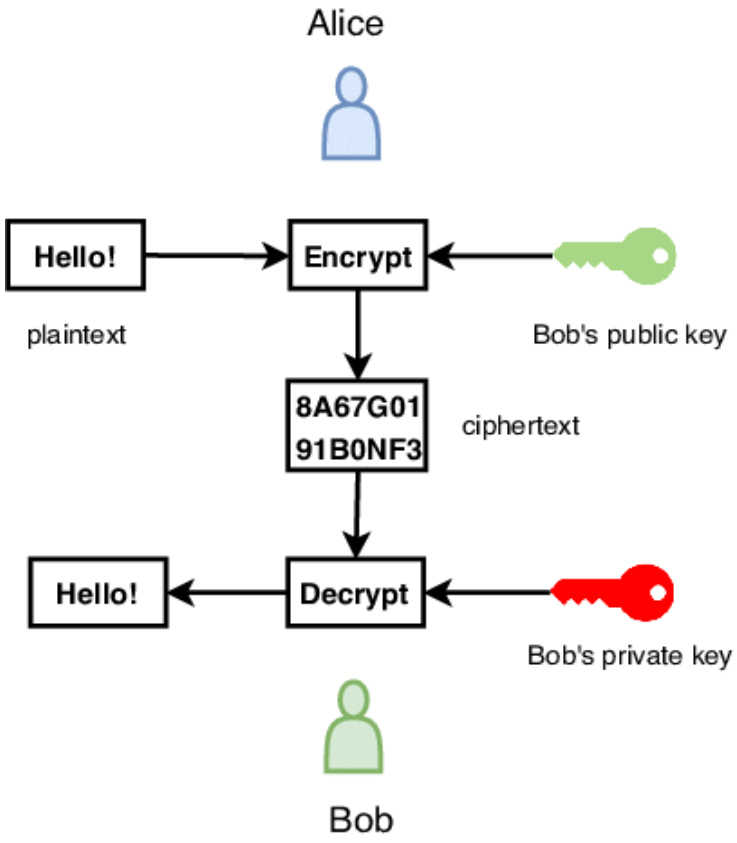
asymmetric encryption : RSA

در این روش از رمزنگاری ، هر شخص دارای یک کلید عمومی و یک کلید خصوصی که یکسان نیستند و منحصر به فرد هستند ، میباشد.

که تفاوت رمزنگاری متقارن و نامتقارن در متفاوت بودن کلید عمومی و خصوصی میباشد.

به عنوان مثال اگر کاربر اول میخواهد متنی را به صورت رمزنگاری شده برای کاربر دوم بفرستد ؛ باید ابتدا با استفاده از کلید عمومی کاربردوم آن را رمزنگاری کند و سپس کاربر دوم با استفاده از کلید خصوصی خود ، آن را رمزگشایی کند. (مانند تصویر زیر)



نحوه کارکرد رمزنگاری و رمزگشایی از طریق این الگوریتم به این صورت است :

ابتدا باید دو عدد اول بزرگ تولید کرد زیرا هرچه اعداد بزرگتر باشند امنیت رمزنگاری بیشتر است. آن هارا p و q مینامیم.

سپس حاصل ضرب آن دو عدد اول را حساب میکنیم و آن را n مینامیم که به عنوان RSA modulus هم شناخته میشود . $n = p \times q$

تابع φ را برای n حساب میکنیم.(Totient of RSA modulus) $\Phi(n)= (p-1).(q-1)$

اکنون باید یک عدد به عنوان e انتخاب کنیم که بین Φ(n) و 1 است و نسبت به Φ(n) اول میباشد.

کلید عمومی به صورت (e,n) میباشد.

و کلید خصوصی که به صورت (d,n) است d باید عددی باشد که در معادله $d.e \pmod{\Phi(n)} = 1$ صدق کند.

اکنون برای رمزنگاری که از کلید عمومی استفاده میکنیم داریم : (modular exponentiaion)

Ciphered msg = msg ^ e mod(n)

و برای رمزگشایی :

Deciphered msg = msg ^ d mod(n)

پیاده سازی و کد :

فایل BigMath.java را برای ساخت توابع مورد نیاز درست میکنیم :
تابع modpow برای modular exponentiation ، و توابع extendedGCD و modInverse برای modular inverse

برای محاسبه d از modular inverse با روش الگوریتم اقلیدسی معکوس ، که در آن باید ابتدا ب.م.م e و $\Phi(n)$ را حساب کنیم و مطمئن شویم که 1 میباشد وگرنه ماژول معکوسی وجود ندارد. در مرحله بعد عملیات هایی که برای محاسبه ب.م.م را طی کرده ایم را به صورت معکوس طی میکنیم تا d را بیابیم ، میرویم.

و برای رمزنگاری و رمزگشایی از modular exponentiation استفاده میکنیم

برای شروع نیاز به تولید دو عدد اول بزرگ داریم که برای آن از تست Miller Rabin استفاده میکنیم.

