EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

jihomoravský kraj

OPERAČNÍ SYSTÉMY

Úvod ho hashingu a šifrování

Metodický list

Autor: doc. Ing. Jaroslav Dočkal, CSc., Metodik: Ing. Roman Koláčný

Recenzenti: Ing. Petr Skyva

Rok vydání: 2023

Obsah

Cíle

▪ Cíle výuky:

- Seznámení se základními metodami hashingu a šifrování
- Procvičení vybraných metod hashingu a šifrování
- Porozumění vývojovým trendům v moderní kryptografii
- Orientace v dostupných informačních zdrojích

▪ Dovednosti

Žáci by si měli osvojit následující dovednosti:

- Zapamatovat si názvy a kategorizaci základních metod hashingu a šifrování
- Rozumět historickým souvislostem jednotlivých kryptografických metod
- Aplikovat východiska tématu z úvodu do teoretické části na posuzování jednotlivých algoritmů
- Analyzovat nové vývojové trendy v kryptografii
- Být schopen ohodnocovat kryptografické sady
- Navrhovat použití kryptografických algoritmů v souladu s moderními standardy a doporučeními

▪ Pracovní prostředí

Úlohu lze realizovat v prostředí:

- Cylab JCEKB
- Offline Security Classroom

Pro práci budeme potřebovat následující nástroje:

- Připojení do internetu
- Program HashCalc
- Vývojové prostředí programovacích jazyků Python a C++

Průběh výuky

1 Teoretická část – kryptografické metody

Úvodem studia kryptografie by si žáci měli ujasnit některá východiska:

- Rozdíl mezi šifrováním a kódováním: Šifry používají něco tajného (klíč), kdežto kódy nic neskrývají.
- Rozdíl mezi heslem s klíčem: Heslo je obecný prostředek k autentizaci uživatele (nemusí to být obecně pouze člověk). Pokud používá heslo člověk, snaží se, aby bylo zároveň snadno zapamatovatelné a zároveň neuhodnutelné někým jiným, což jsou rozporné požadavky. Klíč transformuje zprávy do šifrovaného textu, při dešifrování je tomu naopak. Zde požadujeme, aby byla hodnota klíče náhodná anebo pseudonáhodná.
- Pojem „funkce odvození klíče“ (KDF _ Key derivation function), což je je funkce, která transformuje heslo libovolné délky na pevnou hodnotu klíče.
- Vlastnosti dokonalé šifry: Náhodné údaje musí být generovány fyzikálně. Šifra musí být přinejmenším stejně dlouhá jako šifrovaný text. Jsou vygenerovány pouze dvě kopie klíče, jedna pro šifrující stranu a druhá pro stranu dešifrující (existují některé výjimky pro více příjemců). Každý klíč je použit na straně šifrování i dešifrování výhradně jednou, a po jeho použití musí obě strany zničit své klíče.
- Jsou dva základní koncepční přístupy k šifrování: Algoritmus zveřejnit (Secure by Design) – kryptosystém by měl být bezpečný, i když vše o systému, s výjimkou klíče, je veřejně známo. Algoritmus nezveřejňovat (Security through Obscurity) – poskytovat minimum zneužitelných informací, používat zdržovací opatření a mást co nejvíce způsoby, např. falešným hlášením o chybě.

Mezi kryptografické techniky patří především: hashování, šifrování, výměna klíčů, digitální podpisy a digitální certifikáty. Aktuální aplikace pokrývají zabezpečení hesel operačních systémů, moduly TPM[1] (Trusted Platform Modules), šifrování disků, kryptoměna atd. V této pomůcce se omezíme na hashování a šifrování, ostatní oblasti kryptografie necháme na návazné studium.

1.1 Hashování

Hashovací hodnota, je matematicky generovaná, jednosměrná, alfanumerická hodnota získaná transformací posloupnosti bitů libovolné délka na (relativně) malou délku.

Nejznámější způsob použití hashe je odvozování klíčů z hesel. Pokud by byl soubor hesel zcela bezpečný, hashování by nebylo potřebné. My se však připravujeme se na nejhorší případ. Připravujeme se na to, že útočník získá přístup

[1] Jako příklad může posloužit BitLocker Microsoftu.

k přihlašovacím údajům našich uživatelů. Do hry vstupuje klíčová myšlenka ochrany soukromí: Nemusíte se starat o data, která nemáte. Zde na rychlosti hashe nezáleží, protože jsou hesla relativně krátká.

Druhým způsobem použití je kontrola integrity dat. Pokud změníme byť jediný bit kontrolovaných dat, výsledkem je zcela jiná hodnota hashe. Nejde přitom o nepatrnou změnu hodnoty hashe, ale o významnou změnu. V kryptografii se tomu říká „princip difúze a zmatku“ (diffusion and confusion). Účelem je pomoci eliminovat detekci předvídatelných vzorů. Protože kontrolované soubory mohou být dosti veliké, zde na rychlosti hashování dosti záleží.

Hodnoty hash souborů kompromitovaných malwarem jsou často sdíleny bezpečnostní komunitou, protože jsou cennými indikátory kompromitace (důkaz infekce), což je třetí významná oblast použití. Analytici zpravodajství o hrozbách často sdílejí tyto informace prostřednictvím otevřených zpravodajských databází, jako jsou VirusTotal a OTX.

Hashovací funkce mohou být kryptografického i nekryptografického charakteru. Na *https://asecuritysite.com/hash2* je těch nekryptografických uvedena celá řada. Tyto hodnoty hashe se obvykle používají pro indexování, například u hashovacích tabulek. Existuje i kategorie kryptografických hash funkcí s výstupy libovolné délky – eXtendable-Output Functions **(**XOF). Lze je tedy použít v aplikacích, u kterých je zapotřebí výstup jiné než standardní velikosti.

1.1.1 Zvyšování délek hashů

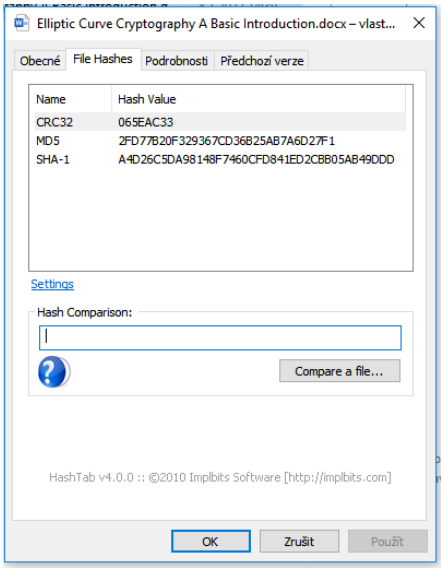
Rychlým a snadným způsobem, jak zobrazit sadu různých hashů z jednoduchého řetězce či souboru, je HashCalc – viz obr. 1.1.1.1. Všimněte si, že nekryptografická funkce CRC (Cyclic Redundancy Code), která je zaměřena na přenosové chyby, je výrazně kratší (32 bitů), protože zde jde o kompromis mezi ztrátami způsobenými neodchycenými přenosovými chybami a ztrátami na přenosovém výkonu v důsledku přenosu režijních dat.

Historicky nejznámější kryptografickou hash funkcí je MD5 (Message Digest 5). Ronald Rivest ji navrhl v roce 1992 jako náhradu svého MD4 z roku 1991[2]. NIST (National Institute of Standards and Technology) nikdy neuznal její délku hashe 128 bitů za dostatečnou, tu uznal až u SHA-1 se 160 bity (návrh National Security Agency). V roce 2008 byl algoritmus MD5 za dva dny prolomen v rámci útoků na RapidSSL pomocí klastru 200 počítačů. Protože na počítač

[2] Na MD4 a MD5 navázal neúspěšný kandidát na standard SHA-3 MD6 (Pumkin Hash).

běžného uživatele se nepředpokládá tak silný útok, je MD5 stále základem zabezpečení hesel u NLTMv2 Microsoftu (První verze NLTM u Windows 2000 používala zastaralý MD4 z roku 1991[3]).

