימת מודעות של אפליקציות userspace ושל מערכת הקבצים עצמה להצפנה הרי שבהצפנת כונן המצב הוא *business as usual* ואף תוכנה אינה ערה לעובדה שפעולות הכתיבה\ קריאה שלה עוברות דרך handle שמתרגם את כלל הפעולות מול הדיסק **בזמן אמת**. אי לכך, כאשר נרצה להגן על קבצים בתצורת Data at Rest מפני דלף מידע (למשל עקב גניבה פיזית של לפטופ\ כונן במחשב\ raid בשרת) נעדיף להשתמש בהצפנה מלאה של הכונן.

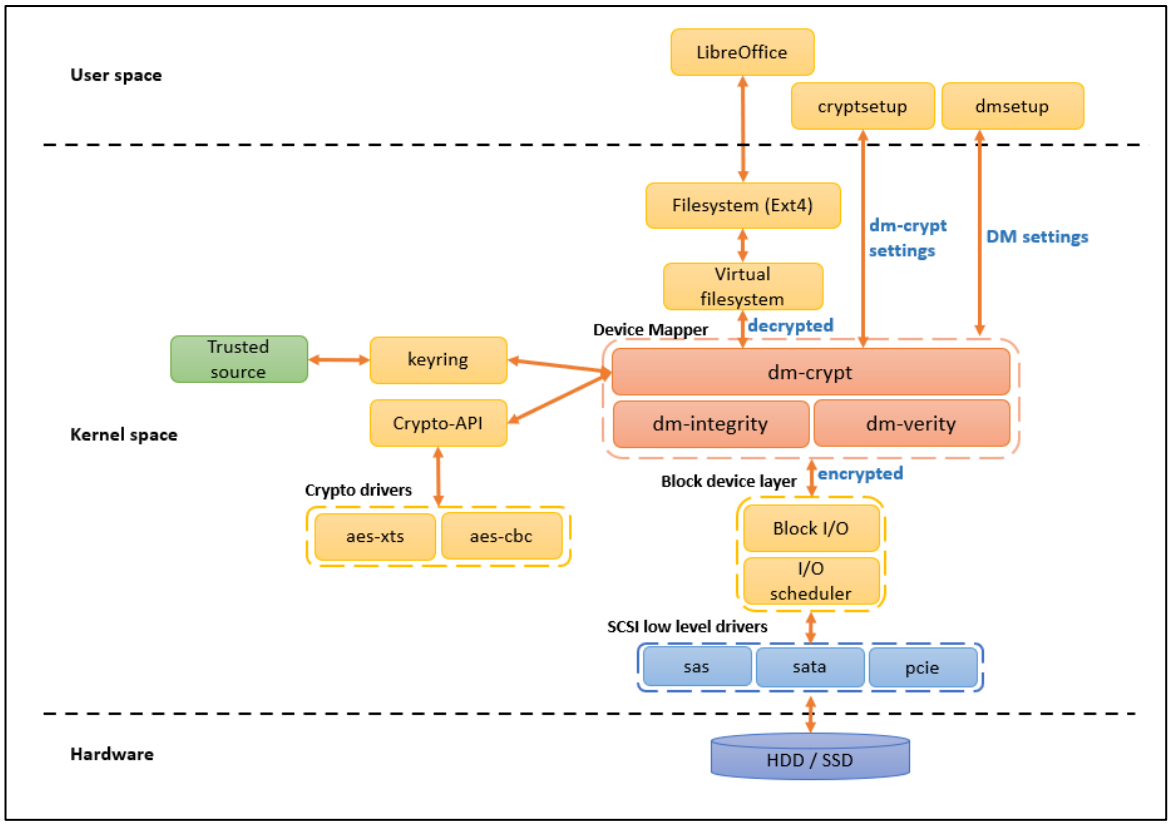
את המאמר הראשון בסדרה (רב הנסתר על הגלוי - עולם ההצפנה ב-Linux) פתחנו עם השאלה "ב-Windows יש לנו את BitLocker אבל מה קיים ב-Linux"? ובמסגרתו הצגנו את רכיבי הקרנל אשר לוקחים יד בפעולות ההצפנה. לאחר מכן פֵּרַשְׁנוּ את מכלול הפתרונות הנפוצים ביותר לשם הצפנה ברמת מערכת הקבצים וברמת הכונן בסביבת Linux במטרה לענות על אותה שאלה.

במאמר הקרוב נצלול לפתרון אחד אשר מאפשר הצפנת כונן בצורה שקופה למשתמש הקצה - **LUKS**. נראה כיצד הוא בנוי, מה הם הרכיבים הנעים בתוכו, נדבר על מערכת היחסים שלו מול LVM, כיצד הוא עובד מאחורי הקלעים ונבין מדוע הוא הסטנדרט.

Linux Storage Internals ב-60 שניות

ניתן להסתכל על הצפנת כונן כשכבה של אבסטרקציה - כל מערכת קבצים (משפחות EXT, XFS, FAT וכד') יכולה לשבת מעל ההצפנה מכיוון שאנחנו למעשה מצפינים **בלוקים** של מידע שיושבים בכונן; זאת בשונה מהצפנת קבצים אשר בה צריך לתת את הדעת לכל סוג של מערכת קבצים וסביבת עבודה (ספריות לוקאליות על המחשב). קרי, קובץ שהוצפן במחשב אחד עשוי לא להיפתח במחשב אחר.

אם לסכם את מרבית החומר התיאורטי של מודלי הקרנל שהוצג במאמר הקודם הרי זה באמצעות הדיאגרמה של ארכיטקטורת dm_crypt והתנהלותה עם שאר מודלי הקרנל:



באיור ניתן לראות כיצד מודלי ההצפנה ומודלי ה-storage פועלים ביחד על מנת לנהל את פעולות ההצפנה. נתחיל את ההסבר מלמעלה למטה.

מה קורה מאחורי הקלעים כאשר קוראים קובץ מוצפן בלינוקס:

כאשר משתמש ב-Linux ניגש לפתוח קובץ באמצעות תוכנת LibreOffice (או כל יישום userspace דומה), התוכנה שולחת open() system call עם שם הקובץ אל מערכת הקבצים. בכללי, מערכת הקבצים קוראת raw bytes מהדיסק ופרסרת תבניות בתים אלה כעץ של קבצים וספריות. בשביל כך, מערכת הקבצים (באמצעות הקרנל) משתמשת ב-block device layer.

Fun fact - לינוקס מאפשרת לנו גם ב-userspace לקרוא את אותם "raw bytes" באמצעות פנייה ישירה אל /dev/sda. זה מידע בינארי אז לא תראו שם יותר מידי דברים מעניינים אבל הפעולה אפשרית.

שכבת ה-block device קוראת לדרייבר של אחסון מידע (device driver, הוא גם חלק מקוד הקרנל) בשביל לתרגם מפקודות ביניים של "read sector X" אל פעולות אינדיבידואליות של input/output בשפה נמוכה יותר. ישנם מספר דרייברים להתקני אחסון, כולל IDE, (S)ATA, SCSI, Firewire ועוד בהתאם לסטנדרט התקשורת שאותם דרייברים יכולים להשתמש בהם.

הוראות ה-I/O משתמשות ביכולות המובנות של שבב המעבד ושל בקר לוח האם כדי לשלוח ולקבל אותות חשמליים על הכבל שעובר לכונן הפיזי. פה אנחנו כבר מדברים על **חומרה**, לא תוכנה. בקצה השני של אותו כבל, הקושחה של הדיסק (קוד בקרה embedded) מפרשת את האותות החשמליים כדי לסובב את הפלטות ולהזיז את הראשים של הדיסק (HDD), או לקרוא תאי הפלאש של ה-ROM (SSD).

בצורה כזו כל המידע עובר "הלוך חזור" מהדיסק אל הממשק הגרפי של תוכנת LibreOffice. *כמובן כמובן שהשמטתי מיליון פרטים כמו פעולות ה-caching ועוד הרבה חלקים בסיפור. אנשים (שהם בבירור לא אני) מסוגלים לכתוב מאמרים וספרים שלמים שמתארים איך כל התהליך הזה עובד בפרטי פרטים. אבל הפסקאות האחרונות מספקות רעיון כללי (ולא רע בכלל) לאיך הכל נראה.*

אז רגע איפה נכנס כל הבלוק האדום הענק הזה בלב האיור? איפה dm_crypt בכל הסיפור?

טוב שאתם שואלים, Device mapper מהווה שכבת אבסטרקציה בין ההתקנים החומרתיים אל שכבת אחסון וירטואלית שהקרנל 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 העובר בקרנל (דיברנו עליו מקודם) ולהצפין אותו לפני פעולת הכתיבה לדיסק. מכיוון שתשתית DM מאפשרת עֶרֵמַת 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. כמו כן, על מנת לבצע את הפעולות הקריפטוגרפים שמעורבות בהצפנה, dm-crypt עושה שימוש במודל ה-**crypto-API**.

בעת קריאת הקובץ תוכנת LibreOffice למעשה פונה ישירות אל מערכת הקבצים מבלי לדעת שהיא עוברת דרך שכבת האבסטקרקציה של device mapper אשר מיוצגת כמערכת קבצים וירטואלית.

באמצעות Cryptsetup ניתן לקנפג את dm-crypt ועל ידי dmsetup ניתן להגדיר את כלל device mapper.

בפרקים הבאים נבין את המוטיבציה לשימוש ב-LUKS, נצלול לעומקו ונראה כיצד הוא עושה שימוש בכל אחד מאותם מודלים קרנליים בשביל לבצע את הצפנת הכונן.

למה לי בכלל?

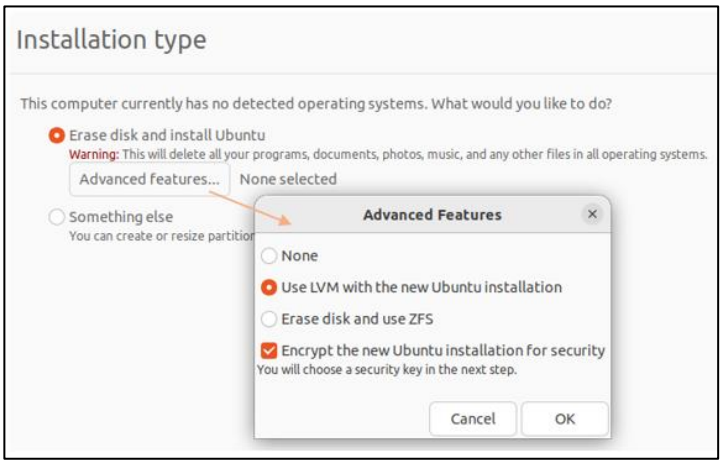
מוטיבציה בריאה צריכה להיות אחת מאבני הביניין של כל פעולה בחיינו. בקו מחשבתי-פילוסופי דומה, מהם היתרונות של LUKS? מה הוא עושה ולמה לכם בכלל לקרוא מאמר עליו?

