Kryptografie – 2. projekt
Hybridní šifrování

Ondřej Ondryáš (xondry02), 30. dubna 2023

Stručný popis implementace

Kód programu je rozdělen na moduly `kry` se vstupním bodem programu, `client`, `server` a `util`, ve kterém je umístěna vlastní implementace některých kryptografických primitiv (zarovnávání a (de)šifrování pomocí RSA) a třída spravující klíče. Pro generování RSA klíčů, hashování algoritmem MD5 a (de)šifrování algoritmem AES využívá knihovnu **PyCryptodome**[1]. Jiné knihovny nejsou využity.

Server není paralelní, v jednu chvíli může obsluhovat pouze jednoho klienta (úprava by byla snadná, věřím však, že cílem projektu nebyla dokonalá implementace síťové komunikace). Klient se serverem komunikuje pomocí jednoduchého protokolu: zprávy klienta začínají 4B číslem označujícím délku zprávy, následuje celá zpráva ve tvaru: 16 B inicializační vektor pro AES-CBC; zašifrovaná zpráva s připojeným MD5 otiskem; 256 B zašifrovaný klíč relace. Server přijetí zprávy potvrzuje zasláním 32B zprávy, která je zašifrovaná klíčem relace a obsahuje informaci o úspěchu ověření integrity.

Pro zarovnávání RSA zpráv na požadovanou velikost 256 B/blok byl implementován algoritmus EME-OAEP (*optimal asymmetric encryption padding*) podle RFC 2347[2] s využitím funkce generující masku MGF1[3] a hashovací funkce SHA3-256.

Pro zarovnávání AES zpráv na požadovanou velikost 16 B/blok byl využit jednoduchý algoritmus popsaný v PKCS #7 (RFC 5652[4]), který vyplňuje do velikosti bloku bajty, jejichž hodnota je rovna počtu doplněných bajtů.

Vytváření a výměna klíčů

RSA klíče jsou generovány použitou knihovnou při spuštění programu. Vygenerované klíče jsou uloženy do složky `cert`, zvlášť je uložen soukromý klíč ve formátu PEM a veřejný klíč ve formátu OpenSSH. Pokud není nalezen veřejný klíč, ale soukromý klíč ano, veřejný klíč je extrahován ze soukromého a uložen. Uložené klíče jsou identifikovány číslem portu a označením strany („client“, nebo „server“, nicméně je možné použít libovolný řetězec: zde je ponechán prostor pro potenciální rozšíření na více různých klientů).

Program v podstatě předpokládá, že si obě strany jednotlivých stran už vyměněny. V souladu se zadáním načítá klient veřejný klíč serveru ze složky `cert` po spuštění; server načítá veřejný klíč klienta po jeho připojení. Pokud není klíč protistrany nalezen, program je ukončen.

Zadání uvádí, že má být využit algoritmus RSA-2048, modulus klíče n je tedy 2048bitový. Tato velikost klíče je v roce 2023 stále považována za bezpečnou, nicméně v horizontu příštích 10 let by bylo vhodné přejít na klíče větší, např. 3072bitové[5] V současné době se však nedá očekávat, že by bezpečnost řešení mohla být narušena útokem silou.

Pro symetrické šifrování je využita šifra AES-128 (tedy se 128b klíči a bloky), také v souladu s úvodem zadání.

[1] `https://www.pycryptodome.org/`

[2] `https://www.rfc-editor.org/rfc/rfc2437#section-9.1.1`

[3] `https://www.rfc-editor.org/rfc/rfc2437#section-10.2.1`

[4] `https://www.rfc-editor.org/rfc/rfc5652#section-6.3`

[5] Viz např. NIST Special Publication 800-57 Part 1 Revision 5 (National Institute of Standards and Technology, květen 2020), kap. 5.6.1.1 a 5.6.3.

Pro symetrickou kryptografii se v současnosti předpokládá, že 128b klíče budou bezpečné i po roce 2030[6], zde také není program ohrožen útokem silou.

Klíče relace jsou generovány s využitím knihovní funkce `get_random_bytes` (v současné verzi knihovny je toto pouze alias pro standardní funkci `os.urandom` využívající systémový generátor (pseudo)náhodných čísel). Klíč relace generuje klient, server se na něm nijak nepodílí (není tedy zajištěna spravedlivost, což může být potenciální bezpečnostní nedostatek). Klíč relace je zašifrován pomocí RSA veřejným klíčem serveru a připojen ke zprávě symetricky zašifrované klíčem relace. Server určí klíč relace dešifrováním příslušné části zprávy svým soukromým klíčem.

Nový klíč relace se generuje pro každou zprávu od klienta. Využit je pouze pro tuto zprávu a následnou potvrzovací zprávu ze strany serveru. Předpokládám, že využití „ephemeral“ klíčů s krátkou dobou života má pozitivní dopad na bezpečnost komunikace.

Asymetrické šifrování

Pro generování klíčů šifry RSA byla využita knihovní funkce. Samotné šifrování a dešifrování bylo implementováno ve vlastních funkcích `util.rsa_encrypt` a `util.rsa_decrypt`, modulární umocňování zajišťuje standardní knihovní funkce `pow`. Nutno podotknout, že funkce „encrypt“ se používá pro šifrování veřejným klíčem příjemce i pro „ověřování podpisu“ veřejným klíčem odesilatele; funkce „decrypt“ se používá pro dešifrování soukromým klíčem příjemce i pro „podepisování“ soukromým klíčem odesilatele – jde o dvojice sémanticky rozdílných, ale matematicky shodných operací.

