Bitlocker šifrování disku v Linuxovém prostředí

Bc. Vojtěch Trefný

Diplomová práce
2019

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

*** Nascanované zadání, strana 1 ***

*** Nascanované zadání, strana 2 ***

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomové práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové práce bude uložen v příruční knihovně Fakulty aplikované informatiky. Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně .....

podpis autora

ABSTRAKT

Text abstraktu česky

Klíčová slova: Přehled klíčových slov

ABSTRACT

Text of the abstract

Keywords: Some keywords

Zde je místo pro případné poděkování, motto, úryvky knih, básní atp.

OBSAH

ÚVOD

První odstavec pod nadpisem se neodsazuje, ostatní ano (pouze první řádek, odsazení vertikální mezy odstavci je typycké pro anglickou sazbu; czech babel toto respektuje, netřeba do textu přidávat jakékoliv explicitní formátování, viz ukázka sazby tohoto textu s následujícím odstavcem).

Formátování druhého odstavce. Text text text text text text text text text text text text.

I. TEORETICKÁ ČÁST

1 Šifrování disku

2 BitLocker

text

2.1 Použité kryptografické funkce

BitLocker používá několik různých kryptografických funkcí a to jak pro šifrování samotných na disku uložených dat, tak pro ochranu klíčů pro jejich (de)šifrování, které jsou uloženy v metadatech. V této kapitole si představíme trojici algoritmů používaných v různých verzích BitLockeru pro šifrování dat na disku: AES-CBC a jeho rozšíření Elephant difuzér a AES-XTS a také algoritmus AES-CCM, kterým jsou zašifrovány BitLockerem používané klíče a stručně si také představíme pro BitLocker specifickou funkci pro odvození klíče z uživatelem zadávaného hesla.

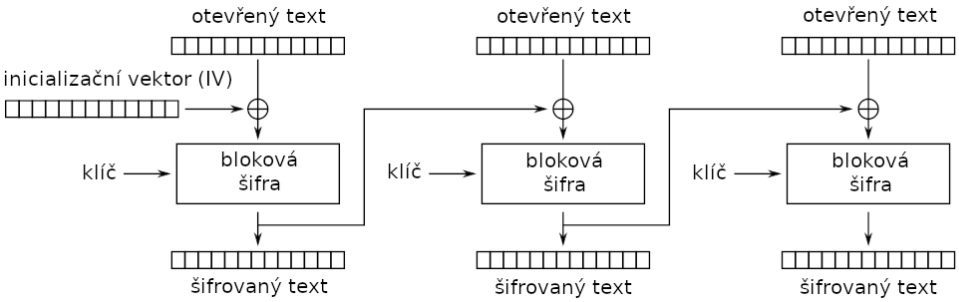
2.1.1 AES-CBC

AES-CBC je standardní široce používaná bloková šifra, která je ve starších verzích BitLockeru používaná pro šifrování dat uložených na šifrovaném zařízení.

Původní návrh CBC módu byl patentován v roce 1978[1] a řešil problém staršího ECB módu, při jehož použití vzniká zašifrováním dvou stejných bloků otevřeného textu stejný šifrovaný text. Tento problém CBC řeší tak, že každý blok otevřeného textu je napřed XORován s předchozím šifrovaným blokem. Standardně tedy šifrování v CBC módu probíhá následovně

$$C_i = E(K, P_i \oplus C_{i-1}) \quad \text{pro } i = 1, \ldots, k$$

kde C_i je výsledný šifrovaný blok, K klíč, P_i blok otevřeného textu a C_{i-1} předchozí šifrovaný blok. Protože pro první blok textu neexistuje žádný předchozí blok C_0, se kterým by mohl být XORován, zavádí se nová hodnota, takzvaný *inicializační vektor*, který se použije pro první blok místo předchozího šifrovaného bloku.[2]



Obr. 2.1 Schéma šifrování pomocí blokové šifry v CBC módu[3]

Pro BitLocker je v případě AES-CBC použito jako inicializační vektor číslo sektoru, navíc ještě zašifrované stejným klíčem, který je využit i pro šifrování textu.

XORování s předchozím blokem zavádí to AES-CBC potenciální slabinu, kdy při znalosti umístění bloku otevřeného textu lze pozměnit předchozí blok tak, aby v následujícím bloku došlo k přehození hodnoty určeného bitu. Tím se sice znehodnotí informace uložené ve změněném bloku (který bude dešifrován na náhodná data), ale můžeme tak změnit důležitou informaci obsaženou v bloku následujícím podle potřeba útočníka.[4]

U BitLockeru může tento útok představovat problém, pokud útočník zná přesné rozložení dat na disku, což je možné obzvlášť u počítačů instalovaných z předpřipravených obrazů, pokud k nim má útočník přístup. Útočník pak může změnit některé důležité části systému, například spouštěné binární soubory.[5, 6] AES-CBC neobsahuje žádnou autentizační část a nechrání tedy před podobným (ale ani náhodnými, například v případě poškození disku) změnami šifrovaných dat tak, jako jiné šifry, jako například AES-CCM.

2.1.2 Elephant difuzér

Elephant difuzér je nový algoritmus navržený pro původní variantu BitLockeru rozšiřující AES-CBC tak, aby byly odstraněny potenciální bezpečnostní problémy AES-CBC při použití pro šifrování blokových zařízeních, popsané v 2.1.1[5].

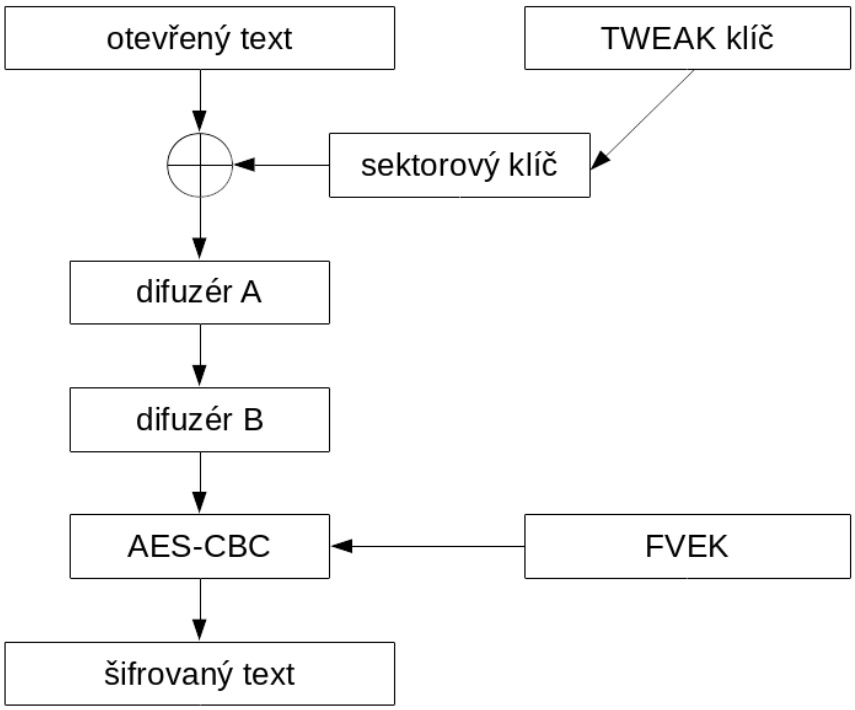
Zjednodušené schéma fungování šifrování pomocí AES-CBC s Elephant difuzérem je znázorněno na obrázku 2.2. Základem je použití dvou různých klíčů — TWEAK klíče pro difuzér a FVEK pro samotné AES-CBC. Podle [5] je tak zajištěno, že výsledek bude minimálně stejně bezpečný jako samotný AES-CBC a eliminuje se tak potenciální problém s použitím nové, nevyzkoušené šifry.

Prvním krokem při použití difuzéru je odvození takzvaného sektorového klíče K_s. Ten se získá z čísla sektoru a TWEAK klíče (K_{TWEAK}) následujícím způsobem:

$$K_s = E(K_{TWEAK}, e(s)) \parallel E(K_{TWEAK}, e'(s))$$

kde $e(s)$ je číslo sektoru uložené jako 64bit little endian (s vynulovanými nepoužitými bity) a $e'(s)$ je stejné jako $e(s)$, kromě posledního bajtu, který je nastaven na 128 (0x80). Výsledný 128bit klíč je pak následně podle potřeby zřetězen, aby odpovídal délce šifrovaného sektoru.[5, 6, 7]

Otevřený text je pak následně XORován s takto získaným klíčem a dále zpracován pomocí dvojice difuzérů A a B.



Obr. 2.2 Schéma šifrování sektoru pomocí AES-CBS s Elephant difuzérem[5, 7]

Přesný popis obou difuzérů je k dispozici v [5], zde si pouze stručně naznačíme jejich funkčnost, jejíž cílem je rozprostřít informaci po celé délce šifrovaného sektoru.

Difuzéry A (2.1) a B (2.2) fungují při šifrování dat následovně:

$$\begin{aligned} &\textbf{for } i = n \cdot A_{cycles} - 1, \ldots 2, 1, 0 \\ &\qquad d_i \leftarrow d_i - (d_{i-2} \oplus (d_{i-5} \lll {R_i^{(a)}}_{\bmod 4})) \end{aligned} \tag{2.1}$$

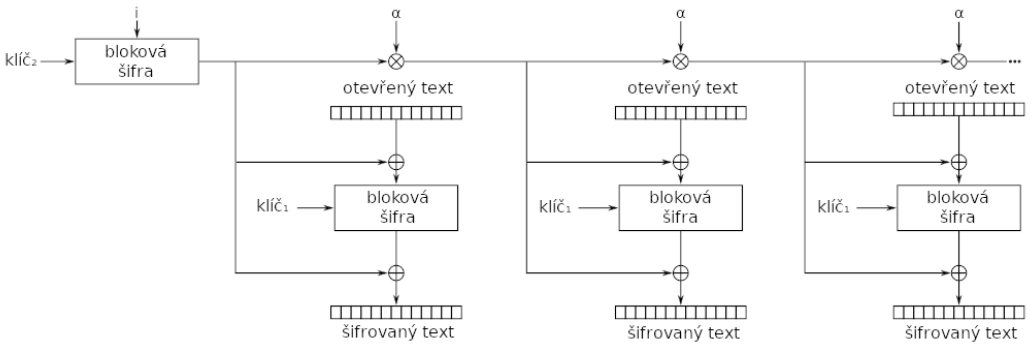
$$\begin{aligned} &\textbf{for } i = n \cdot B_{cycles} - 1, \ldots 2, 1, 0 \\ &\qquad d_i \leftarrow d_i - (d_{i+2} \oplus (d_{i+5} \lll {R_i^{(b)}}_{\bmod 4})) \end{aligned} \tag{2.2}$$

kde n je počet slov v sektoru, $(d_0, d_1, \ldots, d_{n-1})$ jsou slova daného sektoru, i je řídící proměnná cyklu, A_{cycles} a B_{cycles} je počet opakování difuzérů A a B (u BitLockeru je $A_{cycles} := 5$ a $B_{cycles} := 3$) a $R^{(a)}$ a $R^{(b)}$ počet posunutí doleva ($\lll$) v jednotlivých krocích.[5]

2.1.3 AES-XTS

AES-XTS je speciálně navržená bloková šifra pro šifrování dat uložených na blokových zařízeních. Jedná se o rozšíření původního XEX módu, jehož funkcionalita je naznačena na obrázku 2.3, o takzvaný *ciphertext stealing*, který rozšiřuje XEX mód o možnost

šifrovat data i o jiné délce, než jsou násobky 128bit.[8] Vzhledem k tomu, že u blokových zařízení se standardně používá délka sektoru 512 nebo 4096 bajtů, jedná se v tomto případě prakticky o XEX mód.



Obr. 2.3 Schéma šifrování pomocí blokové šifry v XEX módu[9]

Šifrování jednoho bloku pomocí AES-XTS pak probíhá následujícím způsobem

$$T = E(K_2, i) \otimes \alpha^j$$
$$C = E(K_1, P \oplus T) \oplus T$$

kde E je použitá šifrovací funkce (v tomto případě AES), C je výsledný šifrovaný blok, P blok otevřeného textu, K_1 a K_2 jsou klíče, i číslo sektoru, α^j prvek $GF(2^{128})$ odpovídající polynomiálnímu x (například 10_2), přičemž GF je Galoisovo těleso.[10]

Z výše uvedeného vyplývá, že pro AES-XTS jsou použity dva různé klíče — jeden pro šifrování čísla sektoru (inicializačního vektoru) a druhý pro šifrování samotného bloku otevřeného textu. Proto jsou klíče pro AES-XTS dvakrát větší, než kolik by vyžadovalo samotné AES, klíč pro 128bit AES-XTS tedy bude mít délku 256bit.

Stejně jako AES-CBC zmíněný v části 2.1.1, ani AES-XTS neobsahuje žádnou autentizaci a nechrání tedy před změnami šifrovaného textu.

2.1.4 AES-CCM

Poslední z šifrovacích funkcí použitých v BitLockeru je AES s CCM módem, která je použita pouze v rámci metadat na ochranu uložených klíčů. AES-CCM představuje zástupce šifer používaných pro takzvané autentizované šifrování, které zajišťují jako integritu, tak důvěrnost dat. Toho je typicky docíleno spojením šifrování a MAC.[11]

CCM mód představuje kombinaci dvou existujících technologií — CTR módu, který zajišťuje samotné šifrování a CBC-MAC, který zajišťuje autentizaci, přičemž pro obě části šifry se používá stejný klíč.

Postup při dešifrování a verifikace, který nás při práci s BitLocker zařízeními zajímá nejvíce, je následující: Na vstupu funkce jsou zašifrovaná data, řetězec pro verifikaci (MAC tag) a nonce použité při generování šifrovaných dat. Šifrovaný text je dešifrován v CTR módu, jehož výsledkem jsou otevřená data a MAC. Následně se pomocí CBC-MAC ověří dešifrovaný MAC a pokud je verifikace úspěšná, vrátí CCM funkce dešifrovaná data.[12]

Ačkoli je CCM mód obvykle používán pro zajištění, že s šifrovanými daty nebylo nijak manipulováno (jako například v IPsec[13]), v BitLockeru lze verifikační krok využít také k rychlému zjištění, zda bylo uživatelem zadáno správné heslo, protože v případě použití nesprávného hesla MAC autentizace také selže a není tak třeba složitě zjišťovat, zda dešifrovaný heslem chráněný klíč je ve skutečnosti správný či nikoli.

