
بسم الله الرحمن الرحيم

پیک ایریدیم

محمد مهدی اقدسی ۴۰۰۵۲۱۰۸۱

علی شکوهی ۴۰۰۵۲۱۴۷۷

دانیال یگانه ۴۰۰۵۲۲۳۳۷

آخرین ویرایش: ۱۱ دی ۱۴۰۳ در ساعت ۲۳ و ۹ دقیقه

فهرست مطالب

۱ سوالات تئوری

۱.۱ سوال اول

در واقع، هیچ الگوریتم "سریع" شناخته شده ای برای فاکتورگیری اعداد صحیح بزرگ در یک کامپیوتر کلاسیک وجود ندارد. به همین دلیل است که فاکتورسازی اعداد صحیح یک مشکل محاسباتی سخت در نظر گرفته می‌شود و برای امنیت بسیاری از سیستم‌های رمزنگاری مانند RSA بسیار مهم است. دشواری فاکتورگیری اعداد بزرگ از این واقعیت ناشی می شود که شناخته شده ترین الگوریتم ها در تعداد ارقام عدد فاکتورگیری شده زمان نمایی می گیرند.

پیچیدگی زمانی سریعترین الگوریتم شناخته شده برای فاکتورسازی اعداد صحیح به اعداد اول به اندازه عدد ورودی بستگی دارد. الگوریتم های مختلفی وجود دارند که برای اندازه های ورودی مختلف عملکرد بهتری دارند. برای اعداد بسیار بزرگ (اعداد با صدها یا هزاران رقم)، سریعترین الگوریتم شناخته شده غربال میدان اعداد عمومی (GNFS) است. پیچیدگی زمانی الگوریتم GNFS برای فاکتورگیری یک عدد صحیح N برابر است با:

$$O(exp(((\frac{64}{9})^{\frac{1}{3}} + O(1)).(logN)^{1/3}.(loglogN)^{2/3}))$$

این الگوریتم یک الگوریتم زیر نمایی می باشد که به طور قابل‌توجهی سریع‌تر از الگوریتم‌های صرفاً نمایی مانند تقسیم آزمایشی یا الگوریتم rho پولارد است. با این حال، GNFS هنوز از نظر محاسباتی فشرده است و برای اعداد کوچک‌تر کارایی کمتری دارد. برای اعداد با اندازه متوسط تر (تا چند صد رقم)، الگوریتم های دیگر مانند غربال درجه دوم (QS) یا روش منحنی بیضی (ECM) ممکن است در عمل سریعتر باشند، علیرغم داشتن پیچیدگی های زمانی مجانبی بدتر. پیچیدگی زمانی الگوریتم غربال درجه دوم:

$$O(exp((1 + O(1)).(lnN)^{1/2}.(lnlnN)^{1/2}))$$

مقالات مربوط به GNFS:

Gasarch, W. (2005). A Beginner's Guide To The General Number Field Sieve. University of Maryland.

Briggs, M. (2001). An Introduction to the General Number Field Sieve. Virginia Tech.

۲.۱ سوال دوم

اعداد نیمه اول که با نام اعداد نیمه اول یا اعداد دو اول نیز شناخته می شوند، اعداد طبیعی هستند که دقیقا حاصل ضرب دو عدد اول مجزا هستند. به عبارت دیگر، یک عدد نیمه اول را می توان به صورت ضرب دو عدد اول مختلف بیان کرد. به عنوان مثال، اعداد ۶، ۱۰، ۱۴، ۱۵، ۲۱، ۲۲، ۲۶، ۳۳، ۳۴، ۳۵، ۳۸، ۳۹، ۵۱، ۵۵، ۵۷، ۵۸، ۶۲، ۶۵، ۶۹، ۷۷، ۸۵، ۸۷ و ۹۱ همه اعداد نیمه اول هستند زیرا می توان آنها را به عنوان حاصل ضرب دو عدد اول مجزا بیان کرد:

- $3 \times 2 = 6$
- $5 \times 2 = 10$
- $7 \times 2 = 14$
- $5 \times 3 = 15$
- $7 \times 3 = 21$
- ...

در مورد تجزیه اعداد نیمه اول به عوامل اول، به طور کلی ساده تر از تجزیه اعداد ترکیبی که نیمه اول نیستند در نظر گرفته می شود. در اینجا دلیل آن است:

۱. تعداد محدود عامل اول: طبق تعریف، اعداد نیمه اول دقیقاً دو عامل اول مجزا دارند. این محدودیت فرآیند فاکتورسازی را در مقایسه با اعداد ترکیبی که ممکن است بیش از دو عامل اول داشته باشند، ساده می کند.
۲. الگوریتم های فاکتورسازی کارآمد: الگوریتم های کارآمدی وجود دارند که به طور خاص برای فاکتورگیری اعداد نیمه اول طراحی شده اند، مانند غربال درجه دوم و غربال فیلد اعداد. این الگوریتم‌ها از ساختار اعداد نیمه اول بهره می‌برند و فرآیند فاکتورسازی را کارآمدتر از اعداد مرکب عمومی می‌کنند.
۳. مبنای عامل کوچکتر: در زمینه الگوریتم های فاکتورسازی، پایه عامل (مجموعه اعداد اول کوچک مورد استفاده در الگوریتم) برای اعداد نیمه اول کوچکتر است زیرا عوامل به دو عدد اول محدود می شوند. این امر محاسبات را در مقایسه با اعداد مرکب با چندین عامل اول قابل کنترل تر می کند.

در اینجا یک قیاس وجود دارد: تصور کنید یک کیسه با چند تیله رنگی دارید. یافتن یک سنگ مرمر رنگی خاص (ضریب اصلی) در صورتی که سنگ مرمرهای زیادی وجود داشته باشد (عدد مرکب با فاکتورهای زیاد) می تواند دشوار باشد. اما، اگر کیسه فقط دو تیله داشته باشد (نیمه پرایم)، پیدا کردن یک سنگ مرمر رنگی خاص بسیار آسان تر می شود. با این حال، توجه به این نکته مهم است که در حالی که فاکتورسازی اعداد نیمه اول به طور کلی ساده تر از اعداد مرکب عمومی است، مشکل همچنان با افزایش اندازه (تعداد ارقام) اعداد نیمه اول افزایش می یابد. اعداد نیمه اول بزرگ هنوز هم می توانند چالش های محاسباتی قابل توجهی برای الگوریتم های فاکتورسازی ایجاد کنند. معمولاً تجزیه اعداد نیمه اول بسیار بزرگ به ضرایب اول دشوارتر از فاکتورگیری اعدادی است که اول هستند یا دارای عوامل اول کوچک هستند. این به این دلیل است که اعداد نیمه اول به طور خاص به عنوان حاصل ضرب دو عدد اول بزرگ انتخاب می شوند و هیچ الگوریتم کارآمد شناخته شده ای وجود ندارد که آنها را به طور کلی عامل بندی کند.

به عبارت دیگر، تجزیه اعداد نیمه اول بسیار بزرگ زمانی که دو عدد اول بسیار بزرگ و تصادفی انتخاب شده و دارای مقدار نسبتا نزدیک هستند، حتی سریع ترین الگوریتم ها در سریع ترین کامپیوترها آنقدر زمان می برند که در واقع ناکارآمد هستند. به طور خلاصه، اعداد نیمه اول، اعداد مرکبی هستند که حاصل ضرب دو عدد اول مجزا هستند، و تجزیه آنها به ضرایب اول آنها معمولا ساده‌تر از تجزیه اعداد ترکیبی که نیمه اول نیستند، به دلیل تعداد محدود عوامل اول و در دسترس بودن کارآمد است. الگوریتم های فاکتورسازی که به طور خاص برای اعداد نیمه اول طراحی شده اند. اما با افزایش اندازه اعداد اول، فاکتورگیری اعداد نیمه اول به طور تصاعدی دشوارتر می شود و به طور فزاینده ای به چالش کشیدن آنها تبدیل می شود. این دشواری در طرح‌های رمزنگاری مختلف، مانند ،RSA که بر مشکل فاکتورگیری اعداد نیمه‌اول بزرگ برای امنیت خود متکی هستند، استفاده می‌شود.

۳.۱ سوال سوم

در اصل، شرط اول بودن m نسبت به n یک شرط قوی است که عمدتاً برای اطمینان از یکپارچگی ریاضی عملیات رمزنگاری استفاده می‌شود، اما برای عملکرد الگوریتم RSA کاملاً ضروری نیست به شرطی که دو عدد اول انتخاب شده با یکدیگر یکسان نباشند. در اکثر مواقع، m به طور طبیعی نسبت به n به دلیل نحوه ساختن n (به عنوان حاصل ضرب دو عدد اول بزرگ) اول خواهد بود، اما چنین شرطی ضروری نیست.

در مقاله اصلی RSA نیز در بخش VI The Underlying Mathematics این قضیه اثبات می شود.

لینک به سوال مربوط در سایت crypto.stackexchange.com (اثبات‌های مختلفی از این قضیه در پاسخ‌ها وجود دارد): سوال

مرجع به بخش اثبات ریاضیاتی در مقاله اصلی RSA : اثبات

۴.۱ سوال چهارم

در الگوریتم RSA بهتر است پارامتر e فرد باشد تا زوج. این به این دلیل است که e باید نسبت به $\varphi(n)$ اول باشد، که در آن $\varphi(n) = (p-1)(q-1)$. از آنجایی که $\varphi(n)$ همیشه زوج است (چرا که هر دو $p-1$ و $q-1$ زوج هستند)، e زوج نسبت به $\varphi(n)$ اول نخواهد بود، مگر اینکه ۱ باشد، که برای رمزگذاری مناسب نیست. به طور معمول، مقادیر کوچک فرد اول مانند ۳، ۱۷، یا ۶۵۵۳۷ برای e انتخاب می‌شوند، زیرا برای رمزگذاری کارآمد هستند و اطمینان می‌دهند که بزرگترین مقسوم‌کننده مشترک با $\varphi(n)$ برابر با ۱ است.

۵.۱ سوال پنجم

اعداد فرما دنباله‌ای خاص از اعداد هستند که با فرمول زیر تعریف می‌شوند:

$$F_n = 2^{2^n} + 1$$

که در آن n یک عدد صحیح غیرمنفی است. چند عدد اول فرما عبارتند از:

$$F_0 = 3$$

$$F_1 = 5$$

$$F_2 = 17$$

$$F_3 = 257$$

$$F_4 = 65537$$

نقش اعداد فرما در تولید پارامتر RSA

در الگوریتم RSA، انتخاب توان عمومی e به عنوان یک عدد فرما، به ویژه $F_4 = 65537$، رایج است. این انتخاب مطلوب است زیرا:

۱. **کارایی:** استفاده از ۶۵۵۳۷ (که $2^{16}+1$ است) به عنوان e رمزگذاری کارآمد و تأیید امضا را به دلیل وزن کم همینگ آن تضمین می‌کند (یعنی فقط دو بیت ۱ در نمایش باینری خود دارد).

۲. **ضرورت:** اعداد فرما که اعداد اول نسبتاً بزرگ هستند، خاصیت لازم را حفظ می‌کنند که e نسبت به $\varphi(n)$ هم‌اول باشد.

۳. **اعداد فرد:** این اعداد، اعداد فرد هستند و ما ترجیح می‌دهیم e فرد باشد همانطور که در سوال قبلی بیان شد.

بنابراین، به طور خلاصه، در حالی که این انتخاب مورد نیاز نیست، استفاده از یک عدد اول فرما مانند ۶۵۵۳۷ به عنوان توان عمومی e در RSA یک روش توصیه‌شده است. این انتخاب تعادلی بین کارایی محاسباتی و امنیت برقرار می‌کند و آن را به یک انتخاب محبوب در پیاده‌سازی RSA تبدیل می‌کند.