ובכן, Linux Unified Key Setup (LUKS) עומד מאחורי הצפנת כונן קוד פתוח הנפוצה ביותר בסביבת הקרנל. מכיוון שהצפנת כונן לוקחת חלק ברובד נמוך יותר ממערכת הקבצים (ולמעשה יותר נמוך גם ממערכת ההפעלה), תהליך ההצפנה והפיענוח שקוף למשתמש הקצה. בתור משתמשים, כל מה שנדרש עבורכם הוא הכנסת סיסמה אחרי שלב ה-boot (קיימת גם אופציה להצפין את מחיצת ה-boot).

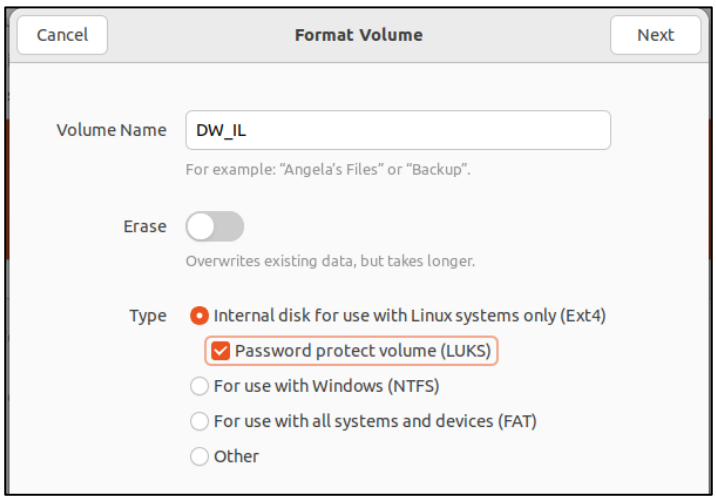
בעוד שרוב תוכנות הצפנת כונן מיישמות פורמטים שונים, ללא תאימות ולפעמים גם ללא תיעוד מעמיק, LUKS מיישמת פורמט **סטנדרטי** ובלתי תלוי בפלטפורמה לשימוש בכלים שונים. זה לא רק מקל על תאימות ותפעול בין תוכניות שונות, אלא גם **מבטיח שכולן מיישמות את אספקט ניהול הסיסמאות בצורה מאובטחת ומתועדת**. רוצים להצפין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

היתרון המשמעותי של LUKS שהוא "as low as it gets" בסביבת מערכת ההפעלה. הסיבה לכך פשוטה, הוא מתממשק עם הקרנל בצורה ישירה דרך device mapper (דיברנו עליו ועל תפקידו בקרנל במאמר הקודם). למעשה אם תסתכלו על רשימת האנשים שתרמו לקוד הפתוח של מודל device mapper בקרנל הלינוקסי תראו שתורם הקוד (המתכנת) העיקרי שם גם כתב את רוב הקוד של cryptsetup - תוכנת userspace אשר מנהלת את LUKS. כלומר, הוא ידע בדיוק מה הוא עושה וכיצד הכל יכול לזמר ביחד בצורה הטובה ביותר.

שנית כל, כבר עכשיו חשוב לומר ש-**LUKS2 הוא הסטנדרט**. מה היתרונות בלהיות סטנדרט ואיפה זה פוגש אותנו? ובכן, במידה ותעלו image נקי של Ubuntu ותבקשו להצפין את כלל הכונן של מערכת ההפעלה - החלון הבא יקפוץ לכם:



או במידה ונרצה להצפין מידע על מדיה נתיקה כמו כונן USB נקבל את האופציה הבאה (**והיחידה**):



כלומר, כמשתמשים, LUKS פוגש אותנו כפתרון הצפנה בצורה ישירה בעידוד מפתחי מ"ה. תזכרו את התמונות האלו מכיוון שעוד נחזור אליהן ונבין מה קורה שם וכיצד הכל מתחבר.

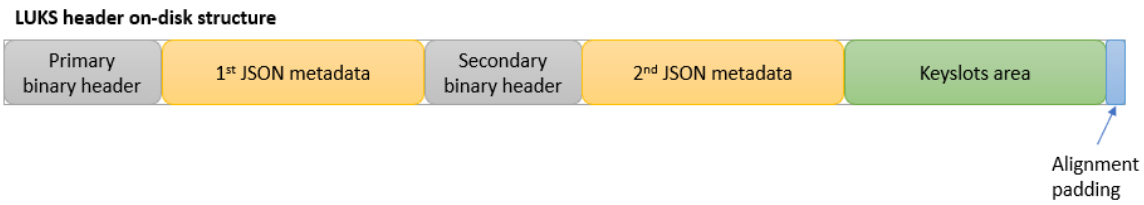
מתחילים - מבינים את LUKS

בכל תחילת partition של בלוק שהוצפן עם LUKS נמצא את ה-LUKS Header. מדובר בנושא ששווה להרחיב עליו קצת כי הוא מכיל את כל ה-metadate שנחוץ בשביל ההצפנה וגם מהסיבה הפשוטה *שרוב השאלות באינטרנט בנושא LUKS מגיעים מאנשים שהצליחו לדפוק את ה-header וכעת לא יכולים לפתוח את ההצפנה ולגשת לכונן שלהם*. ספוילר אלרט - במידה והצלחתם לשבש את ה-LUKS header רוב הסיכויים שלא תצליחו לשחזר את הנתונים שלכם. זו גם כנראה הסיבה מדוע רשום בדוקומנטציה במפורש על ההתחלה:

ATTENTION: If you are going to read just one thing, make it the section on Backup and Data Recovery

למעשה, ה-header של הגרסה החדשה והעדכנית (LUKS2) מכיל **שכפול זהה** של הנתונים הקריטיים לשחזור מהגרסה הקודמת (ולכן אורכו כמעט פי 2 מ-LUKS1); אולי בשביל שיהיה גיבוי למצב שהמשתמשים יהרסו חלק קריטי ממנו. *אבל כמובן שעדיין מצליחים להרוס אותו.*

נתחיל מהסוף, זה המבנה של ה-header:



ה-header מחולק ל-3 חלקים לוגים:

- header בינארי בשני מקטעים (זהים בתכולתם)
- metadata בשני מקטעים בפורמט JSON כפול (זהים בתכולתם)
- אזור אחסון מפתחות (keyslots binary data)

כאמור, חלק בינארי ואחריו מידע בפורמט JSON בחזרה כפולה ואחריהם מפתחות המשתמשים. החלקים הבינאריים בגודל קבוע של סקטור (4096 בתים למרות שרק 512 בתים בשימוש), החלק של המפתחות מוקצה דינאמית (כתלות במספר המפתחות שיש לאותו חלק מוצפן - ניתן 1 עד 8 מפתחות שונים).

השדה הבינארי נועד לסריקה מהירה על ידי ה-blkid (ממשק CLI שמטרתו לזהות את סוג הבלוק ולתת מידע עליו - סוג partitions, סוג ה-filesystem, מה ה-UUID שלו וכד') וייעודו לספק את החתימה של LUKS לפני הבלוק, מידע בסיסי, גודל ה-header ו-checksum של ה-metadata.

נוכל ללמוד לא מעט על מטרת השדה הבינארי אם נסתכל על ה-struct שלו בשפת C בקוד מקור:

```
#define LUKS2_MAGIC_1ST "LUKS\xba\xbe"
#define LUKS2_MAGIC_2ND "SKUL\xba\xbe"
#define LUKS2_MAGIC_L 6
#define LUKS2_UUID_L 40
#define LUKS2_LABEL_L 48
#define LUKS2_SALT_L 64
#define LUKS2_CHECKSUM_ALG_L 32
#define LUKS2_CHECKSUM_L 64

struct luks2_hdr_disk {
        char            magic[LUKS2_MAGIC_L];
        uint16_t        version;        /* Version 2 */
        uint64_t        hdr_size;       /* in bytes, including JSON area */
        uint64_t        seqid;          /* increased on every update */
        char            label[LUKS2_LABEL_L];
        char            checksum_alg[LUKS2_CHECKSUM_ALG_L];
        uint8_t         salt[LUKS2_SALT_L]; /* unique for every header/offset */
        char            uuid[LUKS2_UUID_L];
        char            subsystem[LUKS2_LABEL_L]; /* owner subsystem label */
        uint64_t        hdr_offset;     /* offset from device start in bytes */
        char            _padding[184];
        uint8_t         csum[LUKS2_CHECKSUM_L];
        char            _padding4096[7*512];
        /* JSON area starts here */
} __attribute__ ((packed));
```

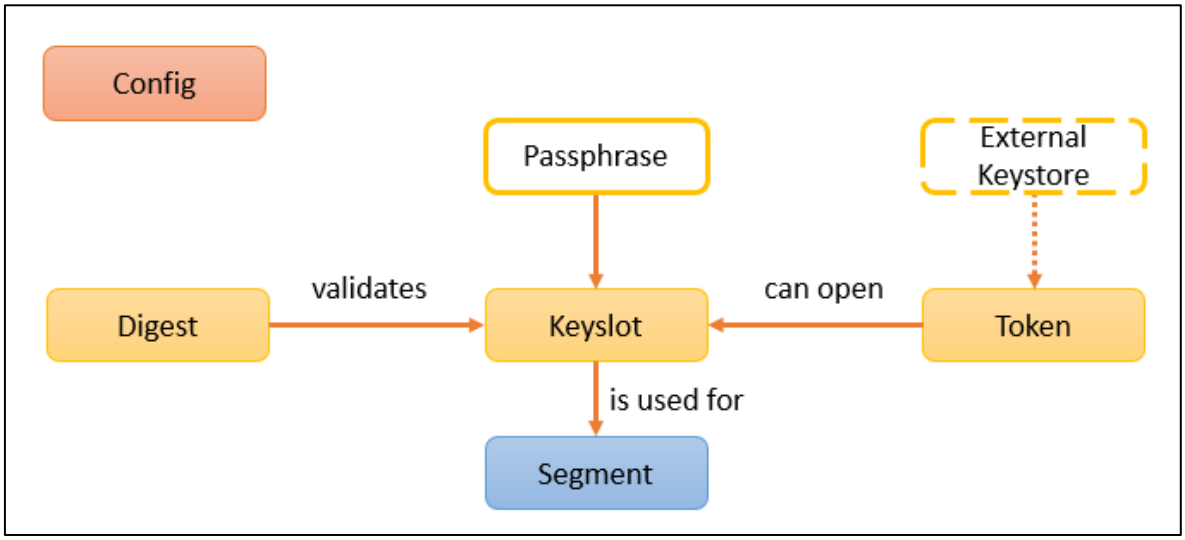
שדות version, uuid, label, subsystem נועדו לשימוש עם מסד הנתונים של udev (מנהל התקנים של הקרנל הלינוקסי) ופעולות מבוססות udev. למשל, מערכת כלשהי יכולה לנהל את כל התקני LUKS2 עם שדה subsystem מסוים בצורה אוטומטית על ידי שימוש בכלי ייעודי שהוגדר מראש. מלבד הרעיון של תפעוליות נוחה, ניתן לראות את ה-padding שתופס את שאר ה-sector. בסה"כ הקוד די מובן אז לא נתעכב על השדות השונים.

