Asymetrická kryptografie

Gymnázium J. V. Jirsíka, České Budějovice, Fráni Šrámka 23

Maturitní práce

Dennis Pražák

únor 2019

Čestné prohlášení

Prohlašuji, že předloženou práci jsem vypracoval samostatně, pouze za užití zdrojů uvedených v závěru práce.

V Českých Budějovicích dne ……

Poděkování

Děkuji RNDr. Leně Kolářové za odborné vedení práce a podnětné připomínky, které mi během vytváření práce poskytovala.

Anotace

V této práci je vysvětlen a prakticky implementován model asymetrického šifrování a porovnán s jinými běžně používanými modely. Výsledkem je program, který uživateli umožní generovat klíčové páry, šifrovat, dešifrovat, podepisovat a ověřovat podpisy.

Asymetrická kryptografie nás obklopuje, ačkoli si to mnozí z nás neuvědomují. Od přístupu k webům, přes populární komunikační mobilní aplikace, až po ransomware – většina dnešního šifrování probíhá se zapojením asymetrické kryptografie.

Znalost matematických principů, na kterých se tato šifra staví, umožní zamyslet se a určit adekvátní množství důvěry, která do ní bude vložena.

Obsah

1 Úvod

Asymetrická kryptografie je matematický model šifrování libovolného typu dat. Základním principem je to, že se jiným klíčem data šifrují a jiným dešifrují, přičemž jeden klíč je prakticky nemožné odvodit z druhého.

Tento princip byl vymyšlen dvěma kryptografy – Whitfieldem Diffiem a Martinem Hellmanem v roce 1976, když předvedli techniku, která se nadále začala označovat jako Diffieho-Hellmanova výměna klíčů. [1]

Problém kryptografie do té doby spočíval zejména v tom, že neexistoval bezpečný způsob, jak sdílet s druhým koncem komunikace klíč (tzv. sdílené tajemství). Když na daném komunikačním kanále totiž někdo odposlouchával, zachytil i klíč, který pak mohl použít nadále a porozumět tak naprosto veškeré komunikaci. Proto se dříve používala technika *pre-shared key*, tj. technika před-sdíleného klíče. Klíč se nejdříve mezi participanty komunikace vyměnil bezpečným způsobem, například i fyzickým setkáním.

Diffieho-Hellmanova výměna klíčů je revoluční v tom, že k takovému před-sdílení vůbec nemusí dojít. Více o Diffie-Hellmanově výměně bude nastíněno v sekci 2.3. Na základě tohoto nápadu přišli v roce 1977 tři profesoři z MIT na způsob šifrování, který byl tak významný, že se používá dodnes. V roce 1978 vydali práci, ve které byl tento způsob popsán, a která změnila celý svět kryptografie. Těmito profesory byli Ron Rivest, Adi Shamir a Leonard Adleman. Jejich principy jsou dnes známy jako RSA (zkratka vznikla z počátečních písmen příjmení těchto profesorů). [2]

RSA spočívá v tom, že každý uživatel má vlastní klíčový pár, který si sám vytvoří. Ten sestává z veřejného a soukromého klíče. Veřejný klíč může (a dokonce by měl) vystavit na veřejně dostupné místo (ověřený osobní web, klíčový server…). Kdokoli pak potřebuje uživateli poslat zašifrovanou zprávu může tento veřejný klíč použít. Zprávy zašifrované veřejným klíčem lze dešifrovat jen klíčem soukromým, který má uživatel pečlivě uschovaný.

Pro snazší pochopení lze použít analogii s visacím zámkem. Veřejný klíč je jako otevřený visací zámek, ke kterému má klíč jen jediná osoba (ta, která zámek s klíčem vytvořila – uživatel). Otevřené visací zámky lze rozeslat ostatním. Kdokoli pak může takový zámek použít

a zamknout s ním zprávu. Jen majitel klíče tuto zprávu dokáže přečíst, protože jen on má klíč od zamčeného zámku.

Podobným způsobem funguje i podepisování a ověřování. Uživatel si vytvoří zámek a klíč. Tentokrát však všem rozešle svůj klíč a zámek ponechá v tajnosti. Uživatel poté může zamknout nějakou zprávu (např. smlouvu, kterou podepisuje) tímto zámkem. Ostatní si můžou ověřit pravost podpisu tím, že se pokusí tento zámek odemknout veřejně dostupným klíčem od uživatele.

Dnes se již aktivně RSA tolik nepoužívá. Pasivně je však velmi rozšířené. Pokud uživatel webového prohlížeče navštíví webovou stránku pomocí protokolu HTTPS, je tato stránka asymetricky podepsána. Webový prohlížeč pak vyzkouší ověřit podpis (podle analogie výše – odemknout zámek) pomocí ověřeného a veřejně dostupného certifikátu (klíče) od určité autority. Pokud podpis sedí, uživatel si může být jist, že tento web opravdu pochází od jeho poskytovatele, a že nebyl pozměněn na cestě od serveru k uživateli. To je možné díky tomu, že odpovídající soukromý klíč zná jen a pouze server.

Existuje mnoho programů, které poskytují RSA pro běžné uživatele. GPG (GNU Privacy Guard) je původně program pro systém Linux (pro Windows existuje port jménem Gpg4win), který se ovládá pomocí příkazové řádky. Protože je rozšířený a má volně dostupný zdrojový kód, jedná se o takřka standard klientské asymetrické kryptografie. Práce s příkazovou řádkou ale nemusí být pro všechny příjemná. Naštěstí existují moderní alternativy, které většinou na GPG staví.

Enigmail je rozšířením pro e-mailový klient Thunderbird, který používá GPG, ale o velkou část celého procesu se postará za uživatele. Při psaní e-mailu stačí zaškrtnout možnost *zašifrovat*. Adresátovi se zpráva automaticky dešifruje, pokud má doplněk také nainstalován. Je samozřejmě stále potřeba nějakým způsobem vyměnit své veřejné klíče. Pro zajištění soukromí by však k výměně nemělo dojít e-mailem, kvůli možnosti útoku způsobem *man-in-the-middle*. Ledaže mezi účastníky funguje tzv. *síť důvěry*, což je řešení vytvořené programem GPG. V jiném případě by se klíče měly veřejně vystavit.

Velmi čerstvým řešením je Keybase. Jedná se o repozitář online identit, kde se každý může zaregistrovat, nechat kryptograficky ověřit své identity na sociálních sítích a nahrát své klíče. Kdokoli si pak daného uživatele může najít, ověřit a použít jejich veřejný klíč. Vše zjednodušuje desktopová aplikace Keybase, která vypadá jako běžný textový komunikační program. Uživatel si zvolí kontakt a může si s ním začít dopisovat. Vše je v zákulisí ověřeno

a zašifrováno.

