مریم رضائی

پروژه‌ی پایانی: سامانه رمز RSA

۹۸۱۸۱۳۰۸۸

شکل کلی برنامه:

برنامه به طور کلی دارای منویی با چهار گزینه‌ی تولید کلیدها، رمزنگاری، رمزگشایی و خروج است که در حلقه‌ی بی‌نهایت قرار گرفته و هر تعداد بار که کاربر بخواهد قابل استفاده است.

۱) **تولید کلیدها:** تابع keys() را از توابع اصلی فراخوانی می‌کند. این عمل به علت بزرگی کلیدهای مورد نظر و پایین بودن احتمال اول بودن عدد در ارقام بالا، چند دقیقه‌ای زمان می‌برد. استفاده از ماژول‌های خود پایتون به جای الگوریتم LCG برای اعداد رندوم قطعا سرعت را بالا می‌برد.

۲) **رمزنگاری:** ابتدا جفت کلید را به صورت دو عددِ با فاصله جداشده و دوماً پیام واضح را دریافت می‌کند. سپس با بررسی طول کاراکترهای پیام درصورت بزرگتر بود آن از طول بلاک‌ها عمل جداسازی را انجام داده و هر بلاک را به عنوان یک عضو لیست قرار می‌دهد. در نهایت هر عضو با تابع اصلی blockOrd() به عدد تبدیل شده و با فراخوانی تابع اصلی encrypt() به رمز تبدیل می‌شود. پیام رمز شده به عنوان خروجی چاپ می‌شود که بلاک‌ها با فاصله از هم جدا شده‌اند.

۳) **رمزگشایی:** ابتدا جفت کلید را به صورت دو عددِ با فاصله جداشده و دوماً پیامِ رمز شده را دریافت می‌کند. سپس هر بلاک را عضوی از لیست قرار داده و با تابع اصلی decrypt() بلاک رمز را به عدد با معنی تبدیل کرده بعد با تابع اصلی blockChr() به پیام واضح تغییر می‌دهد و چاپ می‌کند.

۴) **خروج:** از حلقه‌ی بی‌نهایت خارج می‌شود.

توابع اصلی:

- keys(): با انتخاب عدد رندوم به کمک تابع prng()، اول بودن آن را با تابع primality() می‌سنجد. سپس n و phi را با تابع multiply() انتخاب کرده و e را ۲^۱۶+۱ یعنی ۶۵۵۳۷ قرار می‌دهد و شروط را بررسی می‌کند. آنگاه d را از modinv() یافته و در خروجی دو کلید را درون فایلی متنی ذخیره می‌کند وکلید عمومی را منتشر می‌کند.
- blockOrd(): رشته را ورودی گرفته و بنا به فرمول تغییر مبنا، هر کاراکتر را با توجه به جایگاه و ord() آن به مبنای ۲۵۶ که مربوط به کاراکترهای ASCII هست (از ۰ تا ۲۵۵) تغییر می‌دهد و جمع می‌کند. در خروجی unicode بلاک را return می‌کند.
- blockChr(): برای رفتن مسیری برعکس تابع blockOrd() نیاز است عدد ورودی را بر ۲۵۶ به توان جایگاه تقسیم صحیح کنیم، که دیگر مقادیر جمع شده باقی‌مانده می‌شوند. این تقسیم باید از کاراکتر آخر بلاک شروع شود و به عقب بیاید زیرا برای هر توان ۲۵۶ در مقادیر جمع شده، توانی بالاتر در مقدار بعدی‌ست و پاسخ تقسیم بر توان کوچک‌تر، پاسخ مربوط به توان بزرگ تر را خروجی می‌دهد. یعنی:

$$Hey \rightarrow int = ۲۵۶^{۰} \times ord('H') + ۲۵۶^{۱} \times ord('e') + ۲۵۶^{۲} \times ord('y')$$

$$int \,//\, ۲۵۶^{۱} = ۲۵۶ \times ord('y')$$

پس باید از بزرگ‌ترین توان شروع کرد تا ord('y') را پیدا کرد. از آن‌جا که بلوک شاید از حد تعیین شده کوتاه‌تر باشند نیاز است بزرگ‌ترین توان ممکن را که خروجی‌ای در محدوده مقادیر ASCII می‌دهد بیابیم. سپس با داشتن طول بلاک از آخر به اول محاسبات را انجام دهیم و در انتها reverse کنیم.

- encrypt() و decrypt(): تابع modexpo() را فراخوانی می‌کنند که بازنویسی تابع pow() با سه ورودی‌ست. این تابع مکمل از خاصیت قضیه‌ی «باقی‌مانده چینی» استفاده می‌کند.

توابع ابزاری و مکمل:

- prng(): با به کارگیری الگوریتم linear congruential generator، این pseudorandom number generator از یک seed زمان شروع کرده و با مقادیر پیشفرض m و a فرمول مورد نظر را پیاده سازی می‌کند. با بررسی جدول ارائه شده در ویکیپدیا از مقدار فرض شده برای هر کدام در مکان‌های متفاوت و محدود bit عدد پاسخ، می‌توان نتیجه‌گیری کرد که با ۵۱۲-bit برای a و ۱۰۲۴-bit برای m، اعداد بالای ۵۱۲ بیتی برای p و q تضمین می‌شود.
- primality() و test(): تابع primality() برای سنجش اول بودن عدد n، تست «میلر-رابین» یعنی تابع test() را k بار تکرار می‌کند. این تست بر دسته‌ای از برابری‌ها تکیه می‌کند که بنا بر دانش ریاضی برای اعداد اول صدق می‌کنند. به این علت نیاز است ابتدا موارد واضح و خاص که برای برابری‌های مورد نظر استثنا به وجود می‌آورند را جدا کرده و سپس به بررسی عدد بپردازیم. فرمول‌های ریاضی و توضیحات به صورت کامنت در فایل موجودند.
- modexpo(): برای به‌توان‌رسانی پیمانه‌ای این تابع از قضیه‌ی «باقی‌مانده چینی» کمک گرفته و با محاسبه باقی‌مانده بخش به بخش اعداد را کوچک‌تر کرده و سرعت را افزایش داده است.
- power(): تابعی سریع تر از علامت توان پایتون با فراخوانی تابع سریع multiply().
- multiply(): از الگوریتم ضرب سریع «کاراتسوبا» استفاده می‌کند؛ این الگوریتم که تعداد ضرب‌های دو عدد n رقمی را به حدود $n^{1.58}$ ضرب تک‌رقمی تبدیل می‌کند، بهینه‌تر از الگوریتم عادی ضرب است که نیاز به n^2 ضرب تک رقمی دارد. الگوریتم کاراتسوبا پاسخ ضرب دو عدد بزرگ را از راه ضرب سه عدد کوچک‌تر محاسبه می‌کند که تعداد ارقام هرکدام حدودا نصف تعداد ارقام اعداد اصلی‌ست.
- egcd() و modinv() و gcd(): تابع بازگشتی egcd() با کمک الگوریتم مبسوط اقلیدس علاوه بر بزرگترین مقسوم علیه مشترک، ضرایب قصیه بزو را هم پیدا می‌کند و برای این کار از دنباله‌هایی از خارج قسمت‌ها و باقی‌مانده‌ها به شکل بازگشتی استفاده می‌کند. توابع modinv() و gcd() با فراخوانی egcd()، مقادیر مورد نظر را (یعنی به ترتیب وارون ضربی و ب.م.م) جدا می‌کنند.