Základní údaje

Program je implementován v jazyce `Python`, verze 2.6 minimálně. Vyžaduje přítomnost knihovny `pycrypto` (balíček `Crypto`). Obojí je na serveru Merlin přítomno (i když `Crypto` ne kompletní).
Pro spuštění programu spusťte:

```
$ python messenger -s|-c
```

Komunikační protokol

Pro komunikaci mezi serverem a klientem bylo využito pojmenovaných rour. Aplikace používá 2 takovéto roury: `downlink` a `uplink`.
Pro komunikaci je používán formát CSV, kdy jako první je na řádku uveden příkaz, která se má provést (určuje pozici v protokolu), pak již následují data (pokud se posílá víc než jeden údaj, jsou také odděleny čárkou). Data posílaná v příkazu jsou zakódována pomocí Base64. Tudíž data neobsahují netisknutelé znaky je možné je číst po řádcích.
Jako probíhá komunikace mezi klientem a serverem popisuje následující tabulka.

INIT	Klient zahajuje komunikaci a posílá serveru P a G parametry pro Diffie-Helman
MY_PUBLIC	Server odpovídá svým veřejným klíčem
MY_PUBLIC	Klient si spočítá sdílené tajemství a odpovídá svým veřejným klíčem
AES	Server si také vypočítá sdílené tajemství a rovnou odpovídá inicializačním vektorem pro AES
FFS_X	Nyní již šifrovanou formou pošle klient své vygenerované X pro Feige-Fiat-Shamirovo identifikačná schéma
FFS_A	Server přijme X, vygeneruje A vektor a pošle jej klientovi
FFS_Y	Klient spočítá Y a pošle jej serveru
FFS_OK	Server uvědomí klienta, že se jej podařilo identifikovat a tím jej vyzívá k dalšímu kolu identifikace
FFS_DONE	Server uvědomí klienta, že byl identifikován, a že může začít posílat zprávy (**MSG**)
FFS_FAIL	Server uvědomí klienta, že se nepodařilo prokázat jeho identitu a ukončí komunikaci (čeká opět na **INIT**)

MSG	Klient posílá serveru zprávy
SHA256	Server odpovídá hashí

Pro úplnost je třeba dodat, že informace o operaci se vždy přenáší nešifrovaně, naopak data, od ustanovená AES tunelu, vždy šifrovaně v obou směrech. Pokud dojde serveru požadavek, který neočekával, server vypíše chybu a přeruší komunikaci (čeká opět na **INIT**)

Diffie-Hellman

Tato metoda výměny klíčů je implementována ve třídě `DiffieHellman`, v souboru `dh.py`. Pro ustanovení komunikace klient počítá vlastní sadu P a G proměnných. Pro tenhle účel je použit algoritmus z RFC 3526, Sekce 2[1]. Z důvodů problematické implementace desetinných čísel v Pythonu je použita hodnota PI která je vynásobena 10^{15} (počet desetinných míst na mém počítači pro PI z modulu `math`) a následný výpočet je celočíselně vydělen.
Serveru jsou následně hodnoty P a G poslány a klíč je vypočítán standardním způsobem:

$$Private = rand(1,\ P-1)$$
$$Public = G^{private} \bmod P$$
$$Key = otherpublic^{private} \bmod P$$

Pro redukci výsledného klíče na 256 bitů je použita hashovací funkce SHA256.

Feige-Fiat-Shamirovo identifikační schéma

Identifikační schéma je implementováno pomocí tříd `FFSProver` a `FFSVerifier` v modulu `ffs.py`. První zmíněná třída slouží klientovi k dokazování své identity a druhou využívá server pro jeho ověření.
Pro ověření identity jsou použity hodnoty proměnné N a S vektoru, které byly zaslány e-mailem. Z nich je předpočítán V vektor jako $v_i = s_i^2$, pro každý prvek S vektoru. Tím je příprava dokončena. Pak, když se chce klient identifikovat, vygeneruje si náhodné číslo R a znaménko + nebo – a to pomocí následujícího schématu:

$$R = rand(minint,\ maxint)$$
$$sign = choice(-1,\ 1)$$

Následně vypočítá X a pošle jej serveru:

$$X = sign * R^2 \bmod N$$

Server vygeneruje A vektor, obsahující náhodně 0 a 1 a odpoví jím:

$$A = a_0 ... a_n,\ kde\ a = choice(0,\ 1)\ a\ n = n\ pro\ S\ vektor$$

Klient, která obdržel tento vektor, spočítá hodnotu Y, kterou pošle serveru:

$$Y = R * s_i^{a_i},\ pro\ všechny\ prvky\ S\ vektoru$$

Server ověří správnost odpovědi, a to:

[1] https://www.ietf.org/rfc/rfc3526.txt

$$Y^2 \equiv abs(X * v_i^{a_i}) \; mod \; N \; pro \; všechny \; prvky \; vektoru \; V \;\; a \; A$$

Toto pověření se mi nepodařilo spolehlivě implementovat, takže pro komtrolu je možné jej vypnout pomocí konstanty `SKIP_FFS` v `server.py`.

AES a použitý mód

Pro šifrování AES byla použita knihovna `Crypto` v módu `MODE_CBC`. Inicializační vektor je vygenerován pomocí `/dev/urandom`, jelikož část knihovny `Crypto`, zodpovědná za tuto funkcionalitu není na Merlinovi dostupná. Tento vektor je poslán nešifrovanou podobou klientovi a jsou to poslední obsahová data, která nejsou v komunikaci šifrována.

Úskalí v zabezpečení komunikace

Nespolehlivost mé implementace FFS je zcela jiste první a největší úskalí. Jako další bych měl zmínit špatný návrh komunikačního protokolu, kdy část dat (operace požadavku, příkaz) je posílána bez šifrování.