Veřejné klíče se na výstup vypisují ve formátu „OpenSSH“. Pro soukromé klíče se vypisuje pouze SHA3-256 otisk (konkrétně otisk klíče zakódovaného ve formátu PEM).

Vstupem pro šifrování veřejným RSA-2048 klíčem příjemce je 2048b (256B) blok dat, ten je pak také výstupem po dešifrování. Data je tedy nutné před šifrováním (i podepisováním) zarovnat a po přijetí a dešifrování je nutné získat původní zprávu inverzní operací. Program k tomu využívá vlastní implementaci algoritmu OAEP, což by mělo předcházet některým bezpečnostním problémům spojeným s jednoduššími zarovnávacími schématy. OAEP do výstupu zavádí náhodnost, což je nutné pro předejití klasického CPA útoku na deterministické „učebnicové RSA“. Výstup OAEP závisí na použité hashovací funkci a na funkci generující masku, zde MGF1, jejíž náhodnost je také postavena na hashovací funkci. V obou případech jsem využil hashovací funkci SHA3-256 (ač pro tuto volbu nemám žádný objektivní důvod; RFC 2347 doporučuje pouze SHA1, což by pro použití v OAEP mělo být stále dostačující).

Předpokládám, že by implementace mohla být náchylná na útoky postranními kanály, neboť proti těmto nebyly implementované algoritmy nijak explicitně navrženy a neočekávám, že by tomu bylo jinak v případě použitých funkcí standardní knihovny.

Symetrické šifrování

Bloková šifra AES je použita v režimu CBC, což umožňuje bezpečný přenos zpráv delších než je velikost jednoho bloku (16 B). Vstupem je zde inicializační vektor (IV), který program generuje jako sekvenci (psuedo)náhodných bajtů stejným způsobem jako klíč relace. IV musí být pro dešifování dodán druhé straně, přičemž musí být zajištěna jeho integrita. To je řešeno připojením IV do obsahu, nad kterým se počítá MD5 otisk.

IV i AES klíče i jejich zarovnaná verze se vypisují ve formě hexadecimálního čísla.

Data, která procházejí symetrickou šifrou, obsahují konkatenaci uživatelské zprávy a „podepsaného“ (asymetricky zašifrovaného soukromým klíčem odesilatele) MD5 otisku konkatenace IV a uživatelské zprávy. Server

[6] Viz výše odkazovaný dokument NIST.

odesílá potvrzovací zprávu, ve které je symetricky šifrována konkatenace stavového bajtu a původního MD5 otisku. Dá se říct, že je zajištěna důvěrnost (zpráva je zašifrována symetrickým klíčem známým pouze klientovi a serveru), autenticita (stejný důvod) i integrita odpovědi (otisk je známý pouze klientovi a serveru, útočník bez znalosti sym. klíče a otisku nebo původní zprávy nemůže tuto odpověď napodobit).

Zajištění integrity

K zajištění integrity se využívá hashovací funkce MD5, která vypočítá otisk zasílané zprávy. Otisk je následně zašifrován veřejným klíčem serveru a připojen k datům, která jsou symetricky zašifrována. MD5 otisky jsou 128bitové, otisk zašifrovaný pomocí RSA-2048 se tedy vleze do jednoho zarovnaného 256B bloku. Server po symetrickém dešifrování dat ze zprávy oddělí posledních 256 B, dešifruje je svým soukromým klíčem, čímž získá předpokládaný otisk dat. Následně vypočítá ze zbytku dat MD5 otisk a oba řetězce porovná. Pokud by zpráva byla pozměněna, řetězce se budou lišit.

Jak bylo uvedeno výše, otisk se počítá nad konkatenací zprávy a incializačního vektoru šifry AES, a to z důvodu zajištění integrity IV i zprávy.

Server zasílá potvrzovací zprávu, jejíž první bajt má hodnotu 1 (integrita potvrzena), nebo 0 (integrita narušena). Klient znovu zasílá pouze zprávy, pro které potvrzovací zpráva vůbec nedošla, nicméně uživateli na výstupu signalizuje, jaký stav přijaté potvrzení nese.

Funkce MD5 se v současné době nepovažuje za bezpečnou, mimo jiné proto, že dostatečně nezajišťuje bezkoliznost. Pokud by útočník znal vstupní text, je možné, že by mohl vytvořit zprávu se stejným otiskem. Aby mohl takovou zprávu serveru dodat namísto skutečné původní zprávy, musel by však také znát také symetrický klíč relace, aby mohl celou zprávu i s původním otiskem znovu zašifrovat. Myslím si proto, že v tomto případě nepředstavuje použití slabé hashovací funkce významné bezpečnostní riziko.

Program je náchylný na „replay“ útoky, při kterých útočník zachytí zprávu a poté ji zasílá znovu. Řešením by mohlo být připojení časového razítka (které by muselo být zahrnuto v části, pro kterou je zajištěna integrita).