۶.۱ سوال ششم

مقاله RSA چندین الگوریتم کارآمد را برای انجام توان مدولار، $M^e \mod n$ و $C^d \mod n$ در طول رمزگذاری و رمزگشایی به ترتیب مورد بحث قرار می‌دهد. در اینجا برخی از الگوریتم های بهینه شده تحت پوشش آورده شده است:

۱. توان با مربع و ضرب مکرر (بخش VII.A): این یک الگوریتم اساسی است که در مقاله ارائه شده است. وقتی بیت متناظر در نمایش دودویی e ۱ باشد، $M^e \mod n$ را با مجذور کردن مکرر M و ضرب در M محاسبه می کند. پیچیدگی زمانی آن $O(\log e)$ است.

۲. رویه های کارآمدتر (بخش VII.A): این مقاله اشاره می‌کند که روش‌های کارآمدتری نسبت به مربع‌سازی مکرر پایه شناخته شده‌اند، بدون اینکه وارد جزئیات شوند. برخی از نمونه ها عبارتند از:
   - توان پنجره کشویی: توان های کوچک را از قبل محاسبه می کند تا دوباره استفاده کند و سرعت را افزایش دهد.
   - توان زنجیره جمع: M^e را با استفاده از یک زنجیره جمع برای e محاسبه می کند.

۳. الگوریتم های مطالعه شده توسط Knuth (بخش VII.A): این مقاله به کار اصلی کنوت «هنر برنامه‌نویسی رایانه‌ای» اشاره می‌کند که الگوریتم‌های توان را با جزئیات مورد مطالعه قرار می‌دهد، از جمله:
   - توان دودویی
   - استفاده از زنجیر اضافه
   - بهره برداری از الگوهای توان ویژه
   - ضریب معاملاتی برای تربیع
۴. الگوریتم ضرب مونتگومری: اگرچه به صراحت در مقاله اصلی RSA ذکر نشده است، روش ضرب مونتگومری یک تکنیک بهینه برای انجام ضرب‌های مدولار در طول توان است، و از عملیات تقسیم پرهزینه جلوگیری می کند.
۵. کاهش بارت: این تکنیک کارایی گام کاهش مدولار را در طول توان بهبود می بخشد. مقادیری را از پیش محاسبه می‌کند که به کاهش مدولار اجازه می‌دهد با استفاده از ضرب و تفریق انجام شود و از تقسیم اجتناب شود.

این تکنیک‌ها، همراه با سایر روش‌های پیشرفته مانند استفاده از سخت‌افزار محاسباتی مدولار، می‌توانند به طور قابل‌توجهی عملیات توان‌سازی مدولار هسته را در RSA، که فشرده‌ترین بخش‌های محاسباتی آن هستند، بهینه کنند. الگوریتم‌های بهینه‌سازی قدرت برای پیاده‌سازی RSA با کارایی بالا بسیار مهم هستند.

۷.۱ سوال هفتم

مقایسه سطح امنیتی اندازه‌های کلید RSA با اندازه‌های کلید متقارن برای رمزهای بلوکی به شرح زیر است:

- یک کلید RSA ۱۰۲۴ بیتی تقریباً به اندازه یک کلید متقارن ۸۰ بیتی مانند 2TDEA امنیت دارد.
- یک کلید RSA ۲۰۴۸ بیتی تقریباً به اندازه یک کلید متقارن ۱۱۲ بیتی مانند 3TDEA امنیت دارد.

باید توجه داشت که این‌ها تنها تخمین‌هایی هستند و مفروضات پیچیدگی محاسباتی و مدل‌های حمله برای سیستم‌های رمزنگاری کلید عمومی مانند RSA با رمزهای بلوکی متفاوت است. اما این تخمین‌ها یک حس کلی از سطوح امنیتی مورد هدف برای اندازه‌های مختلف کلید RSA در مقایسه با رمزهای بلوکی ارائه می‌دهد.

مرجع:

Barker, Elaine (May 2020). "Recommendation for Key Management: Part 1 – General" (PDF), Page 54. NIST Special Publication.

۸.۱ سوال هشتم

تولید اعداد اول مناسب برای RSA شامل چندین مرحله برای اطمینان از ایمن بودن و مناسب بودن اعداد برای استفاده رمزنگاری است:

۱. انتخاب تصادفی نامزدها

- طول بیت: طول بیت اول را انتخاب میکنیم (به عنوان مثال، ۱۰۲۴ بیت برای کلیدهای ۲۰۴۸ بیتی RSA).
- تولید عدد تصادفی: یک عدد تصادفی از طول بیت مورد نظر را ایجاد کنید. مطمئن شوید که فرد است (زیرا اعداد زوج بزرگتر از ۲ اول نیستند).

۲. تست اولیه

- تست های پایه: بررسی های اولیه مانند تقسیم پذیری بر اعداد اول کوچک را انجام دهید تا سریعاً اعداد غیر اول را رد کنیم.
- تست های اولیه احتمالی: از آزمون هایی مانند آزمون Miller-Rabin یا آزمون Baillie-PSW برای تعیین اینکه آیا عددی با احتمال زیاد اول است یا خیر، استفاده میکنیم. چندین دور را تکرار میکنیم تا شانس مثبت کاذب را کاهش دهیم

۳. تضمین امنیت رمزنگاری

- راندهای کافی(Sufficient Rounds): برای Miller-Rabin، از دورهای کافی (مثلاً ۴۰ برای امنیت بالا) استفاده میکنیم تا به سطح اطمینان مطلوبی دست یابید.
- اجتناب از عوامل کوچک(Avoiding Small Factors): اطمینان حاصل میکنیم که اعداد اول خیلی به توان های اعداد اول کوچک نزدیک نیست تا از حملات رمزنگاری خاص جلوگیری شود.

۴. تایید و اعتبار سنجی

- منحصر به فرد و به اندازه کافی بزرگ: مطمئن میشویم که اعداد اول p و q متمایز و به اندازه کافی بزرگ هستند تا حاشیه امنیتی لازم را فراهم کنند.
- معیارهای اضافی: به صورت اختیاری، ویژگی های اضافی مانند p-۱ یا q-۱ را که دارای فاکتورهای اصلی بزرگ برای افزایش امنیت هستند، بررسی میکنیم.

مثالی از این فرآیند:

۱. ایجاد نامزد:

- یک عدد فرد تصادفی ۱۰۲۴ بیتی ایجاد میکنیم.

۲. بررسی اولیه:

- بررسی میکنیم که آیا عدد بر هر عدد اول کوچک بخش پذیر است (مثلاً تا ۱۰۰۰).

۳. تست اولیه:

- آزمون Miller-Rabin را برای ۴۰ تکرار اعمال میکنیم.

۴. تکرار:

- اگر عدد تمام تست ها را پشت سر بگذارد، احتمالاً اول است. در غیر این صورت، یک نامزد جدید ایجاد میکنیم و تکرار میکنیم.

تضمین امنیت تولید اعداد اول:

- از مولدهای اعداد تصادفی امن رمزنگاری شده (CSPRNG) استفاده کنیم.

- اعتبار اجرا را بر اساس استانداردهای شناخته شده (به عنوان مثال، FIPS 186-4).

کتابخانه ها و ابزارها:

- کتابخانه های رمزنگاری مانند OpenSSL و GNU MP (GMP) توابع داخلی را برای تولید و آزمایش اعداد اول بزرگ ارائه می کنند و از تولید اعداد اول قابل اعتماد و کارآمد برای RSA اطمینان حاصل می کنند.

الگوریتم های احتمالی تولید اعداد اول احتمال زیادی برای تولید یک عدد اول در یک محدوده مشخص ارائه می کنند، اما ممکن است گاهی اوقات اعداد ترکیبی تولید کنند. این الگوریتم ها اغلب برای آزمایش اولیه یا زمانی که سرعت بر قطعیت مطلق اولویت دارد استفاده میشود. الگوریتم های رایج احتمالی تولید اعداد اول عبارتند از:

- Miller-Rabin
- Probabilistic Lucas

همچنین باید در نظر داشت که در حالت کلی الگوریتم های تولید اعداد اول به دو دسته تقسیم می شوند: الگوریتم های احتمالی و قطعی

الگوریتم های تولید اعداد اول قطعی تضمین می کنند که خروجی یک عدد اول است، اما آنها اغلب کندتر از الگوریتم های احتمالی هستند. این الگوریتم ها معمولا بر ویژگی های ریاضی خاص اعداد اول تکیه می کنند و ممکن است محاسبات پیچیده‌تر و زمان طولانی تری را شامل شوند. الگوریتم های متداول تولید اعداد اول قطعی عبارتند از:

- Pollard's rho algorithm
- Sieve of Eratosthenes

۹.۱ سوال نهم

مقدمه

اثبات دانایی صفر (Zero-Knowledge Proofs) روش‌های رمزنگاری هستند که به یک طرف (اثبات‌کننده) اجازه می‌دهند تا به طرف دیگر (تأییدکننده) ثابت کند که مقداری را می‌داند، بدون اینکه هیچ اطلاعاتی در مورد آن مقدار فاش کند. اثبات دانایی صفر غیرتعاملی (Non-Interactive Zero-Knowledge Proofs) نوعی هستند که تعامل بین اثبات‌کننده و تأییدکننده به حداقل می‌رسد، معمولاً فقط یک پیام از اثبات‌کننده به تأییدکننده.

یکی از معروف‌ترین NIZKPs مورد استفاده در فناوری زنجیره بلوکی، **zk-SNARKs** یا Zero-Knowledge Succinct Non-Interactive Arguments of Knowledge است.

اجزای zk-SNARKs

اجزای اصلی zk-SNARKs عبارتند از:

۱. **زبان بیانی:** توصیف مسئله‌ای که باید اثبات شود.
۲. **تنظیمات اولیه (Setup Phase):** تولید پارامترهای عمومی و خصوصی.
۳. **ساخت اثبات (Proof Generation):** اثبات دانایی بدون افشای اطلاعات.
۴. **تأیید اثبات (Proof Verification):** تأیید اعتبار اثبات بدون نیاز به تعامل.

توضیح فرآیند

فرآیند zk-SNARKs را می‌توان به مراحل زیر تقسیم کرد:

تنظیمات اولیه (Setup Phase)

- یک الگوریتم راه‌اندازی وجود دارد که دو مجموعه پارامتر تولید می‌کند: پارامترهای عمومی و پارامترهای خصوصی.
- پارامترهای عمومی به‌طور عمومی منتشر می‌شوند و برای ساخت و تأیید اثبات‌ها استفاده می‌شوند.
- پارامترهای خصوصی مخفی نگه داشته می‌شوند و تنها برای اطمینان از امنیت پروتکل ضروری هستند.

ساخت اثبات (Proof Generation)

- اثبات‌کننده با استفاده از پارامترهای عمومی و دانش مخفی خود، یک اثبات غیرتعاملی تولید می‌کند.
- این اثبات شامل یک رشته کوتاه از داده‌ها است که ثابت می‌کند اثبات‌کننده واقعاً دانش مخفی مورد نیاز را دارد، بدون اینکه آن دانش را فاش کند.

تأیید اثبات (Proof Verification)

- تأییدکننده با استفاده از پارامترهای عمومی و اثبات تولید شده، می‌تواند صحت اثبات را تأیید کند.
- تأیید اثبات بسیار سریع است و نیازی به تعامل با اثبات‌کننده ندارد.

کاربرد در زنجیره بلوکی

- **حفظ حریم خصوصی:** در ارزهای رمزنگاری شده مانند Zcash، از zk-SNARKs برای مخفی‌سازی جزئیات تراکنش‌ها (مانند فرستنده، گیرنده و مقدار تراکنش) استفاده می‌شود.
- **کاهش بار محاسباتی:** تأیید تراکنش‌ها به‌صورت غیرتعاملی و سریع انجام می‌شود، که می‌تواند بار محاسباتی را کاهش دهد و به مقیاس‌پذیری شبکه کمک کند.