AES-CCM bylo kvůli zajištění autentizace také zvažováno pro šifrování samotných uložených dat. Problémem v tomto případě ale je fakt, že šifrovaná dat nemohou být větší, než data otevřená — potřebný MAC tag není možné uložit společně s šifrovaným blokem a ukládání MAC tagu do jiného bloku společně s MAC tagy pro další bloky porušuje atomicitu zápisu sektoru a v případě poškození sektoru s MAC tagy bude ztraceno více sektorů s daty, která nelze při dešifrování ověřit.[5]

2.1.5 Odvození klíče z hesla

Jedním z problémů, které je třeba řešit v případech, kdy je v kryptografii použito heslo zadávané uživatelem, je převedení tohoto hesla na klíč použitelný pro vybranou šifrovací funkci, v našem případě AES-CCM, která v BitLockeru slouží k ochraně na disku uložených klíčů. Obecně se v těchto případech používají speciální derivační funkce (KDF, Key Derivation Function), které z hesla odvodí klíč o požadované délce. Obvyklým postupem je zkombinování hesla s unikátní solí za použití hashovací funkce, kdy přidání soli ztěžuje možnost vytvoření předpočítaných tabulek hesel. Dalším používaným stupněm ochrany je pak přidání určitého počtu iterací, které je třeba pro odvození klíče provést. Tím se celý výpočet zpomaluje a teoreticky se tak zvyšuje odolnost proti útokům hrubou silou.[14]

Ačkoli existují standardizované funkce pro odvození klíče z hesla, jako například PBKDF2 definovaná v [14], Microsoft pro BitLocker zavedl vlastní funkci postavenou nad hashovací funkcí SHA256 s pevným počtem iterací.

Prvním krokem pro odvození klíče je zkonstruování struktury uvedené na obrázku 2.4.

Následně se počet iterací (`count`) a poslední spočtená hodnota hashe celé struktury `last_sha256` nastaví na nuly. Z uživatelem zadaného hesla se spočte SHA256 a z tohoto výsledku se spočte ještě jednou SHA256 a nastaví se jako výchozí hodnota

```
uint8_t last_sha256[32];
uint8_t initial_sha256[32];
uint8_t salt[8];
uint64_t count;
```

Obr. 2.4 Struktura pro odvození hesla z klíče podle [15]

hashe (`initial_sha256`). Potřebná sůl (`salt`) je uložena v metadatových záznamech u VMK, který je chráněn daným heslem (viz 2.3.2).

Dalším krokem je již samotná iterace, kdy se spočte SHA256 hash celé struktury a výsledek se uloží do proměnné `last_sha256`. Zároveň se zvýší čítač iterací o jedna a vše se opakuje celkem 1048576 (0x100000) krát. Poslední hodnota uložená v `last_sha256` po provedení všech iterací, je pak hledaný odvozený klíč.[15, 16]

2.1.6 Inicializační vektory

Ve výše uvedených popisech šifrovacích funkcí je několikrát zmíněn inicializační vektor. Ten se používá u blokových šifer, které pro zvýšení bezpečnosti před šifrováním XORují otevřený text s předchozím šifrovaným blokem. Inicializační vektor v tomto případě nahrazuje nultý šifrovaný blok pro první blok otevřeného textu.

Inicializační vektor je možné definovat různými způsoby jako je například fixní IV (v tom případě ale dojde ke stejnému problému jako u ECB módu — dva stejně začínající sektory vytvoří stejný šifrovaný text), náhodný IV, posloupnost čísel nebo IV generovaný z unikátního čísla nonce (z anglického „*n*umber used *once*“).[2]

V BitLockeru jsou použity pro výše zmíněné šifry následující inicializační vektory:

- u AES-CBC číslo sektoru zašifrované pomocí AES-ECB s klíčem použitým pro šifrování dat,
- u AES-XTS pouze jednoduché číslo sektoru uložené jako 128bit little indian a
- u AES-CCM nonce uložené společně s klíčem, který je pomocí AES-CCM zašifrován. [15]

2.2 Diskový formát

Pro samotnou práci s BitLocker zařízením v linuxovém prostředí je nejdůležitější formát, tedy způsob, jakým jsou na disku uložena data. Protože je pomocí BitLockeru možné vytvořit šifrovaný flash disk, který lze použít na jiném počítači pouze za znalosti hesla, je zřejmé, že někde na samotném disku jsou uložena všechna potřebná metadata

pro jeho „odemčení“[1] v (alespoň částečně) otevřené podobě[2].

TODO: Přesunout do úvodu

16	68760	128	21440	128	22632	128	91568
Hl.	Data	FVE 1	Data	FVE 2	Data	FVE 3	Data

Obr. 2.5 Zjednodušené schéma struktury 100 MiB BitLocker zařízení

Na obrázku 2.5 je nastíněna zjednodušená struktura uložení dat a metadat na zařízení šifrovaném pomocí BitLockeru. Na začátku zařízení se nachází 8 KiB velká hlavička (podrobněji popsána v části 2.2.1) obsahující základní data pro jeho identifikaci a mezi zašifrovanými daty jsou uloženy tři kopie dalších metadat, každá o velikosti 64 KiB[3] (podrobněji popsány v části 2.2.2). Čísla nad jednotlivými „částmi“ schématu odpovídají jejich velikosti v sektorech pro testovací zařízení o velikosti 100 MiB, které bylo vytvořeno ve Windows 10.

2.2.1 Hlavička

Stejně jako u většiny diskových formátů, je i u BitLockeru na začátku disku takzvaná hlavička, která obsahuje základní informace o použitém formátu a jeho vlastnostech a také slouží k jeho rychlé identifikaci.

BitLocker hlavička zabírá celkem 512 bajtů a je u ní patrná inspirace u souborového systému NTFS. V tabulce 2.1 jsou zobrazeny jednotlivé známé položky hlavičky BitLockeru a pro srovnání také stejné položky v hlavičce souborového systému NTFS. Struktura formátu BitLocker není společností Microsoft nikde veřejně zcela kompletně zdokumentována, význam jednotlivých položek tedy nemusí být vždy přesně znám.

Popis struktury NTFS hlavičky je převzat z [18], popis struktury BitLocker hlavičky je pak částečně převzata z [15], částečně z [5] a částečně výsledkem vlastního zkoumání.

Z pohledu identifikace BitLocker zařízení je nejdůležitější částí hlavičky 8 bajtů na offsetu 3, které se u NTFS formátu nazývají *OEM název* a které slouží pro rychlou identifikace zařízení. V linuxových systémech se podobné identifikátory obvykle nazývají *signatura*. Pro BitLocker formát je (u všech verzí) signatura v ASCII podobě

TODO: To by asi chtělo citaci.

[1] U šifrovaných úložných zařízení se běžně používá termín *odemčení* pro jeho „připravení“ pro čtení. Odemčení dává větší smysl, než dešifrování, protože data se dešifrují až při jejich čtení (abychom se vyhnuli relativně pomalému dešifrování dat, která nebudou čtena). Při odemčení se tedy pouze z metadat (nebo z jiného hardwaru, jako například TPM) získá (de)šifrovací klíč a připraví se (virtuální) zařízení, ze kterého lze číst data v otevřené podobě.

[2] Například populární nástroj VeraCrypt na zašifrovaném disku žádná metadata v otevřené podobě nemá.[17]

[3] Velikosti odpovídají místu, které je pro daná metadata na zařízení vyhrazeno. Ve skutečnosti mohou být metadata mnohem menší.

Tab. 2.1 Porování položek hlaviček BitLocker a NTFS

offset	velikost	BitLocker	NTFS
0	3	boot kód	
3	8	OEM název (signatura)	
11	2	počet bajtů na sektor	
13	1	počet sektorů na cluster	
14	2	rezervované sektory	
16	4	nepoužito	
21	1	popisek média	
22	18	nepoužito	
40	8	počet sektorů	
48	8	adresa prvního clusteru MFT	
56	8	kopie adresy prvního clusteru MFT	
64	1	velikost MFT entry	
65	3	nepoužito	
68	1	velikost indexu	
69	3	nepoužito	
72	8	NTFS serial number	
80	4	nepoužito	
84	76	boot kód	
160	16	BitLocker GUID	boot kód
176	8	offset první kopie FVE metadat	
184	8	offset druhé kopie FVE metadat	
192	8	offset třetí kopie FVE metadat	
200	310	boot kód	
510	2	signatura (`0xaa55`)	

`-FVE-FS-`[19].

Pro další práci s BitLockerem není většina položek hlavičky zajímavá. Výjimku tvoří GUID identifikátor uložený na offsetu 160 (16 bajtů dlouhý UTF-8 textový řetězec) a trojice 32bitových bezznaménkových celočíselných (`uint32`) hodnot na offsetech 176, 184 a 192, které obsahují umístění (jako relativní offset od začátku zkoumaného zařízení) tří bloků FVE metadat. Všechny tyto čtyři hodnoty jsou v BitLocker hlavičce umístěny na offsetech, které jsou v NTFS součástí *bootcode*.

Umístění všech výše zmíněných „důležitých“ částí BitLocker hlavičky je zobrazeno na obrázku 2.6.

2.2.2 FVE metadata

Samotná výše popsaná hlavička formátu BitLocker obsahuje především informace sloužící pro rychlou identifikaci zařízení jako zařízení šifrovaného pomocí technologie BitLocker. Všechny informace potřebné pro práci s tímto zařízením, tedy především způsob uložení

```
00000000  eb 58 90 2d 46 56 45 2d  46 53 2d 00 02 08 00 00 |.X.-FVE-FS-.....|
00000010  00 00 00 00 00 f8 00 00  3f 00 ff 00 00 28 03 00 |........?....(..|
00000020  00 00 00 00 e0 1f 00 00  00 00 00 00 00 00 00 00 |................|
00000030  01 00 06 00 00 00 00 00  00 00 00 00 00 00 00 00 |................|
00000040  80 00 29 00 00 00 00 4e  4f 20 4e 41 4d 45 20 20 |..)....NO NAME  |
00000050  20 20 46 41 54 33 32 20  20 20 33 c9 8e d1 bc f4 |  FAT32   3.....|
00000060  7b 8e c1 8e d9 bd 00 7c  a0 fb 7d b4 7d 8b f0 ac |{......|..}.}...|
00000070  98 40 74 0c 48 74 0e b4  0e bb 07 00 cd 10 eb ef |.@t.Ht..........|
00000080  a0 fd 7d eb e6 cd 16 cd  19 00 00 00 00 00 00 00 |..}.............|
00000090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |................|
000000a0  3b d6 67 49 29 2e d8 4a  83 99 f6 a3 39 e3 d0 01 |;.gI)..J....9...|
000000b0  00 50 19 02 00 00 00 00  00 d0 c1 02 00 00 00 00 |.P..............|
000000c0  00 a0 73 03 00 00 00 00  00 00 00 00 00 00 00 00 |..s.............|
000000d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |................|
*
00000100  0d 0a 52 65 6d 6f 76 65  20 64 69 73 6b 73 20 6f |..Remove disks o|
00000110  72 20 6f 74 68 65 72 20  6d 65 64 69 61 2e ff 0d |r other media...|
00000120  0a 44 69 73 6b 20 65 72  72 6f 72 ff 0d 0a 50 72 |.Disk error...Pr|
00000130  65 73 73 20 61 6e 79 20  6b 65 79 20 74 6f 20 72 |ess any key to r|
00000140  65 73 74 61 72 74 0d 0a  00 00 00 00 00 00 00 00 |estart..........|
00000150  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |................|
*
00000190  00 00 00 00 00 00 00 00  78 78 78 78 78 78 78 78 |........xxxxxxxx|
000001a0  78 78 78 78 78 78 78 78  78 78 78 78 78 78 78 78 |xxxxxxxxxxxxxxxx|
*
000001e0  78 78 78 78 78 78 78 78  ff ff ff ff ff ff ff ff |xxxxxxxx........|
000001f0  ff ff ff ff ff ff ff ff  ff ff ff 00 1f 2c 55 aa |.............,U.|
00000200
```

Obr. 2.6 BitLocker hlavička se zvýrazněnou signaturou, GUID a trojicí offsetů FVE metadat

dat, jejich umístění, způsob jakým jsou šifrována a hlavně klíč pro jejich (de)šifrování, jsou uloženy na třech různých místech[4)] definovaných v hlavičce. Jedná se o tři identické kopie[5)] takzvaných *FVE metadat*.

FVE metadata se skládají z celkem tří částí – hlavičky FVE bloku (*FVE metadata block header*), samotné FVE hlavičky (*FVE metadata header*) a různého množství FVE záznamů (*FVE metadata entry*, které obsahují samotné klíče a další důležité informace[15][6)].

[4)] Na offsetech přibližně ve 33 %, 44 % a 55 % u testovaných BitLocker zařízení.

[5)] Tři kopie jsou zvoleny pravděpodobně jako záloha pro případ náhodného poškození metadat. Vzhledem k tomu, že bez kompletní nepoškozené kopie těchto metadat není možné data na zařízení dešifrovat, je vícenásobná záloha na místě.

[6)] Toto dělení zavádí Joachim Metz v [15]. Teoreticky by se daly dvě první části metadat spojit, protože na disku se nachází vždy hned za sebou, ale rozdělení dává smysl, protože první část se týká popisu samotných metadat (signatura, verze, umístění všech tří bloků), zatímco druhá část už obsahuje samotná metadata (GUID, čas vytvoření, použitý šifrovací algoritmus).

Důležité položky v obou hlavičkách, jejich velikosti a offsety (vztažené vůči začátku dané hlavičky) jsou uvedeny v tabulce 2.2. Kompletní struktura obou hlaviček je součástí přílohy .

TODO: odkaz na přílohu

Tab. 2.2 Zjednodušená struktura FVE metadat

Hlavička FVE bloku		
offset	velikost	popis
0	8	signatura (`-FVE-FS-`)
10	2	verze (1 nebo 2)
32	8	offset první kopie FVE metadat
40	8	offset druhé kopie FVE metadat
48	8	offset třetí kopie FVE metadat

FVE hlavička		
0	4	velikost metadat (včetně záznamů)
16	16	GUID
36	4	šifrovací algoritmus
40	8	datum a čas vytvoření

Mezi pro nás zajímavé položky v hlavičce patří její celková velikost (včetně velikosti samotné hlavičky a velikosti za ní následujících záznamů), šifrovací algoritmus použitý pro zašifrování dat uložených na disku (možné algoritmy jsou popsány v části 2.1) a také čas vytvoření, který je uložen ve formátu `FILETIME`[7)].