Problém tohoto programu je ale to, že klíče musí být nahrány na server Keybase. Jeho uživatelé mu tak musí plně důvěřovat, což není moudré. Vždy může dojít k útoku a úniku klíčů. Uživatel asymetrické kryptografie by měl důvěřovat jen sám sobě, když se jedná o bezpečnost soukromých částí jeho klíčových párů.

Dalším problémem je, že desktopová aplikace Keybase všechna ověření a šifrování schovává do pozadí. Uživatel by ale měl alespoň zčásti mít kontrolu nad tím, co se s jeho zprávou děje.

Tyto problémy se bude práce snažit vyřešit v praktické části, kde bude implementována desktopová aplikace pro práci s asymetrickou kryptografií. Navíc implementuje vlastní způsob vytváření klíčů a jejich použití. Jelikož je její zdrojový kód volně dostupný, vše je naprosto transparentní. Uživateli bude také umožněno použít jiné (například systémové) implementace.

2 Matematický model

V této kapitole bude popsán matematický model pro generování klíčů, šifrování, dešifrování, podepisování a ověřování pomocí principů RSA.

RSA je založeno na existenci tzv. jednosměrných funkcí. To jsou takové funkce

$$f(x, \dots) = y\,,$$

že pro známé $x, \dots$ je snadné vypočítat hodnotu y, ale pokud známe pouze y, je prakticky nemožné najít vstupní hodnoty. Nejjednodušším příkladem je násobení prvočísel. Mějme prvočísla p a q, potom $p \cdot q = n$. Pro velká n je velmi obtížné[1] zjistit původní prvočísla.

RSA využívá určité funkce, která je jednosměrná jen a pouze tehdy, když není známá určitá informace, která při výpočtu pomůže. U předchozího příkladu to může být například znalost jednoho z prvočísel p a q. Pokud je známo jedno prvočíslo, výpočet druhého je jednoduchý: $q = n/p$.

2.1 Modulární aritmetika

Tabulka 1: Ukázka ekvivalenčních tříd modula 13

a	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	65	66	67	68	69	70	71	72	73	74	75	76	77
	52	53	54	55	56	57	58	59	60	61	62	63	64
	39	40	41	42	43	44	45	46	47	48	49	50	51
	26	27	28	29	30	31	32	33	34	35	36	37	38
	13	14	15	16	17	18	19	20	21	22	23	24	25
e	0	1	2	3	4	5	6	7	8	9	10	11	12

RSA využívá pro všechny operace z repertoáru RSA právě modulární kongruenci a vlastnosti modulární inverze.

[1] *velmi obtížné* zde znamená, že tato operace má vysokou časovou složitost

Modulární kongruenci lze definovat následovně

$$a \equiv b \,(\text{mod}\, m) \iff mk + b = a; k \in \mathbb{Z}\,, \tag{1}$$

kde m se nazývá *modulus*. Jinak řečeno, $m \mid (a - b)$. To ale nemusí vždy znamenat, že $a/m = k$, zbytek b. Modulo není ekvivalentní s dělením se zbytkem, protože má jiné výsledky, například když $a < 0$. Navíc modulární kongruence nemá jeden správný výsledek (nejedná se totiž o operaci, ale o kongruenci), protože možných b je pro každý jeden možný pár a, m nekonečně mnoho. Pokud b je nejmenší možné nezáporné, nazývá se *ekvivalenční třída*. Ukázka ekvivalenčních tříd e pro argument a je vidět v tabulce 1.

Například tyto kongruence platí:

$$45 \equiv 6 \,(\text{mod}\, 13)\,,$$
$$45 \equiv 19 \,(\text{mod}\, 13)\,,$$
$$45 \not\equiv 7 \,(\text{mod}\, 13)$$

a zároveň 6 je ekvivalenční třída 45 modulo 13.

Modulární kongruence má mnoho užitečných vlastností, které lze jednoduše dokázat, pomocí její algebraické definice. Určité vlastnosti jsou vhodné při strojovém výpočtu modulární kongruence pro velká a.

$$a + b \equiv [(a \bmod m) + (b \bmod m)] \,(\text{mod}\, m)\,.$$

Důkaz. Nechť $a \equiv c \,(\text{mod}\, m)$ a $b \equiv d \,(\text{mod}\, m)$, kde c a d jsou ekvivalenční třídy. Větu lze potom zjednodušit na $a + b \equiv c + d \,(\text{mod}\, m)$.

$$\begin{aligned} & mi + c = a \ \wedge mj + d = b \\ \implies & mi + mj + c + d = a + b \\ \implies & (i + j)m + c + d = a + b; (i + j) \in \mathbb{Z} \\ \implies & a + b \equiv c + d \,(\text{mod}\, m) \text{ dle definice.} \end{aligned}$$

□

Podobným způsobem lze triviálně dokázat i následující kongruenci

$$a \cdot b \equiv [(a \bmod m) \cdot (b \bmod m)] \pmod{m}. \tag{2}$$

To umožní při výpočtu rozdělit a na násobek, nebo součet, či jejich kombinaci a tím ho zjednodušit.

V RSA je také využita modulární exponenciace, což je výraz ve tvaru $a^e \equiv b \pmod{m}$. Protože v RSA se při modulární exponenciaci používají relativně velké exponenty, bylo potřeba vymyslet způsob, jak výpočet zjednodušit. K tomu lze aplikovat modulární multiplikaci z věty 2. Mocnění je n-krát opakované násobení. Při modulární exponenicaci lze tedy použít mezivýsledky a velmi tak ušetřit paměť. Již tato úvaha optimalizuje úkon modulární exponenciace. Existuje ještě mnoho dalších účinných způsobů optimalizace, jež jsou mimo rozsah této práce.

Další definice, která je esenciální pro RSA je modulární inverze. Inverze je častý matematický nástroj. Jestliže existuje inverze n^{-1} čísla n, potom $n \cdot n^{-1} = 1$. Jestliže existuje inverze $\mathbf{A}^{-1}$ matice $\mathbf{A}$, potom $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{E}$, kde $\mathbf{E}$ je jednotková matice. V modulární aritmetice existuje inverze také – jestliže existuje inverze a^{-1} argumentu a modulární kongruence, potom $a \cdot a^{-1} \equiv 1 \pmod{m}$. Horní index -1 zde neznačí exponent, ale obecně inverzi.

Například $4 \equiv 4 \pmod{13}$ má inverzi 10, protože $4 \cdot 10 \equiv 1 \pmod{13}$.

K nalezení modulární inverze lze použít buď rozšířený Euklidův algoritmus, anebo určitý aritmetický trik, který bude popsán v praktické části práce.

2.2 Eulerova funkce

Eulerova funkce $\varphi(n)$ je počet čísel $k \in \mathbb{N}$ takových, že $1 \leq k \leq n$ a zároveň nejvyšší společný dělitel k a n je 1. Neformálně řečeno – je to počet čísel menších než n a nesoudělných s n. [3]

Eulerova funkce má pro RSA dvě důležité vlastnosti:

1. Jedná se o funkci multiplikativní, neboli $\varphi(nm) = \varphi(n) \cdot \varphi(m)$. [3]
2. Jestliže p je prvočíslo, pak $\varphi(p) = p - 1$. To vychází ze základní vlastnosti prvočísel – všechna přirozená čísla kromě jedné, která jsou menší než prvočíslo jsou s prvočíslem nesoudělná.