مثال کاربردی

فرض کنید یک فرستنده می‌خواهد اثبات کند که یک مقدار خاص از ارز رمزنگاری شده را دارد و می‌تواند آن را به گیرنده منتقل کند، بدون اینکه جزئیات تراکنش (مانند مقدار دقیق) را فاش کند. با استفاده از zk-SNARKs، فرستنده می‌تواند اثبات کند که تراکنش

معتبر است، در حالی که اطلاعات حساس مخفی باقی می‌مانند. گیرنده و سایر اعضای شبکه می‌توانند به راحتی و بدون نیاز به تعامل با فرستنده، صحت این اثبات را تأیید کنند.

نتیجه‌گیری

zk-SNARKs به عنوان یک ابزار قدرتمند در زنجیره بلوکی برای ایجاد تراکنش‌های امن و خصوصی به‌کار گرفته می‌شوند، و به افزایش اعتماد و کارایی در سیستم‌های غیرمتمرکز کمک می‌کنند.

۱۰.۱ سوال دهم

الگوریتم کلید نامتقارن رابین (۱۹۷۹)

الگوریتم کلید نامتقارن **رابین** توسط **مایکل او. رابین** در سال ۱۹۷۹ ارائه شد. این الگوریتم بر مبنای دشواری **فاکتورگیری** (تجزیه به عوامل اول) طراحی شده است و به عنوان یکی از اولین الگوریتم‌های رمزنگاری کلید عمومی محسوب می‌شود.

مفاهیم کلی

الگوریتم رابین از ویژگی‌های **اعداد اول** و **محاسبه جذر** به صورت پیمانه‌ای استفاده می‌کند. امنیت این الگوریتم به سختی فاکتورگیری عدد مرکب n بستگی دارد، که از حاصل‌ضرب دو عدد اول بزرگ p و q به دست می‌آید.

مراحل الگوریتم

۱. تولید کلیدها

۱. دو عدد اول بزرگ p و q انتخاب کنید به طوری که هر دو به پیمانه ۴ با عدد ۳ هم‌نهشت باشند:

$$p \equiv 3 \pmod 4, \quad q \equiv 3 \pmod 4.$$

۲. مقدار n را به صورت زیر محاسبه کنید:

$$n = p \times q.$$

۳. کلید عمومی n است و کلید خصوصی شامل p و q می‌شود.

۲. رمزگذاری

۱. فرض کنید M یک پیام به شکل عددی باشد که $M < n$.

۲. برای رمزگذاری پیام، مقدار C را به صورت زیر محاسبه کنید:

$$C = M^2 \pmod{n}.$$

۳. C متن رمز شده است.

۳. رمزگشایی

۱. برای رمزگشایی C، نیاز به مقادیر خصوصی p و q داریم.
۲. از الگوریتم **چینی باقیمانده** (CRT) برای محاسبه جذرهای C به پیمانه p و q استفاده می‌کنیم.
۳. چهار مقدار ممکن به عنوان جذر C وجود دارد. یکی از این مقادیر همان پیام اصلی M است.

ویژگی‌های الگوریتم

- امنیت این الگوریتم به طور مستقیم به **دشواری فاکتورگیری** عدد n وابسته است.
- رمزگذاری در این الگوریتم بسیار سریع است؛ زیرا شامل یک **عملیات مربع کردن** ساده به پیمانه n می‌شود.
- رمزگشایی پیچیده‌تر از رمزگذاری است، زیرا باید چهار جذر مختلف را محاسبه و بررسی کرد.
- یکی از چالش‌های این الگوریتم، تولید **چهار جذر ممکن** در فرآیند رمزگشایی است، که نیازمند حل ابهام برای یافتن پیام صحیح می‌باشد.

۱۱.۱ سوال یازدهم

الگوریتم **امضای دیجیتال DSA** (Digital Signature Algorithm) که در سال ۱۹۹۱ توسط NIST پیشنهاد و بعدها در قالب استاندارد FIPS 186-1 منتشر شد، مبتنی بر مفاهیم گروه‌های انتخابی و توابع گسسته لگاریتم است. این الگوریتم در سه مرحله‌ی کلی کار می‌کند: تولید کلید، امضا کردن پیام، و در نهایت راستی‌آزمایی (تأیید) امضا. مراحل امضای پیام به‌صورت زیر خلاصه می‌شود:

۱. **تولید پارامترها و کلیدها:**

- ابتدا پارامترهای اصلی الگوریتم (شامل اعداد اول و پایه‌ی مولد) توسط یک مرجع یا کاربر تولید می‌شود. این پارامترها را می‌توان در سطح سیستم یا برای هر کاربر مجزا تولید کرد.
- هر کاربر یک کلید خصوصی و متناظر با آن یک کلید عمومی خواهد داشت. کلید خصوصی عددی تصادفی و کم‌حجم (نسبت به اندازه‌ی ماژول اصلی) است و کلید عمومی با استفاده از یک تابع توان (ماژولار) از کلید خصوصی محاسبه می‌شود.

۲. **تولید امضا:**

- فرستنده ابتدا پیام خود را توسط یک تابع درهم‌ساز (Hash) نظیر SHA-1 یا SHA-2 خلاصه می‌کند تا هش پیام به‌دست آید.
- برای هر امضاکردن، یک عدد تصادفی موقتی (مثلاً k) انتخاب می‌شود. سپس با استفاده از این عدد تصادفی و پارامترهای عمومی، مؤلفه‌ی اول امضا (معمولاً با r نمایش داده می‌شود) تولید می‌گردد.
- مؤلفه‌ی دوم امضا (معمولاً s) نیز با درنظرگرفتن کلید خصوصی فرستنده، هش پیام، و همان عدد تصادفی k (به‌صورت وارون ماژولار) به‌دست می‌آید.
- در نهایت جفت (r, s) به‌عنوان امضای دیجیتال پیام ارسال می‌شود.

۳. **تأیید امضا:**

- گیرنده ابتدا پارامترهای عمومی فرستنده (شامل کلید عمومی او) و امضای دریافت‌شده (r, s) را استخراج می‌کند.
- با محاسبه‌ی دوباره‌ی هش پیام و انجام محاسباتی شامل r، s و کلید عمومی فرستنده، گیرنده تأیید می‌کند که این امضا صحیح است یا خیر.
- اگر نتیجه با r دریافت‌شده یکسان باشد، امضا معتبر خواهد بود؛ در غیر این صورت امضا رد می‌شود.

۱۲.۱ سوال دوازدهم

امروزه نسخه‌های بیضوی از الگوریتم‌های مبتنی بر DH (Diffie-Hellman)، بسیار پرکاربرد هستند. این الگوریتم‌ها در حوزه‌ی رمزنگاری با خم‌های بیضوی (Elliptic Curves) عمل می‌کنند که امنیت بالاتر و اندازه‌ی کلید کوچک‌تری نسبت به روش‌های قدیمی‌تر (مانند RSA) دارند. ایده‌ی کلی مبتنی بر سختیِ مسئله‌ی **لگاریتم گسسته** در گروه نقاط یک خم بیضوی است:

- در فضای خم بیضوی، یک نقطه‌ی پایه (Generator) انتخاب می‌شود. هر کاربر یک عدد تصادفی کوچک (نسبت به مرتبه‌ی خم) به‌عنوان کلید خصوصی تولید می‌کند و با ضرب اسکالر آن عدد در نقطه‌ی پایه، کلید عمومی خود را به‌دست می‌آورد.
- در تبادل کلید ECC-DH، اگر کاربر اول عدد خصوصی d_A و نقطه‌ی عمومی $Q_A = d_A G$ داشته باشد و کاربر دوم هم عدد خصوصی d_B و نقطه‌ی عمومی $Q_B = d_B G$، آنگاه کلید مشترک با محاسبه‌ی $d_A Q_B$ یا $d_B Q_A$ به‌دست می‌آید. این دو مقدار (در ریاضیات خم بیضوی) برابر هستند و عملاً راز مشترک طرفین محسوب می‌شوند.

مثال ساده:

۱. **کاربر اول** (آلیس) کلید خصوصی d_A را برمی‌گزیند و کلید عمومی‌اش را به‌شکل $Q_A = d_A G$ می‌سازد.

۲. **کاربر دوم** (باب) هم کلید خصوصی d_B و کلید عمومی $Q_B = d_B G$ را تولید می‌کند.

۳. آلیس پس از دریافت Q_B، نقطه‌ی $d_A Q_B$ را محاسبه می‌کند.

۴. باب نیز پس از دریافت Q_A، نقطه‌ی $d_B Q_A$ را محاسبه می‌کند.

۵. طبق خواص گروه خم بیضوی، داریم:

$$d_A Q_B = d_A(d_B G) = d_B(d_A G) = d_B Q_A,$$

که همان کلید اشتراکی نهایی بین آن‌هاست.

۱۳.۱ سوال سیزدهم

طاهر الجمال (Taher Elgamal) الگوریتمی مبتنی بر DH را در سال ۱۹۸۴ ارائه داد که بعدها با نام **الگوریتم الگمال** (ElGamal) شناخته شد. این روش یک رمزنگاری نامتقارن (کلید عمومی) است. ایده‌ی اصلی آن به شرح زیر است:

- بر روی یک گروه با عمل ضربی (یا جمعی در خم‌های بیضوی) کار می‌کنیم که **لگاریتم گسسته** در آن دشوار است.
- شخص گیرنده (مثلاً باب) یک کلید خصوصی x برمی‌گزیند و با یک پارامتر عمومی g (پایه) و عدد اول بزرگ p (ماژول اصلی)، کلید عمومی‌اش را $y = g^x \bmod p$ منتشر می‌کند.
- فرستنده (مثلاً آلیس)، برای رمزکردن پیام M:
  ۱. یک عدد تصادفی k انتخاب می‌کند.
  ۲. کامپوننت اول متن رمز را $c_1 = g^k \bmod p$ می‌سازد.
  ۳. کامپوننت دوم متن رمز را $c_2 = M \times (y^k \bmod p)$ می‌سازد.
  ۴. متن رمز را به‌شکل جفت (c_1, c_2) به باب ارسال می‌کند.
- باب برای رمزگشایی از کلید خصوصی x استفاده می‌کند و از طریق محاسبه‌ی $(c_1)^x \bmod p$ به مقدار مشترک y^k دست می‌یابد و سپس $M = c_2 \times (c_1^x)^{-1} \bmod p$ را به‌دست می‌آورد.

مثال ساده:

۱. **تولید کلید:**
- باب یک عدد اول بزرگ p و پایه‌ی g را انتخاب می‌کند.
- عدد خصوصی باب x است و عدد عمومی او $y = g^x \bmod p$.

۲. **رمزگذاری:**
- آلیس پیام M دارد.
- آلیس عدد تصادفی k را برمی‌گزیند و $c_1 = g^k \bmod p$ را محاسبه می‌کند.
- سپس $c_2 = M \times (y^k \bmod p)$ را می‌سازد.
- بسته‌ی رمز را (c_1, c_2) برای باب می‌فرستد.