2.2.3 FVE záznamy

Za výše uvedenou hlavičkou se nachází blíže nespecifikované množství FVE záznamů. Ty slouží v podstatě jako key-value úložiště pro jakékoli další „informace“, které jsou pro práci s BitLockerem potřebné. Tím, že není třeba předem určeno, kolik takových záznamů bude za hlavičkou uloženo, je možné přidávat nové položky při zachování zpětné kompatibility[8)].

Jelikož známe celkovou velikost FVE metadat (je uvedena v hlavičce, viz tabulka 2.2) a celková velikosti hlaviček FVE metadat je pevná (64 a 48 bajtů), pro přečtení všech záznamů stačí číst data ve smyčce, dokud nedojdeme na konec metadat, nebo dokud následující záznam nemá nulovou velikost.

[7)] FILETIME je ve skutečnosti struktura sestávající ze dvou 32bit celočíselných hodnot, které dohromady udávají počet 100 nanosekundových intervalů, které k danému datu uplynuly od 1. ledna 1601.[20]

[8)] Celková největší možná velikost FVE metadat je 64 KiB (alespoň tedy tolik je pro FVE metadata vyhrazeno na vytvořených BitLocker zařízeních), teoreticky je tedy možné mít až 64 KiB - 112 B metadat.

Struktura FVE je relativně jednoduchá a je popsaná v tabulce 2.3. Důležitou součástí je velikost záznamu, protože podle svého typu může mít různou délku.

Tab. 2.3 Struktura FVE záznamu

offset	velikost	popis
0	2	velikost záznamu
2	2	typ záznamu
4	2	typ hodnoty záznamu
6	2	verze (1)
8		data

Typ a hodnota označují, co je v daném záznamu uloženo. Známé typy a hodnoty jsou popsány v tabulce 2.4. U typů se typicky jedná buď o klíč (FVEK, VMK), nebo obecnou *property*, hodnota pak dále specifikuje, jak je daný typ uložen (zašifrovaný klíč, textový řetězec).

Způsob uložení dat záleží na tom, jaká konkrétní data jsou v záznamu uložena. U „jednoduchých“ záznamů, jako je například popisek, je v datech uložen textový řetězec uložený v kódování UTF-16, u „složitějších“ záznamů, jako jsou například klíče, mají data vlastní strukturu včetně dalších záznamů.

Tab. 2.4 Známé typy FVE záznamů

Typy	
typ	popis
0	property
1	VMK
2	FVEK
7	popisek
15	hlavička disku[9)]

Hodnoty	
typ	popis
0	smazáno
1	klíč
2	string
5	AES-CCM šifrovaný klíč
6	TPM klíč
8	VMK
15	offset a velikost

Příklad „jednoduchého“ záznamu je uveden na obrázku 2.7, kde vidíme záznam typu *description* (popisek). Ten v podstatě obsahuje jméno počítače, na kterém bylo dané BitLocker zařízení vytvořeno a také datum vytvoření. Můžeme tedy vidět, že toto konkrétní BitLocker zařízení bylo vytvořeno na počítači `DESKTOP-NPM7RCA` a to 3. února 2019. Tato informace je uložena jako standardní textový řetězec v kódování UTF-16. Kromě tohoto řetězce jsou pak na obrázku zvýrazněny i další údaje: velikost

[9)] Umístění a velikost NTFS hlavičky otevřeného zařízení. Odpovídá hodnotě 15. Podrobnější informace o umístění NTFS hlavičky na šifrovaném zařízení jsou v části 2.4.1.

celého záznamu (64 bajtů), jeho typ (7 — popisek) a hodnota (2 — textový řetězec) a verze (1).

```
02195070  40 00 07 00 02 00 01 00  44 00 45 00 53 00 4b 00 |@.......D.E.S.K.|
02195080  54 00 4f 00 50 00 2d 00  4e 00 50 00 4d 00 37 00 |T.O.P.-.N.P.M.7.|
02195090  52 00 43 00 41 00 20 00  47 00 3a 00 20 00 32 00 |R.C.A. .G.:. .2.|
021950a0  2f 00 33 00 2f 00 32 00  30 00 31 00 39 00 00 00 |/.3./.2.0.1.9...|
```

Obr. 2.7 Příklad FVE záznamu typu „description“ (popisek)

U jednoduchého zařízení — v tomto konkrétním případě USB flash disku — se bude obvykle vyskytovat pouze pět záznamů a to již výše zmíněný popisek, dvojice záznamů typu VMK, jeden záznam typu FVEK (více informací o obou se nachází v části 2.3) a jeden záznam obsahující informace o umístění hlavičky disku (více informací o tomto záznamu se nachází v části 2.4.1).

2.3 Klíče

Pravděpodobně nejdůležitější součástí BitLocker hlavičky jsou šifrovací klíče. Ve FVE metadatech nalezneme celkem dva typy klíčů — Full Volume Encryption Key, neboli FVEK, a Volume Master Key, neboli VMK[10]. Uloženy jsou v metadatových záznamech odpovídajících typů v zašifrované podobě.

2.3.1 Full Volume Encryption Key

Full Volume Encryption Key (dále jen „FVEK“) je nejdůležitějším klíčem pro celý BitLocker. Pomocí tohoto klíče jsou totiž zašifrovaná data uložená na disku. FVEK samotný nejde změnit bez kompletního přešifrování všech dat a v případě jeho poškození nebo náhodného smazání, není možné uložená data nijak dešifrovat.

FVEK je v metadatech uložen v záznamu typu *FVEK* s hodnotou *AES-CCM šifrovaný klíč* a je, jak hodnota naznačuje, zašifrován pomocí šifry AES-CCM (o této šifře a módu více v části 2.1), kdy je jako klíč použit VMK.

Struktura dat pro FVEK v metadatovém záznamu je popsána v tabulce 2.5. Kromě samotného klíče obsahují datum a čas jeho vytvoření a nonce. Nonce (z anglického „*n*umber used *once*“) je unikátní číslo (v tomto případě prostá posloupnost začínající od nuly a zvyšující se s každým klíčem) sloužící společně s datem a časem vytvoření jako inicializační vektor pro dešifrování[2].

[10] Původní varianta BitLockeru má ještě jeden klíč — TWEAK, ten je podrobněji popsán v části 2.5.

[11] Velikost šifrovaného klíče záleží na použité šifře — 12 bajtů vždy připadne na informace o klíči a 32 bajtů v tomto případě připadá na samotný klíč, jelikož je použit 128bit AES.

Tab. 2.5 Způsob uložení FVEK v metadatech

offset	velikost	popis
0	8	datum a čas vytvoření (jako FILETIME)
8	4	nonce
12	16	MAC tag
28	44[11)]	šifrovaný klíč

Samotná zašifrovaná část klíče obsahuje kromě samotného klíče také další data o klíči samotném — velikost, verze a šifrovací metoda použitá pro data zašifrovaná pomocí FVEK. Jejich struktura je popsána v tabulce 2.6.

Tab. 2.6 Obsah FVEK po dešifrování

offset	velikost	popis
0	4	velikost
4	4	verze (1)[12)]
8	4	šifrovací metoda
12	32	klíč

Na obrázku 2.8 je pak vidět příklad dešifrovaného FVEK. Zvýrazněny jsou jeho celková velikost (44 bajtů), verze (1), použitá šifrovací funkce (hexadecimální kód `0x8004` v tomto případě znamená 128bit AES-XTS) a následně samotnou dvojici 128bit klíčů pro AES-XTS.

```
00000000  2c 00 00 00 01 00 00 00  04 80 00 00 a4 d0 11 64  |,..............d|
00000010  0c a0 df ec b2 4d a2 39  b1 4e 4a b7 62 56 f2 e3  |.....M.9.NJ.bV..|
00000020  b2 27 54 40 91 21 0e 98  aa 84 5f 52              |.'T@.!...._R|
```

Obr. 2.8 Dešifrovaný FVEK

2.3.2 Volume Master Key

Jak již bylo řečeno výše, FVEK je na disku uložen zašifrován pomocí Volume Master Key (dále jen „VMK“). Ten je uložen také v metadatových záznamech ve FVE metadatech a je také zašifrován. Na rozdíl od FVEK, který je vždy uložen v pouze v jediné kopii, VMK může být ve FVE metadatech uložen vícekrát, pokaždé chráněný jiným způsobem, tedy pokaždé jinak zašifrovaný.

[12)] Některé zdroje [15] uvádějí verzi pouze jako 2 bajtovou a následující 2 bajty jako „neznámé“. Vzhledem k tomu, že v jiných hlavičkách je verze v některých případech 4 bajtová a v některých 2 bajtová a že na testovacích zařízeních byly tyto dva bajty vždy nulové, domnívám se, že je pravděpodobnější, že verze je zde 4 bajtová.

Tento systém umožňuje, aby byl FVEK (jakožto hlavní a nejdůležitější klíč) uložen na disku pouze v jediné kopii, ale zároveň existovala možnost, jak mít pro jedno zařízení více různých hesel (respektive více různých způsobů odemčení daného zařízení)[21]. Pro přidání nového hesla tak teoreticky stačí jednoduše znát alespoň jedno již existující[22], pomocí kterého se VMK dešifruje a následně uloží zašifrovaný pomocí nového hesla. Analogicky tak lze také snadno změnit heslo — jak již bylo zmíněno výše, FVEK nejde změnit bez přešifrování celého zařízení, ale změna hesla díky tomuto systému znamená pouhé uložení nově zašifrovaného VMK.

V odstavci výše je několikrát zmíněno *heslo*, ale VMK může být chráněn více různými způsoby. Dokumentace BitLockeru [23] zmiňuje celkem deset možných typů *protektorů* klíčů (tedy deset způsobů, jak může být daný klíč chráněn, respektive šifrován) v BitLockeru. Tyto možnosti jsou zapsány v tabulce 2.7.

Tab. 2.7 Možnosti ochrany VMK

hodnota	popis
0	neznámý/jiný
1	TPM
2	externí klíč
3	číselné heslo
4	TPM a PIN
5	TPM klíč
6	TPM, PIN a klíč
7	veřejný klíč
8	heslo
9	TPM certifikát
10	CryptoAPI Next Generation (CNG)

Z pohledu této práce je nejobvyklejším protektorem právě heslo, protože použití BitLocker zařízení v linuxovém prostředí se dá předpokládat primárně u flash disků, u nichž se používá ochrana heslem[13)].

Pro každé vytvořené BitLocker se kromě „primární“ ochrany (v našem případě typicky hesla) vytváří ještě jeden VMK chráněný takzvaným záložním heslem. Způsob ochrany je u něj stejný jako u VMK, který je chráněný heslem, rozdíl je v tom, že heslo zadává uživatel, kdežto záložní heslo je vygenerované a uživateli je při vytváření „předáno“ v podobě souboru, který obsahuje 48 čísel. U něj se předpokládá, že si jej uživatel buď vytiskne nebo bezpečně uloží v elektronické podobě. U strojů přihlášených v síti Active Directory, lze také záložní klíče automaticky zálohovat na doménovém serveru. Pomocí záložního hesla lze pak zařízení odemknout stejně, jako při použití

[13)] Ochrana pomocí TPM nedává u přenosných disků smysl, protože TPM čipy jsou nedělitelnou součástí hardwaru a takto chráněný disk by nešlo na jiném počítači dešifrovat.

„normálního“ hesla a pomocí nástrojů obsažených v základní instalaci Windows nastavit nové heslo (nebo nastavit nové TPM, či jiný způsob ochrany).[24]

Struktura VMK, naznačená v tabulce 2.8, je ovlivněna tím, že samotný klíč může být chráněn různými způsoby a je pro něj tedy třeba ukládat různá metadata, a i samotný klíč může být potřeba v některých případech ukládat v různých podobách.

První část VMK struktury je v celku běžná — obsahuje identifikátor klíče (GUID), čas vytvoření a typ ochrany. Další metadata jsou pak uložena jako záznamy, stejně jako u samotné FVE hlavičky (podrobněji v části 2.2.3 a tabulce 2.3).

Tab. 2.8 Struktura VMK

offset	velikost	popis
0	16	GUID
16	9	datum a čas vytvoření
24	2	neznámé
26	2	typ ochrany
28		metadatové záznamy

Kompletní VMK klíč chráněný záložním heslem je zobrazen na obrázku 2.9. Zvýrazněno je GUID, typ ochrany (8 — heslo) a dva „připojené“ záznamy, oba typu property, první obsahující sůl potřebnou pro odvození klíče potřebného pro dešifrování VMK ze záložního hesla (funkcionalita odvození klíče z hesla je popsána v části 2.1.5) a druhá obsahující samotný klíč (textový výpis je debugovacím výstupem z nástroje vytvořeného v rámci praktické části).

2.4 Šifrovaná data

2.4.1 Způsob uložení data

Po hlavičkách a metadatech zbývá popsat jen způsob, jakým jsou na disku uložena samotná šifrovaná data. Protože BitLocker metadata se vyskytují celkem ve třech kopiích na různých místech „uprostřed“ šifrovaného zařízení, jsou uložená data rozdělena celkem na čtyři části. Až na jednu výjimku jsou šifrovaná data uložena na správných místech — tedy na místě, kde mají být uložena i po dešifrování.

Výjimkou je v tomto případě hlavička souborového systému NTFS. Její umístění je způsobeno poněkud zvláštním rozhodnutím tvůrců BitLockeru, že otevřené zařízení bude mít stejnou velikost, jako zařízení zašifrované, a to i přesto, že si z jeho celkové velikosti BitLocker metadata uberou přibližně 200 KiB[14)].

[14)] U linuxové implementace šifrování disku, technologie LUKS/dm-crypt, byl zvolen jiný přístup — otevřené zařízení je menší a metadata se na něm nijak neřeší — jsou z výsledného zařízení „odstraněna“. První sektor otevřeného zařízení pak obsahuje standardní hlavičku souborového systému bez potřeby dalšího „přesouvání“ jako u BitLockeru.