ה-primary binary header חייב להיות ב-sector 0 של ההתקן שהוצפן עם LUKS2 מכיוון שלפיו מתחיל תהליך הפיענוח. גודל חלק ה-JSON חייב להיות מיושר לפי offset של סקטור בגודל 4096 בתים, כלומר אם ניקח את כל ה-header נוכל לחשב את גודל שדה ה-JSON לפי 4096 - hdr_size.

חלק ה-JSON מתחיל ישר אחרי ה-binary header ומאוחסן על ידי string אשר מורכב מ-5 אובייקטים. אחריו מגיע חלק ה-keyslots area אשר מכיל מידע אודות כלל המפתחות שהתוכנה משתמשת בהם על מנת לפתוח את ההצפנה. כאמור, ניתן להצפין חלקי אחסון שונים החל מ-volume בודד וחלה בכונן שלם. באותו חלק ב-header מאוחסנות גם פונקציות ההצפנה ו-anti-forensic + פרמטרים נוספים. בכל ה-header ניתן להחזיק עד 8 מפתחות שונים *(תזכירו לי כמה יש ב-BitLocker?)*.

בכללי, ניתן לחלק את כלל חלקי ה-JSON ל-5 אובייקטים שפועלים ביניהם על מנת להגדיר את פעולת LUKS2:

- **Config** - מכיל תכונות קבועות של תצורת ה-header
- **Keyslots** - אובייקטים המתארים את אזורי האחסון של המפתחות מוצפנים
- **Digests** - נועד לאימות המפתחות לאחר פיענוח מה-keyslots
- **Segments** - האזורים בדיסק שמכילים את המידע המוצפן של המשתמש
- **Tokens** - מכיל metadata אופציונאלי נוסף לכיצד ניתן לחבור למערכות אחרות. למשל - כיצד לקבל את ה-passphrase עבור ה-keyslots. אם אתם זוכרים דיברנו במאמר הקודם על Keyrings, זו מערכת אחת שממנה יכולה להגיע הסיסמה לפתיחת ה-Keyslot שמשויך למפתח.



כל אחד מ-5 האובייקטים הנ"ל מורכב ממספר תתי-אובייקטים ותכונות נוספות אשר מפורטות בצורה פנטסטית בדוקומנטציה אבל לא נתעכב על הסברתם מכיוון שזה יפספס את מטרת המאמר.

עם זאת, נסתכל ברפרוף על דוגמה מלאה ל-JSON metadata של הטמעת LUKS2 על כונן באמצעות הצפנת AES-XTS אשר מכיל 2 Keyslots ו-token אחד המגיע מ-Keyrings:


```
1  {
2    "keyslots":{
3      "0":{
4        "type":"luks2",
5        "key_size":32,
6        "af":{
7          "type":"luks1",
8          "stripes":4000,
9          "hash":"sha256"
10       },
11       "area":{
12         "type":"raw",
13         "encryption":"aes-xts-plain64",
14         "key_size":32,
15         "offset":"32768",
16         "size":"131072"
17       },
18       "kdf":{
19         "type":"argon2i",
20         "time":4,
21         "memory":235980,
22         "cpus":2,
23         "salt":"z6vz4xK7cjan92rDA5JF806Jk2H
24       }
25     },
26     "1":{
27       "type":"luks2",
28       "key_size":32,
29       "af":{
30         "type":"luks1",
31         "stripes":4000,
32         "hash":"sha256"
33       },
34       "area":{
35         "type":"raw",
36         "encryption":"aes-xts-plain64",
37         "key_size":32,
38         "offset":"163840",
39         "size":"131072"
40       },
41       "kdf":{
42         "type":"pbkdf2",
43         "hash":"sha256",
44         "iterations":1774240,
45         "salt":"vWcwY3rx2fKpXW2Q6oSCNf8j5bv
46       }
47     }
48   },
49   "tokens":{
50     "0":{
51       "type":"luks2-keyring",
52       "keyslots":[
53         "1"
54       ],
55       "key_description":"MyKeyringKeyID"
56     }
57   },
58   "segments":{
59     "0":{
60       "type":"crypt",
61       "offset":"4194304",
62       "iv_tweak":"0",
63       "size":"dynamic",
64       "encryption":"aes-xts-plain64",
65       "sector_size":512
66     }
67   },
68   "digests":{
69     "0":{
70       "type":"pbkdf2",
71       "keyslots":[
72         "0",
73         "1"
74       ],
75       "segments":[
76         "0"
77       ],
78       "hash":"sha256",
79       "iterations":110890,
80       "salt":"G8gqtKbS96IbogHyJL0+t9kmjLkx+DM3HHJqQtgc2Dk=",
81       "digest":"C9JWko5m+oYmjg6R0t/98cGGzLr/4UaG3hImSJMivfc="
82     }
83   },
84   "config":{
85     "json_size":"12288",
86     "keyslots_size":"4161536",
87     "flags":[
88       "allow-discards"
89     ]
90   }
91 }
```

פרקטיקה

דיברנו לא מעט במאמר הקודם על Cryptsetup ו-dm-crypt וכיצד ניתן להשתמש בהם לטובת הצפנה שטחי זיכרון במערכת ההפעלה. בשביל להדגים את השימוש בהם ניקח דוגמה כללית ככול הניתן - הצפנת מחיצה כלשהי באמצעות LUKS. בצורה דומה היה ניתן להצפין את הכונן הראשי של מערכת ההפעלה ובכך למנוע מעלייה של המערכת ללא הזנת סיסמה לאחר שלב ה-boot.

ראשית, על מנת שנוכל לדבר עם dm-crypt השוכנת במרחב הקרנל נדרש להתקין את Cryptsetup. ניתן לבצע זאת באמצעות:

```
$ sudo apt-get instal cryptsetup
$ which cryptsetup
```

במידה וההתקנה הלכה חָלַק פקודת which תחזיר לנו את הנתיב של הקובץ הבינארי:

```
/usr/sbin/cryptsetup
```

מכיוון שעובדים עם storage מעתה נרוץ תחת משתמש root אשר מיוצג באמצעות '#'.

על מנת ליצור כונן וירטואלי של LUKS עם ההגדרות הדיפולטיות כל שצריך לעשות זה להריץ:

```
# cryptsetup luksFormat device
```

הפקודה (למרות השם המבלבל שלה) לא מבצעת format אלא יוצרת header של LUKS בכונן. בצורה דומה, ניתן היה לכתוב:

```
# cryptsetup --type luks2 --cipher aes-xts-plain64 --hash sha256 --iter-time
2000 --key-size 256 --pbkdf argon2id --use-urandom --verify-passphrase
luksFormat device
```

מכיוון שאלו הן ההגדרות הדיפולטיביות.

ניתן לבדוק את התוצאות באמצעות:

```
# cryptsetup luksDump device
```

הפקודה תחזיר לנו את המידע בנוגע ל-header וגם את המידע שנמצא ב-keyslots.

כעת, על מנת לפתוח את ההצפנה ולגשת לקבצים:

```
# cryptsetup open device dm_name
```

ברוח דומה לדוקומנטציה, כאמור ה-Header הוא החלק הפגיע ביותר. ניתן **לגבות** אותו על ידי הפקודה:

```
# cryptsetup luksHeaderBackup device --header-backup-file LuksBackUp.bin
```

הפקודה תיצור קובץ בינארי (כאמור המידע שמוחזק ב-header הינו בפורמט JSON ובינארי) כך שבמידה וגרמנו למצב של corrupt ב-Header ניתן לשלוף את הקובץ גיבוי שיושב בשרת בטוח ופשוט להקליד את הפקודה:

```
# cryptsetup luksHeaderRestore/device --header-backup-file LuksBackUp.bin
```

במידה והיינו רוצים להתאמת עם keyfile בנוסף (או במקום) סיסמה יש לכך 3 אפשרויות:

- **Passphrase keyfile** - סיסמה 'רגילה' שפשוט נמצאת בקובץ
- **Random text keyfile** - סיסמה שמורכבת מתווים רנדומליים בשביל למנוע מתקפות dictionary. לרוב נראה שימוש ב-openssl בשביל ליצור את ה-keyfile.
- **Binary keyfile** - כל קובץ שנרצה להצפין באמצעותו. זה יכול להיות תמונה, סרטון, או כל קובץ בינארי סטטי אחר. בהחלט אופציה מעניינת.

ניתן גם להגדיר עלייה ממפתח השוכן ב-TPM.

לאחר שבחרתם את סוג ה-keyfile המועדף עליכם כל שנוצר לעשות הוא להריץ את LUKS נגדו:

```
# cryptsetup luksFormat device --key-file /path/to/mykeyfile
```

כעת, בכל פעם שנרצה לפתוח את ההצפנה נצטרך להפנות אל הקובץ באמצעות:

```
# cryptsetup luksOpen device dm_name --key-file /path/to/mykeyfile
```

אם מסיבה כזו או אחרת לא הצלחתם לשמור על ה-keyfile שלכם בריא ושלם (*כל הכבוד*) עדיין יהיה ניתן לפתוח את ההצפנה באמצעות הסיסמה הראשונית שבאמצעותה יצרתם את הקובץ, קרי:

```
# cryptsetup open device dm_name
```

כאמור, LUKS מאפשר אחזקה של עד 8 מפתחות להצפנה בודדת. לפיצ'ר הזה קיים פוטנציאל שימוש אדיר - **חלוקת מפתח שונה למספר ישויות לצורך בקרת גישה על הכונן** היא המרכזית, פיצ'ר שחסר מאוד ב-BitLocket. ובכן, במידה ונרצה להוסיף מפתח כל שנדרש לעשות:

```
# cryptsetup luksAddKey /dev/device (/path/to/additionalkeyfile)
```

במידה ו-/path/to/additionalkeyfile קיים הוא ישתמש בו לפעולת ההצפנה, במידה והוא לא יקיים פשוט תקפוץ לנו שורה נוספת שמבקשת הזנה של סיסמה נוספת.