۳. **رمزگشایی:**
- باب با استفاده از x محاسبه می‌کند:

$$(c_1)^x \bmod p = (g^k)^x \bmod p = g^{kx} \bmod p = y^k.$$

- از آنجا که $c_2 = M \times y^k$ است، با ضرب در وارون y^k در مد p می‌تواند M را بازیابی کند:

$$M = c_2 \times (y^k)^{-1} \bmod p.$$

۱۴.۱ سوال چهاردهم

این دو عکس مربوط به افراد برجسته در حوزه امنیت هستند:

۱. **بروس اشنایر (Bruce Schneier):** بروس اشنایر یکی از متخصصین برجسته در حوزه امنیت سایبری و رمزنگاری است. او نویسنده کتاب‌ها و مقالات بسیاری در زمینه امنیت اطلاعات است. برخی از مهم‌ترین کتاب‌های او عبارتند از *Applied Cryptography* و *Lies and Secrets* که به موضوعات مختلف امنیت اطلاعات و تحلیل سیستم‌های امنیتی پرداخته است. مفاهیمی همچون *Theater Security* (نمایش امنیت) که به اقدامات ناکارآمد امنیتی اشاره دارد، توسط او معرفی شده است. او الگوریتم‌های رمزنگاری Blowfish و Twofish را طراحی کرده است که در زمان خود بسیار مورد استفاده قرار گرفته‌اند. همچنین، او به‌طور گسترده درباره حفظ حریم خصوصی، امنیت در فضای دیجیتال، و مسائل امنیت سایبری بحث کرده و به عنوان یکی از منتقدان سرسخت نظارت گسترده دولت‌ها شناخته می‌شود.

۲. **رالف مرکل (Ralph Merkle):** رالف مرکل یکی از بنیان‌گذاران رمزنگاری کلید عمومی و از پیشگامان در علم رمزنگاری مدرن است. او به‌ویژه به دلیل توسعه Merkle Tree (درخت مرکل) شناخته می‌شود. این ساختار داده‌ای برای تأیید یکپارچگی داده‌ها در سیستم‌های توزیع‌شده مانند بلاک‌چین و سیستم‌های همتا به همتا استفاده می‌شود. علاوه بر این، مرکل یکی از اولین افرادی بود که ایده رمزنگاری کلید عمومی را ارائه کرد. کارهای او پایه‌ای برای الگوریتم‌هایی مانند RSA و Elliptic Curve Cryptography بوده است. او همچنین در زمینه محاسبات کوانتومی و کاربردهای آن در رمزنگاری تحقیقات داشته است. دستاوردهای او نقش مهمی در توسعه فناوری‌های مدرن امنیت داده داشته است.

۲ سوالات عملی

۱.۲ سوال اول

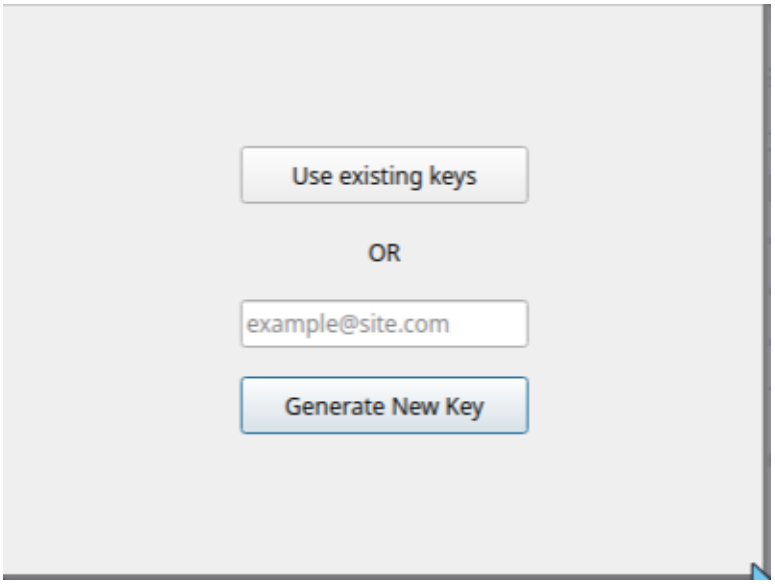
بخش اول: محیط و ابزارهای مورد استفاده

برای پیاده‌سازی محیط کاربری این برنامه از محیط توسعه Qt با زبان C++ استفاده شده است. همچنین برای تامین امنیت و استفاده از رمزنگاری از موتور قدرتمند GnuPG یا Gnu Privacy Guard استفاده شده است. GnuPG یک نرم‌افزار آزاد است که برای رمزنگاری و امضای دیجیتال استفاده می‌شود. این نرم‌افزار از استاندارد OpenPGP پیروی می‌کند و برای تضمین امنیت ارتباطات و حفاظت از داده‌های شخصی طراحی شده است.

به طور خاص در این برنامه از کتابخانه GPGME (GnuPG Made Easy) برای ارتباط با GnuPG استفاده شده است. GPGME (GnuPG Made Easy) یک کتابخانه برای زبان C/C++ است که برای اضافه کردن پشتیبانی از رمزنگاری به برنامه‌ها طراحی شده است. این کتابخانه برای ساده‌تر کردن دسترسی به موتورهای رمزنگاری عمومی مانند GnuPG یا GpgSM طراحی شده است. GPGME از GnuPG به عنوان پشتیبان خود استفاده می‌کند و از پروتکل OpenPGP و Cryptographic Message Syntax (CMS) پشتیبانی می‌کند.

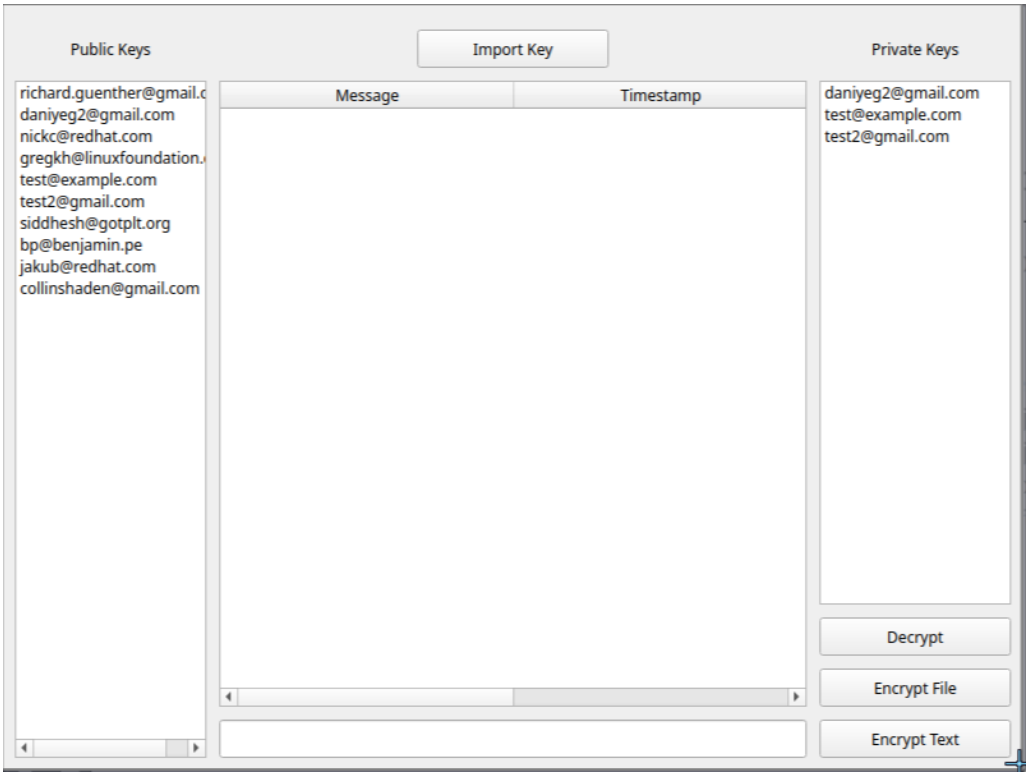
بخش دوم: نحوه استفاده از نرم‌افزار

کاربر اول با صفحه زیر روبرو می‌شود. وی می‌تواند با زدن دکمه بالا به برنامه اصلی رفته یا در همین صفحه جفت کلید جدیدی را با وارد کردن ایمیل و زدن دکمه Generate New Keys برای خود تولید کند.



شکل ۱.۲: صفحه اول برنامه

پس از عبور از صفحه اول، کاربر وارد صفحه اصلی برنامه می‌شود که به صورت زیر می‌باشد.



شکل ۲.۲: صفحه اصلی برنامه

در برنامه اصلی اجزای صفحه به صورت زیر می‌باشد:

- لیست کلیدهای عمومی (Public Key): در این قسمت کاربر می‌تواند لیست ایمیل افرادی را ببیند که به کلید عمومی آنها دسترسی داشته و می‌تواند برای آنها پیامی را رمزنگاری کند.

- لیست کلیدهای خصوصی (Private Key): در این قسمت کاربر می‌تواند لیست ایمیل افرادی را ببیند که به کلید خصوصی آنها دسترسی داشته و از طریق آن می‌تواند پیامهای خود را امضا کند (در این برنامه امضا کردن پیام به صورت اجباری انجام می‌شود).
- دکمه Import Key: کاربر با فشردن این دکمه به صفحه انتخاب فایل هدایت می‌شود که وی در آن می‌تواند فایل مربوطه به کلیدهای موردنظر خود را انتخاب کرده و آنها را وارد برنامه بکند تا بتواند از آنها استفاده کند.
- فیلد وارد کردن متن: در این بخش کاربر می‌تواند پیام مورد نظر خود را تایپ کرده و پس از انتخاب کلیدهای عمومی و خصوصی با فشردن دکمه Encrypt Text پیام خود را برای فرد موردنظر رمز کند.
- دکمه Encrypt Text: با زدن این دکمه پیام بخش قبل برداشته شده و سپس در فایلی به صورت رمز شده ذخیره می‌شود.
- دکمه Encrypt File: کاربر با فشردن این دکمه به صفحه انتخاب فایل هدایت می‌شود و پس از انتخاب، فایل انتخاب شده وی رمزنگاری شده و ذخیره می‌شود.
- دکمه Decrypt: کاربر با زدن این دکمه باز به صفحه انتخاب فایل هدایت می‌شود. در این صفحه کاربر می‌تواند فایل رمز شده‌ای را که برای وی ارسال شده است انتخاب کند. این فایل سپس رمزگشایی شده و در پوشه محلی ذخیره می‌شود.
- بخش گزارش: در این بخش برنامه گزارش‌های مختلف از جمله نام و محل ذخیره‌سازی فایلها و اورهای پیش آمده را به کاربر نشان می‌دهد.

بخش سوم: نحوه پیاده‌سازی

همانگونه که پیشتر ذکر شد این برنامه توسط API های تامین شده توسط کتابخانه GPGME امنیت خود را تامین می‌کند. در اینجا به عنوان مثال توابع رمزنگاری پیام و رمزگشایی فایل را بررسی می‌کنیم.