Všechny algoritmy pro výpočet hashů trpí v jisté míře kolizemi, tj. existencí stejných hashů pro různé zdrojové texty. U MD5 a SHA-1 je problémem, že pomocí soudobých výpočetních prostředků se tyto kolize dají najít (Stevens 2017)[4]. Proto byly později upřednostňovány algoritmy se složitějšími vnitřními logickými funkcemi a delšími hashy – např. u SHA-2 (SHA256) 256 bitů, u SHA-512 512 bitů.



Obr. 1.1.1.1: HashCalc určený k výpočtu hashů ze souboru nebo řetězce znaků.

Soutěž o SHA-3 organizovanou NIST (2008–2012) vyhrál algoritmus Keccak (BERTONI 2011), a protože byl vytvořen na odlišné bázi, byla mu přisouzena role náhradního řešení a vymezení pro oblast senzorových sítí a mikroprocesorů v RFID. Konkurenční algoritmus z finále soutěže BLAKE v jedné z novějších verzí BLAKE2s s 256bitovým výstupem nahrazuje algoritmy SHA od verze Linuxu 5.17.

Při navrhování moderního hashování je zaměření obvykle na pevný hash výstup, ale návrháři rovněž implementují výstup s proměnnou délkou pro XOF. Např. s metodou hashování SHA-3 máme čtyři různé kryptografické hashovací metody s pevným výstupem (SHA3–224, SHA3–256, SHA3–384 a SHA3–512) a dvě funkce typu XOF (SHAKE128 a SHAKE256). S BLAKE2b – jednou z nejrychlejších kryptografických metod hashování – je k dispozici XOF varianta BLAKE2XB a s algoritmem BLAKE2 je k dispozici XOF varianta BLAKE2XS.

Hash, stejně jako ostatní kryptografické metody, je součástí knihoven řady programovacích jazyků, výstup si lze zadat v hexadecimálním či bytovém tvaru – viz obr. 1.1.1.2.

```
>>> import hashlib
>>> h=hashlib.sha256()
>>> h.update('hash v kódu utf-8'.encode('utf-8'))
>>> print(h.hexdigest()) # tisk hashe v podobě hexadecimálního řetězce
c80bc70f941b8f169ef02125f16059d4d0fa82d48a565a0cc79e8b008c94ef81
>>> print(h.digest()) # tisk hashe vyjádřeného v bytech
b'\xc8\x0b\xc7\x0f\x94\x1b\x8f\x16\x9e\xf0!%\xf1`Y\xd4\xd0\xfa\x82\xd4\x8aVZ\x0c\xc7\x9e\x8b\x00\x8c\x94\xef\x81'
>>>
```

Obr. 1.1.1.2: Různé formy výstupu kryptografické funkce

[3] U Active Directory je defaultně vyžadována implementace NTLM a Kerberosu (šifrovací systém na bázi algoritmu DES). Pro hash je zde navíc k dispozici MD5, SHA-1 a bcrypt.
[4] Na *https://www.shattered.io/* si můžete otestovat odolnost svého souboru proti koliznímu útoku.

1.1.2 Slovníkový útok

Slovníkový útok je založen na vyzkoušení všech řetězců v předem připraveném seznamu. Tyto útoky původně používaly slova nalezená ve slovníku (odtud fráze slovníkový útok); nyní jsou však na otevřeném internetu dostupné mnohem větší seznamy obsahující stovky milionů hesel získaných z minulých úniků dat. Existuje také crackovací software, který může takové seznamy používat a vytvářet běžné varianty, jako je nahrazení podobně vypadajících písmen číslicemi.

Slovníkový útok zkouší pouze ty možnosti, které jsou považovány za nejpravděpodobnější. Slovníkové útoky jsou často úspěšné, protože mnoho lidí má tendenci volit krátká hesla, která jsou běžnými slovy nebo běžnými hesly; nebo varianty získané například připojením číslice nebo interpunkčního znaku. Slovníkové útoky jsou často úspěšné, protože mnoho běžně používaných technik vytváření hesel je pokryto dostupnými seznamy v kombinaci s vytvářením vzorů pro crackování softwaru. Bezpečnějším přístupem je náhodné vygenerování dlouhého hesla (15 a více písmen) nebo použití víceslovných přístupových frází.

Pro slovníky se předem vytvářejí seznamy hashů slov ze slovníku a pak ukládají do databáze s použitím hashe jako klíče. To vyžaduje značné množství času na přípravu, ale umožňuje rychlejší provedení skutečného útoku. Požadavky na úložiště pro předem vypočítané tabulky byly kdysi hlavními náklady útoku, ale nyní jsou menším problémem kvůli nízkým nákladům na diskové úložiště.

Předpočítané slovníkové útoky jsou zvláště účinné, když má být prolomeno velké množství hesel. Předpočítaný slovník je potřeba vygenerovat pouze jednou a po jeho dokončení je možné téměř okamžitě vyhledat odpovídající heslo. Propracovanější přístup zahrnuje použití duhových (Rainbow) tabulek (tvořené převypočítanými řetězci hashů), které snižují požadavky na úložiště za cenu mírně delší doby vyhledávání. Předpočítané útoky s využitím Rainbow tabulky mohou být zmařeny použitím solení, protože sůl je pokaždé jiná, výsledný hash je tím pádem i pro stejná hesla pokaždé jiný.

1.1.3 Solení a pepření

Solením se vytvářejí situace, kdy jeden vstup (obvykle heslo) má při použití stejné jednosměrné (hashovací) funkce mnoho různých výstupů (v podstatě tolik, kolik různých kombinací soli lze získat). To útočníkovi ztěžuje prolomení hash hesla. Rozdíl mezi solením a pepřením je v tom, že hodnota pepře je na rozdíl od soli uchovávána v tajnosti a není uložena v databázi s hashovaným heslem.

Pokud není použito solení ani pepření, aktéři hrozeb, kteří získají hashovaná uživatelská hesla (spolu s jejich odpovídajícími uživatelskými jmény), mohou obejít mechanismy ověřování hesel ve formátu prostého textu a přihlásit se pomocí hashe hesla. Tento postup je znám pod termínem Pass the Hash (MITRE ATT&CK technika ID T1550.002). To činí hashe hesel zranitelnými vůči zneužití, což umožňuje protivníkovi pohybovat se v rámci sítě laterálně (postranně) – např. ukradne účet jednoho zaměstnance organizace a použije ho k phishingovému útoky na další pracovníky organizace.

1.1.4 PBKDF2 – funkce odvozování klíče na základě hesla

Cílem je převést heslo na klíč pomocí funkce PBKDF2(heslo, sůl, počet cyklů), kde *PBKGF2* (Password-Based Key Derivation Function) je jedna ze standardizovaných hashovacích funkcí, *sůl* má zajisti rozptyl hodnot, tj. aby stejné heslo nevedlo vždy ke stejnému klíči, a *počet cyklů* udává, kolikrát výstup z hashovací funkce použiji jako cyklický vstup do hashovací funkce v dalším kroku (norma udává 1024 cyklů). Fragment programu v Pythonu pro výpočet hashe pomocí soli s hodnotou KB3 a 4096 rundami ukazuje obr. 1.1.4.1.