Pokud se tyto dvě vlastnosti zkombinují, vznikne jednosměrná operace se „zadními vrátky“.

Zvolí-li se totiž dvě dostatečně velká prvočísla p a q, spočítá se $\varphi(n), n = pq$ takto:

$$\varphi(pq) = \varphi(p) \cdot \varphi(q) = (p-1) \cdot (q-1)\,.$$

Není-li znám rozklad n na prvočísla p a q, výpočet $\varphi(n)$ začne být u vysokých čísel velmi zdlouhavý.

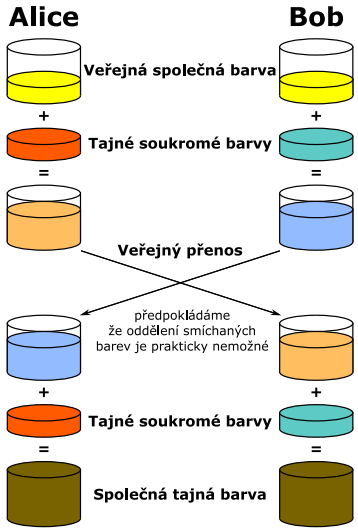
Tato funkce má ještě jednu významnou vlastnost, která je spojená s modulární exponenciací. Eulerova věta říká, že pro každé $n, a \in \mathbb{N}$, kde a je nesoudělné s n (nejvyšší společný dělitel je 1) platí: [4]

$$a^{\varphi(n)} \equiv 1 \pmod{n}\,. \tag{3}$$

2.3 Diffieho-Hellmanova výměna klíčů

Diffieho-Hellmanova výměna klíčů je technika pro tvorbu protokolu výměny klíčů z roku 1976 [1]. Jelikož na ní bylo později postaveno fungování RSA, bude zde krátce popsán její princip. Běžně se jako analogie ke konkrétní matematické implementaci používá subtraktivní míchání barev, které zde odpovídá jednosměrné operaci.

Alice a Bob chtějí bezpečně komunikovat. Musejí se dohodnout na sdíleném tajemství (kryptografickém klíči), ale nemohou ho sdílet přes komunikační kanál, protože na něm naslouchá Eva.



Obrázek 1: Diffieho-Hellmanova výměna klíčů znázorněna pomocí míchání barev (plná velikost v příloze)

Autor: A.J. Han Vinck, Univerzita Duisburg-Essen, volné dílo

Alice s Bobem se domluví na veřejné společné barvě, například *žluté*. Každý si zvolí svou soukromou barvu. Alice si zvolí například *oranžovou* a Bob si zvolí *zelenou*. Alice a Bob smíchají své soukromé barvy s veřejnou. Alice získá směs *žluté* a *oranžové* (vznikne *béžová*). Bob získá směs *žluté* a *zelené* (vznikne *modrá*). Výslednou směs si navzájem pošlou. K těmto směsím přidají své soukromé barvy. Alice přidá k Bobově směsi (*modré*) svou soukromou barvu (*oranžovou*) a vznikne směs *žluté*, *zelené* a *oranžové* (*hnědá*). Bob přidá k Alicině směsi svou soukromou *zelenou* barvu a vznikne směs *žluté*, *zelené* a *modré* (*hnědá*). Nyní mají Alice a Bob sdílené tajemství – *hnědou* barvu.

Naslouchající Eva získá jen původní barvu (*žlutou*) a následné směsi (*béžovou* a *modrou*). Pokud považujeme rozdělení barev za prakticky nemožné, Eva nikdy nezjistí sdílené tajemství (*hnědou* barvu). V praxi se místo míchání barev používá modulární exponenciace.

Tuto techniku je možné použít i při komunikaci mezi více než dvěma konci. Modulární exponenciace totiž pracuje s jedním až nekonečně mnoha exponenty.

Diffieho-Hellmanova výměna klíčů je i dnes velmi často používanou technikou, která (nebo výměna na ní založená) se používá v mnoha praktických aplikacích, kde je potřeba často a bezpečně komunikovat. Konkrétním příkladem může být mobilní komunikační aplikace WhatsApp [5] nebo protokol MTProto aplikace Telegram [6].

2.4 RSA

V této sekci budou popsány operace RSA a jejich matematický princip.

Šifrování: Bob posílá tajnou zprávu Alici. Pro tento účel si Alice vygeneruje klíčový pár. Ten se vždy skládá ze soukromého a veřejného klíče (*private & public key*). Veřejný klíč může libovolně sdílet s kýmkoli. Soukromý klíč nesmí znát nikdo jiný než Alice, jinak je klíčový pár kompromitován. Bob zašifruje zprávu pro Alici jejím veřejným klíčem a výslednou šifru pošle Alici. Jen Alice zná soukromý klíč a jen soukromým klíčem lze šifru dešifrovat.

Podepisování: Bob chce poslat zprávu Alici a zároveň chce zajistit, aby si mohla Alice ověřit, že zprávu opravdu poslal Bob. Bob si vytvoří klíčový pár. Zprávu podepíše (zašifruje) svým soukromým klíčem. Alice zná Bobův veřejný klíč a může jím tak ověřit (dešifrovat) zprávu. V tomto případě může zprávu dešifrovat kdokoli, protože Bobův veřejný klíč je veřejně znám.

Tyto dvě operace lze zkombinovat a získat tím tak při komunikaci jak bezpečnost, tak autenticitu zpráv.

Po celou tuto sekci platí, že

1. m je zpráva, kterou šifrujeme,
2. e je veřejný exponent,
3. c je šifra (zašifrovaná zpráva),
4. d je soukromý exponent (tajná informace),
5. p a q jsou libovolná různá prvočísla,
6. $p \cdot q = n$.