تابع رمزنگاری پیام: (در اینجا از خطهای مربوط به مدیریت اورها صرف نظر شده است)

```
۱  void MainWindow::on_encryptTextButton_clicked()
۲  {
۳      gpgme_ctx_t ctx = nullptr;
۴      gpgme_key_t pubkey = nullptr, privkey = nullptr;
۵      gpgme_error_t err;
۶      gpgme_data_t plain = nullptr, encrypted = nullptr;
۷
۸      QString plainText = ui->textLE->text();
۹      ui->textLE->clear();
۱۰     if (plainText.isEmpty()) {
۱۱         return;
۱۲     }
```

```
    if (!ui->privateKeyList->currentItem() || !ui->publicKeyList->currentItem())
    {
        addLogEntry("Please select both public and private keys");
        return;
    }
    QString privateEmail = ui->privateKeyList->currentItem()->text();
    QString publicEmail = ui->publicKeyList->currentItem()->text();

    err = gpgme_new(&ctx);
    ...
    err = gpgme_set_protocol(ctx, GPGME_PROTOCOL_OpenPGP);
    ...
    err = gpgme_get_key(ctx, publicEmail.toUtf8().constData(), &pubkey, 0);
    ...
    err = gpgme_get_key(ctx, privateEmail.toUtf8().constData(), &privkey, 1);
    ...
    err = gpgme_data_new_from_mem(&plain, plainText.toUtf8().constData(), plainText.size(), 1);
    ...
    err = gpgme_data_new(&encrypted);
    ...
    err = gpgme_signers_add(ctx, privkey);
    ...
    gpgme_key_t keys[] = {pubkey, nullptr};
    err = gpgme_op_encrypt_sign(ctx, keys, GPGME_ENCRYPT_ALWAYS_TRUST, plain, encrypted);
    ...
    QString timestamp = QDateTime::currentDateTime().toString("yyyyMMddhhmmss");
    QString filename = QString("%1.gpg").arg(timestamp);
    QFile file(filename);
    ...
    QByteArray encryptedData;
    char buffer[512];
    int n;
```

```
۴۶
۴۷      if (gpgme_data_seek(encrypted, 0, SEEK_SET) == -1)
۴۸      {
۴۹          ...
۵۰      }
۵۱
۵۲      while ((n = gpgme_data_read(encrypted, buffer, sizeof(buffer))) > 0) {
۵۳          encryptedData.append(buffer, n);
۵۴      }
۵۵
۵۶      file.write(encryptedData);
۵۷      file.close();
۵۸
۵۹      addLogEntry("The file has been successfully encrypted and saved as " + filename);
۶۰
۶۱      gpgme_data_release(plain);
۶۲      gpgme_data_release(encrypted);
۶۳      gpgme_key_unref(pubkey);
۶۴      gpgme_key_unref(privkey);
۶۵      gpgme_release(ctx);
۶۶  }
```

در این کد اول وجود پیام و انتخاب شدن کلیدهای عمومی و خصوصی بررسی می‌شود. سپس با تابع gpgme_new یک context جدید برای عملیات تشکیل می‌شود. سپس با تابع gpgme_set_protocol پروتکل مورد استفاده برابر با OpenPGP قرار می‌گیرد. سپس با استفاده از تابع gpgme_get_key کلیدهای متناظر با ایمیل‌ها دریافت می‌شود. سپس با تابع gpgme_signers_add کلید خصوصی انتخاب شده به عنوان یک امضا کننده اضافه می‌شود، و در نهایت با استفاده از تابع gpgme_op_encrypt_sign پیام مورد نظر رمز می‌شود. بعد از انجام عملیات رمزگذاری محتوای رمزگذاری شده در فایلی به نام timestamp فعلی ذخیره می شود و نام این فایل به کاربر اطلاع داده می‌شود.

تابع رمزگشایی پیام: (در اینجا از خطهای مربوط به مدیریت اورها صرف نظر شده است)

```
۱  void MainWindow::on_decryptButton_clicked()
۲  {
۳      gpgme_ctx_t ctx;
۴      gpgme_error_t err;
```

```
    gpgme_data_t encrypted, plain;

    QString fileName = QFileDialog::getOpenFileName(nullptr, "Select Encrypted File");
    if (fileName.isEmpty()) {
        return;
    }

    QFile file(fileName);
    if (!file.open(QIODevice::ReadOnly)) {
    ...
    }

    QByteArray encryptedData = file.readAll();
    file.close();

    err = gpgme_new(&ctx);
    ...
    err = gpgme_set_protocol(ctx, GPGME_PROTOCOL_OpenPGP);
    ...
    err = gpgme_data_new_from_mem(&encrypted, encryptedData.constData(), encryptedData.size(), 1);
    ...
    err = gpgme_data_new(&plain);
    ...
    gpgme_decrypt_result_t decrypt_result;
    gpgme_verify_result_t verify_result;
    err = gpgme_op_decrypt_verify(ctx, encrypted, plain);
    ...
    decrypt_result = gpgme_op_decrypt_result(ctx);
    verify_result = gpgme_op_verify_result(ctx);

    if (decrypt_result->recipients->status != GPG_ERR_NO_ERROR) {
        ...
    }
```

```
    QString decryptedFileName = fileName + ".decrypted";
    QFile decryptedFile(decryptedFileName);
    if (!decryptedFile.open(QIODevice::WriteOnly)) {
        ...
    }

    QByteArray decryptedData;
    char buffer[512];
    int n;

    if (gpgme_data_seek(plain, 0, SEEK_SET) == -1)
    {
        ...
    }

    while ((n = gpgme_data_read(plain, buffer, sizeof(buffer))) > 0) {
        decryptedData.append(buffer, n);
    }

    decryptedFile.write(decryptedData);
    decryptedFile.close();

    addLogEntry("The file has been successfully decrypted and saved as " + decryptedFileName);

    gpgme_data_release(encrypted);
    gpgme_data_release(plain);
    gpgme_release(ctx);
}
```

در این کد اول دیالوگ انتخاب فایل نمایش داده می‌شود تا کاربر فایل خود را انتخاب کند. سپس با تابع gpgme_new یک context جدید برای عملیات تشکیل می‌شود. سپس با تابع gpgme_set_protocol پروتکل مورد استفاده برابر با OpenPGP قرار می‌گیرد. سپس با استفاده از تابع gpgme_op_decrypt_verify پیام مورد نظر رمزگشایی شده و امضای آن تایید می‌شود. سپس با استفاده از توابع gpgme_op_decrypt_result و gpgme_op_verify_result وضعیت نهایی این عملیات‌ها خوانده می‌شود. در

نهایت نیز محتوای فایل رمزگشایی شده در فایلی با پسوند decrypted. ذخیره می‌شود و نام فایل به کاربر اطلاع داده می‌شود.

۲.۲ سوال دوم

بخش اول: محیط و ابزارهای مورد استفاده

محیط توسعه

برای پیاده‌سازی و مقایسه الگوریتم‌های AES و RSA از محیط توسعه Qt با زبان C++ استفاده شده است. Qt به دلیل داشتن ابزارهای قدرتمند برای طراحی رابط کاربری گرافیکی و همچنین امکانات گسترده برای ترسیم نمودارها، انتخاب مناسبی برای این پروژه بوده است. نسخه مورد استفاده شامل ماژول‌های core، gui، widgets و charts می‌باشد که برای طراحی رابط کاربری و نمایش نتایج به کار گرفته شده‌اند.

کتابخانه‌های مورد استفاده

در این پروژه، از کتابخانه OpenSSL برای پیاده‌سازی الگوریتم‌های AES و RSA استفاده شده است. OpenSSL یک کتابخانه معتبر و قدرتمند در زمینه رمزنگاری است که امکانات متنوعی را برای پیاده‌سازی الگوریتم‌های رمزنگاری فراهم می‌کند. علاوه بر این، از کلاس‌های استاندارد Qt مانند QElapsedTimer برای اندازه‌گیری زمان و QChart برای ترسیم نمودارها بهره گرفته شده است.

ساختار پروژه

ساختار پروژه شامل فایل‌های زیر می‌باشد:

- **mainwindow.cpp** و **mainwindow.h**: پیاده‌سازی رابط کاربری و تعاملات آن.
- **cryptotest.cpp** و **cryptotest.h**: پیاده‌سازی عملکردهای رمزنگاری و اندازه‌گیری زمان پاسخ‌دهی.
- **mainwindow.ui**: تعریف رابط کاربری با استفاده از ابزار طراحی Qt.
- **Q2.pro**: فایل پروژه Qt شامل تنظیمات و وابستگی‌های پروژه.

بخش دوم: پیاده‌سازی الگوریتم‌های AES و RSA

پیاده‌سازی AES

برای پیاده‌سازی AES-128-CBC از توابع ارائه شده توسط OpenSSL استفاده شده است. در این پیاده‌سازی، یک کلید AES به طول ۱۲۸ بیت به صورت تصادفی تولید شده و برای هر پیام، یک IV (Initialization Vector) نیز به صورت تصادفی ایجاد می‌شود تا امنیت عملیات رمزنگاری افزایش یابد. فرآیند رمزنگاری شامل مراحل Initialization، EncryptUpdate و EncryptFinal است که به ترتیب برای راه‌اندازی کانتکست، رمزنگاری داده‌ها و پایان عملیات رمزنگاری استفاده می‌شوند.

یکی از توابع مهم در این بخش تابع `testAES` در فایل `cryptotest.cpp` است که زمان پاسخ‌دهی AES را برای هر پیام اندازه‌گیری می‌کند. این تابع برای هر پیام یک کانتکست جدید AES ایجاد می‌کند، پیام را رمزنگاری می‌کند و زمان مورد نیاز برای این عملیات را ثبت می‌نماید.

پیاده‌سازی RSA

برای پیاده‌سازی RSA با طول کلید ۳۰۷۲ بیت، از توابع موجود در OpenSSL استفاده شده است. ابتدا یک جفت کلید عمومی و خصوصی RSA تولید می‌شود. سپس برای هر پیام، از کلید عمومی RSA و پدینگ `RSA_PKCS1_OAEP_PADDING` برای رمزنگاری استفاده می‌شود. به دلیل محدودیت‌های RSA در طول پیام قابل رمزنگاری، طول هر پیام به گونه‌ای انتخاب شده است که با توجه به طول کلید و پدینگ، امکان رمزنگاری آن فراهم باشد.

یکی از توابع مهم در این بخش تابع `testRSA` در فایل `cryptotest.cpp` است که زمان پاسخ‌دهی RSA را برای هر پیام اندازه‌گیری می‌کند. این تابع برای هر پیام، آن را با استفاده از کلید عمومی RSA رمزنگاری کرده و زمان مورد نیاز برای این عملیات ثبت می‌گردد.

بخش سوم: مقایسه

انتخاب طول کلید

برای مقایسه عادلانه AES و RSA از نظر امنیت، طول کلیدهای این دو الگوریتم به گونه‌ای انتخاب شده‌اند که معادل از نظر امنیتی باشند. بر اساس منابع معتبر، کلید ۱۲۸ بیتی AES معادل امنیتی کلید ۳۰۷۲ بیتی RSA می‌باشد. این انتخاب تضمین می‌کند که هر دو الگوریتم از نظر مقاومت در برابر حملات برابر هستند و نتایج مقایسه به طور مستقیم قابل مقایسه خواهند بود.

تولید پیام‌های تصادفی

برای انجام مقایسه، هزار پیام تصادفی به طول ۱۲۸ بایت تولید شده‌اند. از توابع `RAND_bytes` در OpenSSL برای تولید داده‌های تصادفی استفاده شده است تا تضمین شود که داده‌ها به صورت کاملاً تصادفی و بدون الگوی خاصی تولید شوند. این پیام‌ها به عنوان ورودی برای عملیات رمزنگاری توسط هر دو الگوریتم AES و RSA استفاده شده‌اند.