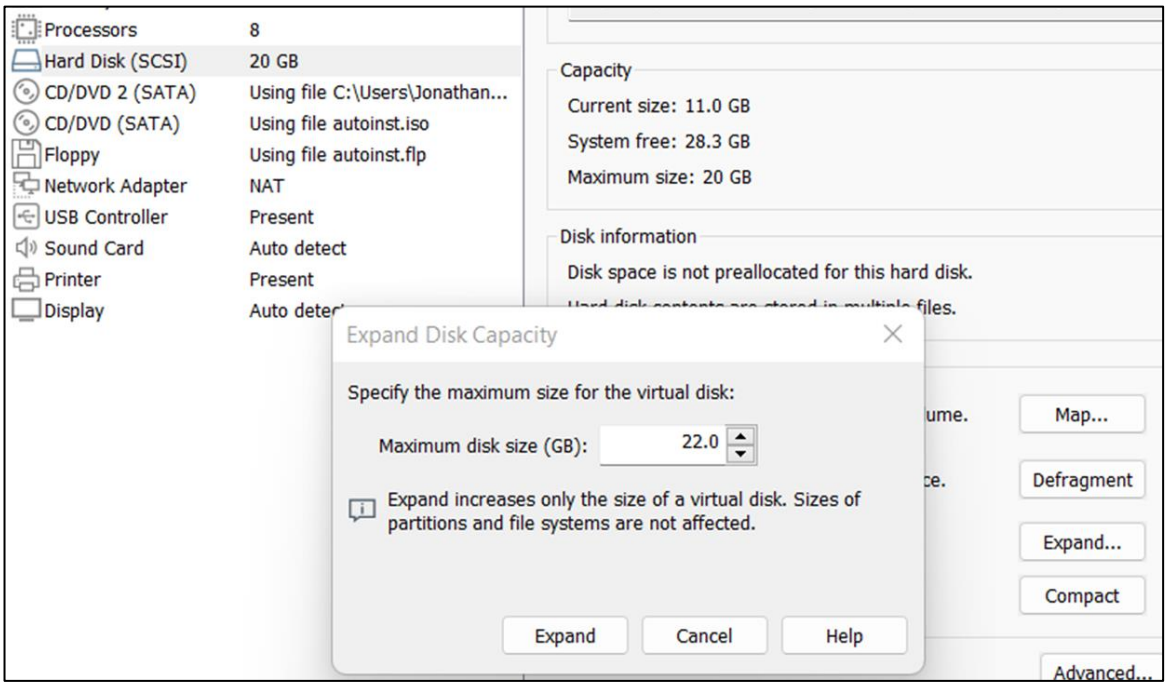
אין צורך לזכור את הכל אלא רק להבין את עקרון הפעולה. בפרק הבא נדגים את השימוש ב-LUKS בצורה מלאה על מנת להבהיר את השימוש ולצאת מהמשבצת של *"פקודות בלי הקשר לכלום"*.

החיים האמיתיים

ניקח דוגמה ממשית בשביל להראות את השימוש הפרקטי בכלי - יצירת מחיצה חדשה, הצפנתה, ביצוע mount וניהול שטח האחסון המוצפן. היה ניתן לבחור להדגים באמצעות הצפנת תיקיית הבית (/home) של המשתמש אבל לשם ההדגמה ניקח מקרה כללי ככול האפשר. כמו כן, כשמקנפגים את LUKS על איזור מסוים כל התוכן שהיה בו קודם נדרס.

הערה: כשמתחילים לעבוד עם הצפנה זה תמיד רעיון טוב להתחיל מסביבת VM ורק לאחר מכן לעבור למכונה אמיתית. לכן, בשביל ה-PoC במסגרת המאמר אמחיש עבודה ב-VM (ובפרט בסביבת VMware Workstation) וכיצד גם אתם יכולים לשכפל את הביצוע. מכיוון ש-Ubuntu נמצא בשימוש הרחב ביותר אדגים על מכונה כזו אבל באותה מידה ניתן היה לעשות שימוש ב-REHL או בכל הפצה אחרת (עם שינויים מינוריים).

נתחיל - נגדיל את האחסון במכונה הוירטואלית ב-2Gb בשביל להקצות מקום (אנחנו מעוניינים לוודא שלא נדרוס מידע קיים):



באמצעות fdisk (פקודה unix-ית המספקת יכולות חלוקה של כונן פיזי אל מחיצות) נפנה אל /dev/sda (קובץ הממפה את הכונן הראשון במערכת ההפעלה) ונבקש ליצור מחיצה חדשה בשם sda42 עם שטח האחסון שהוספנו. אפשר היה (ואולי אפילו עדיף) לבצע את הפעולה הנ"ל עם LVM במקום.

זהו רצף הפקודות:

```
root@ubuntu22:~# fdisk /dev/sda
Welcome to fdisk (util-linux 2.37.2).
Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

GPT PMBR size mismatch (41943039 != 46137343) will be corrected by write.
This disk is currently in use - repartitioning is probably a bad idea.
It's recommended to umount all file systems, and swapoff all swap
partitions on this disk.

Command (m for help): p
Disk /dev/sda: 22 GiB, 23622320128 bytes, 46137344 sectors
Disk model: VMware Virtual S
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 1FA7B94D-D31E-422F-914A-CFFE349F4782

Device       Start      End  Sectors  Size Type
/dev/sda1     2048     4095     2048    1M BIOS boot
/dev/sda2     4096  1054719  1050624  513M EFI System
/dev/sda3  1054720 41940991 40886272 19.5G Linux filesystem

Command (m for help): n
Partition number (4-128, default 4): 42
First sector (41940992-46137310, default 41940992):
Last sector, +/-sectors or +/-size{K,M,G,T,P} (41940992-46137310, default 46137310):

Created a new partition 42 of type 'Linux filesystem' and of size 2 GiB.

Command (m for help): p
Disk /dev/sda: 22 GiB, 23622320128 bytes, 46137344 sectors
Disk model: VMware Virtual S
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: gpt
Disk identifier: 1FA7B94D-D31E-422F-914A-CFFE349F4782

Device        Start      End  Sectors  Size Type
/dev/sda1      2048     4095     2048    1M BIOS boot
/dev/sda2      4096  1054719  1050624  513M EFI System
/dev/sda3   1054720 41940991 40886272 19.5G Linux filesystem
/dev/sda42 41940992 46137310  4196319    2G Linux filesystem

Command (m for help): w
The partition table has been altered.
Syncing disks.
```

ניתן לראות שבאמצעות הממשק של fdisk קודם בדקנו מהם המחיצות הזמינות לנו (באמצעות הזנה של התו p), לאחר מכן יצרנו מחיצה חדשה בשם 42 בגודל 2GB (באמצעות הזנה של התו n) ולאחר מכן בדקנו שהוא אכן נוצר. לסיום, שמרנו את טבלת המחיצות העדכנית (באמצעות התו w).

לאחר שיצרנו את המחיצה נכיל בה מערכת קבצים של ext4 ונכין תיקייה אליה נבצע mount. נבצע את ה-mount וניתן לתיקייה הרשאות כלליות בשביל להוסיף אליה קובץ משלנו:

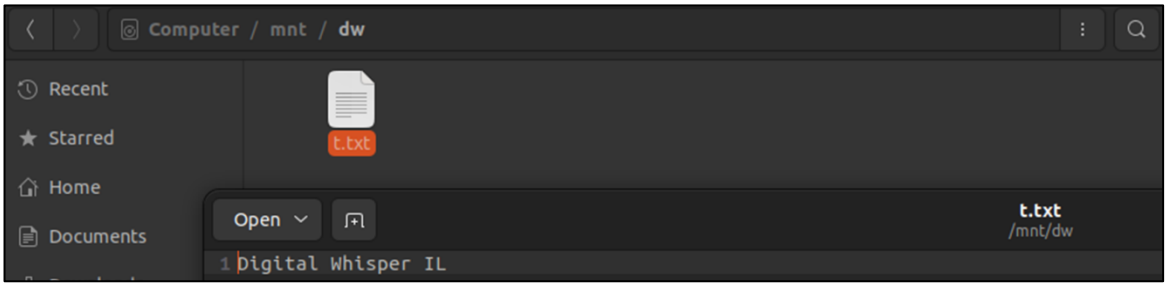
```
root@ubuntu22:~# mkdir /mnt/dw
root@ubuntu22:~# mkfs.ext4 /dev/sda42
mke2fs 1.46.5 (30-Dec-2021)
Creating filesystem with 524288 4k blocks and 131328 inodes
Filesystem UUID: af596f55-50f0-4bc9-b280-6c5e5035d55d
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done

root@ubuntu22:~# mount /dev/sda42 /mnt/dw
root@ubuntu22:~# chmod 777 /mnt/dw/
root@ubuntu22:~# echo "Digital Whisper IL" > /mnt/dw/t.txt
```

אפשר לראות שהכלנו במחיצה מערכת קבצים מסוג ext4, יצרנו תיקייה חדשה וביצענו אליה את ה-mount. לסיום הכנסנו קובץ txt פשוט בשביל להדגים שהכל עובד כשורה.

באמצעות ה-file explorer הלינוקסי נוכל לגשת למיקום התיקייה ולראות שהקובץ אכן נוצר.



עד כה, עבודה רגילה עם מערכת ההפעלה. כעת נבקש להצפין את המחיצה שיצרנו. נריץ את cryptsetup דיפולטיבית להצפנת AES-XTS באורך 256 (הצפנת XTS מפצלת את המפתח הראשי ל-2 ולכן צריך להכניס אורך מפתח של 512 סיביות).

```
root@ubuntu22:~# cryptsetup -s 512 luksFormat /dev/sda42
WARNING: Device /dev/sda42 already contains a 'ext4' superblock signature.

WARNING!
========
This will overwrite data on /dev/sda42 irrevocably.

Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /dev/sda42:
Verify passphrase:
root@ubuntu22:~# mount /dev/sda42 /mnt/dw
mount: /mnt/dw: unknown filesystem type 'crypto_LUKS'.
root@ubuntu22:~# cryptsetup open /dev/sda42 dm_name
Enter passphrase for /dev/sda42:
root@ubuntu22:~# mkfs -t ext4 /dev/mapper/dm_name
mke2fs 1.46.5 (30-Dec-2021)
Creating filesystem with 520443 4k blocks and 130304 inodes
Filesystem UUID: 3f8ae143-5f0a-4568-8a12-9c2c2338c5ce
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done
```

פה נכנס חלק חשוב שצריך להבין בהצפת המחיצה - מכיוון שמערכת הקבצים שהייתה קיימת קודם נדרסת, לא יהיה ניתן לשחזר את כל המידע שהיה נוכח קודם (זוכרים מה הם אמרו בהתחלה? גיבוי גיבוי גיבוי). באדום ניתן לראות מה יקרה כעת כשנבקש לבצע mount. כעת ה-header החדש הוא של LUKS (ולא של ext4) ולכן לא ניתן לבצע את ה-mount.

על מנת למפות את המחיצה המוצפנת אל מערכת הקבצים הלוקאלית נדרש לערב את **device mapper**. בשביל כך, נפתח את ההצפנה באמצעות פקודת *cryptsetup open* וניתן לה שם (בחרתי ב-*dm_name* בשביל להבהיר את הנקודה, אפשר היה גם לקרוא לו יוסי או שם בנאלי יותר כמו root). בצורה כזו הקרנל תדע שמעתה ואילך sda42 היא למעשה מחיצה מוצפנת ונדרש לגשת אליה דרך LUKS אך ורק באמצעות */dev/mapper/dm_name* בשביל לא לדרוס בטעות חלקים מהמידע המוצפן (כאמור חלק ה-header של המחיצה הוא קריטי לתפקוד ההצפנה).