Potom platí, že

$$m^e \equiv c \pmod{n}\,, \tag{4}$$

$$c^d \equiv m \pmod{n} \implies (m^e)^d \equiv m \pmod{n} \implies m^{ed} \equiv m \pmod{n}\,, \tag{5}$$

kde kongruence 4 odpovídá operaci šifrování a kongruence 5 dešifrování. Šifrování i dešifrování je tedy intuitivní, problém je už jen s vyhledáním takových čísel e a d, aby tyto kongruence platily. Ty platí v případě, že $ed \equiv 1 \pmod{n}$, neboli d je modulární inverze e. K výpočtu modulární inverze využijeme Eulerovu větu z kongruence 3, ale k tomuto účelu ji nejprve trochu rozšíříme. Předpokládejme, že $k \geq 1; k \in \mathbb{Z}$. Poté platí, že

$$\begin{aligned} m^{k\cdot\varphi(n)} &\equiv 1^k = 1 \pmod{n}\,, \\ m \cdot m^{k\cdot\varphi(n)} = m^{k\cdot\varphi(n)+1} &\equiv m \pmod{n}\,. \end{aligned}$$

Exponent $k \cdot \varphi(n) + 1$ je tedy roven ed. Soukromý exponent lze zjistit následovně

$$d = \frac{k \cdot \varphi(n) + 1}{e}\,. \tag{6}$$

Je vidět, že k výpočtu soukromého exponentu d je potřeba znát hodnotu $\varphi(n)$. Tu pro vysoká n ale prakticky nelze spočítat, pokud není znám rozklad n na prvočísla. Prvním krokem při vytváření klíčového páru je tedy nalézt dvě vysoká prvočísla p a q a spočítat hodnotu Eulerovy funkce v bodě jejich součinu n.

Nyní je také již zřejmé, proč bylo potřeba zavést koeficient k. Soukromý exponent d se totiž spočítá dělením, ovšem pro použití modulární exponenciace musí být d celé číslo. Při implementaci se tedy musí najít k takové, aby e dělilo beze zbytku čitatel zlomku.

Takto je vytvořen klíčový pár. Sestává ze soukromé části (d) a veřejné části (e, n).

2.5 Příklad použití RSA s konkrétními čísly

V této sekci bude názorně ukázán konkrétní příklad šifrování a dešifrování pomocí RSA na malých číslech.

Bob posílá tajnou zprávu Alici a Eva naslouchá na stejném komunikačním kanále. Alice najde dvě prvočísla $p = 163$, $q = 181$ (v praxi se používají velmi vysoká prvočísla). Potom tedy $n = pq = 29503$. Alice jednoduše spočítá $\varphi(n)$, protože

$$\varphi(n) = \varphi(pq) = \varphi(p) \cdot \varphi(q) = (p-1) \cdot (q-1) = 162 \cdot 180 = 29160\,.$$

Poté si Alice zvolí veřejný exponent $e = 7$, který nemá žádného společného dělitele s $\varphi(n)$. To proto, aby se dala nalézt modulární inverze. Nakonec vypočítá svůj soukromý exponent d pomocí vzorce

$$d = \frac{k \cdot \varphi(n) + 1}{e} = \frac{4 \cdot 29160 + 1}{7} = 16663\,,$$

kde k byl zvolen tak, aby d bylo celé číslo.

Alicin soukromý klíč je informace $d = 16663$. Veřejný klíč jsou informace $e = 7$ a $n = 29503$. Veřejný klíč pošle Alice Bobovi přes komunikační kanál. Tyto informace tedy zachytí i Eva. Nyní mohou Bob i Eva šifrovat zprávy pro Alici, ale nikdo jiný je nemůže dešifrovat než Alice.

Bob chce zašifrovat zprávu $m = 42$ pro Alici. Šifru vypočte modulární exponenciací

$$\begin{aligned} m^e &\equiv c \pmod{n}\,, \\ 42^7 &\equiv 29457 \pmod{29503}\,, \\ c &= 29457\,. \end{aligned}$$

Bob pošle Alici šifru 29457. Tu zachytí i Eva, ale Eva nemá soukromý klíč, takže nikdy nezjistí hodnotu původní zprávy m. Když šifra přijde Alici, může spočítat původní zprávu

pomocí modulární exponenciace

$$m^d \equiv m \pmod{n},$$
$$29457^{16663} \equiv 42 \pmod{29503},$$
$$m = 42.$$

Alice získala původní zprávu m. Může se zdát že výpočet 29457^{16663} mod 29503 je složitý, ale jak již bylo nastíněno v sekci 2.1, existuje mnoho optimalizací, díky kterým se tato operace stává relativně paměťově a časově nenáročnou.

Protože Eva zná jen n, a ne jeho rozklad na prvočísla p a q, nemůže zprávu c dešifrovat. Musela by totiž spočítat $\varphi(n)$, což je pro vysoká n velmi zdlouhavé. V praxi je n 1024-bitové číslo (nejméně $\lfloor \log 2^{1023} + 1 \rfloor = 308$ decimálních cifer). Pro přímý výpočet $\varphi(n)$ by bylo potřeba vykonat přibližně 10^{153} operací. Rychlejší je rozložit n na prvočísla a poté spočítat $\varphi(n) = (p-1)(q-1)$. Pro rozklad n na prvočísla, kde $n > 10^{100}$ je nejefektivnější algoritmus GNFS, jehož složitost je [7]

$$\mathcal{O}\left(e^{\sqrt[3]{\frac{64}{9} b \log^2 b}}\right),$$

kde b je počet bitů čísla n. Pro 1024-bitové n by bylo potřeba asi 10^{30} operací, což by s dnešním výkonem běžného počítače trvalo asi $32 \cdot 10^{12}$ let.

3 Implementace Hitai

V této praktické části práce bude popsána implementace vlastního programu s podporou vytváření a správy klíčů, šifrování, dešifrování, podepisování a ověřování kryptografických podpisů. Zdrojový kód je dostupný na adrese `https://github.com/sorashi/hitai`.

Hitai byl implementován pro platformu .NET 4.5 v jazyce C#, což je objektově orientovaný, typově zabezpečený, moderní jazyk s částečnou podporou funkcionálního programování v podobě technologie LINQ. Pro tvorbu uživatelského rozhraní byla použita technologie Windows Forms (také známá jako WinForms).

Projekt je distribuován pod licencí MIT a jeho zdrojový kód je volně dostupný na webu GitHub. To umožňuje komukoli navrhovat změny, nahlašovat chyby, anebo dokonce založit vlastní projekt na základě Hitai.

3.1 Použité technologie

Jako vývojové prostředí bylo použito Visual Studio 2019 Preview 2. Visual Studio bylo původně integrovaným vývojovým prostředím jen pro technologii .NET, ale s postupem času Microsoft ve svém produktu začal plně podporovat mnoho jiných technologií a platforem. Příkladem mohou být platformy Python a Node.js. Ve středu pozornosti však zůstalo .NET a zvláště pak jazyk C#, který se za posledních 10 let rychle vyvíjel. To mohlo být mimo jiné zapříčiněno migrací společnosti Microsoft k open-source politice. Technologie .NET Core je plně open-sourcová a multiplatformní implementace platformy .NET (k dispozici na webu GitHub pod adresou dotnet/core). Roslyn je open-source kompilér a analyzátor kódu pro C# (na GitHub pod adresou dotnet/roslyn).