اندازه‌گیری زمان پاسخ‌دهی

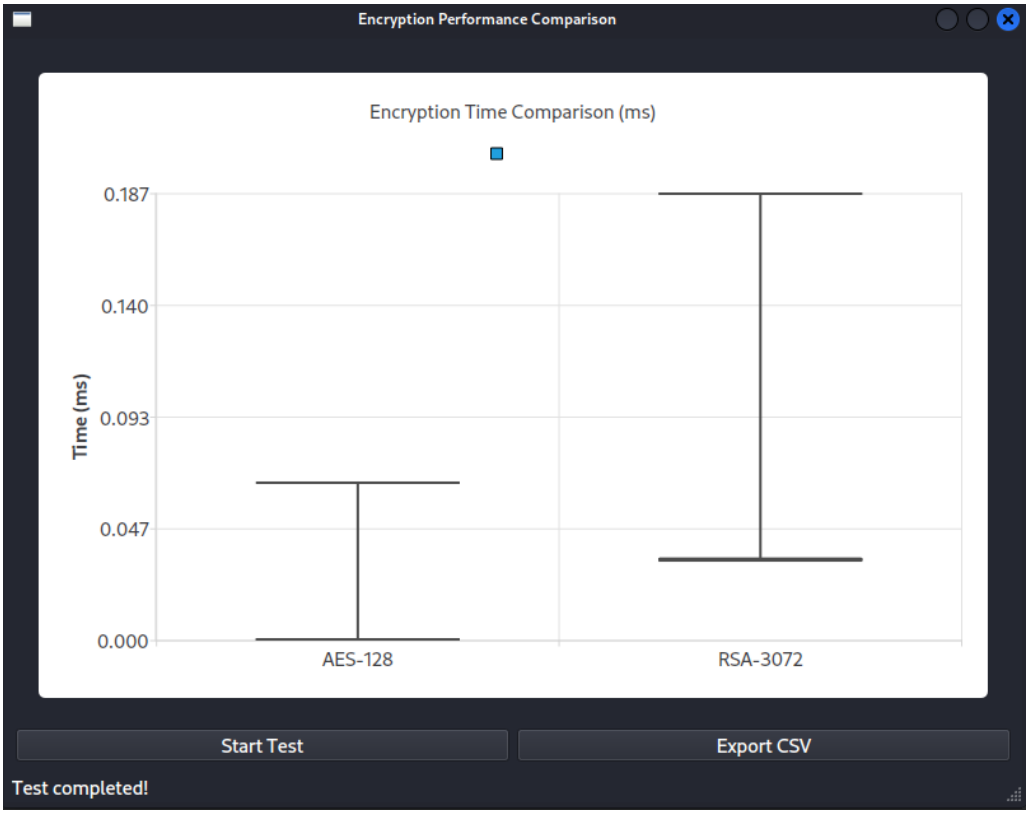
زمان پاسخ‌دهی هر دو الگوریتم برای رمزنگاری هر پیام به صورت جداگانه اندازه‌گیری شده است. برای این منظور، از کلاس `QElapsedTimer` در Qt استفاده شده است که امکان اندازه‌گیری دقیق زمان‌های بسیار کوتاه را فراهم می‌کند. زمان‌های اندازه‌گیری شده به میلی‌ثانیه تبدیل و در وکتورهای جداگانه برای AES و RSA ذخیره شده‌اند. این داده‌ها سپس برای تحلیل و ترسیم نمودار BoxPlot استفاده خواهند شد.

ترسیم نمودار BoxPlot

برای نمایش توزیع زمان‌های پاسخ‌دهی AES و RSA، از نمودار BoxPlot استفاده شده است. این نمودار شامل مقادیر مینیمم، بیشینه، میانه، چارک اول و سوم است که اطلاعات جامعی درباره پراکندگی و توزیع داده‌ها ارائه می‌دهد. استفاده از BoxPlot امکان مقایسه بصری دقیق‌تر بین دو الگوریتم را فراهم می‌کند و نقاط قوت و ضعف هر یک را به وضوح نشان می‌دهد.

بخش چهارم: نتایج و تحلیل

نمودار BoxPlot زمان پاسخ‌دهی



شکل ۳.۲: نمودار BoxPlot مقایسه زمان پاسخ‌دهی AES و RSA

تحلیل نتایج

از نمودار BoxPlot مشاهده می‌شود که الگوریتم AES-128-CBC نسبت به RSA-3072 به طور قابل توجهی زمان پاسخ‌دهی کمتری دارد. این تفاوت عمدتاً به دلیل طبیعت الگوریتم‌های رمزنگاری متقارن و نامتقارن است. AES به عنوان یک الگوریتم متقارن، برای رمزنگاری و رمزگشایی داده‌ها سریع‌تر عمل می‌کند زیرا عملیات ریاضی ساده‌تری را نسبت به RSA انجام می‌دهد. از سوی دیگر، RSA به دلیل پیچیدگی‌های بیشتری که در عملیات رمزنگاری دارد و نیاز به محاسبات عددی بزرگ، زمان بیشتری را صرف می‌کند.

همچنین، پراکندگی داده‌های زمان پاسخ‌دهی RSA بیشتر از AES است که نشان‌دهنده نوسانات بیشتری در زمان رمزنگاری RSA نسبت به AES می‌باشد. این امر می‌تواند به دلیل عوامل مختلفی مانند بار پردازشی سیستم یا پیچیدگی‌های داخلی RSA

باشد. به طور کلی، نتایج نشان‌دهنده این است که برای کاربردهایی که نیاز به رمزنگاری سریع دارند، AES گزینه بهتری نسبت به RSA است.

بخش پنجم: نتیجه‌گیری

در این گزارش، به مقایسه زمان پاسخ‌دهی الگوریتم‌های AES و RSA در بستر Qt با استفاده از زبان C++ پرداخته شد. نتایج نشان داد که AES-128-CBC نسبت به RSA-3072 در رمزنگاری پیام‌ها سریع‌تر عمل می‌کند. این تفاوت عمدتاً به دلیل نوع الگوریتم‌های متقارن و نامتقارن و پیچیدگی‌های داخلی آن‌ها می‌باشد. با توجه به امنیت معادل انتخاب شده برای طول کلیدهای AES و RSA، می‌توان نتیجه گرفت که برای کاربردهایی که نیاز به رمزنگاری سریع دارند، AES گزینه مناسبی است. از سوی دیگر، RSA برای مواردی که نیاز به تبادل کلید یا ایجاد امضاهای دیجیتال دارند، همچنان کاربردی و مهم باقی می‌ماند.

بخش ششم: توابع مهم پیاده‌سازی

در این بخش، به توضیح توابع کلیدی پیاده‌سازی در دو فایل `cryptotest.cpp` و `mainwindow.cpp` پرداخته می‌شود.

توابع اصلی در cryptotest.cpp

تابع `testAES`:

این تابع زمان پاسخ‌دهی الگوریتم AES را برای هر پیام در وکتور ورودی اندازه‌گیری می‌کند. برای هر پیام، یک کانتکست جدید AES ایجاد شده، پیام رمزنگاری می‌شود و زمان مورد نیاز برای این عملیات ثبت می‌گردد.

```
QVector<double> CryptoTest::testAES(const QVector<QByteArray> &messages)
{
    QVector<double> times;
    times.reserve(messages.size());

    for (const QByteArray &msg : messages) {
        EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
        if (!ctx) {
            qWarning() << "Failed to create cipher context";
            continue;
        }

        unsigned char iv[16];
        RAND_bytes(iv, sizeof(iv));

```

```
        QElapsedTimer timer;
        timer.start();

        if (EVP_EncryptInit_ex(ctx, EVP_aes_128_cbc(), nullptr, aesKey, iv) != 1) {
            qWarning() << "AES init failed";
            EVP_CIPHER_CTX_free(ctx);
            continue;
        }

        QByteArray ciphertext(msg.size() + EVP_MAX_BLOCK_LENGTH, 0);
        int len = 0;
        int ciphertext_len = 0;

        if (EVP_EncryptUpdate(ctx,
        reinterpret_cast<unsigned char*>(ciphertext.data()),
        &len,
        reinterpret_cast<const unsigned char*>(msg.constData()),
        msg.size()) != 1) {
            qWarning() << "AES encrypt update failed";
            EVP_CIPHER_CTX_free(ctx);
            continue;
        }
        ciphertext_len = len;

        if (EVP_EncryptFinal_ex(ctx,
        reinterpret_cast<unsigned char*>(ciphertext.data()) + len,
        &len) != 1) {
            qWarning() << "AES encrypt final failed";
            EVP_CIPHER_CTX_free(ctx);
            continue;
        }
        ciphertext_len += len;
        ciphertext.resize(ciphertext_len);
```

```
        double elapsedMs = timer.nsecsElapsed() / 1e6;
        times.push_back(elapsedMs);

        EVP_CIPHER_CTX_free(ctx);
    }

    return times;
}
```

تابع testRSA:

این تابع زمان پاسخ‌دهی الگوریتم RSA را برای هر پیام در وکتور ورودی اندازه‌گیری می‌کند. برای هر پیام، آن را با استفاده از کلید عمومی RSA رمزنگاری کرده و زمان مورد نیاز برای این عملیات ثبت می‌گردد.

```
QVector<double> CryptoTest::testRSA(const QVector<QByteArray>& messages)
{
    QVector<double> times;
    times.reserve(messages.size());

    int rsa_size = RSA_size(rsaKeyPair);

    for (const QByteArray &msg : messages) {
        QByteArray encrypted(rsa_size, 0);

        QElapsedTimer timer;
        timer.start();

        int encrypted_length = RSA_public_encrypt(
        qMin(msg.size(), rsa_size - 42),
        reinterpret_cast<const unsigned char*>(msg.constData()),
        reinterpret_cast<unsigned char*>(encrypted.data()),
        rsaKeyPair,
        RSA_PKCS1_OAEP_PADDING
        );

```

```
        double elapsedMs = timer.nsecsElapsed() / 1e6;

        if (encrypted_length == -1) {
            qWarning() << "RSA encryption failed:"
            << ERR_error_string(ERR_get_error(), nullptr);
            continue;
        }

        times.push_back(elapsedMs);
    }

    return times;
}
```

توابع اصلی در mainwindow.cpp

تابع `onStartTest`:

این تابع هنگامی که کاربر بر روی دکمه `Start Test` کلیک می‌کند، اجرا می‌شود. وظیفه این تابع تولید پیام‌های تصادفی، اجرای آزمون‌های AES و RSA، و ترسیم نمودار BoxPlot است.

تابع `onExportCSV`:

این تابع هنگامی که کاربر بر روی دکمه `Export CSV` کلیک می‌کند، اجرا می‌شود. وظیفه این تابع صادر کردن نتایج آزمون‌های AES و RSA به یک فایل CSV است.

تابع `createBoxPlot`:

این تابع داده‌های زمان پاسخ‌دهی AES و RSA را دریافت کرده و نمودار BoxPlot مربوطه را ترسیم می‌کند. نمودار شامل مقادیر مینیمم، بیشینه، میانه، چارک اول و سوم برای هر دو الگوریتم است.

بخش هفتم: چالش‌ها و راهکارها

در فرآیند پیاده‌سازی و مقایسه الگوریتم‌های AES و RSA با استفاده از Qt و C++، با چالش‌های متعددی مواجه شدیم که مهم‌ترین آن‌ها عبارت بودند از:

- **انتخاب و تنظیمات صحیح کتابخانه OpenSSL:** اطمینان از سازگاری نسخه‌های مختلف OpenSSL با پروژه و تنظیمات مناسب برای استفاده از توابع رمزنگاری از چالش‌های اصلی بود. مطالعه مستندات OpenSSL و انجام تست‌های متعدد به ما کمک کرد تا این مشکلات را برطرف کنیم.

- **اندازه‌گیری دقیق زمان پاسخ‌دهی:** استفاده از ابزارهای دقیق مانند `QElapsedTimer` برای اندازه‌گیری زمان رمزنگاری بدون دخالت عوامل خارجی ضروری بود. اجرای تست‌ها در محیط‌های کنترل‌شده و کاهش بار پردازشی سیستم به دقت اندازه‌گیری کمک کرد.
- **طراحی رابط کاربری مناسب:** ایجاد یک رابط کاربری کاربرپسند که امکان شروع تست، نمایش وضعیت و ترسیم نمودارها را به راحتی فراهم کند، از دیگر چالش‌های پروژه بود. استفاده از ابزارهای طراحی Qt Designer و انجام تست‌های کاربری به بهبود تجربه کاربر منجر شد.

برای مقابله با این چالش‌ها، راهکارهای زیر اتخاذ شد:

- مطالعه مستندات OpenSSL و Qt برای درک بهتر از نحوه استفاده و تعامل آن‌ها با C++.
- اجرای تست‌های متعدد برای اطمینان از دقت اندازه‌گیری زمان پاسخ‌دهی و کاهش تأثیر عوامل خارجی.
- طراحی رابط کاربری با استفاده از ابزارهای طراحی Qt Designer و تست‌های کاربری برای بهبود تجربه کاربر.