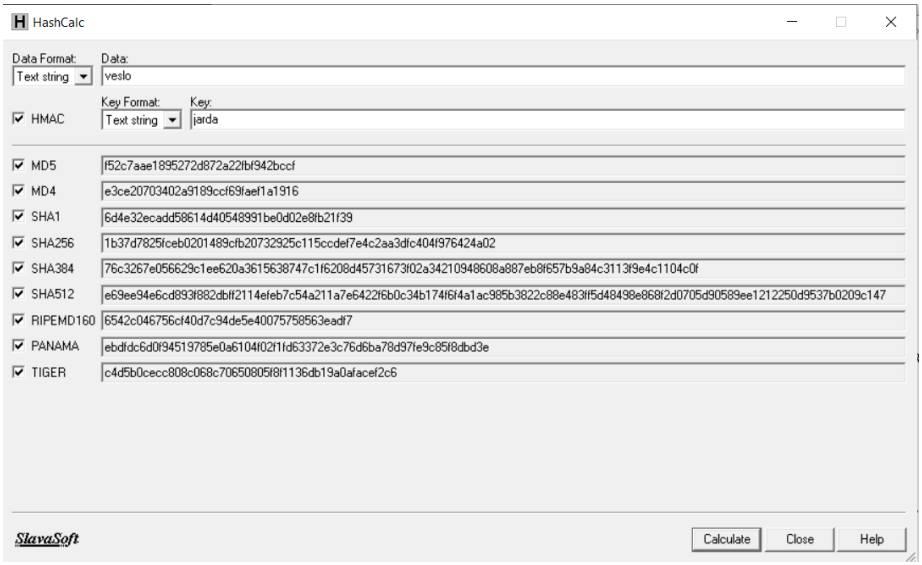
Pro generování samotných náhodných čísel je používán kód HMAC (Hash-based Message Authentication Code[5]) viz obr. 3, což je specifický typ ověřovacího kódu (MAC – Message Authentication Code) zprávy veslo zahrnující kryptografickou hashovací funkci a tajný kryptografický klíč (na obr. 1.1.4.2 jarda). V systémech výzva – odpověď ale náhodně generované soli nejsou praktické, protože je pak třeba posílat sůl pokaždé s výzvou, to už pak je efeltivnější používat jméno.

```
>>> from passlib.utils.pbkdf2 import pbkdf2
>>> import sys
>>>
>>> if (rounds>4096):
... print („Příliš rund")
  File "<stdin>", line 2
    print („Příliš rund")
>>> s2 = pbkdf2(st, salt, rounds, keylen=keylen, prf=method)
>>> print ("String:\t\t",st)
String:         veslo
>>> print ("Salt:\t\t",salt)
Salt:           KB3
>>> print ("Rounds:\t\t",rounds)
Rounds:         2048
>>> print ("Key length:\t",keylen)
Key length:     64
>>> print ("Method:\t\t",method)
Method:         hmac-sha512
>>>
>>> print ("\nHash: ",s2.hex())

Hash:  a215379be598e834266e2c44f1977a2df87160660dbd1b405880f2
```

Obr. 1.1.4.1: Fragment programu v Pythonu pro výpočet hashe pomocí soli (zde KB3) a 4096 rund.

[5] HMAC zabíjí dvě mouchy jednou ranou – kromě kontroly integrity slouží k autentizaci.



Obr. 1.1.4.2: Příklad použití funkce HMAC v programu HashCalc

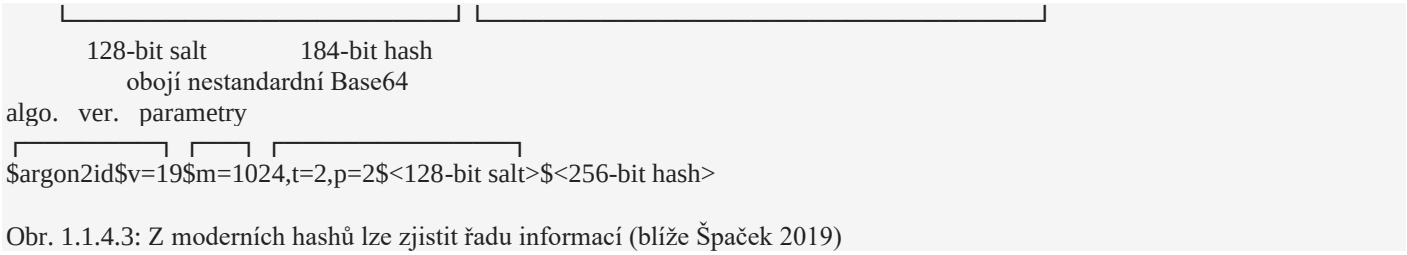
PBKDF2 je součástí série de facto standardů Public-Key Cryptography Standards (PKCS) společnosti RSA Laboratories, konkrétně PKCS #5 v2.0. RFC 8018 (PKCS #5 v2.1) z roku 2017 doporučuje PBKDF2 pro hashování hesel, používá ho TrueCrypt i jeho nástupce VeraCrypt.

Funkce odvozování klíče na základě hesla (KDF založená na heslu) je obecně navržena tak, aby byla výpočetně náročná, takže její výpočet trvá relativně dlouho (řekněme v řádu několika set milisekund). Oprávněným uživatelům stačí provést funkci pouze jednou za operaci (např. autentizace), a proto je potřebný čas zanedbatelný. Útok hrubou silou by však pravděpodobně musel provést operaci miliardkrát, v tomto okamžiku se časové požadavky stanou významnými a v ideálním případě neúměrnými.

Realizace funkce PBKDF2 mají jednu slabinu: nejsou dostatečně odolné vůči slovníkovým útokům, útokům GPU a útokům ASIC. Naopak nejnovější algoritmy jako Bcrypt (1999) používaný Microsoftem, Scrypt (2009) oblíbený v oblasti digitální měny a u Cisco, Argon2 (2015) a Ballon[6] (2016) doporučený v (NIST 2017), všechny používají sůl + mnoho iterací + spoustu CPU + spoustu paměti RAM a to velmi ztěžuje navrhování vlastního hardwaru, který by výrazně urychlil prolomení hesla. Ze struktury hashe lze zjistit řadu informací – viz obr. 1.1.4.3. Pracovní princip šifrovacího algoritmu spočívá v tom, že uměle komplikuje výběr možností řešení kryptografické úlohy tím, že jej vyplní „šumem“. Tento šum jsou náhodně generovaná čísla, na která se vztahuje šifrovací algoritmus, čímž se prodlužuje doba zpracování.

```
bcrypt (2y)
 ┌┐ cost (2¹⁰ = 1024 opakování)
 ┌┐┌┐
$2y$10$7REcgj13ZZTW9XSYGWfZVODMB0uIPn3c2jZmse1kjz7LHGzTdUnGm
```

[6] Jako dílčí algoritmus může používat jakékoli standardní kryptografické hashovací funkce, které nejsou náročné na prostor (např. SHA-3, SHA-512).



Obr. 1.1.4.3: Z moderních hashů lze zjistit řadu informací (blíže Špaček 2019)

Když je k odvození klíče z daného hesla použito velké množství CPU a RAM, je prolamování hesel pomalé a neefektivní (např. 5–10 pokusů za sekundu), a to i při použití velmi dobrého hardwaru a softwaru pro prolomení hesla. Cílem moderních funkcí KDF je prakticky znemožnit provedení útoku hrubou silou ke zvrácení hesla z jeho hashe.

Svědectvím vývoje hashovacích metod je hashování hesel u síťových prvků Cisco: Ve verzi IOS 13 byly použity algoritmy MD5 (Typ 5) a SHA-256 (Typ 4). Oba může aplikace Cain prolomit, jen u typu 4 to bude trvat déle. Poté Cisco zkoušelo Vigenerovu šifru (bude o ní později) jako Typ 7 a zjistil, že jde o prolomitelnou variantu. Od verze IOS 15.3(3) Cisco používá Typ 8 s PBKDF2-SHA-256 a TYP 9 Scrypt.

Vraťme se ještě k (NIST 2017): „Funkce odvozování klíče BY MĚLA používat schválenou jednosměrnou funkci, jako je Keyed Hash Message Authentication Code (HMAC), …, Secure Hash Algorithm 3 (SHA-3), CMAC[7] nebo Keccak Message Authentication Code (KMAC), Customizable SHAKE (cSHAKE) nebo ParallelHash[8].” Samá moderna.

1.1.5 Scrypt – moderní funkce odvození kryptografického klíče

Svědectvím vývoje hashovacích metod je hashování hesel u síťových prvků Cisco: Ve verzi IOS 13 byly použity algoritmy MD5 (Typ 5) a SHA-256 (Typ 4). Oba může aplikace Cain prolomit, jen u typu 4 to bude trvat déle. Poté Cisco zkoušelo Vigenerovu šifru (bude o ní později) jako Typ 7 a zjistil, že jde o prolomitelnou variantu. Od verze IOS 15.3(3) Cisco používá Typ 8 s PBKDF2-SHA-256 a TYP 9 Scrypt, který si nyní rozebereme podrobněji.

Scrypt (RFC 7914) je silná funkce odvození kryptografického klíče (KDF). Je náročný na paměť, navržený tak, aby zabránil útokům GPU, ASIC a FPGA (vysoce účinný hardware pro prolomení hesel).