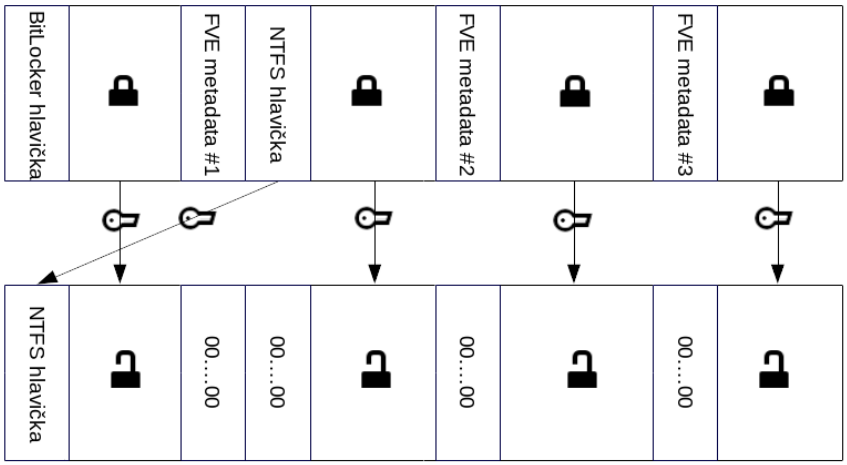
```
00000000 c1 56 2e 01 d6 4e 27 45  8a bf 7a 9f 29 e0 b5 21  |.V...N'E..z.)..!|
00000010 40 c5 cd 54 a0 bb d4 01  00 00 00 08 ac 00 00 00  |@..T............|
00000020 03 00 01 00 00 10 00 00  46 ee b7 10 0e 43 4d d4  |........F....CM.|
00000030 f1 84 a5 ab eb c6 21 f4  40 00 12 00 05 00 01 00  |....!...@.......|
00000040 40 7d e5 52 a0 bb d4 01  04 00 00 00 72 b0 71 f4  |@}.R........r.q.|
00000050 20 9e c9 8e b7 1b 5e 42  71 b5 bc 21 c6 57 9b 29  | .....^Bq..!.W.)|
00000060 56 2c 92 ad db d7 73 75  a9 78 c2 94 c5 a5 07 d1  |V,....su.x......|
00000070 62 61 0c 56 d8 ca 9d ac  50 00 13 00 05 00 01 00  |ba.V....P.......|
00000080 40 7d e5 52 a0 bb d4 01  05 00 00 00 3e d9 ac 58  |@}.R........>..X|
00000090 e6 86 ba ac 05 48 ea 0b  64 ee 77 7a b4 77 ba cb  |.....H..d.wz.w..|
000000a0 c0 83 83 b0 7b ab 52 c7  0d 9e 8f 62 d7 cb a3 90  |....{.R....b....|
000000b0 cc b8 8e 39 a4 be 8a 0a  5c 16 86 62 c9 64 81 4d  |...9....\..b.d.M|
000000c0 91 9d 27 24 3a 8e a3 7c  50 00 00 00 05 00 01 00  |..'$:..|P.......|
000000d0 40 7d e5 52 a0 bb d4 01  06 00 00 00 97 18 2f d6  |@}.R........../.|
000000e0 83 de e7 63 0a fa 57 48  44 2b 66 90 91 a0 ad e9  |...c..WHD+f.....|
000000f0 0c 08 e8 1e 3d 2f 7d 3b  cc 9f ba e4 ed b5 6b c2  |....=/};......k.|
00000100 e1 a4 53 cf c5 60 2a 92  2d c8 1d 85 10 b7 99 87  |..S..`*.-.......|
00000110 9d 1d 1e 36 46 40 6b e7                          |...6F@k.        |

VMK
        Identifier:     012e56c1-4ed6-4527-8abf-7a9f29e0b521
        Type:           VMK protected with recovery password
        Salt:           46 ee b7 10 0e 43 4d d4 a5 ab eb c6 21 f4 f1 84
        AES-CCM encrypted key
                Nonce data:     2019-02-03 09:10:36.052000
                Nonce counter:  6
                Key:     91 a0 ad e9 0c 08 ... 1d 1e 36 46 40 6b e7
```

Obr. 2.9 VMK chráněný záložním heslem

Speciální zacházení vyžaduje NTFS hlavička proto, že na výsledném otevřeném zařízení musí být na jeho začátku, aby toto zařízení bylo systémem správně rozpoznáno jako NTFS a jako takové připojeno. Proto je třeba NTFS hlavičku přesunout na začátek disku a nahradit jí původní BitLocker hlavičku. Umístění NTFS hlavičky v šifrovaných datech je zapsáno ve FVE metadatech ve speciálním záznamu typu *hlavička disku* (viz tabulka 2.4). V záznamu je uveden offset (relativně k začátku disku), na kterém se zašifrovaná NTFS hlavička nachází a její velikost (u testovaných zařízení 8 KiB, což také odpovídá velikosti vyhrazené pro BitLocker hlavičku).

Vzhledem k tomu, že výsledné otevřené zařízení má mít stejnou velikosti, jako šifrované zařízení, zbývá ještě vyřešit, jak bude v otevřeném zařízení naloženo s metadaty — na jejich místě v otevřeném NTFS musí „něco“ být a zároveň je třeba ochránit je proti náhodnému přepsání nebo smazání. Teoreticky je možné tato metadata prostě dešifrovat stejně jako ostatní šifrovaná data. Výsledkem by pak sice byla nesmyslná data, ale pokud je zařízení již otevřeno, není třeba k metadatům již znovu přistupo-



Obr. 2.10 Schéma „mapování“ mezi šifrovaným a otevřeným BitLocker zařízením

vat a je tedy jedno, že nejsou „čitelná“. Takováto „nesmyslná“ data by už jen stačilo v rámci NTFS ochránit před přepsáním. Pokud by bylo třeba k metadatům přistupovat i u otevřeného zařízení, bylo by další možností nechat je prostě viditelná tak, jak jsou (a opět je ochránit před přepsání).

Autoři BitLockeru ale nakonec sáhli po třetí možnosti — metadata jsou v otevřeném zařízení nahrazena nulami. Ve výsledném otevřeném NTFS tak jsou metadata viditelná jako speciální systémové soubory uložené ve složce `System Volume Information`. Soubory jsou samozřejmě prázdné, respektive plné nul, ale zabraňují přepsání míst, na kterých se skutečná metadata vyskytují. Ve Windows je tato složka ve výchozím nastavení skryta.

2.4.2 Postup při dešifrování

Při znalosti struktury metadat a způsobu uložení šifrovaných dat, je další postup dešifrování následující: z FVE hlavičky (tabulka 2.2) zjistíme, jaký byl použit šifrovací algoritmus (v nejnovějších verzích BitLockeru to bude AES-XTS), pomocí uživatelem zadaného hesla dešifrujeme VMK s odpovídajícícm typem ochrany (tabulka 2.7) a pomocí něj dešifrujeme FVEK, kterým jsou zašifrována samotná data.

Jedinou neznámou potřebnou pro dešifrování dat tak zůstává inicializační vektor. Ten u nejnovější verze BitLockeru odpovídá offsetu (v sektorech), na kterém jsou daná šifrovaná data uložena na zašifrovaném zařízení (první sektor NTFS hlavičky, který bude v dešifrovaných datech uložen na začátku tak má inicializační vektor daný svou pozicí v šifrovaných datech, nikoli nulový, jak by se mohlo zdát). O způsobu použití inicializačního vektoru v BitLockeru podrobněji pojednává část 2.1.6.

Dešifrovaný první sektor je vidět na obrázku 2.11. Jde zde velmi dobře poznat podob-

```
00000000  eb 52 90 4e 54 46 53 20  20 20 20 00 02 08 00 00  |.R.NTFS    .....|
00000010  00 00 00 00 00 f8 00 00  3f 00 ff 00 00 28 03 00  |........?....(..|
00000020  00 00 00 00 80 00 00 00  ff 1f 03 00 00 00 00 00  |................|
00000030  55 21 00 00 00 00 00 00  02 00 00 00 00 00 00 00  |U!..............|
00000040  f6 00 00 00 01 00 00 00  52 53 3d 84 7d 3d 84 a4  |........RS=.}=..|
00000050  00 00 00 00 fa 33 c0 8e  d0 bc 00 7c fb 68 c0 07  |.....3.....|.h..|
00000060  1f 1e 68 66 00 cb 88 16  0e 00 66 81 3e 03 00 4e  |..hf......f.>..N|
00000070  54 46 53 75 15 b4 41 bb  aa 55 cd 13 72 0c 81 fb  |TFSu..A..U..r...|
00000080  55 aa 75 06 f7 c1 01 00  75 03 e9 dd 00 1e 83 ec  |U.u.....u.......|
00000090  18 68 1a 00 b4 48 8a 16  0e 00 8b f4 16 1f cd 13  |.h...H..........|
000000a0  9f 83 c4 18 9e 58 1f 72  e1 3b 06 0b 00 75 db a3  |.....X.r.;...u..|
000000b0  0f 00 c1 2e 0f 00 04 1e  5a 33 db b9 00 20 2b c8  |........Z3... +.|
000000c0  66 ff 06 11 00 03 16 0f  00 8e c2 ff 06 16 00 e8  |f...............|
000000d0  4b 00 2b c8 77 ef b8 00  bb cd 1a 66 23 c0 75 2d  |K.+.w......f#.u-|
000000e0  66 81 fb 54 43 50 41 75  24 81 f9 02 01 72 1e 16  |f..TCPAu$....r..|
000000f0  68 07 bb 16 68 52 11 16  68 09 00 66 53 66 53 66  |h...hR..h..fSfSf|
00000100  55 16 16 16 68 b8 01 66  61 0e 07 cd 1a 33 c0 bf  |U...h..fa....3..|
00000110  0a 13 b9 f6 0c fc f3 aa  e9 fe 01 90 90 66 60 1e  |.............f`.|
...
00000150  0f 82 16 00 66 ff 06 11  00 03 16 0f 00 8e c2 ff  |....f...........|
00000160  0e 16 00 75 bc 07 1f 66  61 c3 a1 f6 01 e8 09 00  |...u...fa.......|
00000170  a1 fa 01 e8 03 00 f4 eb  fd 8b f0 ac 3c 00 74 09  |............<.t.|
00000180  b4 0e bb 07 00 cd 10 eb  f2 c3 0d 0a 41 20 64 69  |............A di|
00000190  73 6b 20 72 65 61 64 20  65 72 72 6f 72 20 6f 63  |sk read error oc|
000001a0  63 75 72 72 65 64 00 0d  0a 42 4f 4f 54 4d 47 52  |curred...BOOTMGR|
000001b0  20 69 73 20 63 6f 6d 70  72 65 73 73 65 64 00 0d  | is compressed..|
000001c0  0a 50 72 65 73 73 20 43  74 72 6c 2b 41 6c 74 2b  |.Press Ctrl+Alt+|
000001d0  44 65 6c 20 74 6f 20 72  65 73 74 61 72 74 0d 0a  |Del to restart..|
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001f0  00 00 00 00 00 00 8a 01  a7 01 bf 01 00 00 55 aa  |..............U.|
00000200
```

Obr. 2.11 První sektor dešifrovaného BitLocker zařízení (NTFS hlavička)

nost NTFS hlavičky s BitLocker hlavičkou (obrázek 2.6). Hlavní odlišností je signatura, která je zde jasně viditelná jako `NTFS`, na rozdíl od `-FVE-FS-` u BitLockeru. Chybí také offsety BitLocker metadat, které u BitLocker hlavičky „zabírají“ část boot kódu, který je u NTFS kompletní. Zajímavá je stejná boot signatura (`55 aa`) u obou hlaviček.

U zbývajících dat pak dešifrování probíhá stejně — po 512 B sektorech s inicializačním vektorem nastaveným na číslo sektoru odpovídající jejich umístění na šifrovaném zařízení. Jedinou výjimkou jsou oblasti BitLocker metadata, která jsou nahrazena nulami, jak bylo popsáno v části 2.4.1.

2.5 Odlišnosti ve starších verzích

Výše popsaná struktura diskového formátu BitLocker, způsob uložení klíčů a rozložení dat na šifrovaném a dešifrovaném zařízení, odpovídají aktuální nejnovější verzi BitLockeru dostupné ve Windows 7 a novějších. Původní verze dostupná ve Windows Vista se v některých drobnostech mírně liší. Tato práce se primárně zaobírá nejnovější verzí, protože je v současné době jedinou podporovanou (oficiální podpora Windows Vista byla ukončena 11. dubna 2017[25]). Podpora pro starší verze však může být také v některých případech vyžadovaná, a proto si ve stručnosti představíme nejvýznamnější odlišnosti mezi těmito verzemi.

Nejvýraznější změnou je nejspíše změna algoritmu použitého pro šifrování data z AES-CBC s Elephant difuzérem nejdříve pouze na AES-CBC[6] a později na AES-XTS[26] (rozdíl mezi těmito algoritmy a pravděpodobný důvod pro změnu je popsán v části 2.1), ale menší změny se týkají i samotných metadat a hlavičky.

2.5.1 Hlavička

Samotná BitLocker hlavička se změnila jen minimálně. Zajímavé je, že starší verze obsahuje „odkaz“ pouze na první kopii FVE metadat a to přesto, že i tato verze obsahuje tři kopie. Offsety ostatních kopií je tak třeba vyčíst ze samotných metadat. Díky tomu se všechna metadata specifická pro BitLocker „vešla“ do nevyužitých oblastí v NTFS hlavičce (od které je BitLocker hlavička odvozen, viz 2.2.1) a nezasahují tak do boot kódu. Na druhou stranu v případě poškození první kopie FVE metadat bude složitější najít na disku další dvě „záložní“ kopie.

2.5.2 FVE metadata

FVE metadata se u starší verze liší pouze v drobných detailech. Výhodou FVE metadat je, že jsou verzovaná a lze tak snadno rozpoznat, u jakou variantu BitLockeru se jedná. Starší varianta má verzi 1, novější varianta má verzi 2. Důležité hodnoty (signatura, velikost, umístěné offsetů všech tří kopií FVE metadat) jsou v obou verzích stejné.

2.5.3 Klíče

Struktura klíčů VMK a FVEK se u starší verze BitLockeru nijak neliší. Jediný rozdíl představuje další klíč TWEAK, který se používá pro šifrování inicializačního vektoru. TWEAK klíč je uložen, podobně jako FVEK, zašifrovaný pomocí VMK ve FVE metadatech jako speciální záznam (viz 2.2.3).

2.5.4 Šifrovaná data

Asi největší odlišnost u starších verzí je ve způsobu uložení šifrovaných dat a to především v umístění NTFS hlavičky otevřeného zařízení. Zatímco u novější verze BitLockeru je tato hlavička zašifrovaná a uložená na speciálním místě zapsaném ve FVE metadatech (způsob umístění šifrované NTFS hlavičky je popsán v části 2.4.1), v původní variantě BitLockeru je NTFS hlavička uložena nezašifrovaná a to přímo na svém „původním“ místě na začátku disku. Vzhledem k malé odlišnosti původní hlavičky BitLockeru a hlavičky NTFS stačí při dešifrování nahradit určité části BitLocker hlavičky a výsledkem je validní NTFS hlavička pro dešifrované zařízení.