בצורה דומה, מעתה ואילך כשנרצה לפנות אל המחיצה שיצרנו נשתמש ב-*/dev/mapper/dm_name* במקום ב-/dev/sda42. ניתן לראות זאת בשורה האחרונה בריבוע האדום אשר בו ייחסנו אל המחיצה מערכת קבצים מסוג ext4.

זהו, סיימנו. יצרנו אזור בשטח האחסון אשר מוצפן ומוגן באמצעות סיסמה. כעת, כל פעם שנרצה לגשת אליו ולקרוא\ לערוך מידע מתוכו נצטרך לפתוח את ההצפנה, לבצע mount באמצעות device mapper ולסגור את ההצפנה לאחר שנסיים. בקצרה זה יראה כך:

```
root@ubuntu22:~# cryptsetup open /dev/sda42 dm_name
Enter passphrase for /dev/sda42:
root@ubuntu22:~#
root@ubuntu22:~# mount /dev/mapper/dm_name /mnt/dw
root@ubuntu22:~# echo "Digital Whisper IR" > /mnt/dw/t.txt
root@ubuntu22:~# umount /mnt/dw
root@ubuntu22:~# cryptsetup close dm_name
```

אומנם זה נראה קצת ארוך ומסורבל אבל ב**קצת אוטומציה** ניתן לבצע את הכל באנטר אחד:

```
# Create strong LUKS key
openssl genrsa -out /root/luks.key 4096
chmod 400 /root/luks.key

# Fill random data to the device
shred -v --iterations=1 /dev/xvdb

# Format device
cryptsetup -q luksFormat /dev/xvdb --key-file /root/luks.key

# Open device
cryptsetup luksOpen /dev/xvdb data1 --key-file /root/luks.key

# Format device
mkfs.ext4 /dev/mapper/data1

# Mount
mount /dev/mapper/data1 /data1

# Verify
cryptsetup -v isLuks /dev/xvdb
df -h /data1

# --- not mandatory but helpful ---

# Persist at boot
Add to /etc/crypttab: data1 /dev/xvdb /root/luks.key luks
Add to /etc/fstab: /dev/mapper/data1       /data1  ext4    defaults 1 2
```

אלו פחות או יותר כלל השלבים שביצענו עם טוויסט לפתרון גנרי יותר. בשלב הראשון אנחנו יוצרים קובץ keyfile באמצעות openssl, דורסים לחלוטין את התוכן שהיה בכונן מקודם (בשביל זיכוי) באמצעות shred, לאחר מכן מכילים עליו את LUKS עם ה-keyfile שיצרנו קודם, מבצעים mount ולבסוף בודקים שהכל עבד.

ניתן גם להוסיף הוראה שיתבצע set up לאותו כונן בזמן תהליך ה-boot ולהוסיף את מיקומו לטבלת מערכת הקבצים (על מנת למנוע ביצוע של mount בכל פעם).

כמובן שיש מקום להוספת תנאים ובדיקות על מנת שהסקריפט לא יקריס ו\או יקרוס אבל אותו מקום הוא מחוץ לגבולות המאמר.

זה תזמון מעולה להזכיר כלי קוד-פתוח מעולה שמבצע אוטומציה בצמוד ל-LUKS. Clevis הוא כלי שמאפשר הצמדת LUKS volume למערכת ספציפית באמצעות (או לא) ה-TPM (ואמצעים נוספים) ולאחר מכן שימוש בקוד PIN לפתיחה. Clevis עובד בצורה צמודה אל Cryptsetup ומאפשר אוטומציה לתהליך די מסורבל (ומסוכן). במידה ולא יכולים ו\או לא רוצים לעשות שימוש ב-TPM ניתן לעשות שימוש ב-YubiKey - חומרה ייעודית בתצורת USB לאימות פיזי.

בשלב זה חלכם בוודאי שואלים *"בסופו של דבר כל מה שראינו פה זה שימוש בהצפנת AES-XTS, זאת הצפנה 'רגילה ופשוטה' שמלא תוכנות אחרות גם יכולות לעשות. למה אנחנו צריכים את LUKS?"*

ובכן, **LUKS2 הוא הסטנדרט**. מה היתרונות בלהיות סטנדרט? ובכן, במידה ותעלו image נקי של Ubuntu ותבקשו להצפין את כלל הכונן של מערכת ההפעלה - יעשה שימוש ב-LUKS אוטומטית. באופן ותצורה דומה גם תתרחש הצפנת מדיה נתיקה ב-Linux. כיצד זה קורה? עוד מעט נראה.

לפני כן, שווה לעשות עצירה קטנה ולהסביר בקצרה על כיצד LVM עובד ומצפין את המחיצות שהוא מנהל.

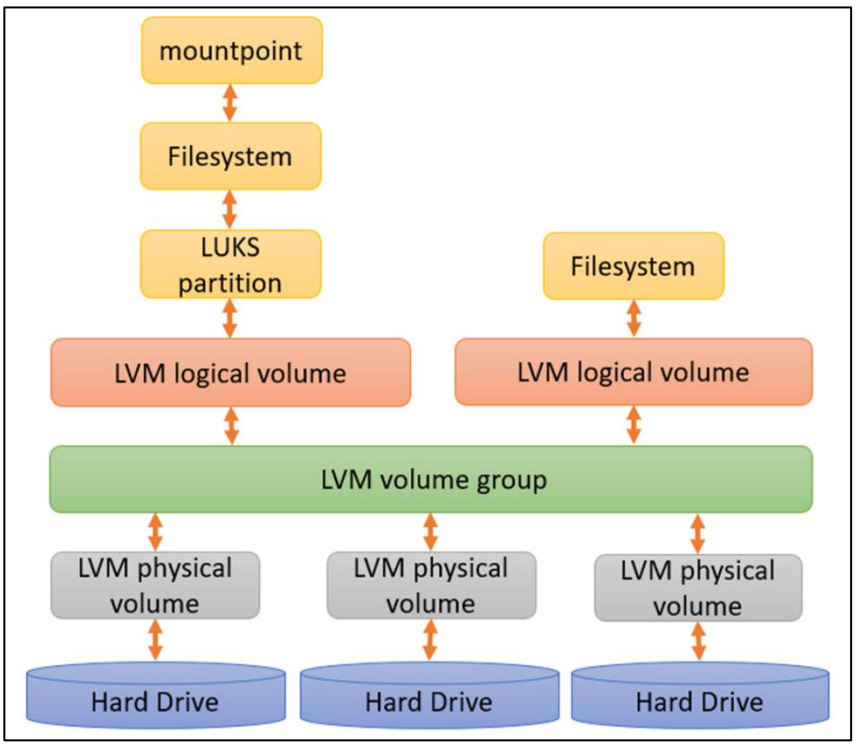
LVM encryption 101

Logical Volume Manager מסתמך על התשתית של device mapper על מנת לאפשר ניהול volume-ים לוגיים עבור הקרנל. למעשה הוא דומה במהותו אל fdisk שהראנו בו שימוש מקודם (למעשה LVM הוא גרסה חדשה יותר שנועדה להוסיף שכבות של אבסטרקציה בכל הקשור לניהול האחסון ובכך מאפשר שימוש בשטח אחסון רחב יותר משמעותית, עם זאת ישנם מצבים מעטים בהם אין ברירה אלא לעשות שימוש ב-fdisk ולכן שווה לזכור את קיומו).

LVM יוצר volume group (vg בקצרה) באמצעות אחד או יותר מהכוננים, אפשר לחשוב על vg בתור pool של כתובות זיכרון שכעת עומדות לרשות ה-LVM והוא יכול לנהל ולהקצות אותן כרצונו. לאחר מכן מתבצע שימוש ב-vg בשביל ליצור שטחי אחסון לוגיים ונוספות אליהם מערכת קבצים. לבסוף, ניתן להכיל הצפנה כזו או אחרת על מערכת הקבצים עצמה שנמצאת בתוך ה-logical volumes.

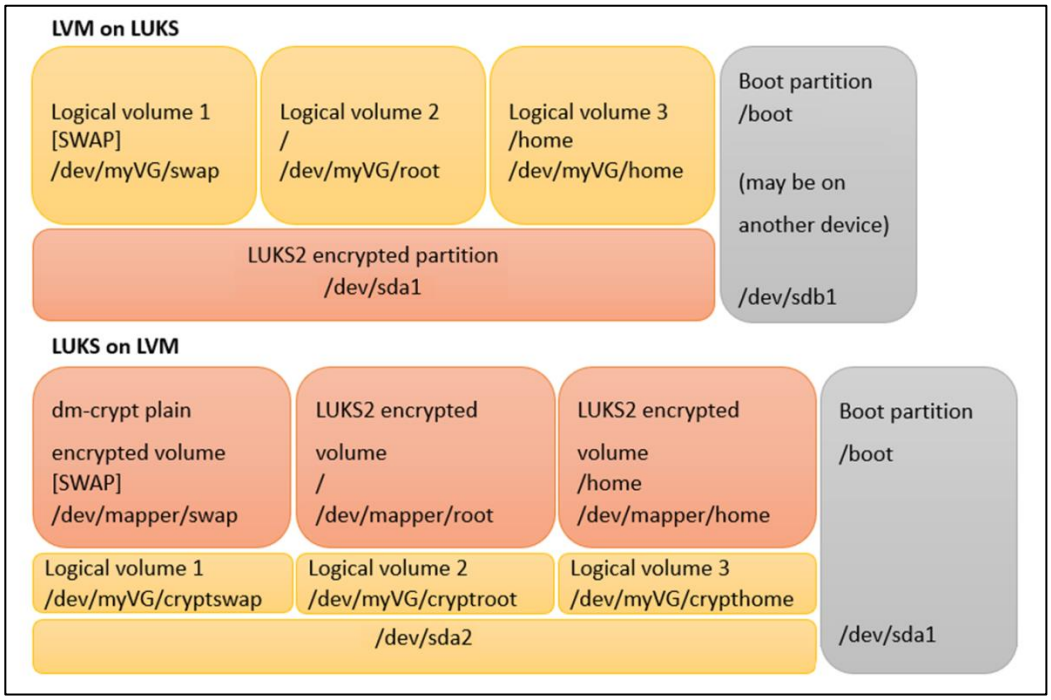
בצורה הזו ניתן לעשות שימוש ב-logical volumes בשביל לבצע דרכם mount לכונן עצמו. קובץ ה-swap הוא למקרה שה-RAM מתמלא וצריכים לשמור דפים מחוצה לו. עיקרון דומה למקבילו בממלכת ה-Windows למי שמכיר.