V integrovaných vývojových prostředích jsou nástroje pro debugging samozřejmostí. Visual Studio umožňuje také navigovat pozastavený program, spouštět instrukce řádku po řádce (a to dokonce i když tok kódu vstoupí do jiného vlákna), pokládat podmíněné breakpointy a sledovat nebo měnit hodnoty proměnných za běhu. Zabudovaný C# REPL lze použít pro experimentování s API aktuálního řešení. Užitečná je také podpora pro *unit testing* a Git.

```
[Test]
public async Task TransformAsyncTest()
{
  var cleartext = "Lorem ipsum dolor sit amet consequer";
  var password = "heslo";
  aes.GenerateKeyFromPassword(password);
  var salt = new byte[aes.Salt.Length];
  var iv = new byte[aes.IV.Length];
  Array.Copy(aes.Salt, salt, salt.Length);
  Array.Copy(aes.IV, iv, iv.Length);
  byte[] cipher = await aes.TransformAsync(
    Encoding.ASCII.GetBytes(cleartext));
  aes = new AesSymmetricEncryptionProvider();
  aes.GenerateKeyFromPassword(password, salt);
  aes.IV = iv;
  var result = Encoding.ASCII.GetString(
    await aes.TransformAsync(cipher, false));
  Assert.AreEqual(cleartext, result);
}
```

Obrázek 2: Ukázka testu z unit testů projektu Hitai

Tento test vyzkouší, zda funguje implementace symetrické kryptografie podle očekávání. Očekávané chování se vyjadřuje pomocí *asercí* – tj. tvrzení (viz poslední řádka). V tomto případě test tvrdí, že zašifrovaný a následně dešifrovaný text je stejný, jako text původní. Pokud tvrzení není pravdivé, test selže.

Visual Studio je rozšiřitelné pomocí doplňků. Při vývoji Hitai byl použit proprietární doplněk ReSharper od české firmy JetBrains. Ten přidává mnoho funkcí a možností, například nastavení a zajištění jednotného stylu kódu. Celý kód je potom konzistentní a lze ho jednodušeji spravovat. Disponuje také mnoha funkcemi pro refaktorování, zrychluje psaní kódu pomocí šablon a *postfixů*.

Při sestavování projektu je důležité otestovat, zda jeho chování odpovídá daným předpokladům. K tomu byl použit unit-testovací framework NUnit 3. S jeho pomocí lze napsat kód v jazyce C#, který zkontroluje tyto předpoklady. Příklad takového testu je vidět na obrázku 2.

Pro konfiguraci sestavení kódu byl použit projekt Cake (C# make), který se snaží o napodobení funkce příkazu a souborů *make* z prostředí Linuxu. Cake umožňuje napsat sestavovací skript v jazyce C#, který sestaví projekt, otestuje ho pomocí unit-testů, zabalí ho a případně distribuuje. To všechno lze poté spustit jedním příkazem, a proto si kdokoli může Hitai sám jednoduše sestavit na vlastním zařízení. Jednotný sestavovací skript také umožňuje jednoduché použití služeb *průběžné integrace*. Prostředí Cake definuje mnoho globálních funkcí a tzv. *builderů* (návrhový vzor), takže skripty jsou velmi přehledné a pochopitelné, viz obrá-

```
Task("Unit-Tests")
  .IsDependentOn("Build")
  .Does(() => {
      NUnit3("./**/bin/" + configuration + "/*.Test.dll",
        new NUnit3Settings { NoResults = false });
      if(AppVeyor.IsRunningOnAppVeyor){
          AppVeyor.UploadTestResults("TestResult.xml",
            AppVeyorTestResultsType.NUnit3);
      }
  });
```

Obrázek 3: Ukázka úlohy ve skriptu Cake

Skript definuje úlohu `Unit-Tests`, která je závislá na sestavení (tj. před otestováním musí být projekt sestaven).

zek 3. Cake je open-source a zdarma, distribuován pod licencí MIT.

Při sestavování aplikace je také použit nástroj Fody, který umožňuje do procesu vytváření binárních souborů ze zdrojového kódu „protkat“ (*weave*) různé doplňky. Hitai používá doplněk Costura, který zabalí všechny závislosti projektu do jedno výstupního souboru. Fody i Costura jsou open-source a dostupné na webu GitHub.

Pro průběžnou integraci (CI – continuous integration) byla použita služba AppVeyor, která ji poskytuje zcela zdarma pro open-source projekty, kterým Hitai je. V praxi to vypadá tak, že jakmile se v kódu něco změní, AppVeyor je upozorněn a spustí virtuální stroj, ve kterém je Hitai sestaven, otestován a případně distribuován, a to vše pomocí jednoho příkazu díky Cake. CI umožňuje vývojářům v open-source komunitě efektivně spolupracovat a rychle zachytávat chyby, protože při neúspěšném sestavení nebo testování jsou upozorněni na webu GitHub, případně e-mailem. Sestavení je také spuštěno při návrhu změn v repozitáři (*pull request* – žádost o natažení změn) a tak navrhovatelé vědí, kdy jejich změny správně fungují. Díky tomu a mnoha dalším opatřením efektivně funguje vývoj extrémně velkých a aktivních open-source projektů, jako je například torvalds/linux nebo microsoft/vscode, kde se průměrně každých 5 minut objeví nový návrh nebo nahlásí nová chyba.

Samozřejmostí je použití verzovacího systému (VCS), kterým byl v případě Hitai zvolen Git. Ten se stará o zachování veškeré historie změn ve formě balíků změn (tzv. commitů), poskytuje možnost kdykoli se vrátit do bodu v minulosti nebo například obrátit jeden commit. Je esenciánlím nástrojem pro spolupráci, protože využívá princip větveného vývoje (tzv. branching), kdy se historie může rozvětvovat do různých směrů a poté se dají tyto větve sloučit (tzv. merge), nebo lze commity jedné větve postupně naskládat na jinou větev (tzv. rebase).

To není zdaleka vše, co Git umožňuje a v čem vývojářům pomáhá. Zároveň jsou jeho principy založeny na grafové teorii, takže kdokoli se základní znalostí této oblasti teoretické informatiky rychle pochopí, jak Git funguje. Je samozřejmě také zdarma a open-source.

Jako způsob instalace a aktualizace programu Hitai byl použit projekt Squirrel, jehož se v moderních desktopových aplikacích často využívá (zejména v těch založených na technologii Electron – například Slack, Discord a Skype). Když uživatel spustí instalační soubor, program je potichu na pozadí nainstalován, vytvoří zástupce na ploše a v menu Start a poté je ihned spuštěn. Uživatelská interakce je naprosto minimální. To samé platí i pro aktualizaci – při zjištění nové verze ji program automaticky stáhne, nainstaluje vedle bežící verze a nahradí všechny zástupce. Při dalším spuštění si uživatel mnohdy ani nevšimne, že aktualizace proběhla. Tohoto aktualizačního cyklu využívají také moderní webové prohlížeče Chrome a Firefox.