Algoritmus Scrypt bere několik vstupních parametrů a vytváří odvozený klíč jako výstup:

klíč = Scrypt(heslo, sůl, N, r, p, derived-key-len)

Parametry konfigurace Scrypt jsou:

- N – počet iterací (ovlivňuje využití paměti a CPU), např. 16384 nebo 2048
- r – velikost bloku (ovlivňuje využití paměti a CPU), např. 8

[7] CMAC je autentizační algoritmus založený na blokové šifře typu CBC (Cipher Block Chaining).
[8] Určený pro velmi dlouhé zprávy.

- p – faktor paralelismu (vlákna běží paralelně – ovlivňuje paměť, využití CPU), obvykle 1 heslo – vstupní heslo (doporučuje se minimální délka 8–10 znaků)
- sůl – bezpečně generované náhodné bajty (minimálně 64 bitů, doporučeno 128 bitů)
- derived-key-len – kolik bajtů se má vygenerovat jako výstup, např. 32 bajtů (256 bitů)

K paměti v Scryptu se přistupuje v každém kroku v silně závislém pořadí, takže rychlost přístupu do paměti je úzkým hrdlem algoritmu. Paměť potřebná k výpočtu odvození klíče Scrypt se vypočítá takto:

Potřebná paměť = 128 * N * r * p bajtů

Příklad: např. 128 * N * r * p = 128 * 16384 * 8 * 1 = 16 MB (nebo 128 * N * r * p = 128 * 2048 * 8 * 1 = 2 MB)

Výběr parametrů závisí na tom, jak dlouho chcete čekat a jaké úrovně zabezpečení (odolnosti proti prolomení hesla) chcete dosáhnout. Vzorové parametry pro interaktivní přihlášení: N=16384, r=8, p=1 (RAM = 2 MB). Pro interaktivní přihlášení pravděpodobně nebudete chtít čekat déle než 0,5 sekundy, takže výpočty by měly být velmi pomalé. Také na straně serveru je obvyklé, že se může přihlásit mnoho uživatelů současně, takže pomalý výpočet Scrypt zpomalí celý systém.

Nechť jsou parametry pro šifrování souborů: N= 2048, r=8, p=1 (RAM = 1 GB) a délka klíče 32 byte, pak lze použít program v Pythonu z obr. 1.1.5.1 – převzato z *https://replit.com/@nakov/Scrypt-in-Python*.

```
import pyscrypt
salt = b'aa1f2d3f4d23ac44e9c5a6c3d8f9ee8c'
passwd = b'p@$Sw0rD~7'
key = pyscrypt.hash(passwd, salt, 2048, 8, 1, 32)
print("Derived key:", key.hex())
```

```
Derived key: b'e813a6f6ccc4e9110193bf9efb7c0a489d76655f9e36629dccbeaf2a73bc0c6f'
```

Obr. 1.1.5.1: Program v Pythonu pro odvození šifrovacího klíče

Během návrhu a vývoje aplikace nebo systému lze provádět testy a volit parametry Scrypt pro daný jazyk a platformu. V kryptopeněžence MyEtherWallet jsou výchozí parametry Scrypt N=8192, r=8, p=1, což není dostatečně silné nastavení. Řešením je použití dlouhého a složitého hesla, aby se zabránilo útokům na prolomení hesla.

1.2 Šifrování

1.2.1 Symetrické šifrování

Šifrovací metody dělíme na symetrické a asymetrické. Symetrické metody používají stejný klíč pro šifrování i dešifrování, asymetrické metody používají pro šifrování a dešifrování jiné klíče. U symetrického šifrování je výhodou použití jednoho klíče, nevýhodou nutnost jeho distribuce. U asymetrického šifrování jeden z klíčů páru zveřejňujeme (veřejný klíč) a druhý utajujeme (tajný klíč). V šifrovacím režimu odesilatel zprávy zprávu šifruje veřejným klíčem a

příjemce ji kontroluje tajným klíčem. V režimu digitálního podpisu odesilatel obvykle nepodepisuje zprávu, ale její hash a zašifrovaný (podepsaný) hash posílá spolu se zprávou. Příjemce přijatý hash zašifruje veřejným klíčem a tato hodnota by měla být shodná s hodnotou hashe zašifrovaného tajným klíčem odesilatele. Binární hodnoty působí problémy při zobrazování a editaci, proto jsou zašifrované hodnoty konvertovány do hexadecimálního tvaru nebo formátu Base64[9].

V komunikačních systémech jde hlavně o rychlost, a tak jsou pro symetrické šifrování využívána hardwarová řešení založená na generování pseudonáhodných posloupností čísel pomocí zpětnovazebních metod – tyto systémy fungují obdobně jako nekryptografické systémy určené pro kontrolu přenášených hodnot na chyby. Kryptografické systémy se liší propojením LFSR (Linear Feedback Shift Register) pomocí vybraných logických funkcí a funkce XOR (exklusivní disjunkce – někdy se označuje jako minišifrování) zajišťujících jednosměrnost šifrování. Jako příklad lze uvést Lineární posuvný registr se zpětnou vazbou A5/1 použitý u standardu GSM celulární telefonie.

Při šifrované komunikaci se tradičně vychází z cyklického posunu (substituce), nakonec Cesarova šifra byla založena na posunu o standardní počet znaků abecedy (formálně zapsáno $f_i(a) = (a + k) \bmod n$, kde a je pozice znaku, k posun, např. pokud je k=1, jde o posun o jednu pozici, A na B či Z na A). Posun je počet pozic posunu určený pozicí znaků hesla v abecedě ($f_i(a) = (a + k_i) \bmod n$, ki označuje i-tý znak abecedy), jde o přístup Vigenèrovy šifry odolné vůči frekvenční analýze[10]. Pokud je pozice znaku určena generátorem pseudonáhodných čísel, je systém odolný vůči frekvenční analýze (využívá toho, že jednotlivé znaky abecedy se vyskytují s různou frekvencí).

Po technické stránce se řadu let používaly funkce XOR, substituce a transpozice (permutace, prohození bitů). Určující byly substituční tabulky a ty právě u DES (Data Encryption Standard používaný v letech 1972–2005) a jejich nedostatečná velikost byla slabinou algoritmu s faktickou délkou klíče 56 bitů[11] (z 64 bitů klíče byly vyřazovány paritní bity bez kryptografické síly) pro šifrování 64bitových bloků. Řešením problému slabých substitučních polí byl Triple DES, pod P je otevřený text a C zašifrovaný, E proces šifrování a D dešifrování a k klíč, $C = E_{k3}(D_{k2}(E_{k1}(P)))$ a $P = D_{k1}(E_{k2}(D_{k3}(C)))$[12], což vyhovovalo pouze pro šifrování klíč, navíc celý DES byl postaven jako hardwarové řešení.

Během postupného odchodu DES ze scény padla celá řada návrhů, celou sitaci ale vyřešila soutěž U.S. Secretary of Commerce o AES (Advanced Encryption Standard) v roce 2002, kterou vyhrál algoritmus Rijndael Jonase Daemena a Vincenta Rijmena[13]. Rijndael je určen pro softwarové použití, je pružně navržen, jak klíč tak blok mohou být velké 128, 192 nebo 256 bitů (lze rozšířit o násobky 32 bitů). Šifra je přitom dostatečně silná, tento algoritmus může být velmi účinně implementován na širokém spektru procesorů a v hardwaru.

---

[9] Zde se předpokládá, že práci s číselnými formáty žáci zvládli v předmětu Základy informatiky.
[10] Analýza na základě frekvence výskytu daného znaku v abecedě.
[11] Původně měl být klíč dlouhý 128 bitů, ale NSA (National Bureau of Standards) přinutila IBM zkrátit klíč na 64 bitů. Návrháři poslali NSA návrhy substitučních boxů, zpět se vrátily poněkud odlišné atd. Kryptografická zpráva o DES a návrhová schémata substitučních bloků byly utajeny.
[12] Někdy se používá varianta 3DES-EDE, kdy se šifruje, pak dešifruje a pak znovu šifruje stejným klíčem, jako při prvním šifrování.
[13] Mezi poraženými konkurenty byl mj. RC6 (Rivest Code). Rivest odstranil bezpečnostní slabiny svých ranných kódů RC2 a RC4, kódy RC5 a RC6 již byly dostatečně bezpečné, tratily však na rychlosti zpracování.