پیاده سازی آن را در فایل MillerRabin.java به کمک تابع modpow انجام دادیم.

برای نگهداری p,q,n,e,d ، فایل RSAKeyPair.java را درست میکنیم.

RSA.java را برای تولید کلید ، انتخاب e ، محاسبه d با modInverse و encrypt وdycrypt درست میکنیم.

و فایل SmallNAttack.java را برای نشان دادن ناامنی زمانی که n کوچک است. به این دلیل که :

امنیت RSA بر اساس یک فرض ساده بنا شده است :تجزیه اعداد بزرگ به عوامل اول آن‌ها فوق‌العاده دشوار است.

حمله با تقسیم آزمایشی، این فرض را به چالش می‌کشد. این حمله سعی می‌کند تا عوامل اول pو qرا با امتحان کردن تقسیم n بر اعداد کوچک، به صورت پی‌درپی، پیدا کند. این روش از سادگی ریاضیات استفاده می‌کند و هیچ تکنیک پیچیده‌ای ندارد.این نوع حمله، به عنوان حمله با تقسیم آزمایشی (Trial Division Attack)شناخته می‌شود. این حمله از یک ضعف اساسی در رمزنگاری RSA سوء استفاده می‌کند: اگر کلیدها به اندازه کافی بزرگ نباشند، می‌توان آن‌ها را به سادگی شکست.

و در نهایت App.java که یک main ساده است که کلید تولید می‌کند، پیام عددی را رمزگذاری میکند و رمز و باز می‌کند و نهایتا دموی حمله روی n کوچک.

تحلیل عملکرد تولید کلید

زمان تولید جفت کلید عمومی و خصوصی مستقیماً به دو عامل اصلی وابسته است:

طول بیت اعداد اول :(primeBits) هرچه primeBits بزرگ‌تر باشد، اندازه اعداد p و q افزایش می‌یابد و به تبع آن، جستجو برای یافتن آن‌ها به زمان بیشتری نیاز دارد.

تعداد تکرارهای آزمون میلر-رابین :(mrRounds) این پارامتر دقت آزمون اول بودن را تعیین می‌کند. افزایش تعداد تکرارها، اطمینان از اول بودن اعداد را بالا می‌برد، اما به زمان تولید کلید نیز می‌افزاید.

در عمل، پیاده‌سازی ما نشان داد که با افزایش primeBits، زمان تولید کلید به صورت قابل توجهی افزایش می‌یابد. این نتیجه، هزینه محاسباتی بالای مورد نیاز برای تأمین امنیت در رمزنگاری نامتقارن را تأیید می‌کند.

تحلیل عملکرد رمزنگاری و رمزگشایی

عملکرد توابع encrypt و decrypt به سرعت الگوریتم توان مادولار (پیاده‌سازی‌شده در تابع BigMath.modPow) بستگی دارد. نتایج نشان داد که این عملیات‌ها حتی برای کلیدهای بزرگ نیز در کسری از ثانیه انجام می‌شوند. این کارایی بالا به دلیل استفاده از الگوریتم بهینه "مربع کردن و ضرب" است که تعداد عملیات ضرب را به شکل چشمگیری کاهش می‌دهد.

این یافته اهمیت ویژه‌ای دارد؛ چرا که نشان می‌دهد افزایش طول کلید برای حفظ امنیت، به بهای غیرعملی شدن فرآیندهای رمزنگاری و رمزگشایی نیست. به عبارت دیگر، می‌توان امنیت را بدون فدا کردن کارایی به دست آورد.

تحلیل عملکرد حمله و نتایج

من با پیاده‌سازی یک حمله ساده با عنوان "Trial Division Attack" در فایل SmallNAttack.java، امنیت RSA را در شرایطی که nکوچک باشد، مورد بررسی قرار دادم. نتایج این بخش، فرضیه اصلی امنیت RSA را تأیید کرد:

موفقیت حمله برای n کوچک :برای nهای کوچک (با طول بیت کم)، تابع factor به سرعت عوامل اول p و q را پیدا کرد. این امر نشان می‌دهد که استفاده از کلیدهای کوچک، سیستم را کاملاً در برابر حملات قابل شکست می‌کند.

شکست حمله برای n بزرگ :با افزایش طول بیت n، زمان لازم برای فاکتورگیری به صورت نمایی افزایش یافت. به عنوان مثال، در آزمایش‌های ما، تلاش برای فاکتورگیری یک n با طول ۲۰۴۸ بیت عملاً غیرممکن و خارج از توان محاسباتی رایانه‌های امروزی است.

در نتیجه، متوجه میشویم که امنیت RSA نه به پیچیدگی الگوریتم، بلکه به دشواری ریاضیاتی فاکتورگیری اعداد بسیار بزرگ وابسته است. پیاده‌سازی و نتایج ما، این اصل بنیادین را به صورت عملی تأیید می‌کند.