Nahradit je třeba signaturu — místo původního `-FVE-FS-` dosadíme `NTFS` (standardní signatura souborového systému NTFS) a offset první kopie FVE metadat — ten nahradí adresa prvního clusteru MFT, která je uložen ve FVE metadatech.

Poslední rozdíl v šifrovaných datech spočívá v inicializačním vektoru použitém pro jejich (de)šifrování. Stejně jako u novější verze BitLockeru se zde použije číslo sektoru, ale nikoli „prosté“, ale zašifrované pomocí TWEAK klíče.

3 Existující řešení pro práci s BitLockerem v Linuxu

Pro Linux již v současné době existují nástroje, které umí s BitLockerem více či méně pracovat. Podle aktivity vývoje a pokrytí funkcionality BitLockeru jsou nejvýznamnější dva projekty — knihovna libbde[27] a nástroj Dislocker[28].

3.1 Knihovna libbde

Knihovna libbde vytvořená Joachimem Metzem představuje asi nejlepší software pro práci s BitLockerem v linuxových systémech a nejen tam, protože podporuje i systémy Microsoft Windows a MacOS X[29]. Kromě knihovny jsou součástí projektu i nástroje pro koncové uživatele `bdemount` a `bdeinfo`. Ukázka výstupu nástroje `bdeinfo`, který slouží primárně pro analýzu existujících zařízení, je vidět na obrázku 3.1. Užitečná může být také dostupnost rozhraní pro jazyk Python (samotná knihovna je implementována v jazyce C).

```
BitLocker Drive Encryption information:
        Encryption method               : AES-XTS 128-bit
        Volume identifier               : 1f8bf933-8323-4c97-8a89-a67625ac8f40
        Creation time                   : Feb 03, 2019 09:10:22.265405900 UTC
        Description                     : DESKTOP-NPM7RCA G: 2/3/2019
        Number of key protectors        : 2

Key protector 0:
        Identifier                      : f0f61678-fb6f-4ab1-934a-7094f5b68a85
        Type                            : Password

Key protector 1:
        Identifier                      : 012e56c1-4ed6-4527-8abf-7a9f29e0b521
        Type                            : Recovery password
```

Obr. 3.1 Ukázka výstupu nástroje `bdeinfo`

Podpora BitLockeru, kterou libbde poskytuje, je velice rozsáhlá a dokáže pracovat se všemi existujícími formáty a verzemi. Dokumentace pro tuto knihovnu také obsahuje obsáhlý popis BitLockeru, formátu hlaviček a metadat[15], který byl neocenitelný při přípravě této diplomové práce.

Bohužel i přes tyto rozsáhlé možnosti má knihovna libbde několik vlastností, které z ní dělají nevhodného kandidáta na nástroj pro každodenní použití. Předně je zde problém s neexistující podporou pro zápis — otevřené zařízení je připojitelné pouze pro čtení a ačkoli je podpora pro zápis plánována již od roku 2014[1], stále není k dispozici. Kvůli tomu se může jednat o nástroj vhodný pro forenzní analýzu nebo záchranu dat,

TODO: citaci nebo stačí takhle?

[1] libbde Issue #1: Add write support — https://github.com/libyal/libbde/issues/1

ale například pro vytvoření šifrovaného flash disku, který bude sloužit ke sdílení dat mezi Windows a Linuxem, je toto řešení nepoužitelné.

Z hlediska případného dalšího vývoje nebo použití v jiných projektech, je také minimálně diskutabilní použití některých technologií. Knihovna libbde je součástí většího projektu *libyal*[2)], který obsahuje několik desítek různých knihoven. Jednou z nich je i knihovna *libaes*, která poskytuje multiplatformní implementaci AES a kterou libbde používá pro dešifrování dat. Používání „vlastních“ implementací šifrování je obecně nedoporučováno a preferuje se použití standardních knihoven jako například *libopenssl* nebo *libgcrypt*, které mají teoreticky zaručit správnost implementace kryptografických algoritmů. Použitá knihovna libaes například při dešifrování klíčů kvůli špatné implementaci AES-CCM vůbec nekontroluje přiložený MAC tag[3)].

Pro potenciální uživatele může být problematická také implementace v *user space* pomocí FUSE[4)], které sice výrazně zjednodušuje vývoj, ale může mít velmi výrazný vliv na výkon — zpomalení oproti implementaci v kernelu může nastat až o 83 % a zatížení CPU může narůst až o 31 %[31].

Nevýhodou FUSE je také, že vytvořené otevřené „zařízení“ ve skutečnosti není systémový nástroji rozpoznané jako blokové zařízení a to právě proto, že bylo vytvořeno v user space. Takto vytvořené zařízení sice lze připojit pomocí příkazu `mount`, kdy je na něm úspěšně rozpoznám souborový systém NTFS a jako takový je úspěšně připojen, ale protože jej systém nerozpozná jako nově přidané zařízení, není možné jej detekovat a připojit automaticky.

Pro případnou integraci do existujících nástrojů pro práci s úložnými zařízeními, může být teoreticky problém také licence — libbde je dostupná pod licencí GNU LGPL verze 3, která je zpětně nekompatibilní s verzí 2 a použití takové knihovny by (i u nástrojů a knihoven a dostupných pod licencí GNU LGPL verze 2 a novější) automaticky změnilo licenci výsledného programu na GNU LGPL verze pouze 3[32].

Obecně lze říci, že knihovna libbde a s ní dostupné uživatelské nástroje jsou velmi užitečné pro případnou záchranu dat ze zařízení zašifrovaného pomocí BitLockeru, při nemožnosti použít Windows, případně pro různou analýzu dat, ale bohužel nevhodné pro každodenní použití. Volba použitých technologií (FUSE, vlastní implementace kryptografických funkcí) z libbde také dělá nevhodného kandidáta na další rozšíření a případné začlenění do existujících aplikací a nástrojů.

[2)] Overview of the libyal projects — https://github.com/libyal/libyal/wiki/Overview

[3)] libaes Issue #2: AES CCM implementation seems to different / non-compliant than others — https://github.com/libyal/libcaes/issues/2

[4)] FUSE je interface pro práci se souborovými systémy v user space, bez potřeby programovat přímo v kernel space, a je tedy dostupný i pro neprivilegované uživatele[30].

3.2 Dislocker

Druhým projektem, který se zabývá podporou BitLockeru v linuxových systémech, je nástroj Dislocker. Velkou výhodou tohoto nástroje je, že (na rozdíl od knihovny libbde) podporuje i zápis na otevřené BitLocker zařízení. Kromě Linuxu podporuje také MacOS X a BSD systémy. Součástí Dislockeru jsou kromě samotného nástroje `dislocker` i nástroj pro analýzu metadat `dislocker-metadata` (ukázka z jeho výstupu je na obrázku 3.2, nástroj `dislocker-file` sloužící pro dešifrování celého zařízení a uložení dat do souboru a také nástroj `dislocker-find` pro nalezení blokových zařízení s BitLocker formátem.

```
====================[ BitLocker information structure ]====================
  Signature: '-FVE-FS-'
  Total Size: 0x0370 (880) bytes (including signature and data)
  Version: 2
  Current state: ENCRYPTED (4)
  Next state: ENCRYPTED (4)
  Encrypted volume size: 104857600 bytes (0x6400000), ~100 MB
  Size of convertion region: 0 (0)
  Number of boot sectors backuped: 16 sectors (0x10)
  First metadata header offset:  0x2195000
  Second metadata header offset: 0x2c1d000
  Third metadata header offset:  0x373a000
  Boot sectors backup address:   0x21a5000
  ----------------------------{ Dataset header }----------------------------
    Dataset size: 0x00000324 (804) bytes (including data)
    Unknown data: 0x00000001 (always 0x00000001)
    Dataset header size: 0x00000030 (always 0x00000030)
    Dataset copy size: 0x00000324 (804) bytes
    Dataset GUID: '1F8BF933-8323-4C97-8A89-A67625AC8F40'
    Next counter: 10
    Encryption Type: AES-XTS-128 (0x8004)
    Epoch Timestamp: 1549185022 sec, that to say Sun Feb  3 09:10:22 2019
  ------------------------------------------------------------------------
```

Obr. 3.2 Ukázka výstupu nástroje `dislocker-metadata`

Nevýhody Dislockeru jsou pak podobné jako u výše zmíněné knihovny libbde — implementace využívá FUSE a součástí nástroje je také vlastní implementace AES. Jedná se také pouze o nástroj pro koncové uživatele, nikoli o knihovnu a případná integrace se systémovými nástroji by tak byla složitější. Hlavní nevýhodou je však (pravděpodobně) ukončený vývoj tohoto nástroje — poslední commit v Git repozitáři je z roku 2017.

Obecně je nástroj Dislocker pro koncové uživatele vhodnější, než knihovna libbde a s ní spojené nástroje a to především proto, že umožňuje na BitLocker zařízení také

zapisovat. Sdílí ovšem stejné problémy, které z něj nedělají ideální řešení pro další použití či případné rozšíření nebo začlenění do existujících aplikací a nástrojů.

II. PROJEKTOVÁ ČÁST

4 Nadpis

Hlavním cílem této práce je navrhnout a případně implementovat řešení pro práci s BitLocker zařízeními v linuxovém prostředí, které bude jednoduché na použití pro běžné uživatele a které vyřeší problémy stávajících řešení, jako je používání nestandardních kryptografických knihoven nebo používání technologie FUSE, která má negativní dopad jak na výkon, tak na uživatelskou přívětivost.

Splnění výše uvedených požadavků lze rozdělit do tří částí:

1. podpora pro práci pro přímou práci se šifrovanými daty, tedy jejich dešifrování při čtení a šifrování při zápisu,
2. podpora pro práci s metadaty BitLockeru, tedy identifikaci BitLocker zařízení a získání důležitých informací pro práci s BitLockerem, jako je použitý šifrovací algoritmus, způsob uložení dat na disku a především klíče pro (de)šifrování dat a
3. integrace s existujícími systémový nástroji a démony pro práci s úložnými zařízeními tak, aby z pohledu uživatele fungovala práce s BitLocker zařízeními stejně, jako práce se zařízeními šifrovanými pomocí nativní technologie LUKS/dm-crypt.

První část lze vyřešit pomocí existujícího kernelového ovladače Device Mapper. Přímá nízkoúrovňová práce s blokovými zařízeními je tak vyřešena přímo v kernelu a pro (de)šifrování dat bude využito kryptografických funkcí z kernel crypto API. Pro podporu nejnovější verze BitLockeru s šifrováním AES-XTS dokonce není potřeba Device Mapper nijak upravovat. Podrobněji je toto řešení probráno v kapitole 5.

Pro druhou část byl v rámci této práce naimplementován nový nástroj, který z BitLocker hlavičky a FVE metadat zjistí strukturu daného zařízení a se znalostí hesla získá FVEK a připraví správnou konfiguraci pro Device Mapper pro vytvoření otevřeného zařízení. Tento nový nástroj je podrobněji popsán v části 6.

Aby byla zajištěna podpora BitLockeru v grafických rozhraních, je třeba integrovat vytvořený nástroj do existujících nástrojů pro práci s úložnými zařízeními, především do démona UDisks, který zajistí, že BitLocker zařízení bude rozpoznáno jako podporované šifrované zařízení a uživateli nabídnuto jeho odemčení. Změny potřebné v těchto nástrojích jsou popsány v části 7.

5 Device Mapper

Device Mapper je kernelový ovladač, který poskytuje rozhraní pro vytváření nových blokových zařízení pomocí jejich mapování na zařízení existující. Umožňuje tak například vytvořit z jednoho disku několik menších blokových zařízení, nebo naopak spojit více disků do jednoho zařízení.[33]

Device mapper umožňuje vytváření různých zařízení s různými vlastnostmi, díky různým modulům, takzvaným *targetům*. Ty poskytují jak funkcionalitu pro prosté mapování mezi dvěma či více zařízeními v podobě targetu *dm-linear*, ale existují i složitější targety, které poskytují i další funkcionalitu nad rámec tohoto mapování, jako například target *dm-raid*, který přidává logiku RAID zařízení.

Z pohledu této práce jsou však důležité dva targety — *dm-zero* a především *dm-crypt*.

5.1 dm-zero

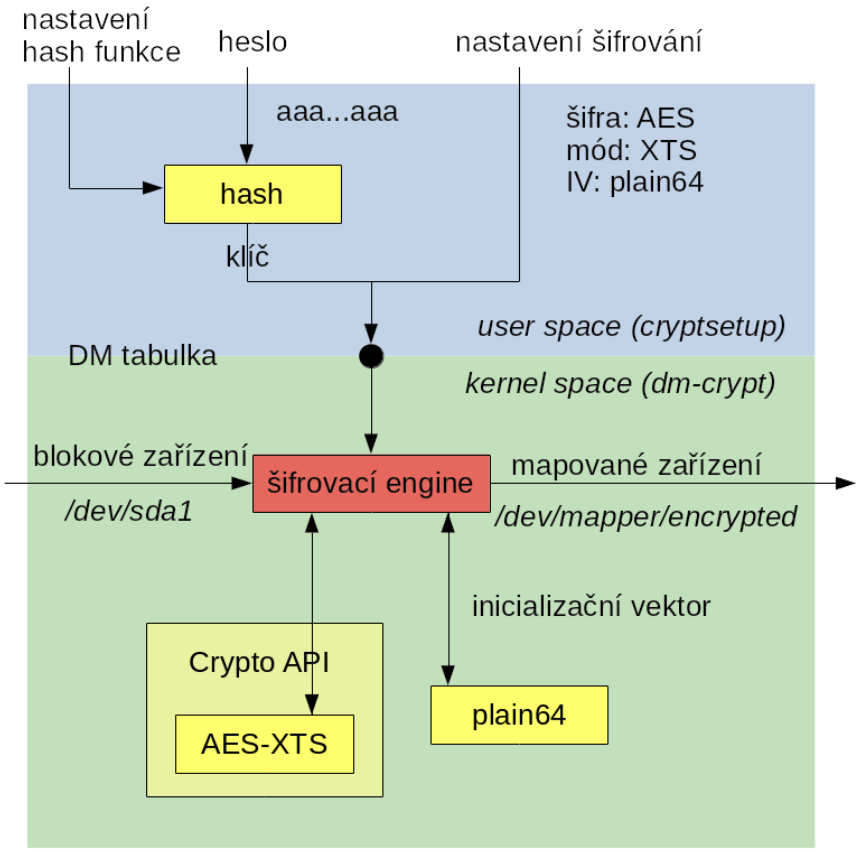
Dm-zero je velice jednoduchý target. V podstatě se jedná o blokové zařízení, které se chová jako `/dev/zero` — při čtení vrací nuly a jakékoli zápisy prostě zahazuje bez chyb.[34]

5.2 dm-crypt

Pro tuto práci nejdůležitějším targetem je dm-crypt, který slouží pro tvorbu šifrovaných zařízení. Kromě samotného mapování mezi existujícím zařízením a zařízením nově vytvořeným, navíc všechny zápisy na mapované zařízení a čtení z něj (de)šifruje.[35]

Z pohledu nástrojů pracujících s mapovaným zařízením, se tak jedná o normální nešifrované zařízení — při čtení bloku jej device mapper podle své tabulky napřed přečte ze zašifrovaného disku a dešifruje, takže uživatel dostane již otevřený text. V případě zápisu je pak situace opačná — uživatel zapisuje otevřená data, která Device Mapper nejprve zašifruje a teprve poté skutečně zapíše na disk. Zjednodušené schéma fungování crypt targetu je na obrázku 5.1.