DES i AES mají jednu společnou nevýhodu a to že ke stejnému původnímu bloku textu lze přiřadit odpovídající blok zašifrovaného textu a vytvářet slovníky pro slovníkové útoky. Takovýto způsob kódování nazýváme Electronic Code Book a nahrazujeme ho zpětnovazebními metodami, například CBC (Cipher Block Chaining), kdy výstup šifrování jednoho bloku dat binárně přičítáme ke vstupním datům dalšího bloku dat, chybějícího předchůdce prvního bloku nahrazujeme tzv. inicializačním vektorem (IV). Metody CBC mají (při stejném IV) tu špatnou vlastnost, že stejné otevřené texty produkují stejné zašifrované texty. Útoky na AES-CBC mohou vycházet z použitého doplnění na velikost celého bloku dat a manipulaci s inicializačním vektorem IV[14].

Každý blok navíc čeká na šifrování těch předchozích, což je nevhodné pro paralelní zpracování[15], proto se v takových případech používá proudová[16] metoda šifrování zvaná CTR (counter), kdy náhodně vygenerovaný inicializační vektor pro každý další blok navýšíme o 1, přivedeme na blok šifrovaný klíčem a výsledek binárně přičteme k bloku dat – neboli to lze provádět pro více bloků současně. Pokd pracujeme ve zpětnovazebním řetězci v jednotlivými bity, hovoříme o tokových metodách, ty jsou ale více používány v systémech přenosu dat než systémech zpracování dat, a proto zde nebudou popisovány.

Pokud CTR doplníme o autentizaci, dostaneme řadu dalších módů, ze kterych je nejznámější AES-256-GCM. Algoritmus patří do kategorie AEAD (Authenticated encryption with associated data) váže přidružená data na šifrovaný text a na kontext, kde se má objevit, takže jsou detekovány a odmítnuty pokusy „vyjmout a vložit“ platný zašifrovaný text do jiného kontextu. Jedním nástrojem je zde tím pádem zajištěna důvěrnost, integrita a autentičnost dat. Pro slabiny některých implementací GCM vůči celému spektru útoků bývá často dávána přednost kombinaci proudové šifry ChaCha20[17] a autentizátora Poly1305[18], což je MAC funkce, neboli funkce určená k ověření integrity dat a autentizaci zprávy. Jak tuto šifru a autentizátora zkombinovat k vytvoření schématu AEAD specifikuje RFC7539. Oba algoritmy jsou určeny pro vysoké rychlost a lehké aplikace IoT (Internet of Things)[19], např. k nasazení rychlých a bezpečných připojení TLS mezi uzly IoT a vzdálenými cloudovými servery, když nejsou k dispozici možnosti hardwarové akcelerace AES[20].

[14] Využívání chybových hlášení zde dostalo název *padding oracle attack*: Vycpávka má dle defacto standardu PKCS#7 standardizovanou velikost a pokud změníme IV, změní se i tato hodnota a přijde o tom chybové hlášení, tímto způsobem se lze dotestovat ke správnému heslu.

[15] Navíc je zde problém, jak dorovnat text na násobky velikosti bloků dat.

[16] U proudových šifer je vstupní datový tok je kombinován s pseudonáhodným proudem bitů vytvořeným z šifrovacího klíče a šifrovacího algoritmu.

[17] ChaCha20 používá 256bitový klíč a 32bitové náhodné číslo (nounce) a z nich v 20 kolech vytvoří klíčový proud, který je pak XORován s proudem prostého textu. Vyzkoušet si lze na *https://asecuritysite.com/encryption/poly1305*.

[18] Jméno je odvozeno od použitého prvočísla $2^{130} – 5$ a od skutečnosti, že jádrem algoritmu je vyhodnocení polynomu.

[19] Proudovou šifru ChaCha používají hash funkce BLAKE, BLAKE2 and BLAKE3, které díky tomu rovněž bodují vysokými rychlostmi zpracování potřebnými pro IoT.

[20] Např. u OpenSSL lze volit jak AES-GCM, tak ChaCha20-Poly1305.

1.2.2 Asymetrické šifrování

Nejznámější algoritmus veřejného klíče je RSA autorů R. Rivest, Shamir a Adleman: Za předpokladu, že n výsledek součinu dvou prvočísel p,q (tyto dvě hodnoty utajujeme) a e (oblíbené hodnoty jsou 3 a 65 537) je nesoudělné prvočíslo k (p-1)(q-1) odvozujeme tajný klíč pomocí vztahu $d = e^{-1} \bmod (p-1)(q-1)$. Pak probíhá šifrování pomocí vztahu $c = m^e \bmod n$ a dešifrování pomocí vztahu $m = c^d \bmod n$. RSA je standardizován pomocí "de facto" PKCS#1 (Public Key Crypto Standard). Problém vycpávky (zde doplnění šifrovaného textu, zatímco u CTR šlo o doplnění zašifrovaného textu), tzv. RSA padding, je zde řešen dvojím způsobem, dle PKCS#1.5 a dle *Optimal Asymmetric Encryption Padding (OAEP)*.

Systém RSA má řadu nevýhod: Je nutné počítat velmi dlouhé veřejné klíče, šifrování i dešifrování je značně pomalé, což vymezují jeho použití v počítačových sítích na distribuci klíčů a digitální podpisy. Pokud se píše o "prolomení RSA" míní se tím jeho rozklad (faktorizaci) na dvě prvočísla. Dosud byl prolomen klíč dlouhý 768 bitů (232 dekadických čísel). RSA má i své slabé klíče, jejich seznam je na *https://freakattack.com/*, Tester pro klienta je na *https://www.smacktls.com/freak/*, pro server na *https://tools.keycdn.com/freak*.

Výhodou RSA je, že algoritmus lze použít i na digitální podpis (opačné použití šifry), podepisování je ale relativně zdlouhavé. Algoritmus DSA[21] (Digital Signature Algorithm) standardizoval NIST v roce 1991 jako DSS (Digital Signature Standard), tento algoritmus naopak rychleji podepisuje (je navržen tak, aby si mohl výpočty předzpracovávat) a zase u něj trvá déle verifikace.

RSA je vytěsňován eliptickými křivkami, které mají vysokou kryptografickou sílu ve vztahu k velikosti klíče, nízké nároky na výpočty, šířku pásma i paměť ze všech algoritmů veřejného klíče, vysokou rychlost šifrování i podepisování, a to jak u hardwarové, tak i u softwarové realizace.

Internet hledal vylepšené řešení oproti RSA a Elliptic Curve je jedním z řešení. Má výhody:

- Mnohem kratší klíče. Prvočíslo p je normálně pouze 160 bitů a je mnohem menší než RSA. To značně urychluje proces šifrování.
- Vytváření křivek je obtížnější než generování prvočísel, což ztěžuje jejich prolomení než RSA.
- Mohou být použity k faktorizaci (rozkladu na prvočísla) hodnot, jako je hledání faktorů prvočísel v RSA.

Celková eliptická křivka je považována za náhradu RSA, zejména u vestavěných systémů (maloformátové počítače, které zajišťují specifické úkoly), které by se obtížně vypořádaly s požadavky na zpracování RSA. Výhodou algoritmu RSA je, že ho lze použít i pro digitální podpisy, eliptické křivky pracují s některou svojí variantou (např. ECDHE) algoritmu Diffie-Hellmana (DHE) či DSA (Digital Signature Algorithm), který je určen výhradně pro výměnu klíčů.

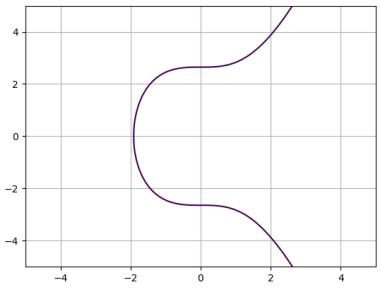
[21] Algoritmus je nápadně podobný algoritmu ElGamal, německý matematik Claus P. Schnorr zase obviňoval NIST z porušení jeho patentových práv (Schnorrův podpis) – tak také někdy vznikají standardy.

Standardizovanými eliptickými křivkami jsou Montgomoryho křivka tvaru $by^2=x^3+ax^2+x$, Weierstrassova křivka[22] má tvar $y^2=x^3+ax+b$ a kroucená Edwardsova křivka $ax^2+y^2=1+dx^2y^2$ (NIST 2019). Vzájemně se liší použitým prvočíslem, počtem bodů eliptické křivky, hodnotami koeficientů a výchozím bodem generátoru.