بخش هشتم: پیشنهادات

برای بهبود عملکرد و افزایش دقت مقایسه‌ها، پیشنهاد می‌شود:

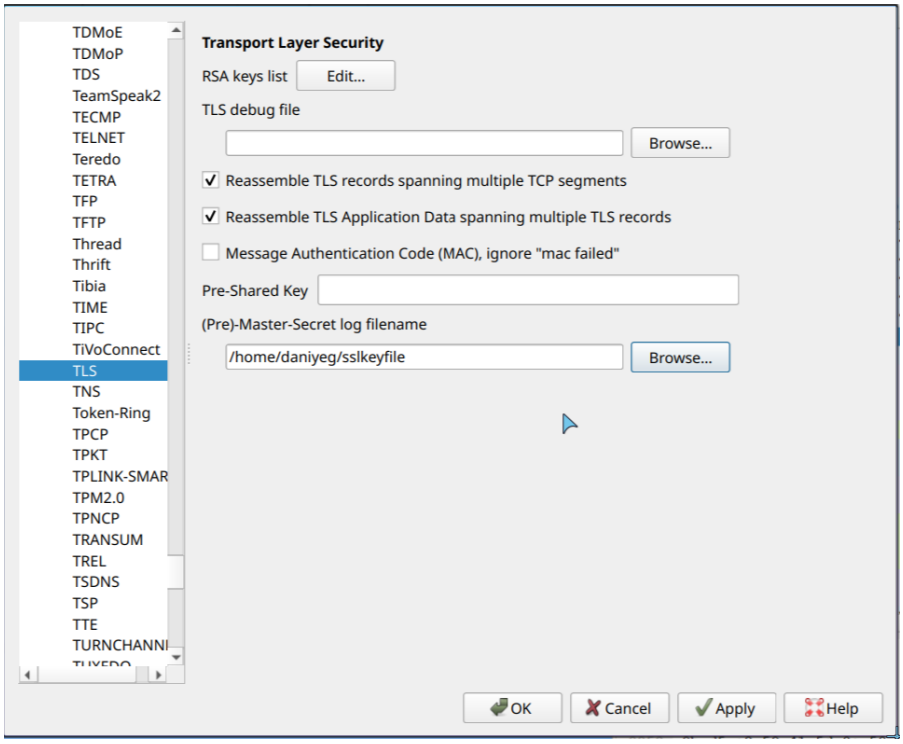
- **افزایش تعداد پیام‌های تست شده:** افزایش تعداد پیام‌های تست شده به بیش از هزار پیام برای دستیابی به نتایج دقیق‌تر و کاهش تاثیر نوسانات تصادفی.
- **بررسی عملکرد الگوریتم‌ها در شرایط مختلف بار پردازشی سیستم:** تست عملکرد AES و RSA در سیستم‌هایی با بار پردازشی متفاوت به منظور درک بهتر از تاثیر شرایط محیطی بر زمان پاسخ‌دهی.
- **بهینه‌سازی کد:** بهینه‌سازی کدهای پیاده‌سازی شده برای کاهش زمان پاسخ‌دهی و افزایش کارایی کلی سیستم.

۳.۲ سوال سوم

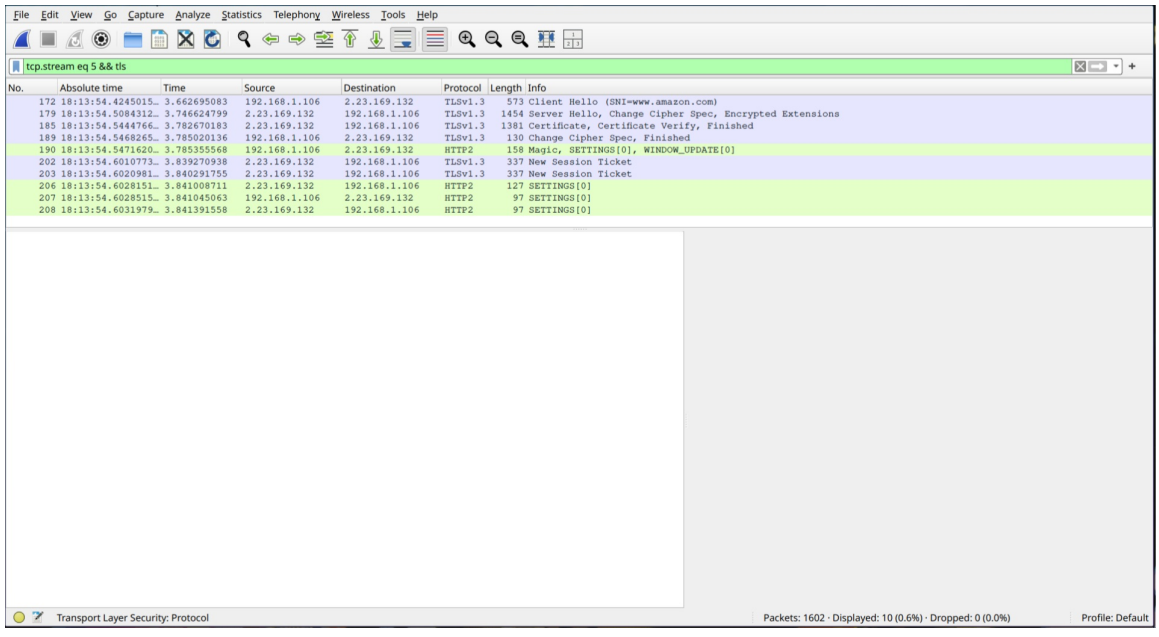
برای دسترسی به محتویات رمزنگاری شده در درون بسته‌ها نیاز داریم تا به مرورگر خود بفهمانیم که باید Pre-Master-Secret ها را در مکانی ذخیره کند تا بتوانیم از آنها در برنامه Wireshark استفاده کنیم. در مرورگر فایرفاکس می‌توانیم با استفاده از متغیر محیطی SLLKEYLOGFILE فایلی را برای ذخیره Pre-Master-Secret ها تعیین کنیم.

```
% export SSLKEYLOGFILE="/home/daniyeg/sslkeyfile"
% firefox
```

سپس در برنامه Wireshark از منوی بالا به بخش Edit -> Preferences رفته و در منوی بغل Protocols -> TLS را انتخاب کرده و فایل مورد نظر خود را در بخش (Pre)-Master-Secret log filename انتخاب می‌کنیم.



حالا می‌توانیم به محتویات داخل بسته‌ها دسترسی داشته باشیم. در مثال پایین به سایت www.amazon.com وصل شده‌ایم و همانطور که مشاهده می‌شود فریم‌های http2 تبادل شده (با اینکه رمزگذاری شده‌اند) قابل مشاهده هستند. همچنین می‌توان مشاهده کرد که ۴ پیام برای برقراری ارتباط TLS رد و بدل شده است که در ادامه به محتویات آنها خواهیم پرداخت.



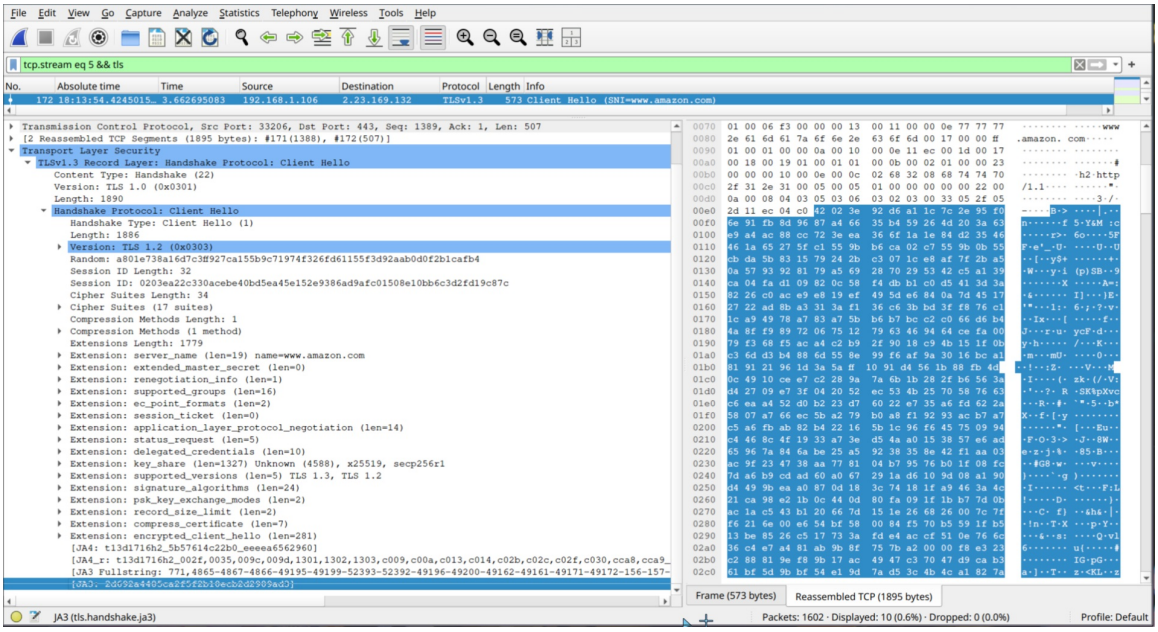
تصویر زیر محتویات پیام اول را نشان می‌دهد که شامل رکورد Client Hello می‌باشد. این رکورد شامل فیلدهای زیر می‌باشد:

- Version: فیلد نسخه که نسخه ارتباطات را مشخص می‌کند. این فیلد در TLS 1.3 استفاده نمی‌شود و همواره دارای مقادیر 0x301 به معنای TLS 1.0 یا 0x303 به معنای TLS 1.2 می‌باشد.

- Random: یک عدد رندوم ۳۲ بایتی که توسط کلاینت تولید شده است و در تولید کلیدهای نهایی برای رمزنگاری نقش دارد.
- Session ID: یک شناسه یکتا که مختص این اتصال می‌باشد و برای سرگیری اتصال در اتصال‌های بعد استفاده می‌شود تا سرعت اتصال مجدد بهبود یافته و میزان اطلاعات تکراری کمتر شود.
- Cipher Suites: الگوریتم‌های رمزنگاری را مشخص می‌کند که کلاینت از آنها پشتیبانی می‌کند و مایل است که در این اتصال از آن استفاده کند.
- Compression Methods: الگوریتم‌های فشرده‌سازی را مشخص می‌کند که کلاینت مایل است از آنها استفاده شود (در این مثال این فیلد خالی می‌باشد که به معنای فشرده نبودن پیام‌ها می‌باشد).

سپس چندین extension رد و بدل می‌شود که محتویات این اکستنشن‌ها برای عملکرد این پروتکل مهم است. اکستنشن‌های مهم در این پیام عبارتند از:

- server_name: دامنه سروری که کلاینت می‌خواهد به آن متصل شود.
- supported_groups: منحنی بیضی و گروه‌های میدان محدودی را که کلاینت برای تبادل کلید از آنها پشتیبانی می کند نشان می‌دهد و مربوط به رمزنگاری بیضوی دیفی-هلمن می‌باشد.
- session_ticket: این فیلد مشخص می‌کند که آیا کلاینت دارای Session Ticket ای می‌باشد یا خیر. از Session Ticket برای کاهش طول فرآیند دست دادن در اتصال مجدد به یک سرور استفاده می‌شود.
- application_layer_protocol_negotiation: پروتکل‌های لایه کاربرد که کلاینت مایل است از آنها استفاده کند (در این مثال پروتکل‌های موجود h۲ و http۱.۱ می‌باشند).
- status_request: روش بررسی ملغی شدن گواهی که در این مثال OCSP می‌باشد. از این اکستنشن معمولا در روش OCSP Stapling برای درخواست انجام این بررسی در سمت سرور استفاده می‌شود.
- key_share: اطلاعات مربوط به توافق کلید کلاینت در روش‌های مبتنی بر دیفی-هلمن.
- supported_versions: نسخه‌های TLS پشتیبانی شده توسط کلاینت. در این اکستنشن نسخه‌های واقعی پروتکل تعیین می‌شود.
- signature_algorithms: الگوریتم‌های امضای دیجیتالی که کلاینت از انها پشتیبانی می‌کند.
- record_size_limit: این اکستنشن سایز هر رکورد را برای این ارتباط محدود می‌کند.
- compress_certificate: این اکستنشن از سرور درخواست می‌کند که گواهی خود را فشرده کرده و الگوریتم‌های فشرده‌سازی پشتیبانی شده توسط کلاینت را تعیین می‌کند.
- encrypted_client_hello: این اکستنشن شامل فیلدهای حساس Client Hello می‌باشد که به صورت رمز شده برای سرور ارسال می‌شود.