Crypt target se používá primárně pro šifrování disku v linuxových systémech společně s technologií LUKS. LUKS se stará o správu klíčů a definuje také diskový formát pro ukládání metadat. Podobně jako u BitLockeru je i u LUKSu klíč pro (de)šifrování dat uložen přímo na disku v metadatech chráněný heslem, případně dalšími způsoby. Crypt target má tedy v tomto případě na starosti ukládání dat a jejich dešifrování při čtení a šifrování při zápisu a LUKS slouží pro zjednodušení práce se šifrovaným zařízením, aby si uživatel nemusel pamatovat klíč a další informace nutné pro správné nastavení šifrování.[36]



Obr. 5.1 Fungování dm-crypt targetu z pohledu user space a kernel space.[36]

Jak může vypadat Device Mapper tabulka jednoduchého šifrovaného zařízení vytvořeného pomocí technologie LUKS/dm-crypt je vidět v tabulce 5.1.

Tab. 5.1 Mapa dm-crypt zařízení v Linuxu

jméno	start	velikost	target	šifra	klíč	IV offset	zařízení	offset
luks	0	172032	crypt	aes-xts-plain64	f5..d0	0	8:3	32768

Zde se jedná o jednoduché šifrované zařízení, které zabírá celé zařízení zašifrované pomocí šifry AES-XTS, kdy inicializačním vektorem je číslo sektoru. Výsledné mapované zařízení je o 16 MiB, které zabírá LUKS hlavička, menší (to udává poslední položka *offset* uváděná v sektorech).

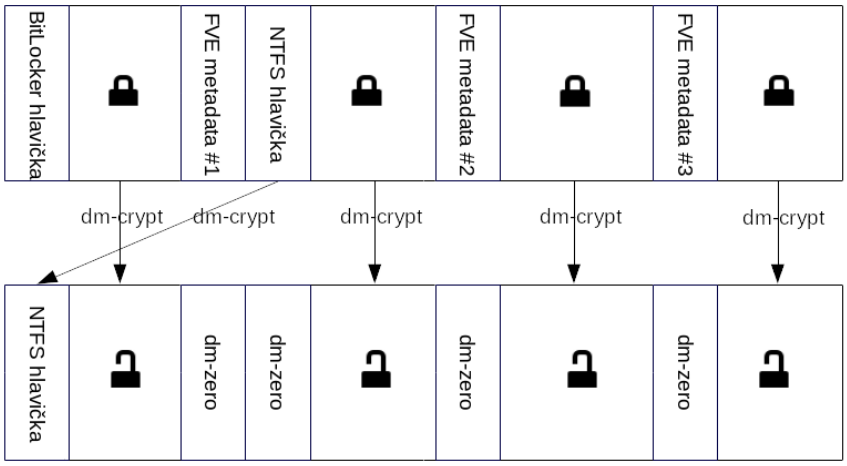
Crypto target podporuje všechny šifry, které jsou podporovány v linuxovém kernel crypto API a širokou škálu definic inicializačního vektoru a teoreticky tak umožňuje podporovat různé technologie šifrování disku — linuxová implementace technologie TrueCrypt/VeraCrypt je také postavena nad Device Mapperem a crypto targetem[37].

Nekecám náhodou?

5.3 Device Mapper a BitLocker

Z výše uvedeného popisu device mapperu, crypto targetu a stručného představení technologie LUKS, vyplývá, že BitLocker a LUKS/dm-crypt jsou velice podobné technologie. Obě mají na disku metadata v otevřené podobě, která obsahují zašifrovaný klíč použitý k šifrování dat uložených na disku. Ve výchozím nastavení dokonce BitLocker i LUKS/dm-crypt používají stejnou šifrovací funkci — AES-XTS a stejný inicializační vektor — číslo sektoru. Jsou zde rozdíly ve způsobu uložení dat a metadat na disku a práci s klíči, ale z pohledu Device Mapperu stačí znát (de)šifrovací klíč pro data a mapu šifrovaných dat na disku a může s BitLockerem pracovat stejně jako s jakýmikoli jinými šifrovanými daty.

Způsob, jak jsou v BitLocker metadatech uloženy klíče je popsán v části 2.3. Pro použití s Device Mapperem tedy stačí pouze získat z FVE metadat FVEK a použít ho pro vytvoření dm-crypt zařízení.



Obr. 5.2 Schéma použitých DM targetů při odemykání BitLocker zařízení

Jak bylo popsáno v části 2.4.1 a naznačeno na obrázku 2.10, struktura uložení dat na BitLocker zařízení je relativně složitá a dešifrované zařízení má některá data, jako například NTFS hlavičku, přesunutá na jiná místa. I toto lze pomocí Device Mapperu vyřešit jednoduše — mapování různých sektorů původního (šifrovaného) zařízení na zařízení nové je jednou ze základních funkcí, kterou Device Mapper nabízí, a tato funkcionalita je dostupná pro všechny targety, tedy i pro crypto target.

Stačí tedy vědět, které sektory šifrovaného zařízení je třeba namapovat na které sektory otevřeného zařízení. Metadatové části, které jsou v otevřeném zařízení v BitLockeru nahrazeny nulami, mohou být stejně snadno nahrazeny pomocí Device Mapper targetu zero, který je mimo jiné také ochrání proti náhodnému přepsání, protože zápisy

na dm-zero zařízení jsou zahazovány[34].

Příklad, jak může fungovat mapování mezi šifrovaným BitLocker zařízením a otevřeným zařízením v linuxovém prostředí, je vidět na obrázku 5.2.

Kompletní Device Mapper tabulka pro testovací 100 MiB zařízení je uvedena v tabulce 5.2.

Tab. 5.2 DM tabulka odemčeného BitLocker zařízení v Linuxu

start	**velikost**	**target**	**šifra**	**klíč**	**IV offset**	**zařízení**	**offset**
0	16	crypt	aes-xts-plain64	a4..52	68904	7:0	68904
16	68760	crypt	aes-xts-plain64	a4..52	16	7:0	16
68776	128	zero					
68904	16	zero					
68920	21424	crypt	aes-xts-plain64	a4..52	68920	7:0	68920
90344	128	zero					
90472	22632	crypt	aes-xts-plain64	a4..52	90472	7:0	90472
113104	128	zero					
113232	91568	crypt	aes-xts-plain64	a4..52	113232	7:0	113232

Můžeme zde vidět strukturu otevřeného zařízení, kdy prvních 16 sektorů zabírá crypt target, který mapuje data původně uložená na offsetu 68904 sektorů — to je NTFS hlavička, která je v původním BitLocker zařízení uložena za prvními metadaty — tomu odpovídá i nastavení IV offsetu, protože inicializační vektor v BitLockeru je číslo sektoru, na kterém jsou data skutečně uložena, nikoli číslo sektoru, na kterém se data nachází na otevřeném zařízení. Následuje první část dat, opět mapovaná pomocí crypt targetu. Za ní jsou dvě dm-zero zařízení, která nahrazují první kopii FVE metadat a NTFS hlavičku, přesunutou na začátek zařízení. Zbývají již jen dvě kopie FVE metadat, opět nahrazené pomocí dm-zero, a tři datové oblasti mapované pomocí dm-crypt.

6 Nadpis

Z výše popsaného Device Mapper targetu dm-crypt a jeho možností vyplývá, že pro správnou podporu BitLockeru v linuxovém prostředí stačí dodat mu klíč, šifru a správné umístění šifrovaných dat a výsledkem bude blokové zařízení, které lze používat pro čtení i pro zápis stejně, jako jakékoli jiné nešifrované blokové zařízení.

Protože žádné z existujících řešení neumožňuje tyto informace získat, bylo rozhodnuto vytvořit nový jednoduchý nástroj, který z BitLocker hlavičky a FVE metadat všechny potřebné informace získá a pomocí nástroje `dmsetup` vytvoří potřebné Device Mapper zařízení.

6.1 Implementace

Jako programovací jazyk pro implementaci byl zvolen Python, který umožňuje rychlý vývoj a díky široké škále vestavěných i externích knihoven má dostatek funkcí pro práci s blokovými zařízení i šifrováním. Menší nevýhodou této volby je nemožnost použít výsledný kód jako knihovnu pro programy napsané v jazyce C (respektive v jiném jazyce než Python), ale jako vhodné API v tomto případě může posloužit i volání externí utility, pokud má k dispozici vhodné módy a přepínače, které umožní přijímat heslo poslané na standardní vstup a omezí interakci s uživatelem. Systémové nástroje, které budeme dále používat, navíc s takovým použitím konzolových nástrojů používají a knihovna libblockdev (viz 7.1.2) umí s takovými nástroji bez problémů pracovat.

Vzhledem k tomu, že celá práce s metadaty je uceleným ohraničeným krokem, kdy po nastavení Device Mapperu již není potřeba další interakce s blokovým zařízením (jelikož se o vše stará kernel), dává částečně smysl celou tuto záležitost pojmout jako samostatný proces, který z disku načte potřebné informace, vytvoří Device Mapper zařízení a ukončí se.

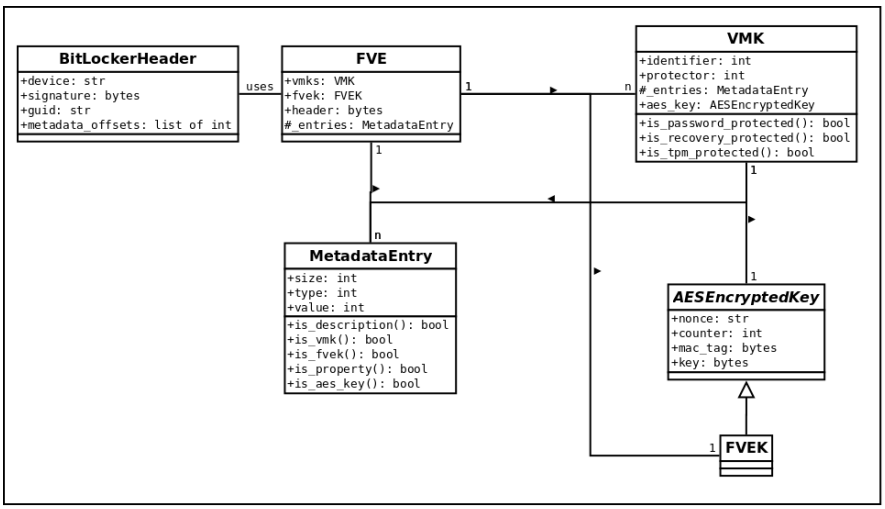
Součástí nástroje jsou následné moduly, které lze využívat i samostatně, případně z jiných Python projektů:

- **constants** obsahující různé konstanty vztahující se k BitLocker metadatům (velikosti hlaviček, offsety...),
- **dm** obsahující pomocné funkce pro práci s Device Mapper zařízeními,
- **entry** obsahující třídu `MetadataEntry` a pokrývající funkcionalitu potřebnou pro práci s metadatovými FVE záznamy (2.2.3),
- **fve** obsahující třídu `FVE`, která využívá ostatní zmíněné moduly pro kompletní parsování FVE metadat,

- **header** obsahující třídu `BitLockerHeader` sloužící pro práci s BitLocker hlavičkou a extrakci potřebných dat z ní,
- **keys** obsahující třídy `AESEncryptedKey`, `FVEK`, `UnencryptedKey`, `VMK` a sloužící pro práci s různými klíči nacházejícími se ve FVE metadatech včetně jejich dešifrování a
- **utils** obsahující různé pomocné funkce a nástroje pro práci blokovými zařízeními a pro dekódování přečtených dat.

Rozčlenění do jednotlivých tříd s definovaným rozhraním umožňuje snadné budoucí rozšíření pro starší verze BitLocker metadat, které jsou často jen mírně odlišné (více informací o rozdílech ve starších verzích BitLockeru v části 2.5) a bude tedy možné při jejich implementaci znovu použít většinu kódu.

Na obrázku 6.1 je k dispozici diagram tříd komponent implementovaného nástroje bitlockersetup. V přiložených zdrojových kódech je také k dispozici kompletní automaticky generovaná API dokumentace.



Obr. 6.1 Diagram tříd vytvořeného nástroje bitlockersetup

6.2 Uživatelské rozhraní

Implementovaný nástroj, pojmenovaný `bitlockersetup`, má několik podpříkazů, které umožňují volit mezi základními operacemi. Jsou to především:

- **open** sloužící pro odemčení BitLocker zařízení,
- **close** sloužící pro jeho opětovné uzamčení a tedy odstraněný vytvořeného Device Mapper zařízení,

- **dump** který vypíše veškeré informace o vybraném zařízení a
- **uuid** který vypíše UUID vybraného BitLocker zařízení.

Kompletní manuálová stránka pro bitlockersetup je dostupná v příloze . Ukázka odemčení BitLocker zařízení pomocí bitlockersetup je na obrázku 6.2.