Tvorba loga (ikony) probíhala v programu Paint.NET od firmy dotPDN LLC, konkrétně ve vydání Classic, které je zdarma distribuováno pod vlastní licencí. Program umožňuje pracovat s rastrovou grafikou. Logo bylo exportováno ve formátu PNG a poté převedeno do formátu ICO pomocí software ImageMagick, který je open-source a zdarma, pod licencí Apache 2.0. Příkaz k převodu vypadá následovně

```
convert logo.png -define icon:auto-resize=256,64,32,16 logo.ico
```

3.2 Vývoj

Při vývoji byl kladen důraz na zachování defenzivní strategie programování – tj. při vstupu toku kódu do funkce zajistit všechny výjimky a předpoklady předem a až poté činit. To zajistilo vyšší přehlednost a spravovatelnost kódu.

Také byla zachována modularita – možnost rozšířit funkcionalitu programu. Toho bylo dosaženo pomocí rozhraní a dědičnosti. Při rozhodování o použití jednoho z těchto dvou způsobů vrstvení lze použít jednoduchou pomůcku – rozhraní existují pro sjednocení funkcionality objektů, zatímco třídy jako předpisy pro objekty. Při takovém rozhodování ale může nastat dilema. Například u společného předka pro poskytovatele asymetrické kryptografie lze použít obě přirovnání. Takový poskytovatel umí *poskytovat* asymetrickou kryptografii a jeho potomci jsou *poskytovateli* asymetrické kryptografie. Nakonec bylo v tomto případě zvoleno rozhraní, protože třída, která ho implementuje, může implementovat i rozhraní jiná. Lze na-

příklad vytvořit třídu, která poskytuje jak asymetrickou, tak symetrickou funkcionalitu. Toho lze dědičností dosáhnout jen velmi těžko, protože v jazyce C# neexistuje *mnohonásobná dědičnost*.

Při vývoji však nešlo vše podle plánu. Tím byl diagram hierarchie tříd, rozhraní a jejich vlastností. Při aplikaci této teorie však často vznikaly problémy, protože se nepočítalo s potřebou implementovat určitou funkcionalitu. Například po implementaci třídy Keypair, která drží informace o klíčovém páru, bylo zjištěno, že je potřeba zajistit to, aby uživatel při jeho použití vždy zadal heslo, zejména kvůli bezpečnosti. Bylo proto zajištěno, že soukromý exponent je v této reprezentaci *vždy* zašifrován. Při jeho extrakci z tohoto objektu je potom potřeba volat metodu `GetPrivateExponentAsync(string password)`, která ho symetricky dešifruje pomocí AES. Nemožnost použít soukromý exponent jinde, bez použití hesla částečně zkomplikovala hierarchii.

3.3 Modely

Hitai reprezentuje objekty jako jsou klíče, zprávy a podpisy pomocí tříd, které práce označuje jako modely. Každý model obsahuje jen potřebné vlastnosti a někdy i základní funkcionalitu, ale zachovává snadnou serializovatelnost. Díky tomu lze každý model převést do reprezentace standardu MessagePack. K tomu Hitai využívá knihovnu třetí strany dostupnou ze správce balíků NuGet pod jménem MessagePack (zdrojový kód dostupný pod licencí MIT). Tato knihovna poskytuje i kompresi pomocí algoritmu LZ4, takže je rychlá a efektivní (výstup je krátký).

Výstupem serializace MessagePack je pole bajtů. Pokud Hitai serializuje zprávu, řetězec bajtů je pro uživatele nevyhovující a těžko se s ním pracuje. Proto Hitai převede tyto bajty na textovou reprezentaci, které se říká armor.

Příkladem často používaného armoru ve světě asymetrické kryptografie je standard OpenPGP (používaný i v GPG), který lze vidět na obrázku 4. Hitai implementuje vlastní armor (ukázka na obrázku 5), který je v mnoha ohledech lepší pro moderní využití, než OpenPGP. Místo čtyřiašedesátkové soustavy používá dvaašedesátkovou. Tím se zamezuje nutnosti využít znaky jako je `=`, `/`, `+` anebo `-`, které někdy způsobují problémy s kompatibilitou ve webových prohlížečích. Hitai armor používá jen znaky `0-9`, `a-z` a `A-Z` (celkem 62 znaků). Další výhodou tohoto armoru je, že u něj není nutné používat znaky nové řádky, které se při vložení na web mnohdy ztratí, což je přirozené chování HTML. Hitai armor může být jen jedinou

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBFi+d9oBEADJaovjNabOnQLRqfevsrSFeMejGTGTdRm6O8MCoxoC/hTezzin
QknVvpKnCSNxdOLWm5rkomAl5IlzYfdWxjfMnFDEOFBUVOrC6VeMeNubA5dqBW5Q
fv2PsB+jT2/6YIM/GwBfSJXERpS9RzXr6o85TI3mXXa9eSM21rpMnGwpJCNJABnM
YMRIRf6q6iwogxm7//HKPMR9tuo0D2MAQoGZkAC12CqVYXUAL8gNl1iqNJXedBMp
LK6R7BtkCqxjCL4pw6mQElM1uG8sCpGXmhxSAq3L7DbE0Fz9E74zUso0bnwxadUx
ROM2jsRgcUJefM+wTWoC7ZjUO/6J9y3e773nXoTY5drkFp1wG/ZhOnIH3LX5Uoa2
kDMGDHzoK4L5XxNzdLhvnV9PCBE+dTMFQ14rGfm1KYVLPk9uCsh7bIXOIflkgC0O
Ore8CRAa3GTwrESXQysBBXYHLdltj7UR8rZVWPVIepLmH7HS18twzqyN10NiSvs=
=XLdm
-----END PGP PUBLIC KEY BLOCK-----
```

Obrázek 4: Ukázka formátu ASCII armoru standardu PGP (zkráceno)