תמונה אחת שמסכמת את 3 הפסקאות האחרונות:



כאשר מדברים על מימוש LVM עם LUKS ישנם 2 קונפיגורציות אפשריות: LUKS on LVM ו- LVM on LUKS.

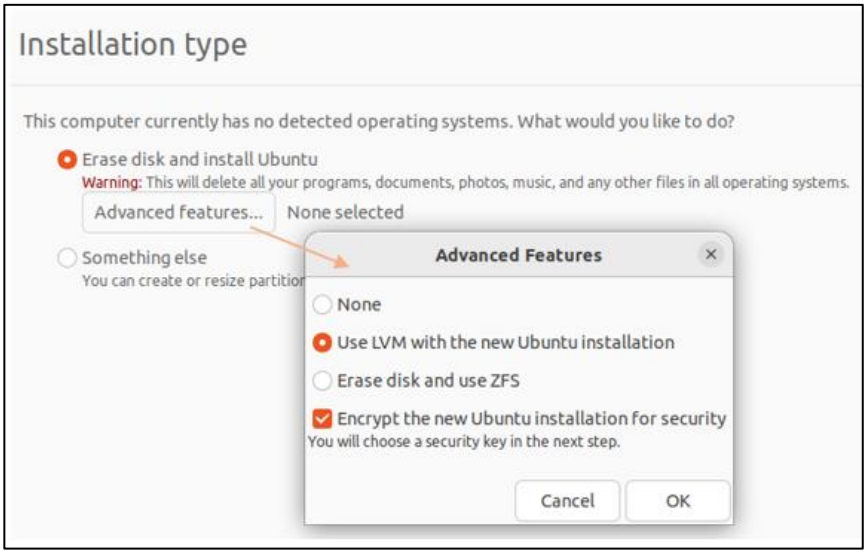
מדובר במקרה קלאסי שבו איור אחד יכול להסביר יותר טוב מדף שלם אז פשוט נסתפק באיור:



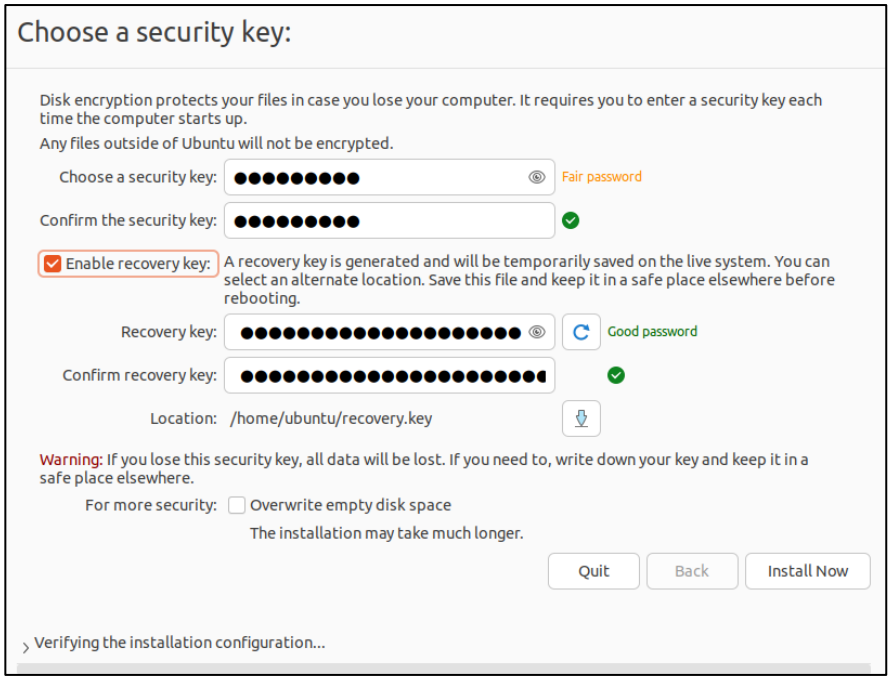
כאמור, LVM מיוצג על ידי הצבע הצהוב ו-LUKS על ידי הצבע הכתום. תזכרו את הארכיטקטורות השונות כי בקרוב נראה באיזו תצורה Ubuntu בחרו לממש את ההצפנה של מערכת ההפעלה שלהם.

מתחילים

במידה ותעלו image נקי של Ubuntu ותבקשו להצפין את כלל כונן מערכת ההפעלה, זה מה שתראו:



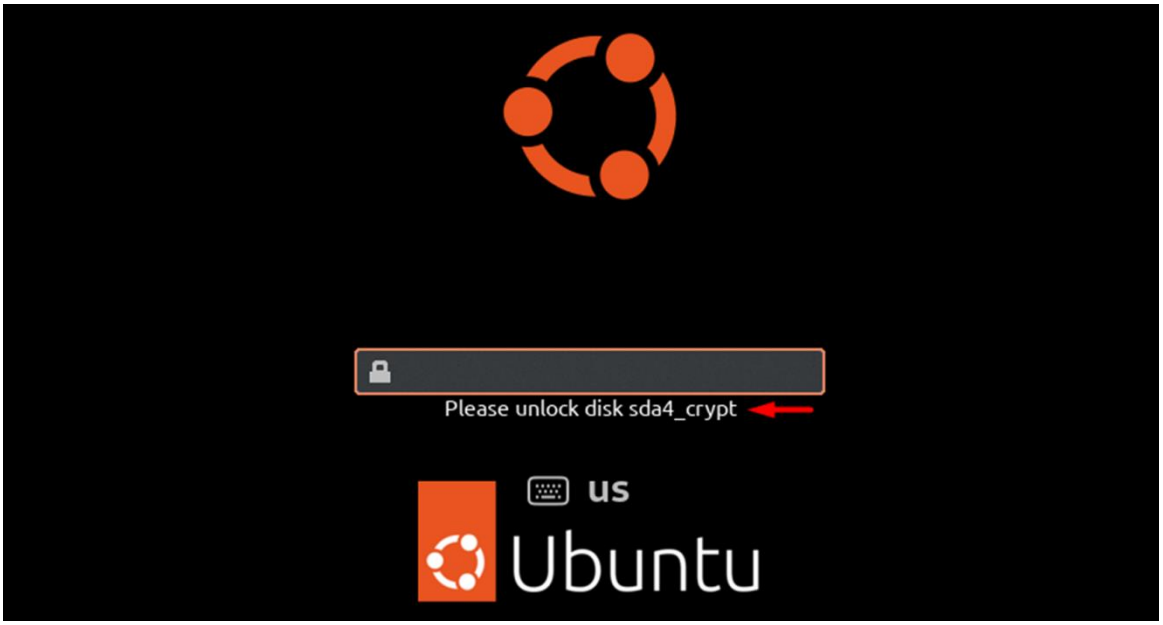
נבחר באופציה השנייה. כלומר, שימוש ב-LVM בשביל להצפין את המחיצה של מערכת ההפעלה. עם לחיצה על אישור יפתח לנו החלון הבא ליצירת מפתח ההצפנה:



בשורה הראשונה מבקשים מאיתנו להזין את הסיסמה של הצפנת הכונן ולאחר מכן לג'אנרט מפתח לגיבוי (string של 32 מספרים אקראיים). קיימת אפשרות לבצע כתיבה אקראית גם על שאר החלקים בכונן (אופציה אחרונה למטה) אבל נוותר על כך.

לאחר מכן עם עלייה של מ"ה יפתח לנו חלון שמבקש הזנה של שם משתמש, שם המחשב וסיסמת המשתמש. זהו, נגמר תהליך ההתקנה.

מעתה, כשנרצה להיכנס למערכת יופיע לנו החלון הבא:



חשוב להבין שהחלון הנ"ל יעלה לפני דף ה-login הרגיל - כלומר כעת למערכת יש 2 שדות להקלדת סיסמה - אחד לפתיחת הצפנת הכונן ולאחר מכן סיסמה נוספת להתחברות משתמש. בהחלט קד"ר.

אבל איזו הצפנה זו? בואו נחקור טיפה מה הוצפן וכיצד זה נראה מאחורי הקלעים. ראשית, עם הקלדת הסיסמה לפתיחת הכונן תקפוץ לנו לכמה רגעים קצרים השורה הבאה:

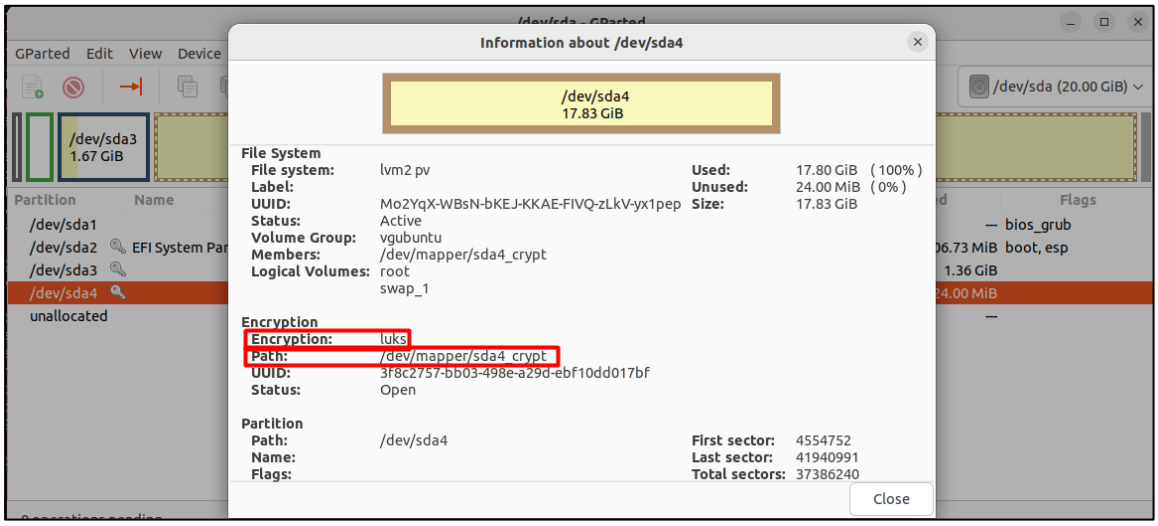
cryptsetup היא חברה וותיקה במאמר ואתם כבר אמורים להכיר אותה היטב - היא מנהלת את dm-crypt שבתורו אחראי על ביצוע ההצפה להתקנים במערכת.