TODO: link na přílohu

```
$ sudo bitlockersetup open /dev/sdb2 bitlocker
Password for '/dev/sdb2':
Created device mapper device '/dev/mapper/bitlocker'.
```

Obr. 6.2 Ukázka použití nástroje `bitlockersetup` pro odemčení BitLocker zařízení

```
$ sudo bitlockersetup dump /dev/sdb2
Encryption method:      AES-XTS 128-bit encryption
Volume identifier:      1f8bf933-8323-4c97-8a89-a67625ac8f40
Creation time:          2019-02-03 09:10:22.265406
Description:            DESKTOP-NPM7RCA G: 2/3/2019
VMK
        Identifier:     f0f61678-fb6f-4ab1-934a-7094f5b68a85
        Type:           VMK protected with password
        Salt:           03 d1 b4 23 6b f4 5b df e4 bd dd f4 83 ec 47 ee
        AES-CCM encrypted key
                Nonce data:     2019-02-03 09:10:36.052000
                Nonce counter:  3
                Key:     0d a8 61 01 72 46 9b 7b 34 40 0c ... 1d 21 0a b7 b8 c4
VMK
        Identifier:     012e56c1-4ed6-4527-8abf-7a9f29e0b521
        Type:           VMK protected with recovery password
        Salt:           46 ee b7 10 0e 43 4d d4 a5 ab eb c6 21 f4 f1 84
        AES-CCM encrypted key
                Nonce data:     2019-02-03 09:10:36.052000
                Nonce counter:  6
                Key:     91 a0 ad e9 0c 08 e8 1e 3d 2f 7d 3b ... 36 46 40 6b e7
FVEK
        AES-CCM encrypted key
                Nonce data:     2019-02-03 09:10:36.052000
                Nonce counter:  8
                Key:     13 08 8b fd e9 12 5a a6 7d 7b 53 73 ... 94 df c8 49 66
```

Obr. 6.3 Ukázka použití nástroje `bitlockersetup` pro zobrazení informací o BitLocker zařízení

6.3 Instalace

Pro snadnou instalaci nástroje bitlockersetup je k dispozici standardní `Makefile` skript, instalaci lze tedy provést ze složky `bitlockersetup` pomocí příkazu `make install`. K dispozici je také SPEC soubor pro tvorbu RPM balíčků pro distribuci Fedora, balíčky pro Fedoru 30 jsou součástí přiložených zdrojových kódů.

Podporována je také instalace pomocí standardních metod pro Python balíčky pomocí skriptu `setup.py`, případně pomocí nástroje `pip`.

7 Integrace se systémovými nástroji

Samotný nástroj pro práci s BitLockerem, vytvořený v rámci této práce, není dostatečným řešením pro kompletní podporu BitLocker zařízení v Linuxu. Cílem je, aby uživatel nepoznal, že nepracuje s nativním zařízením — tedy, aby po připojení přenosného disku zašifrovaného pomocí technologie BitLocker, byl tento automaticky rozpoznán a uživateli bylo v grafickém rozhraní nabídnuto jeho odemčení, bez nutnosti další interakce ze strany uživatele. Tímto se zabývá druhá polovina praktické části této práce, kdy bude vytvořený nástroj integrován do existujících nástrojů a knihoven pro práci s úložnými zařízeními.

7.1 UDisks

I když je často kritizována roztříštěnost linuxových systémů, která se projevuje také velkým množstvím různých grafických pracovních prostředí, pro práci s úložnými nástroji toto naštěstí neplatí a nejoblíbenější pracovní prostředí jako GNOME, KDE a Xfce používají jednotné API poskytované nástrojem UDisks, byť obvykle nepřímo, jako například v případě prostředí GNOME, které nad UDisks používá ještě další vrstvu v podobě nástroje GVfs[38].

Díky výše popsanému stačí, aby UDisks správně označil vybrané BitLocker zařízení jako šifrované a umožnil jeho odemčení, a bez dalších úprav pak nástroje, které UDisks používají, nabídnou uživateli jeho odemčení.

UDisks není jediným nástrojem, ale spíše projektem, který poskytuje hned několik různých možností pro práci s úložnými zařízeními — démona *udisksd*, knihovnu *libudisks* a konzolový nástroj *udisksctl*[39]. Pro uživatele API, které UDisks nabízí, je nejdůležitější, že poskytuje DBus API[1)], kdy na interface *org.freedesktop.UDisks2* vystavuje všechna existující bloková zařízení a umožňuje snadný přístup k jejich vlastnostem a přes nabízené funkce také jejich úpravy a správu.

7.1.1 Identifikace BitLockeru

První věcí, kterou je v případě implementace podpory BitLockeru v UDisks vyřešit, je schopnost identifikace BitLocker zařízení. Jak již bylo řečeno, pro rychlou identifikaci BitLocker zařízení slouží jeho hlavička a především signatura `-FVE-FS-` (o BitLocker hlavičce více v části 2.2.1). UDisks ale sám jednotlivá zařízení neskenuje a jejich hlavičky a signatury nečte, na to používá existující systémovou databázi blokových zařízení

[1)] DBus je softwarová sběrnice, která umožňuje komunikaci mezi jednotlivými aplikacemi a démony (IPC) a také spouštění funkcí z API, které poskytují systémový démoni (RPC). V grafických prostředích jako GNOME a KDE se používají právě pro komunikaci se systémovými démony jako je UDisks a další.[40]

vytvořenou démonem UDev.

UDev slouží primárně pro zprostředkování událostí o přidání, odstranění a změnách blokových zařízení z kernelu do user space. Vytváří také speciální soubory a symbolické odkazy pro přímý přístup k blokovým zařízení v adresáři `/dev`. Důležitou součástí UDevu jsou pravidla, která určují, co má UDev s daným zařízením v jakém případě dělat — umožňují například jejich inicializaci, nahrání potřebných ovladačů nebo prosté spuštění programů a démonů, které dané zařízení mají ovládat.[41] Kromě toho také s využitím různých systémových nástrojů zjistí o nově připojeném zařízení všechny potřebné informace, tedy také přečte jeho signaturu z hlavičky. K tomu využívá nástroje `blkid` z projektu `util-linux`.

Do knihovny libblkid (a tedy i do nástroje blkid) byla podpora pro detekci BitLocker signatury přidána ve verzi 2.33 vydané v listopadu 2018[42]. Díky tomu UDev správně identifikuje BitLocker zařízení a v jeho databázi jsou tato zařízení vedena jako *crypto* zařízení typu *BitLocker*, jak je vidět na obrázku 7.1.

```
E: DEVNAME=/dev/sda2
E: DEVTYPE=partition
E: SUBSYSTEM=block
E: ID_BUS=usb
E: ID_FS_TYPE=BitLocker
E: ID_FS_USAGE=crypto
```

Obr. 7.1 Vybrané informace z UDev databáze o připojeném BitLocker zařízení

Udev (a díky němu i UDisks) tak má vše potřebné pro identifikaci nově připojeného zařízení jako BitLocker zařízení. UDisks tuto identifikaci z UDevu převezme a správně nastaví hodnoty property `IdType` a `IdUsage`, jak je vidět na obrázku 7.2.

V UDisks však stále chybí některé informace o BitLocker zařízení jako například UUID, které UDev nezná, protože jeho získání z BitLocker hlavičky knihovna libblkid nepodporuje.

UDisks také pro toto zařízení nevytvoří interface *org.freedesktop.UDisks2.Encrypted*, které normálně poskytuje další funkce pro práci se šifrovanými zařízeními. To je logické, protože i když je zařízení identifikováno jako šifrované, UDisks s ním neumí nijak pracovat. O přidání této funkcionality a zmíněného interface pro BitLocker zařízení pojednává část 7.1.3.

```
/org/freedesktop/UDisks2/block_devices/sda2:
  org.freedesktop.UDisks2.Block:
    Configuration:              []
    CryptoBackingDevice:        '/'
    Device:                     /dev/sda2
    DeviceNumber:               2050
    HintAuto:                   true
    HintIconName:
    HintIgnore:                 false
    HintName:
    HintPartitionable:          true
    HintSymbolicIconName:
    HintSystem:                 false
    Id:
    IdLabel:
    IdType:                     BitLocker
    IdUUID:
    IdUsage:                    crypto
```

Obr. 7.2 Informace o BitLocker zařízení z pohledu UDisks

7.1.2 Knihovna libblockdev

UDisks nepracuje s blokovými zařízeními na přímo, ale využívá buď Udev databázi (pro zjištění informací o existujících zařízeních, viz část o identifikaci BitLocker zařízení 7.1.1), nebo knihovnu libblockdev (pro samotnou manipulaci se zařízeními)[**?**]. Jakákoli nová vlastnost, která má být přidána do UDisks, musí tedy být napřed přidána do libblockdev.

Libblockdev je relativně jednoduchá knihovna napsaná v jazyce C, která poskytuje jednotné API pro různé technologie pro práci s úložnými zařízeními jako je LVM, Btrfs nebo LUKS. Je také modulární a poskytované funkce jsou rozděleny do pluginů, které je možné používat samostatně. Kromě jazyka C nabízí také rozhraní pro Python díky využití technologie GObject introspection.[43]

Pro implementace nových funkcí pro práci s BitLockerem byl zvolen *crypto* plugin této knihovny, který již obsahuje funkce pro práci s šifrovacími technologiemi LUKS/dm-crypt a TrueCrypt/VeraCrypt. Součástí implementace jsou celkem čtyři nové funkce pro práci s BitLocker zařízeními. Jejich pojmenování a navrhované API vychází z obdobných funkcí pro technologii LUKS/dm-crypt. Deklarace nových funkcí jsou k dispozici na obrázku 7.3.

Nově implementované funkce slouží k odemčení a opětovnému uzamčení (`open/close`) BitLocker zařízení, zjištění zda se skutečně jedná o BitLocker zařízení (`is_bitlocker`)

```
gboolean bd_crypto_bitlocker_open (const gchar *device,
                                   const gchar *name,
                                   const gchar *passphrase,
                                   GError **error);
gboolean bd_crypto_bitlocker_close (const gchar *bitlocker_device,
                                    GError **error);
gboolean bd_crypto_device_is_bitlocker (const gchar *device,
                                        GError **error);
gchar* bd_crypto_bitlocker_uuid (const gchar *device,
                                 GError **error);
```

Obr. 7.3 Interface nově implemetovaných funkcí pro práci s BitLockerem v knihovně libblockdev

a zjištění UUID[2)] BitLocker zařízení (`uuid`). API nových funkcí se drží standardu daném ostatními funkcemi v libblockdev, kdy funkce vrací boolean hodnotu určující, zda bylo její volání úspěšné, či nikoli (s výjimkou funkcí, které vrací nějakou konkrétní hodnotu, jako zde UUID, v takové případě vrací buď dané UUID jako textový řetězec, nebo `NULL` v případě neúspěchu) a v případě chyby nastavují chybu do uživatelem předané výstupní proměnné `error`.

Díky tomu, že mnohé technologie pro práci s úložnými zařízeními neposkytují API v podobě knihoven, má libblockdev propracovaný systém pomocných funkcí pro volání konzolových nástrojů a samotná implementace s použitím vytvořeného nástroje pro práci s BitLockerem byla relativně jednoduchá.

Patche pro knihovnu libblockdev implementující výše zmíněné funkce jsou součástí k této práci přiložených zdrojových kódů.

7.1.3 Implementace v UDisks

V samotném UDisks je nejprve třeba vyřešit identifikaci BitLocker zařízení jako podporovaného šifrovaného zařízení, jak je popsáno v části 7.1.1. K tomu slouží nově přidaná funkce `udisks_linux_block_is_bitlocker`, která využívá informace z UDev databáze pro identifikaci blokového zařízení jako BitLocker. S pomocí této funkce pak stačí UDisks již jen upravit tak, aby BitLocker zařízení přidal DBus interface *org.freedesktop.org.UDisks2.Encrypted* a zpřístupnil tak obecné property a funkce pro šifrovaná zařízení. Které bude třeba následně upravit tak, aby správně podporovaly i BitLocker.

Díky tomu, že UDisks podporuje kromě technologie LUKS/dm-crypt také zařízení šifrovaná pomocí TrueCrypt/VeraCrypt, je přidání další technologie relativně jednoduché

[2)] V terminologii BitLockeru spíše GUID, ale zde se držíme linuxové terminologie, kde se tento univerzální identifikátor nazývá obvykle právě UUID.

— kód je již připraven na podporu více různých technologií, kdy jsou k dispozici obecné implementace funkcí sloužící k obsluze API volání pro odemykání a uzamykání šifrovaných zařízení a pro přidání podpory pro nové technologie stačí vytvořit pomocné funkce s minimálními úpravami ve veřejných funkcích.

Pro BitLocker takto byly vytvořeny pomocné funkce `bitlocker_open_job_func` a `bitlocker_close_job_func`, které jsou volány z obecných funkcí `handle_unlock` a `handle_lock`, které obsluhují API volání veřejných funkcí *Unlock* a *Lock*. Jejich API je definováno velmi obecně (povinným parametrem funkce *Unlock* je pouze heslo, ostatní jsou předávány jako nepovinný seznam klíč-hodnota[45]), takže pro BitLocker není potřeba jej nijak měnit. Ostatní funkce veřejného API (*ChangePassphrase* a *Resize*), které nejsou pro BitLocker podporovány byly upraveny tak, aby při zavolání vracely chybovou hlášku informující o tom, že tato funkcionalita není pro BitLocker podporována, protože u existujícího DBus interface nelze funkce dynamicky odebírat.

Díky tomu, že všechny součásti UDisks — tedy DBus démon, knihovna i konzolový nástroj — jsou generovány ze stejného kódu, není potřeba upravovat žádné další části kódu, kromě výše zmíněných, a podpora pro odemykání BitLocker zařízení bude automaticky propagována i do nástroje `udisksctl`.

Na obrázku 7.4 je viditelné BitLocker zařízení z pohledu UDisks po aplikaci výše zmíněných změn. Nově je přidáno *org.freedesktop.org.UDisks2.Encrypted* interface poskytující další informace o zařízení (funkce zde nejsou viditelné, jedná se pouze o informační výpis z nástroje `udisksctl`).

Patche pro nástroj UDisks implementující výše zmíněné funkce jsou součástí k této práci přiložených zdrojových kódů.

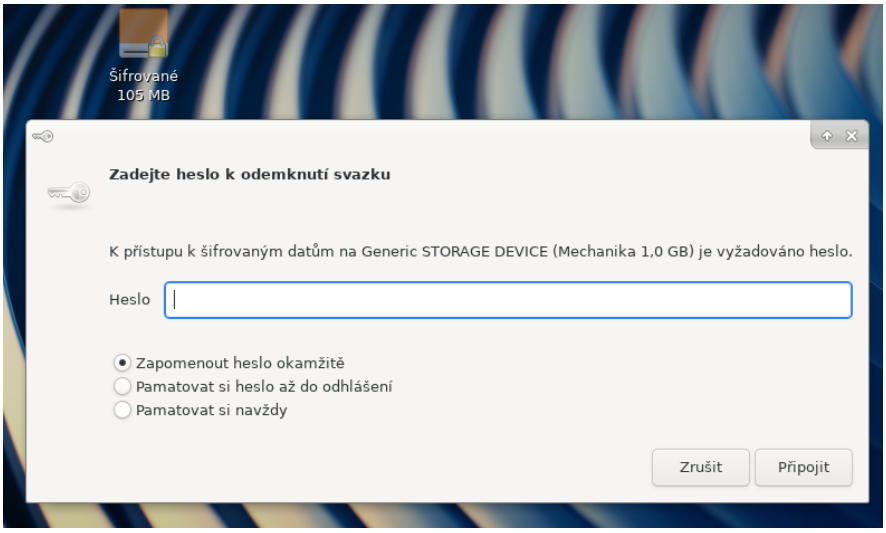
7.1.4 Podpora v grafických prostředích

Jak již bylo zmíněno výše (7.1), jednotlivá grafická rozhraní spoléhají při identifikaci šifrovaných zařízení na UDisks — pokud má dané zařízení typ *crypto* a má k dispozici interface *org.freedesktop.org.UDisks2.Encrypted*, označí jej jako šifrované a nabídnou jeho odemčení. Po instalaci UDisks s aplikovanými změnami zmíněnými výše jsou tak BitLocker zařízení viditelná v grafickém prostředí jako šifrovaná a při pokusu o jejich připojení bude uživatel dotázán na heslo po jehož zadání mu bude zpřístupněno nově vytvořené otevřené zařízení.