Například u algoritmu secp256k1 se pracuje s rovnicí:

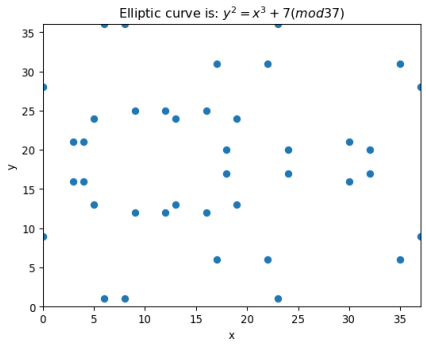
$y^2 = x^3+7 \pmod p$. (rovnice 1.2.2.1)

Pokud rovnici vykreslujeme bez části (mod p), dostaneme, viz obr. 1.2.2.1.



Obr. 1.2.2.1: Graf funkce $y^2 = x^3+7$ převzatý z *https://asecuritysite.com/ecc/plot05*

V eliptických křivkách vybereme jeden z dvojice bodů základní a poté provedeme operace, jako je násobení bodu konstantou, sčítání dvou bodů (P3=P1 + P2) nebo zdvojnásobení bodu (P2=P1+P1) (Buchanan 2021). Např. zavedení části (mod p) nám u rovnice 1.2.2.1) dává dva body y pro hodnoty souřadnic x – viz obr. 1.2.2.2, např. pro x=2 jsou řešením dvojice y (4,18) a (4,5).



Obr. 1.2.2.2: Graf funkce $y^2 = x^3+7 \pmod p$ převzatý z *https://asecuritysite.com/ecc/ecc_pointsv*

Bohužel eliptické křivky má hlavní slabinu a to, že metoda může být prolomena očekávanými kvantovými počítači. Abychom to překonali, potřebujeme nové metody, které jsou kvantově robustní. Navrženy byly metody založené na bodových mřížkách, na hashovacích funkcích, na teorii kódování, na polynomiálních rovnicích, a na supersinguláních

[22] Používá se u bitcoinu.

eliptických křivkách. Téma přesahuje rozsah látky pro tento Pracovní list, zájemci o tyto metody se s nimi mohou seznámit na *https://asecuritysite.com/pqc*, v českém jazyce přehledně např. v bakalářské práci (Popelová 2018). Zřejmě bude brzy třeba doporučení v (NUKIB 2018), kde o těchto metodách není ještě ani zmínka, aktualizovat.

Nové kryptografické algoritmy jsou jedna věc a ověření jejich nasazení druhá a často to není snadné[23]. Jako pozitivní příklad lze uvést knihovnu CIRCL (Cloudflare Interoperable, Pro experimentální ověřování kryptografických algoritmů zaměřených na kryptografii typu Post-Quantum (PQ) a Elliptic Curve Cryptography (ECC) (Kwiatkowski 2019) byla např. navržena knihovna CIRCL (Cloudflare Reusable Cryptographic Library). Je napsána v jazyce Go a dostupnou z *https://github.com/cloudflare/circl*. Pro zařízení CIRCLearn USB Sanitizer společnost Cloudflare použila knihovnu v Pythonu PyCIRCLearn (https://github.com/CIRCL/PyCIRCLean), tento modul byl následně oddělen od skriptů specifických pro zařízení a lze jej použít pro speciální bezpečnostní aplikace k dezinfekci dokumentů z nepřátelských prostředí do důvěryhodných prostředí.

Obr. 1.2.2.3: CIRCLearn USB Sanitizer společnosti Cloudflare

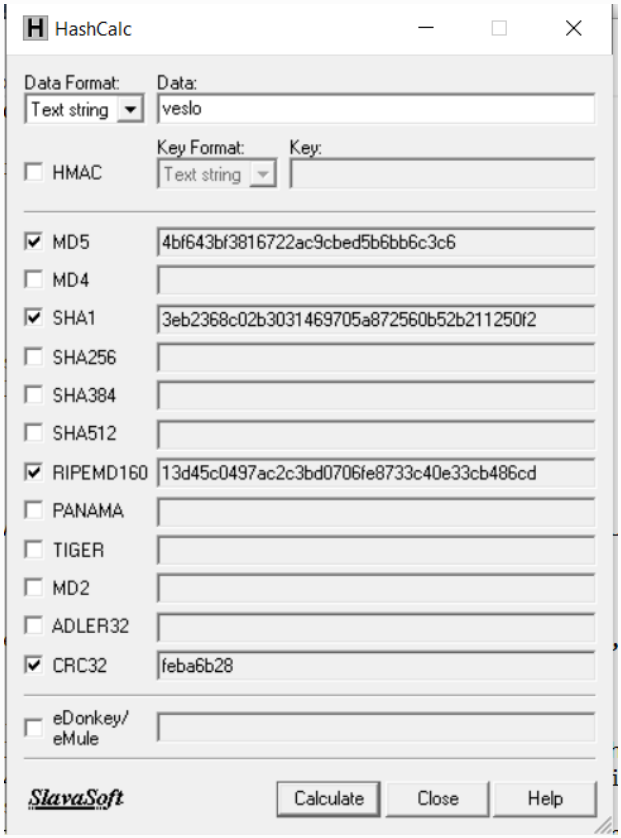
[23] Zajímavé je se např. podívat na předlouhý seznam CVE (Common Vulnerabilities and Exposures) OpenSSH (nástroj pro vzdálené přihlášení pomocí protokolu SSH) na *https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=openssh*.

2 Praktická část – příklady použití kryptografických funkcí

Tato část pracovního listu obsahuje praktické cvičení s aktivním zapojením žáků.

2.1.1 Testování kvality hashe

- Vytvořte si pomocí hash generátoru na *http://www.danstools.com/md5-hash-generator/* svůj MD5 hash svého hesla, např. pro heslo **`veslo`** získáte hexadecimální hodnotu: 4bf643bf3816722ac9cbed5b6bb6c3c6, viz obr. 2.1.1.1.
- Ověřte si, zda jiné realizce stejného algoritmu vedou ke stejnému výsledku, například hashováním našeho vesla na *https://asecuritysite.com/hash/md5_2* získáme hodnotu S/ZDvzgWcirJy+1ba7bDxg==, tato je ale ve formátu Base64, takže je třeba na *https://asecuritysite.com/coding/ascii* převést výsledek do hexadecimálního tvaru a jak si snadno ověříte, získáte stejný výsledek. Stejný výsledek získáte i hashováním hesla na *https://www.pelock.com/products/hash-calculator*, *https://defuse.ca/checksums.htm#checksums* nebo HashCalcu.



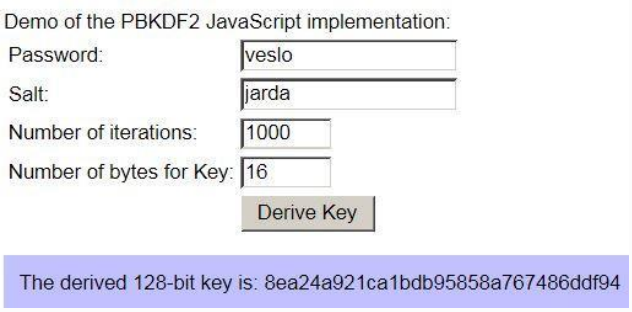
Obr. 2.1.1.1: Použití programu HashCalc pro výpočet vybraných hashů z textového řetězce

- Nainstalujte si program HashCalc a porovnejte výsledek zahashování svého hesla s dosavadními výsledky.
- Na *https://crackstation.net/* (případně použijte *https://www.onlinehashcrack.com/)* otestujte odolnost zahashovaného hesla, např. systém rychle pomocí rain tables odhalí, že hash

4bf643bf3816722ac9cbed5b6bb6c3c6, patří k heslu veslo zahashovaným pomocí algoritmu MD5, viz obr. 2.1.1.2.