עם עליית המערכת נפתח טרמינל ונריץ את lsblk בשביל לראות כיצד כל התקני האחסון ממופים:

```
jon@ubuntu22:~$ lsblk
NAME                    MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINTS
fd0                       2:0    1     4K  0 disk
loop0                     7:0    0     4K  1 loop  /snap/bare/5
loop1                     7:1    0    62M  1 loop  /snap/core20/1587
loop2                     7:2    0  61.9M  1 loop  /snap/core20/1405
loop3                     7:3    0  81.3M  1 loop  /snap/gtk-common-themes/1534
loop4                     7:4    0 248.8M  1 loop  /snap/gnome-3-38-2004/99
loop5                     7:5    0 400.8M  1 loop  /snap/gnome-3-38-2004/112
loop6                     7:6    0 155.6M  1 loop  /snap/firefox/1232
loop7                     7:7    0  91.7M  1 loop  /snap/gtk-common-themes/1535
loop8                     7:8    0  45.9M  1 loop  /snap/snap-store/575
loop9                     7:9    0  45.9M  1 loop  /snap/snap-store/582
loop10                    7:10   0  43.6M  1 loop  /snap/snapd/15177
loop11                    7:11   0   284K  1 loop  /snap/snapd-desktop-integration/10
loop12                    7:12   0    47M  1 loop  /snap/snapd/16292
loop13                    7:13   0   284K  1 loop  /snap/snapd-desktop-integration/14
sda                       8:0    0    22G  0 disk
├─sda1                    8:1    0     1M  0 part
├─sda2                    8:2    0   513M  0 part  /boot/efi
├─sda3                    8:3    0   1.7G  0 part  /boot
└─sda4                    8:4    0  17.8G  0 part
  └─sda4_crypt          253:0    0  17.8G  0 crypt
    ├─vgubuntu-root     253:1    0  16.8G  0 lvm   /
    └─vgubuntu-swap_1   253:2    0   976M  0 lvm   [SWAP]
sr0                      11:0    1 126.4M  0 rom   /media/jon/CDROM
sr1                      11:1    1   3.4G  0 rom   /media/jon/Ubuntu 22.04 LTS amd64
```

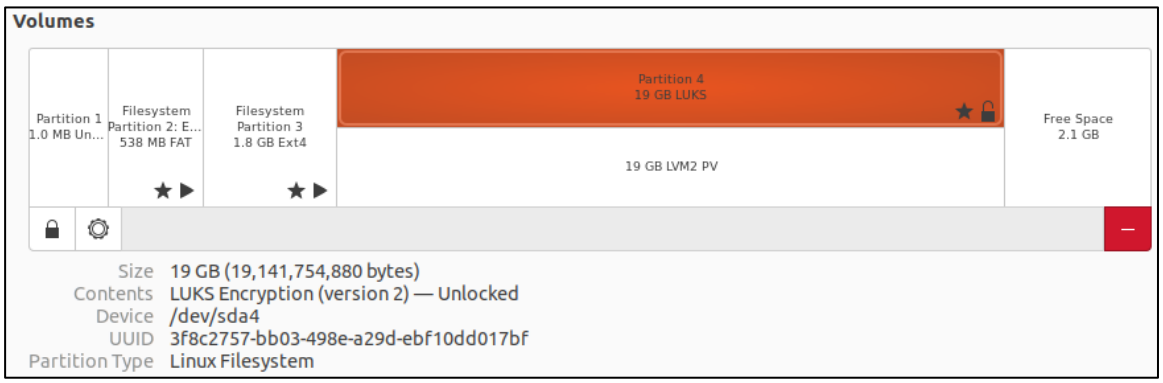
זוכרים שהצגנו את ההבדל בין LVM on LUKS לעומת LUKS on LVM? מה מבין השניים המפתחים של Ubuntu בחרו לממש אם כך? ובכן, התשובה היא פשוטה - **LUKS on LVM**. אישית הייתי מעדיף שיהיה הפוך והכל כולל הכל יהיה מוצפן (ניתן לראות למשל ש-/boot נשאר מחוץ להצפנה) אבל אני בטוח שהמפתחים של הקרנל ראו סיבות ותקורות נוספות שאני לא מודע אליהן.

עם זאת, אנחנו עדיין לא יודעים אם מדובר ב-LUKS או לא, על מנת לוודא סופית שאכן מדובר ב-LUKS נתקין את gparted ונסתכל על הממשק משתמש שלו:



כעת כבר אין עוררין כי המחיצה הנ"ל מוצפנת באמצעות LUKS. ניתן לראות פרטים נוספים על ההצפנה של המחיצה כמו ה-UUID אבל לא נתעמק במילים על זה.

ניקח צעד קדימה ועל הדרך ובאמצעות Disk-Utility נסביר את **החלקים השונים של מערכת ההפעלה:**



- בצד שמאל זה החלק של ה-BIOS boot. מקום אחסון בגודל של 1MB השוכן ב-ROM על גבי לוח האם (ב-userspace הוא מיוצג על ידי איזור /dev/sda1). חלק הקוד של ה-BIOS למעשה לא תלוי באיזו מערכת הפעלה יושבת מעליו, זו יכולה להיות Windows, Linux ואפילו MAC. הוא ללא mountpoint וללא מערכת קבצים. מדובר בחלק קוד הראשון שכל מטרתו לעלות, לבדוק שכל רכיבי החומרה מתקשרים (וזמינים) ואז להתחיל את תהליך עליית המערכת מהסקטור הראשון שזמין לו - ה-MBR.
- לאחר שנמצאה ה-MBR והיא מתחילה לרוץ, תהליך ה-bootloader נטען לזיכרון. הגרסה המודרנית ל-bootloader בלינוקס הוא multiboot בשם GRUB2. הוא רץ ב-RAM ואחראי לטעינה והעברת השליטה לתוכנת הקרנל אשר בתורה מאתחלת את שאר הקוד של מערכת ההפעלה. אבל עוד לא הגענו לאיפה ששוכנת הקרנל.

  התוכן של ה-bootloader מגיע בבלוק השני משמאל בתמונה בגודל של 538MB (רק 7MB בשימוש) שמאחסנים את מערכת ה-EFI במערכת קבצים מסוג FAT 32 bit תחת /dev/sda2 עם מיקום mount של /boot/efi. בקיצור, כאשר המחשב עולה, EFI טוען את GRUB2 מהמחיצה של /boot/efi.
- לאחר מכן מגיע המחיצה של /boot עצמו אשר נמצאת ב-/dev/sda3 ובגודל 1.8GB (מתוכם רק 300MB בשימוש). הקרנל עצמה לרוב תהיה דחוסה בקובץ תחת השם /boot/vmlinuz או במידה והקרנל לא דחוסה הקובץ שלה יהיה בשם vmlinux.
- ב-/dev/sda4 נמצא החלק המעניין שלשמו התכנסנו - זה ה-volume שמנוהל על ידי LVM2 ומוצפן על ידי LUK2 . ניתן לראות שהוא "מיוצג" על ידי 2 חלקים. זאת בגלל היכולת של device mapper לבצע את ההתאמה בין הכונן הפיזי אל הכונן הלוגי כפי שהוא מייצג אותו באמצעות /dev/mapper/sda4_crypt.

נחזור למכונה שלנו ונסתכל על כיצד LUKS מוגדר במחיצה:

```
jon@ubuntu22:~$ sudo cryptsetup luksDump /dev/sda4
LUKS header information
Version:        2
Epoch:          4
Metadata area:  16384 [bytes]
Keyslots area:  16744448 [bytes]
UUID:           3f8c2757-bb03-498e-a29d-ebf10dd017bf
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          (no flags)

Data segments:
  0: crypt
        offset: 16777216 [bytes]
        length: (whole device)
        cipher: aes-xts-plain64
        sector: 512 [bytes]
```

```
Keyslots:
  0: luks2
        Key:        512 bits
        Priority:   normal
        Cipher:     aes-xts-plain64
        Cipher key: 512 bits
        PBKDF:      argon2id
        Time cost:  4
        Memory:     440302
        Threads:    4
        Salt:       f8 b4 ff e7 3c f4 39 e0 35 39 26 b5 39 c5 23 4d
                    28 12 a3 f2 9a 64 cf 08 bb bf f0 9a 3a aa 85 87
        AF stripes: 4000
        AF hash:    sha256
        Area offset:32768 [bytes]
        Area length:258048 [bytes]
        Digest ID:  0
  1: luks2
        Key:        512 bits
        Priority:   normal
        Cipher:     aes-xts-plain64
        Cipher key: 512 bits
        PBKDF:      argon2id
        Time cost:  4
        Memory:     300550
        Threads:    4
        Salt:       c2 fa a8 c9 8c fe 8f 2d bf af e6 9b 85 9f 9e 92
                    ee 10 9f 5f e3 16 b3 c3 6f a0 93 e6 dc 59 94 a6
        AF stripes: 4000
        AF hash:    sha256
        Area offset:290816 [bytes]
        Area length:258048 [bytes]
        Digest ID:  0
```

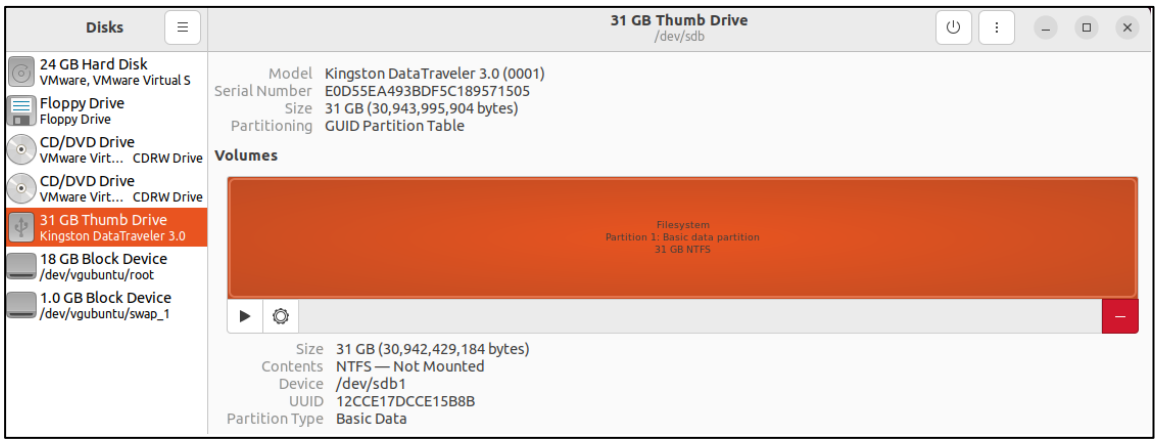
זו הרשימה של כלל המפתחות שיש לנו עבור LUKS. ניתן לפתוח באמצעותם את ההצפנה, כלומר יכול להיות מפתח שונה לכל משתמש. במידה ואנחנו רוצים להוסיף עוד מפתח ניתן לבצע זאת על ידי הפקודה luksAddkey אשר במהלכה תפתח לנו אפשרות להוספת סיסמה לאחר שנזין את אחת הסיסמאות הנוכחיות.

```
jon@ubuntu22:~$ sudo cryptsetup luksAddKey /dev/sda4
Enter any existing passphrase:
Enter new passphrase for key slot:
Verify passphrase:
```