Na obrázku 7.5 je ukázka toho chování z prostředí Xfce.

```
/org/freedesktop/UDisks2/block_devices/sda2:
  org.freedesktop.UDisks2.Block:
    Configuration:              []
...
    HintPartitionable:          true
    HintSymbolicIconName:
    HintSystem:                 false
    Id:
    IdLabel:
    IdType:                     BitLocker
    IdUUID:                     1f8bf933-8323-4c97-...
    IdUsage:                    crypto
  org.freedesktop.UDisks2.Encrypted:
    ChildConfiguration:         []
    CleartextDevice:            '/'
    HintEncryptionType:          BitLocker
    MetadataSize:               0
```

Obr. 7.4 Informace o BitLocker zařízení z pohledu UDisks s přidanou podporou pro BitLocker



Obr. 7.5 Připojení BitLocker zařízení v prostředí Xfce

ZÁVĚR

Text závěru

SEZNAM POUŽITÉ LITERATURY

[1] EHRSAM, William, Carl MEYER, John SMITH a Walter TUCHMAN. *Message verification and transmission error detection by block chaining*. United States. US4074066A. Uděleno 1978-02-14. Zapsáno 1978-02-14.

[2] KOHNO, Tadayoshi, Niels FERGUSON a Bruce SCHNEIER. *Cryptography engineering: design principles and practical applications*. Indianapolis, IN: Wiley Pub., c2010. ISBN 978-0-470-47424-2.

[3] Encryption using the Cipher Block Chaining (CBC) mode. In: *Wikimedia Commons* [online]. [cit. 2019-04-22]. Dostupné z: https://commons.wikimedia.org/wiki/File:CBC_encryption.svg

[4] REGALADO, Daniel. CBC Byte Flipping Attack-101 Approach. In: *Infosec Resources* [online]. Infosec Institute, 2013 [cit. 2019-04-22]. Dostupné z: https://resources.infosecinstitute.com/cbc-byte-flipping-attack-101-approach/

[5] FERGUSON, Niels. *AES-CBC + Elephant diffuser: A Disk Encryption Algorithm for Windows Vista*. Microsoft, 2006.

[6] ROSENDORF, Dan. The BitLocker schema with a view towards Windows 8. In: *43. konference EurOpen.CZ* [online]. Plzeň, 2013, s. 91-101 [cit. 2019-03-04]. ISBN 978-80-86583-26-6. Dostupné z: https://europen.cz/Anot/43/eo-2-13.pdf

[7] KUMAR, Nitin a Vipin KUMAR. Bitlocker and Windows Vista. In: *NVLabs* [online]. 2008 [cit. 2019-04-20]. Dostupné z: http://www.nvlabs.in/uploads/projects/nvbit/nvbit_bitlocker_white_paper.pdf

[8] DWORKIN, Morris. SP 800-38E. *Recommendation for Block Cipher Modes of Operation: The XTS-AES Mode for Confidentiality on Storage Devices*. Gaithersburg, MD, United States: National Institute of Standards and Technology, 2010.

[9] Schema of an XEX encryption. In: *Wikimedia Commons* [online]. [cit. 2019-04-23]. Dostupné z: https://commons.wikimedia.org/wiki/File:XEX_mode_encryption.svg

[10] IEEE STD 1619 -2007. *IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices*. New York, NY, United States: IEEE Computer Society, 2008.

[11] ISO/IEC 19772:2009. *Information technology – Security techniques – Authenticated encryption*. Switzerland: International Organization for Standardization, 2009.

[12] DWORKIN, Morris. SP 800-38C. *Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality*. Gaithersburg, MD, United States: National Institute of Standards and Technology, 2004. Dostupné také z: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38c.pdf

[13] HOUSLEY, R. IETF RFC 4309. *Using Advanced Encryption Standard (AES) CCM Mode with IPsec Encapsulating Security Payload (ESP)*. Herndon, VA, United States: The Internet Society, 2005. Dostupné také z: https://www.ietf.org/rfc/rfc4309.txt

[14] KALISKI, B. IETF RFC 2898. *PKCS #5: Password-Based Cryptography Specification*. 2. Bedford (Massachuttes): RSA Laboratories, 2000. Dostupné také z: https://www.ietf.org/rfc/rfc2898.txt

[15] METZ, Joachim. BitLocker Drive Encryption (BDE) format specification. In: *GitHub: Library and tools to access the BitLocker Drive Encryption (BDE) encrypted volumes* [online]. 2011 [cit. 2019-04-15]. Dostupné z: https://github.com/libyal/libbde/blob/master/documentation/BitLocker%20Drive%20Encryption%20(BDE)%20format.asciidoc

[16] AGOSTINI, Elena a Massimo BERNASCHI. *BitCracker: BitLocker meets GPUs* [online]. 2019. Dostupné z: https://arxiv.org/abs/1901.01337

[17] VeraCrypt Volume Format Specification. In: *VeraCrypt Documentation* [online]. [cit. 2019-04-15]. Dostupné z: https://www.veracrypt.fr/en/VeraCrypt%20Volume%20Format%20Specification.html

[18] CARRIER, Brian. *File system forensic analysis*. 1. London: Addison-Wesley, 2005. ISBN 978-0321268174.

[19] CASEY, Eoghan. *Handbook of digital forensics and investigation*. Boston: Academic, c2010. ISBN 978-0123742674.

[20] Programming reference for Windows API: FILETIME structure. *Microsoft Docs* [online]. [cit. 2019-04-07]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/api/minwinbase/ns-minwinbase-filetime

[21] KORNBLUM, Jesse D. Implementing BitLocker Drive Encryption for forensic analysis. *Digital Investigation*. 2009, **5**(3-4), 75-84. DOI: 10.1016/j.diin.2009.01.001. ISSN 17422876. Dostupné také z: https://linkinghub.elsevier.com/retrieve/pii/S1742287609000024

[22] LICH, Brian a Nick BROWER. Using Your PIN or Password: Changing your PIN or Password. In: *Microsoft Docs* [online]. 2016 [cit. 2019-04-18]. Dostupné z: https://docs.microsoft.com/en-us/microsoft-desktop-optimization-pack/mbam-v2/using-your-pin-or-password

[23] Security WMI Providers: BitLocker Drive Encryption Provider. *Microsoft Docs* [online]. [cit. 2019-04-09]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/secprov/bitlocker-drive-encryption-provider

[24] HALL, Justin a Liza POGGEMEYER. BitLocker recovery guide. In: *Microsoft Docs* [online]. 2019 [cit. 2019-04-15]. Dostupné z: https://docs.microsoft.com/en-us/windows/security/information-protection/bitlocker/bitlocker-recovery-guide-plan

[25] Windows Vista support has ended. In: *Windows Support* [online]. Microsoft, 2017 [cit. 2019-04-15]. Dostupné z: https://support.microsoft.com/en-us/help/22882/windows-vista-end-of-support

[26] SOSNOWSKI, Rafal. Bitlocker: AES-3 (new encryption type). In: *Microsoft TechNet* [online]. 2016 [cit. 2019-04-18]. Dostupné z: https://blogs.technet.microsoft.com/dubaisec/2016/03/04/bitlocker-aes-xts-new-encryption-type/

[27] METZ, Joachim. Library and tools to access the BitLocker Drive Encryption (BDE) encrypted volumes. In: *GitHub* [online]. 2018 [cit. 2018-11-29]. Dostupné z: https://github.com/libyal/libbde

[28] COLTEL, Romain a Hervé SCHAUER. Dislocker: FUSE driver to read/write Windows' BitLocker-ed volumes under Linux / Mac OSX. In: *GitHub* [online]. 2017 [cit. 2019-04-15]. Dostupné z: https://github.com/Aorimn/dislocker

[29] METZ, Joachim. Building. In: *Libbde Wiki* [online]. 2016 [cit. 2019-04-15]. Dostupné z: https://github.com/libyal/libbde/wiki/Building

[30] SINGH, Sumit. Develop your own filesystem with FUSE. In: *IBM Developer: Linux Development* [online]. 2014 [cit. 2019-04-15]. Dostupné z: https://developer.ibm.com/articles/l-fuse/

[31] VANGOOR, Bharath Kumar Reddy a Vasily TARASOV. To FUSE or Not to FUSE: Performance of User-Space File Systems. In: *FAST'17 Proceedings of the 15th Usenix Conference on File and Storage Technologies*. Santa Clara, CA, USA: USENIX Association Berkeley, 2017, s. 59-72. ISBN 978-1-931971-36-2.

[32] Frequently Asked Questions about the GNU Licenses: Is GPLv3 compatible with GPLv2?. In: *GNU Project* [online]. [cit. 2019-04-15]. Dostupné z: https://www.gnu.org/licenses/gpl-faq.html.en#v2v3Compatibility

[33] Logical Volume Manager Administration: APPENDIX A. The Device Mapper. In: *Red Hat Customer Portal* [online]. [cit. 2019-04-19]. Dostupné z: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/logical_volume_manager_administration/device_mapper

[34] *Linux Kernel Documentation: dm-zero* [online]. In: . [cit. 2019-04-19]. Dostupné z: https://www.kernel.org/doc/Documentation/device-mapper/zero.txt

[35] BROŽ, Milan. Dm-crypt: Linux kernel device-mapper crypto target. In: *Cryptsetup Wiki* [online]. 2018 [cit. 2019-04-19]. Dostupné z: https://gitlab.com/cryptsetup/cryptsetup/wikis/DMCrypt

[36] FRUHWIRTH, Clemens a Markus SCHUSTER. Secret Messages: Hard disk encryption with DM-Crypt, LUKS, and cryptsetup. *Linux Magazine* [online]. 2005, **2005**(61) [cit. 2018-11-30]. ISSN 1752-9050. Dostupné z: https://nnc3.com/mags/LM10/Magazine/Archive/2005/61/065-071_encrypt/article.html

[37] BROŽ, Milan. Cryptsetup 1.6.0 Release Notes. In: *The Linux Kernel Archives* [online]. 2014 [cit. 2019-04-20]. Dostupné z: https://kernel.org/pub/linux/utils/cryptsetup/v1.6/v1.6.0-ReleaseNotes

[38] GVfs. In: *GNOME Wiki* [online]. 2019 [cit. 2019-04-18]. Dostupné z: https://wiki.gnome.org/Projects/gvfs

[39] Udisks. In: *FreeDesktop* [online]. 2018 [cit. 2019-04-18]. Dostupné z: https://www.freedesktop.org/wiki/Software/udisks/

[40] PALMIERI, John. Get on D-BUS. *Red Hat Magazine* [online]. 2005, (3) [cit. 2019-04-18]. Dostupné z: http://www.redhat.com/magazine/003jan05/features/dbus/

[41] KENLON, Seth. An introduction to Udev: The Linux subsystem for managing device events. In: *Opensource.com* [online]. 2018 [cit. 2019-04-18]. Dostupné z: https://opensource.com/article/18/11/udev

[42] ŽÁK, Karel. Util-linux 2.33 Release Notes. In: *The Linux Kernel Archives* [online]. 2018 [cit. 2019-04-18]. Dostupné z: https://kernel.org/pub/linux/utils/util-linux/v2.33/v2.33-ReleaseNotes

[43] PODZIMEK, Vratislav. Libblockdev reaches the 1.0 milestone!. In: *Storage APIs* [online]. 2015 [cit. 2019-04-18]. Dostupné z: https://storageapis.wordpress.com/2015/05/21/libblockdev-reaches-the-1-0-milestone/

[44] PODZIMEK, Vratislav. UDisks to build on libblockdev!?. In: *Storage APIs* [online]. 2017 [cit. 2019-04-18]. Dostupné z: https://storageapis.wordpress.com/2017/05/22/udisks-to-build-on-libblockdev/

[45] *UDisks Reference Manual*. 2019. Dostupné také z: http://storaged.org/doc/udisks2-api/latest/

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AES	Advanced Encryption Standard
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BSD	Berkeley Software Distribution
Btrfs	B-tree file system
CBC	Cipher Block Chaining
CCM	Counter with CBC-MAC
CNG	CryptoAPI Next Generation
CPU	Central Processing Unit
CTR	Counter
DBus	Desktop Bus
DM	Device Mapper
ECB	Electronic Code Book
FS	File System
FVE	Full Volume Encryption
FVEK	Full Volume Encryption Key
FUSE	Filesystem in Userspace
GNOME	GNU Network Object Model Environment
GNU	GNU is Not Unix
GUID	Globally Unique Identifier
GVfs	GNOME Virtual file system
IPC	Inter Process Communication
IV	Inicializační vektor
KDE	K Desktop Environment
KDF	Key Derivation Function
LGPL	Lesser General Public License
LUKS	Linux Unified Key Setup
LVM	Logical Volume Management
MAC	Message Authentication Code
MFT	Master File Table
NTFS	New Technology File System
OEM	Original Equipment Manufacturer
PBKDF	Password-Based Key Derivation Function
PIN	Personal Identification Number
RAID	Redundant Array of Independent Disks
RPC	Remote Procedure Call

RPM	RPM Package Manager
SHA	Secure Hash Algorithm
TPM	Trusted Platform Module
USB	Universal Serial Bus
UTF	Unicode Transformation Format
UUID	Universally Unique Identifier
VMK	Volume Master Key
XEX	Xor-Encrypt-Xor
Xfce	XForms Common Environment
XTS	XEX-based Tweaked-codebook with Ciphertext Stealing

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PŘÍLOHA P I. NÁZEV PŘÍLOHY

Obsah přílohy