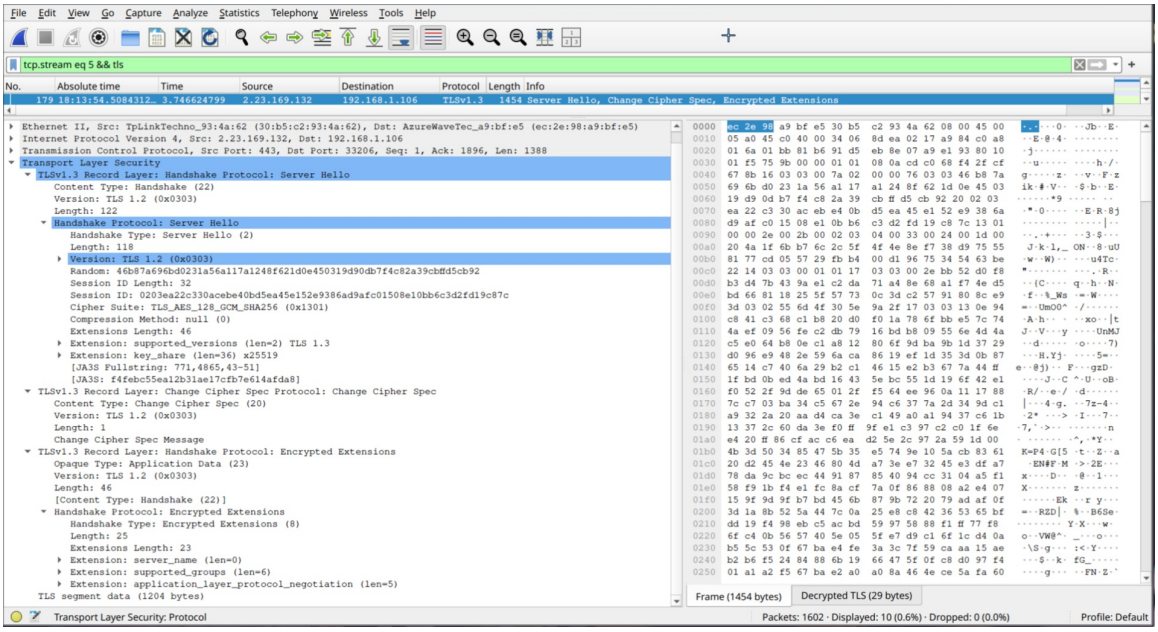


تصویر زیر محتویات پیام دوم را نشان می‌دهد که شامل چندین رکورد می‌باشد. اولین رکورد یعنی Hello Server شامل فیلدهای زیر و اکستنشن‌های می‌باشد:

- Random: یک عدد رندوم ۳۲ بایتی که توسط سرور تولید شده است و در تولید کلیدهای نهایی برای رمزنگاری نقش دارد.
- Session ID: شناسه یکتایی که توسط کلاینت تولید شده و توسط سرور تائید شده است.
- Cipher Suite: الگوریتم رمزنگاری که قرار است از آن در این اتصال از آن استفاده شود.
- Compression Method: الگوریتم فشرده‌سازی که قرار است از آن استفاده شود (در این مثال از فشرده‌سازی استفاده نشده است).
- supported_versions: نسخه پروتکل TLS که از آن استفاده می‌شود. نسخه واقعی پروتکل در این فیلد مشخص شده است.
- key_share: اطلاعات مربوط به توافق کلید سرور در روش‌های مبتنی بر دیفی-هلمن.

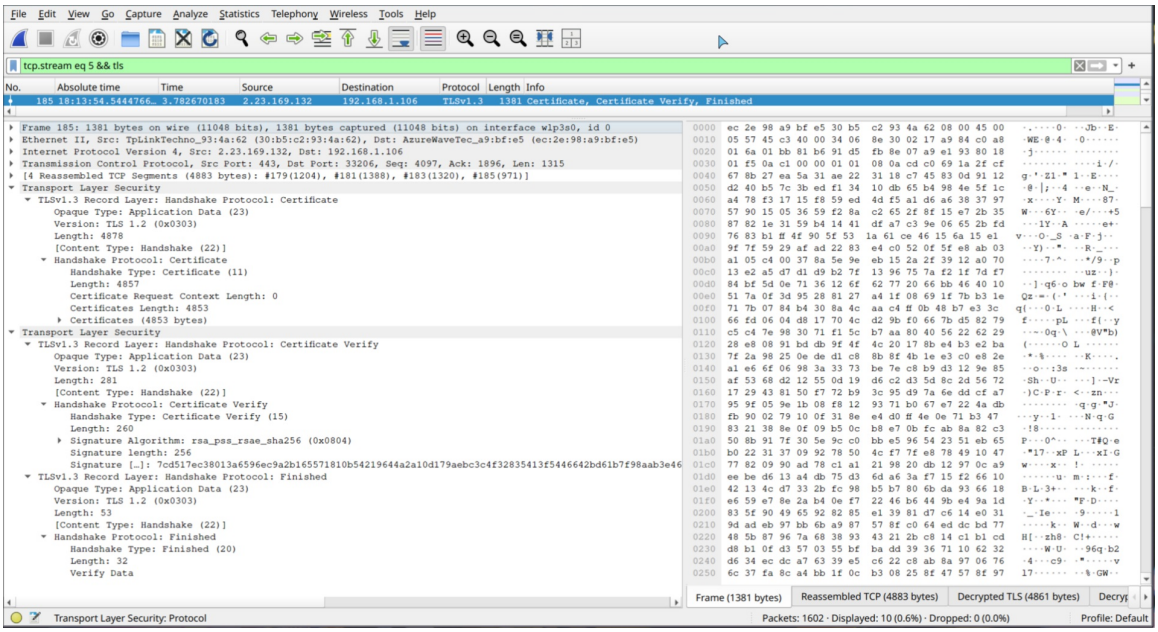
رکورد بعدی Change Cipher Spec می‌باشد که شامل فیلدی نبوده و صرفا به کلاینت اطلاع می‌دهد که از اینجا به بعد اطلاعات رد و بدل شده به صورت رمزی فرستاده می‌شوند. رکورد بعدی Encrypted Extensions می‌باشد که اکستنشن‌های زیر را به صورت رمزی برای کلاینت می‌فرستد:

- server_name: دامنه سروری که کلاینت به آن وصل شده است که در این مثال خالی می‌باشد (یعنی سرور وصل شده برابر با SNI اولیه می‌باشد).
- supported_groups: منحنی بیضی و گروه‌های میدان محدودی را که سرور برای تبادل کلید از آنها استفاده می‌کند نشان می‌دهد و مربوط به رمزنگاری بیضوی دیفی-هلمن می‌باشد.
- application_layer_protocol_negotiation: پروتکل لایه کاربردی می‌باشد که سرور قبول کرده است (در این مثال این مقدار h۲ یا همان http۲ می‌باشد).

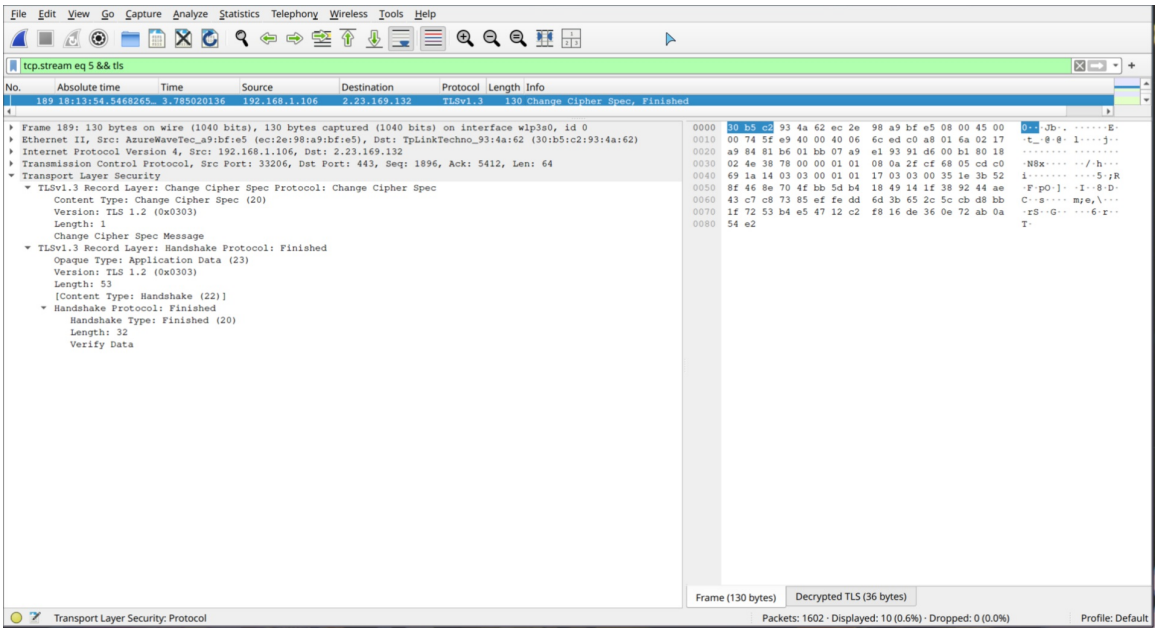


تصویر زیر محتویات پیام سوم را نشان می‌دهد که در آن سه رکورد TLS وجود دارد. رکورد اول صرفا شامل گواهی سرور برای تائید صحت کلید عمومی به اشتراک گذاشته شده در key_share پیام قبل می‌باشد. در رکورد بعدی به نام Certificate Verify سرور در اختیار داشتن کلید خصوصی متناظر با کلید عمومی درج شده در گواهی را به کلاینت اثبات می‌کند. این رکورد شامل یک امضا (که الگوریتم هش آن در این رکورد نیز ذکر شده است) از تمام محتویات دست دادن TLS تا پایان ارسال گواهی می‌باشد که در صورتی که کلاینت بتواند این امضا را تایید کند در دست داشتن کلید خصوصی گواهی برای کلاینت اثبات می‌شود.

در آخر نیز رکورد Finished می‌باشد. این رکورد دارای یک هش (Verify Data) از تمامی پیام‌های ردوبدل شده در طول دست دادن می‌باشد تا کلاینت بتواند صحت اطلاعات پیام‌های دریافتی را با استفاده از مقایسه این هش با هش محاسبه شده توسط خود تایید کند.



تصویر زیر محتویات پیام چهارم و آخر را نشان می‌دهد که در آن دو رکورد TLS وجود دارد. رکورد اول Change Cipher Spec می‌باشد که به سرور اطلاع می‌دهد از این به بعد کلاینت پیام‌های خود را به صورت رمز شده ارسال می‌کند. رکورد دوم نیز رکورد Finished می‌باشد که هدف آن در بخش قبلی توضیح داده شد.



پس از دست دادن سرور برای کلاینت دو Session Ticket می‌فرستد که برای کوتاه کردن فرآیند دست دادن در اتصال‌های بعدی می‌باشد و از دامنه این سوال خارج می‌باشد. در نهایت بعد از طی این مراحل دو طرف می‌توانند برای یکدیگر اطلاعات خود را به صورت رمزی با اطمینان از صحت آنها ارسال کنند.