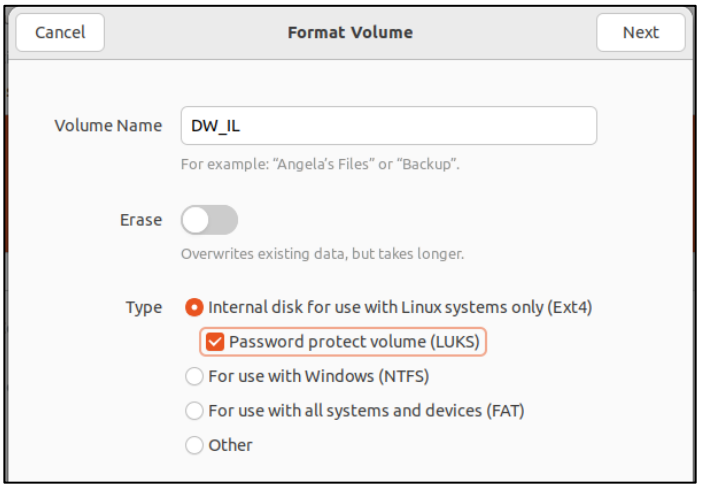
אם נבצע כעת luksDump נוכל לראות את תוכן המפתח החדש שהוספנו:

```
  2: luks2
        Key:        512 bits
        Priority:   normal
        Cipher:     aes-xts-plain64
        Cipher key: 512 bits
        PBKDF:      argon2id
        Time cost:  4
        Memory:     1048576
        Threads:    4
        Salt:       03 ee 8a c8 27 b1 31 47 02 67 b8 d6 47 c9 b6 36
                    31 e5 6a da da 91 24 16 1d 5b f7 35 d1 86 2a 10
        AF stripes: 4000
        AF hash:    sha256
        Area offset:548864 [bytes]
        Area length:258048 [bytes]
        Digest ID:  0
```

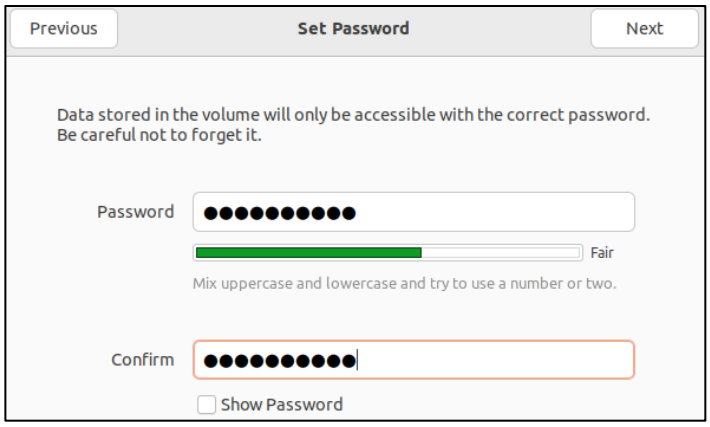
אין ספק שהיתרון המשמעותי בהיותך סטנדרט הוא העובדה שכולם עובדים לפי מה שהגדרת. ניקח דוגמה נוספת ויותר שכיחה - **הצפנת Thumb drive** (דיסק-און-קי בעברית שבורה). בעת חיבור כונן ה-USB למחשב נוכל להסתכל על התוכן השוכן בו בצורה דומה למה שעשינו מקודם באמצעות Disk-Utility:



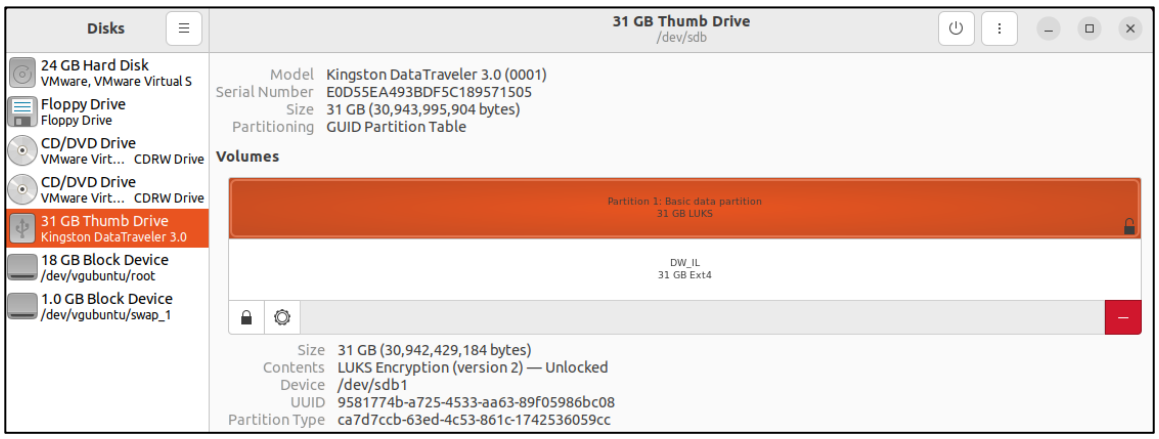
אפשר לראות שכרגע יש בו מערכת קבצים מסוג NTFS והוא מורכב ממחיצה אחת בלבד. נחליף את מערכת הקבצים אל Ext4 על ידי ביצוע format לכונן. בעת שנלחץ על הבקשה תפתח לנו רובריקה קטנה שתציעה לנו להצפין את תוכן הכונן מעתה ואילך באמצעות לא אחר מאשר LUKS:



בצורה דומה למה שעשינו מקודם, נקליד את סיסמת ההצפנה. שימו לב שכעת בשונה מהצפת מערכת ההפעלה אין אופציה לג'נרט מפתח לגיבוי.



כמה אנטרים טריוויאליים אחר כך ונקבל:



כעת, למרות שנראה כאילו מדובר ב-2 מחיצות נפרדות למעשה מדובר במחיצה אחת בלבד. על מנת לפשט את השימוש בכונן ליישומים אחרים במערכת ההפעלה הקרנל מייצרת mapped device המהווה שכבת אבסטרקציה אל הכונן המוצפן וכל האפליקציות יכולות לגשת לקרוא\לכתוב אליו כאילו היה מדובר בכונן רגיל מבלי להבין דבר בהצפנה - **הקרנל לוקחת אחריות על הכל**.

במקרה שלנו, הכונן המוצפן נמצא ב-/dev/sdb1 והכונן "הפתוח" נמצא ב-/dev/mapper/luks-UUID כאשר את UUID מוקצה לכל כונן בנפרד (ניתן לראות בתמונה את הנוכחי).

מכיוון ש-LUKS מיושם ברמת הקרנל כבר מעל 15 שנה כנראה שלא תופתעו לשמוע שעניין התאימות עובד חָלַק- במידה ונחבר את ה-Thumb drive אל הפצה אחרת של Linux ונזין את הסיסמה הכל יעבוד כרגיל.

הערה: כמובן שיכולנו לעשות להצפין את כונן מערכת ההפעלה ואת ה-Thumb drive דרך ה-CMD בצורה זהה למה שעשינו בפרק הקודם. מטרת הפרק הנוכחי הייתה להמחיש את היותו של LUKS סטנדרט - הוא האופציה היחידה שקופצת למשתמשים בממשקי ה-GUI של מערכת ההפעלה.

אנקדוטה אקראית: מסגרת המחקר שעשיתי בנוגע לכלל הכלים שלוקחים חלק בהצפנה, קראתי לא מעט דוקומנטציות, כולל אלו של LUKS1 ו-LUKS2. משהו שנראה לי תְּמוּהַ הוא העובדה שלפי הדוקומטציה של LUKS1 הוא מאפשר decrypt מלא של ה-volume המוצפן (כלומר הסרת ההצפנה ונתינת האפשרות לגשת אליו כרגיל) בעוד שהגרסה החדשה LUKS2 מערימה מלא קשיים (וסיכונים) בשביל לבצע זאת. כתוצאה מכך, עלתה לי המחשבה *אולי אפשר פשוט לשנמך את הצפנת הכונן מ-LUKS2 אל LUKS1 ואז לבטל את ההצפנה?*

ובכן, מסתבר ש**זה אפשרי**. כל שצריך לעשות זה לשנות את סוג המפתח כך שתהיה תאימות לאחור, להמיר את תצורת ההצפנה אל LUKS1 ואז לבצע decrypt בפורמט הישן:

```
root@ubuntu22:~# cryptsetup luksChangeKey  --pbkdf pbkdf2 /dev/sda42
Enter passphrase to be changed:
Enter new passphrase:
Verify passphrase:
root@ubuntu22:~# cryptsetup convert --type luks1 /dev/sda42

WARNING!
========
This operation will convert /dev/sda42 to LUKS1 format.


Are you sure? (Type 'yes' in capital letters): YES
root@ubuntu22:~# cryptsetup-reencrypt --decrypt /dev/sda42
Enter any existing passphrase:
Finished, time 00:26.888, 2032 MiB written, speed  75.6 MiB/s
root@ubuntu22:~#
root@ubuntu22:~# mount /dev/sda42 /mnt/dw
```

אני די בטוח שמדובר ב-"פיצ'ר" לא מתוכנן. אולי אני טועה.

סיכום

פוטנציאל הנזק של מתקפות המנצלות את הסטטוס הפאסיבי של המערכת גדול ומאפשר לתוקף לקרוא, לערוך ולמחוק מידע מהמערכת ולעיתים אף להכניס נוזקות בשלב ה-boot מה שימנע ממוצרי EDR ואנטי-וירוס לטרגר את מנגנוני ההגנה האקטיבים. כאן נכנסים לתמונה פתרונות Data at Rest Protection ובפרט LUKS כאשר מדברים על מערכות מבוססות Linux.

בעוד שרוב תוכנות הצפנת כונן ב-Linux מיישמות פורמטים שונים, ללא תאימות ולפעמים גם ללא תיעוד מעמיק, LUKS מיישמת פורמט סטנדרטי ובלתי תלוי בפלטפורמה לשימוש בכלים שונים (גם בשרתי NAS משתמשים ב-LUKS לא מעט). זה לא רק מקל על תאימות ותפעול בין תוכנות שונות, אלא גם מבטיח שכולן מיישמות את אספקט ניהול הסיסמאות בצורה מאובטחת ומתועדת.

פה מגיעה לסיום הסדרה הקצרה שלנו בנושא אבטחת DAR בסביבת Linux. המאמר הראשון בסדרה עסק בסקירה של רכיבי הקרנל שלוקחים חלק בתהליכי ההצפנה והמאמר השני היווה drill down לכיצד פתרון הצפנה בודד (LUKS) עובד מאחורי הקלעים. מקווה שלמדתם ולקחתם מהם משהו חדש ☺

על הכותב

יהונתן אלקבס, בן 27, חוקר אבטחת מידע בחברה לא קטנה ולא פרטית. חובב סוקולנטים, סודה וקפה.