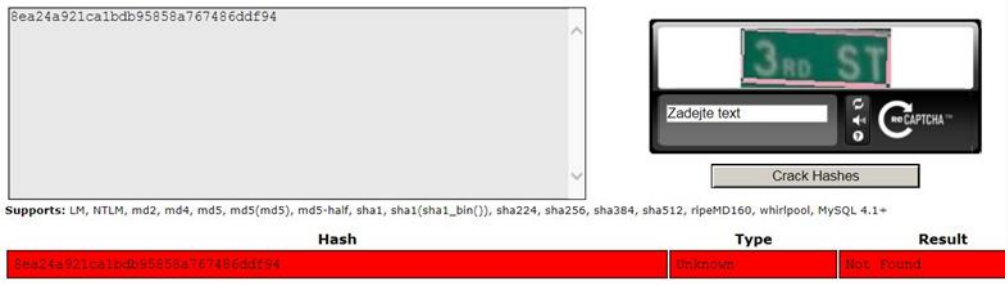
Obr. 2.1.1.2: Testování odolnosti hesla na https://crackstation.net/

- Posolte si své heslo veslo solí (např. Jarda) a použijte generátor klíčů *http://anandam.name/pbkdf2* – viz obr. *2.1.1.3.*



Obr. 2.1.1.3: Příklad posolení hesla veslo solí jarda při 1000 cyklických opakováních

- A zkuste výsledek znovu otestovat na *https://crackstation.net* – viz obr. *2.1.1.4.*



Obr. 2.1.1.4: Opětovné testování odolnosti hesla na https://crackstation.net/

- V našem případě se rainbow tables ukázaly jako neefektivní, 1000 posolených hashů je pro ně příliš. Vyzkoušejte si to jako úkol na vybraném heslu a výsledek pošlete učiteli.

2.1.2 Testování použití PBKDF2

- Zvolte heslo, sůl, počet iterací a počet bitů klíče a spusťte algoritmus na třech realizacích: *https://stuff.birkenstab.de/pbkdf2/*, *http://anandam.name/pbkdf2/* a *https://asecuritysite.com/encryption/pb2*. Pokud nedostanete shodné výsledky, diskutujte s učitelem příčiny.
- Námět pro domácí práci: Zpracujte v jazyce C# fragment programu pro výpočet hashe, obdobný tomu zpracovanému v Pythonu na obr. 1.1.4.1.

2.1.3 Použití moderního algoritmu Scrypt

Úkol: nastavte si potřebné parametry na *https://8gwifi.org/scrypt.jsp* a zašifrujte text heslo s parametry dle obrázku 2.1.3.

Obr. 2.1.3: Generování a verifikace hashe Scrypt pomocí zadaných parametrů obtížnosti útoku.

2.1.4 Použití webové aplikace CyberChef

Úvodní seznámení: CyberChef je webová aplikace pro provádění všech druhů „kybernetických“ operací ve webovém prohlížeči. Tyto operace zahrnují jednoduché kódování jako XOR nebo Base64, složitější šifrování jako AES, DES a Blowfish, vytváření binárních a hexdumpů, kompresi a dekompresi dat, výpočet hashů a kontrolních součtů, analýzu IPv6 a X.509, změnu kódování znaků atd.

CyberChef na https://gchq.github.io/CyberChef/ má čtyři hlavní oblasti:

- Vstupní pole vpravo nahoře, kam lze vložit, napsat nebo přetáhnout text nebo soubor, se kterým chcete pracovat.
- Výstupní pole vpravo dole, kde se zobrazí výsledek zpracování.
- Zcela vlevo seznam operací, kde jsou uvedeny všechny operace, které CyberChef umí.

- Oblast „receptury“ uprostřed, do které lze přetáhnout potřebné operace a zadat argumenty a možnosti.

Úkol: Připojte se na stránku CyberChefu a zašifrujte text „veslo“ pomocí algoritmu Bcrypt s 10 iteracemi (rundami) a zjistěte parametry šifry pomocí Bcrypt parse.

2.1.5 Vigenève kalkulačka

Na *https://asecuritysite.com/coding/vigcalc* zakódujte text 20 slov heslem jarda a proveďte frekvenční analýzu textu.

2.1.6 Generování ECDH

Na *https://asecuritysite.com/ecc/ecc_types* si můžete vyzkoušet generování široké škály eliptických křivek: Curve25519, Curve448, ED25519, Ed448, secp160r2, secp256k1, secp521r, NIST-P192, NIST-P224, NIST-P256.

Prvním úkolem je výběr typu křivky secp256k1 používané v bitcoinech a Ethereu a nastavení počtu bodů eliptické křivky n =1000. Výstupem jsou koeficienty eliptické křivky.

Dále na *https://asecuritysite.com/encryption/ecdh2* generujte symetrický klíč a dvojici (veřejný, privátní) klíčů pro vzájemnou komunikaci Elliptic Curve Diffie Hellman (ECDH).

2.1.7 Zabezpečení webových tokenů JSON (JWT)

JWT si získalo oblibu s rozšířením architektury mikroslužeb: zpracování autentizačních dat svěřuje mikroslužbám, a proto umožňuje vyhnout se různým chybám při autorizaci, zvýšit produktivitu a zlepšit škálovatelnost aplikací.

V souladu s RFC-7519 jsou JSON Web Tokeny (JWT) jedním ze způsobů, jak zobrazit data pro jejich přenos mezi dvěma nebo více stranami jako objekt JSON. JWT se zpravidla skládá ze tří částí: Záhlaví, užitečné zátěže a digitálního podpisu[24]. Každá z částí – záhlaví a užitečné zatížení – je obyčejný objekt JSON, který je třeba dodatečně zakódovat pomocí algoritmu base64url. Poté se zakódované části propojí a na základě toho se detekuje podpis, který se také stane součástí tokenu.

Obecně je token zachycen na *https://jwt.io/*: Záhlaví je servisní součástí tokenu. Pomáhá aplikaci definovat, jak zpracovat přijatý token. Defaultně je použit algoritmus HS256 (HMAC-SHA256), ve kterém je pro generování a ověřování podpisu použit jeden tajný klíč. Pro podpis JWT lze použít symetrické šifrování/algoritmy podpisu, např. RS256 (RSA-SHA256). Standard umožňuje použití dalších algoritmů, včetně HS512, RS512, ES256, ES512, none atd.

Podpis se generuje následovně: Části podpisu a užitečné zátěže jsou zašifrovány algoritmem base64url a poté sjednoceny do jednoho rámečku pomocí tečky („.“) jako oddělovače. Poté je vygenerován podpis (v našem příkladu algoritmem HMAC-SHA256) a přidán do počátečního rámečku za tečkou.

[24] Existují výjimky, kdy JWT postrádá podpis.

Úkol: Na *https://jwt.io/* změňte v užitečné zátěži jméno „John Doe“ za své a změňte algoritmus podpisu na HS512, což je HMAC používající pro hash algoritmus SHA-512.

JWT podporuje „žádný“ algoritmus. Pokud je pole alg nastaveno na „none“, jakýkoli token by byl považován za platný, pokud je jeho sekce podpisu nastavena na prázdnou. Například následující token by byl považován za platný:

eyAiYWxnIiA6ICJOb25lIiwgInR5cCIgOiAiSldUIiB9Cg.eyB1c2VyX25hbWUgOiBhZG1pbiB9Cg

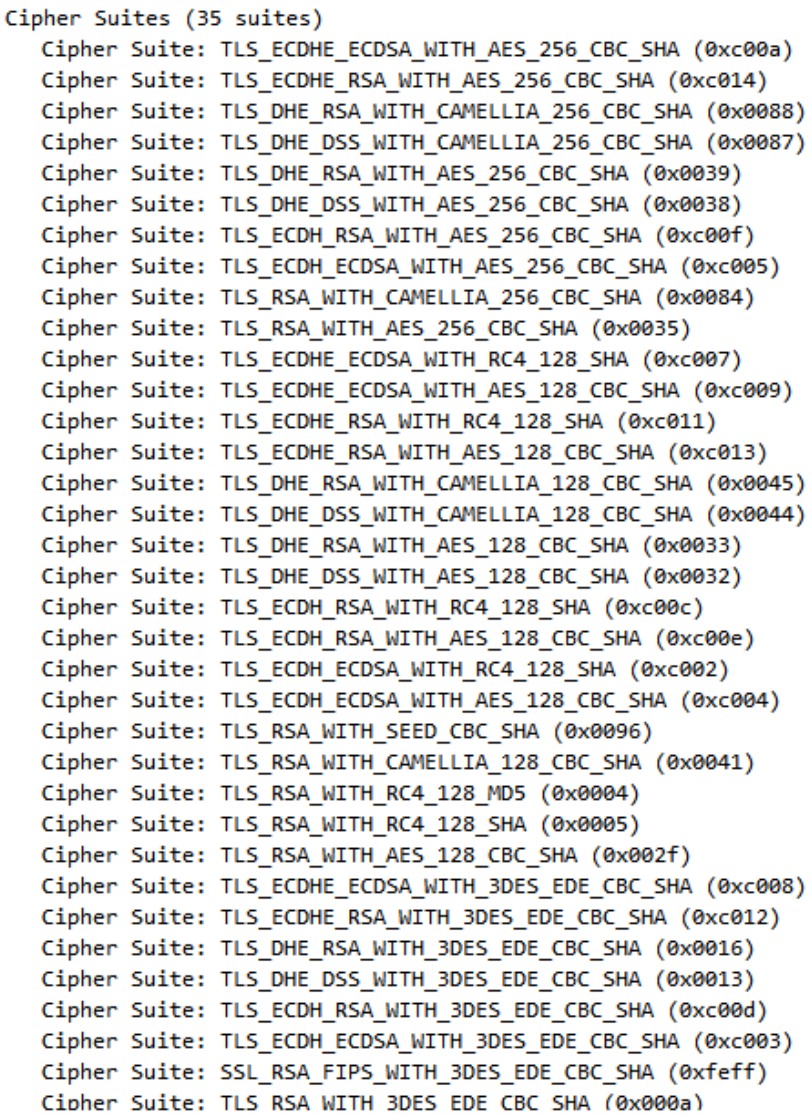
Vyzkoušejte.

2.1.8 Porovnání kryptosad

Jsou dány dvě starší kryptosady (filtr pro Wireshark `tls.handshake.ciphersuite`). Logickou úvahou odhadněte, které sadě dal přednost Twitter a které přetížený Facebook. Porovnejte s kryptosadami používané malwarem (Belajová 2018).

```
Cipher Suites (12 suites)
   Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
   Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
   Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
   Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
   Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
   Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
   Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
   Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
   Cipher Suite: TLS_DHE_DSS_WITH_AES_128_CBC_SHA (0x0032)
   Cipher Suite: TLS_DHE_DSS_WITH_AES_256_CBC_SHA (0x0038)
   Cipher Suite: TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA (0x0013)
   Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
```

Obr. 2.1.8.1: Příklad návrhu starší kryptografické sady při přihlašování klienta na Twitter – používá se TLS (Transport Layer protokol), dále asymetrický algoritmus, symetrický algoritmus o dané velikosti klíče a v příslušném módu a hash algoritmus.



Obr. 2.1.6.2: Příklad návrhu starší kryptografické sady při přihlašování klienta na Facebook

Shrnutí a závěr

Tento metodický list je věnován úvodním hodinám problematiky kryptografie na střední škole se zaměřením na informatiku. Je třeba pokrýt celou širokou škálu mechanismů a jejich aplikací. Závěrem je třeba upozornit, že kryptografické nástroje se ve větších systémech nepoužívají osamoceně, nýbrž se různě kombinují a doplňují. Např. TOR používá symetrické šifrování (typicky AES), šifrování pomocí veřejného klíče (RSA nebo eliptické křivky) pro zajištění identity přenosového uzlu a ECDH pro potvrzování (Curve25519) a SHA-1 coby hash zajišťující kontrolu integrity.

Další závěrečná poznámka se týká generování pseudonáhodných čísel. Například aplikace Avast BreachGuard obsahuje generátor náhodných čísel, pokud důvěřujete Avastu, můžete generátor použít a následně někam uložit, např. do některého ze správců hesel typu LastPass. Složitější situace je při psaní různých aplikací, které pracují s inicializačními vektrory. Podle (Buchanan 2022) „generování náhodných čísel je zásadní pro jádro kybernetické bezpečnosti“, a proto vytvořené generátory náhodných čísel je nezbytné pečlivě testovat.

Při programování kryptografických aplikací po léta převládala volba jazyků C nebo C++, které sice poskytují kontrolu nad správou zdrojů (na rozdíl např. od Javy nebo Go), ale na druhé straně umožňují útočníkům využití slabin souvisejících s pamětí (např. práce s pamětí po jejím uvolnění – viz CWE-416: Use After Free (https://cwe.mitre.org/data/definitions/416.html). Dnes se problém pokouší vyřešit jazyk Rust; přehled jeho kryptoknihoven byl publikován v (Ugochi 2020).

Seznam použitých zdrojů

(Belajová 2017) BELAJOVÁ, Daniela. Analysis of SSL/TLS handshakes in malware encrypted traffic. Bakalářská práce Masarykova univerzita Brno 2018. Dostupné z: *https://is.muni.cz/th/ll69s/thesis.pdf*

(Bertoni 2011) BERTONI, Guido; DAEMEN, Joan; PEETERS, Michaël, VAN ASSCHE, Gilles. *The Keccak sponge function family — Specifications summary* [online]. 2011-01-27 [cit. 2012-10-05].

(Buchanan 2021) BUCHANAN, William J. Determine public key point for ECC with Kryptology. Asecuritysite.com Blog. (2021). Dostupné z: *https://asecuritysite.com/kryptology/ecc_simple*

(Buchanan 2022) BUCHANAN, William J. A Rusty Key: At The Core of Cybersecurity is Random Numbers. Feb 6 2022. Dostupné z: *https://medium.com/asecuritysite-when-bob-met-alice/a-rust-key-at-the-core-of-cybersecurity-is-random-numbers-d741d35ee56c*

(Kwiatkowski 2019) KWIATKOWSKI, Kris, FAZ-HERNÁNDEZ. Introducing CIRCL: An Advanced Cryptographic Library. 20. 06. 2019. Dostupné z: *https://blog.cloudflare.com/introducing-circl/*

(Matějíček, 2020) MATĚJÍČEK, Pavel. Šifrujeme s VeraCryptem. 29.11.2020. *https://spajk.cz/sifrujeme-s-veracryptem/*

(NIST 2017) Digital Identity Guidelines Authentication and Lifecycle Management. Graffi, P. A. etc. NIST 2017. Dostupné z: *https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf*

(NIST 2019) Draft NIST Special Publication 800-186. Lily Chen, Dustin Moody, Andrew Regenscheid, Karen Randall. Recommendations for Discrete Logarithm-Based Cryptography: Elliptic Curve Domain Parameters. NIST October 2019. Dostupné z: *https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-186-draft.pdf*

(NUKIB 2018) Minimální požadavky na kryptografické algoritmy, doporučení v oblasti kryptografických prostředků. Verze 1.0, platná ke dni 28.11.2018. Dostupné z: *https://www.nukib.cz/download/uredni_deska/Kryptograficke_prostredky_doporuceni_v1.0.pdf*

(Šaroun 2019) ŠAROUN, Petr. Ukryjte svá data aneb šifrování disku s VeraCrypt a BitLocker. Pctuning 17. 4. 2019. Dostupné z: *https://pctuning.cz/article/ukryjte-sva-data-aneb-sifrovani-disku-s-veracrypt-a-bitlocker?chapter=7*

(Popelová 2018) POPELOVÁ, Lucie. Metody post-kvantové kryptografie. Bakalářská práce VUT Brno, 2018. Dostupné z: *https://www.vut.cz/www_base/zav_prace_soubor_verejne.php?file_id=175435*

(Stevens 2017) STEVENS, Marc. The First Collision for Full SHA-1. CRYPTO 2017**.** Lecture Notes in Computer Science book series (LNCS, volume 10401) pp 570-596. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-319-63688-7_19

(Špaček 2019) ŠPAČEK, Michal. Změna hashování existujících hesel. 2. srpna 2019. Dostupné z: *https://www.michalspacek.cz/zmena-hashovani-existujicich-hesel*

(Ugochi 2020) UGOCHI, Ukpai. Rust cryptography libraries: A comprehensive list. LogRocket 2020. Dostupné z: *https://blog.logrocket.com/rust-cryptography-libraries-a-comprehensive-list/*