```
BEGIN HITAI MESSAGE. 1IvuEbfX70oBsqG GULXj3I6X3VABuX h9yb6OWBODeATf1
KCcRaMCNhca6mle UubaE0NZsviARQ1 wQITZSeq2j0aIT7 fj0DZ2hUuaz2c3p
unrdGEzHwiFvHRu ZlVXcGtplDBEsuo zyBkpBCHyjjay7z euFFp7dAi3D0foI
21Jr3wIQdfA66AA 065m424EL2n7sdy dcUlrPjeSuhDJxv Qw5drLCoGUktkQd
17Uf4G4iogNBTDk tlNdKZ3y1Wdm1YV AsqcQ2FENJQEjoI
alkh7X2B1. END HITAI MESSAGE
```

Obrázek 5: Ukázka formátu ASCII armor Hitai

řádkou textu, která je proložena mezerami, díky čemuž se o správné zobrazení může postarat textový procesor, příp. webový prohlížeč pomocí běžného zalomování textu. Hitai armor totiž vypadá jako dlouhá věta o mnoha slovech.

Většina komponent řešení Hitai je modulární, tj. implementována pomocí rozhraní nebo abstraktních tříd. Ani armor není výjimkou, a proto lze jednoduše přidat vlastní.

Jedním z modelů je Keypair – klíčový pár. Ten udržuje veřejný exponent, modulus a symetricky zašifrovaný soukromý exponent. Generování klíčového páru mají vždy na starosti konkrétní implementace poskytovatelů asymetrické kryptografie. `HitaiAsymmetricProvider` je jedním z těchto poskytovatelů a využívá vlastní třídy `PrimeGenerator` k vytvoření tohoto páru. Generování probíhá v několika krocích. Postup algoritmu zde bude popsán slovně.

Nejprve jsou zvolena dvě náhodná lichá čísla o velikosti 512 bitů pomocí `RNGCryptoServiceProvider`, což je třída .NET, která poskytuje kryptograficky bezpečný generátor čísel (na rozdíl od třídy `Random`, jež je pseudonáhodný generátor založený na algoritmu Donalda E. Knutha). Zvolený generátor vyplní dvě pole náhodnými bajty. Poté bude použita třída `BigInteger`, pomocí které lze reprezentovat libovolně velké číslo. Protože bajty jsou

náhodné, může se stát, že číslo bude záporné. Jelikož `BigInteger` používá na reprezentaci dvojkový doplněk, lze zajistit nezápornost čísla nastavením MSB[1] na hodnotu 0 (pomocí operace `AND`).

Protože se hledají prvočísla, můžou se nechat vyloučit všechna sudá čísla, jelikož jsou dělitelná dvěma. Číslo je sudé, pokud má jeho LSB[2] hodnotu 0. Vždy se proto nastaví LSB na hodnotu 1 pomocí operace OR.

Nyní se zkontroluje, zda je dané číslo prvočíslem. Protože hustota prvočísel se s velikostí čísel logaritmicky zmenšuje (*prvočíselná věta* z teorie čísel), trvala by přímočará kontrola definitivní prvočíselnosti moc dlouho. Proto budou použity tři testy pravděpodobné prvočíselnosti. Pokud číslo některý z testů nesplní, bude přičteno 2 (tím bude získán další kandidát, protože je přeskočeno sudé číslo) a test bude spuštěn znovu.

Prvním testem je zkouška dělitelnosti prvními 400 prvočísly. Pokud některé z těchto prvočísel dělí kandidáta, kandidát není prvočíslem.

Druhým testem je tzv. Fermatův test pravděpodobné prvočíselnosti. Ten říká, že pokud se testuje číslo n a zvolí se libovolné číslo a, pro které platí

$$2 < a < n \wedge a^{n-1} \not\equiv 1 \pmod{n},$$

potom n určitě není prvočíslo. Čím více různých a je vyzkoušeno, tím větší je pravděpodobnost, že n je prvočíslo. Tato úvaha se zakládá na Eulerově větě z kongruence 3. Tento test je velmi rychlý, protože operaci modulární exponenciace lze velmi dobře optimalizovat, jak již bylo zmíněno v sekci 2.1.

Třetím testem je Millerův-Rabinův test pravděpodobné prvočíselnosti. Jedná se o rozšíření Fermatova testu, které je sice přesnější, ale má vyšší asymptotickou složitost. Millerův-Rabinův test říká, že pokud je n prvočíslo, lze $n - 1$ zapsat jako $2^s \cdot d$, kde d je liché (to vychází z rozkladu na násobek prvočísel). Pokud existuje a, takové, že platí

$$a^d \not\equiv 1 \pmod{n} \wedge a^{2^r d} \not\equiv -1 \pmod{n},$$

pro všechna $0 \leq r \leq s\ -1$, potom n není prvočíslo. Pokud nastane opačný případ, kandidát je s vysokou pravděpodobností prvočíslo. [8]

[1] z anglického *most significant bit* – bit nejvyššího řádu
[2] *least significant bit*

Tabulka 2: Statistika běhu algoritmu pro nalezení prvočísla (odpovídající grafy jsou v příloze)

Počet bitů čísla	64	128	256	512
Prům. neprošlo testem dělitelnosti prvních 400 prvočísel	2274	4785	9542	12386
Prům. nesplnilo Fermatův test	251	661	1439	1952
Prům. nesplnilo Millerův-Rabinův test	122	126	131	88
Prům. počet kandidátů před nalezením prvočísla	2648	5574	11113	14426
Prům. potřebný čas k nalezení prvočísla	424 ms	2 s	11 s	53 s
Počet testů	100	50	10	4

```
3,191,162,340,869,987,341,580,947,986,524,328,096,312,168,627,603,352,
  477,779,523,673,952,525,519,933,761,336,636,441,542,957,506,062,978,
  187,431,304,157,160,256,542,710,145,569,657,533,554,474,134,174,909
```

Obrázek 6: Jedno z nalezených prvočísel

Pokud kandidát projde všemi třemi testy, je s vysokou pravděpodobností prvočíslem. Časová statistika neparalelního běhu tohoto algoritmu je vidět v tabulce 2.

V Hitai běží algoritmus pro vyhledání prvočísel paralelně v tolika vláknech, kolik celkově jader má aktuální procesor.

Dalším krokem je vypočítat soukromý exponent d s pomocí vzorce 6. K tomu je nejprve potřeba vybrat veřejný exponent e tak aby byl nesoudělný s $\varphi(pq)$. Exponent e je vybrán ze seznamu běžně používaných veřejných exponentů: $3, 5, 17, 257, 65537$. Důvod použití těchto čísel je zřejmý – všechna lze snadno vyjádřit bitovými operacemi (např. `65537=1<<16|1`). Pravděpodobnost, že žádný z těchto exponentů nebude nesoudělný s $\varphi(pq)$ je extrémně nízká (cca $2,3 \cdot 10^{-8}$ %). Pokud k tomu však dojde, ukáže se chybová hláška a generování proběhne znovu. Poté se iterací nalezne koeficient k tak, aby platilo $e \mid (k \cdot \varphi(pq) + 1)$. Nakonec lze vypočítat d.

4 Závěr

Implementací RSA práce nastínila, jak funguje. Uvedla, na jakých matematických principech se tato šifra staví a poskytla zamyšlení nad tím, jak moc jí lze důvěřovat, když je využívána, ať už aktivně (Hitai, GPG), či pasivně (používání internetu). Pokud se někomu podaří prolomit tuto šifru, tak jen tehdy, dokáže-li rozložit velmi vysoké číslo na součin prvočísel. Možná se tak stane v době, kdy budou dostatečně vyvinuty kvantové počítače. Znamenalo by to konec bezpečnosti tak, jak ji dnes známe.

Hlavním výstupem práce je desktopový program Hitai, který je open-source a umožňuje uživateli vytvořit klíče, šifrovat, dešifrovat, podepisovat a ověřovat podpisy. Práce na tomto programu bude nadále aktivně pokračovat, jak je u projektů s otevřeným zdrojovým kódem přirozené.

Práce byla vysázena pomocí systému LaTeX. Všechny její části jsou přiloženy v nevysázené formě.

Seznam použitých zdrojů

[1] Diffie, W. a Hellman, M. New directions in cryptography. *IEEE Transactions on Information Theory*. Listopad 1976, roč. 22, č. 6. S. 644–654.

[2] Rivest, R. L., Shamir, A. a Adleman, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*. únor 1978, roč. 21, č. 2. S. 120–126.

[3] Long, C. T. *Elementary Introduction to Number Theory*. 2. vydání. 125 Spring Street, Lexington, Mass.: D.C. Heath and Company, 1972. ISBN 0669627038.

[4] Euler, L. *Commentarii Academiae scientiarum imperialis Petropolitanae*. Sv. 8. [b.m.]: Typis Academiae, 1741. Theorematum quorundam ad numeros primos spectantium demonstratio (Důkaz určitých vět týkajících se prvočísel), s. 141–146. Dostupné na: <http://www.17centurymaths.com/contents/euler/e026&54tr.pdf>.

[5] WhatsApp. *WhatsApp Encryption Overview*. Duben 2016. Techreport. Dostupné na: <https://www.whatsapp.com/security/WhatsApp-Security-Whitepaper.pdf>.

[6] *MTProto, mobilní protokol aplikace Telegram*. Dostupné na: <https://core.telegram.org/mtproto/description>.

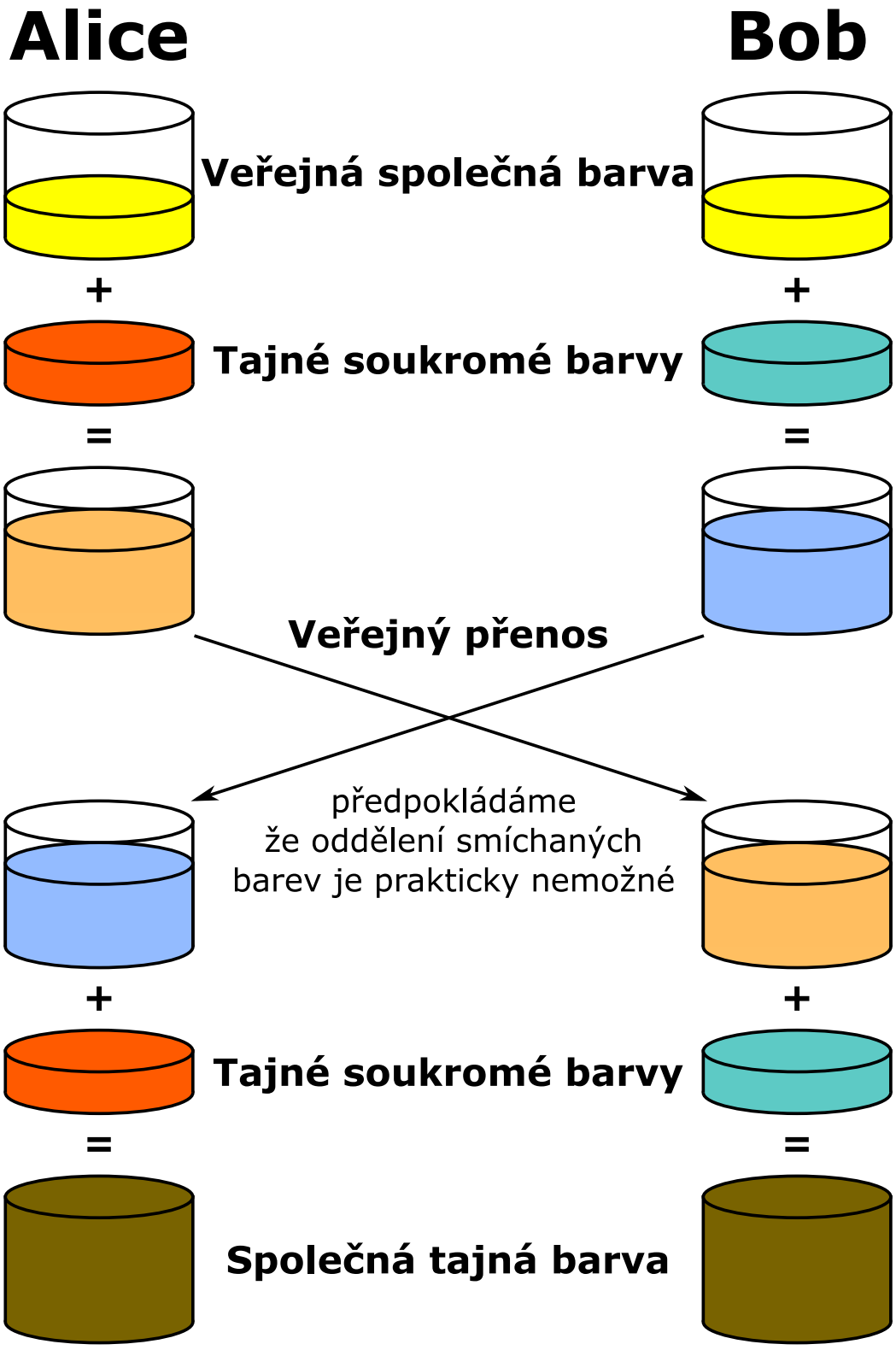
[7] Pomerance, C. A tale of two sieves. *Notices Amer. Math. Soc*. 1996, roč. 43. S. 1473–1485. Dostupné na: <https://www.ams.org/notices/199612/pomerance.pdf>.

[8] Rabin, M. O. Probabilistic algorithm for testing primality. *Journal of Number Theory*. Feb 1980, roč. 12, č. 1. S. 128–138.

Seznam příloh

1. Diffieho-Hellmanova výměna klíčů, ilustrační obrázek v plné velikosti
2. Grafy časové statistiky běhu algoritmu pro vyhledání prvočísel, data z tabulky 2 na str. 23
3. Vstupní soubory pro systém LaTeX, pomocí kterých byla táto práce vysázena. Hlavním vstupním souborem je `main.tex`. K vysázení se musí použít spustitelný soubor `xelatex` s přepínačem `-shell-escape`.
4. Zdrojový kód programu Hitai. Pro sestavení stačí spustit příkaz `.\build.ps1` z Windows PowerShell. Spouštění skriptů může být na daném systému omezené, v takovém případě je potřeba toto nastavení upravit pomocí příkazu `Set-ExecutionPolicy RemoteSigned`.
5. Instalační soubor programu Hitai. Jedná se o spustitelný soubor, který nainstaluje Hitai do uživatelské složky aktuálního uživatele a spustí ho.

Příloha č.1 – Diffieho-Hellmanova výměna klíčů



Příloha č.2 – Grafy časové statistiky algoritmu

U všech následujících grafů jsou uvedeny průměrné hodnoty z celkem 100 testů pro 64-bitová čísla, 50 testů pro 128-bitová čísla, 10 testů pro 256-bitová čísla a 4 testy pro 512-bitová čísla. Na ose y je vždy uveden počet bitů testovaného čísla.