Yasin ÖZTÜRK 704171007

# BLG520E Cryptography

## 3rd Homework

Bu projede basit station-to-station (StoS) kullanılarak güvenli kimlik doğrulama, anahtar paylaşımı ve veri şifreleme protokolleri geliştirilmiştir. Geliştirilen protokollerde CBC modunda 128 bit AES, 2048 bit RSA, SHA256 ve Diffie-Hellman anahtar paylaşımı kripto sistemleri kullanılmıştır. AES veri şifrelemede, RSA kimlik doğrulama için imzalamada ve SHA256 veri bütünlüğünün kontrolünde kullanılmıştır. Protokolün ayrıntıları, C# dilindeki implementasyonu ve sonuçları aşağıda paylaşılmıştır.

### Basit Station-to-Station Protokolü

Basit StoS protokülünde aşağıdaki 3 adımın gerçekleştirilmesi gerekmektedir:

1. Alice → Bob: $g^x \bmod p$
2. Bob → Alice: $g^y \bmod p$, $E_K(S_B(g^y \bmod , g^x \bmod p))$
3. Alice → Bob: $E_K(S_A(g^x \bmod , g^y \bmod p))$

Basit StoS protokolüne başlamadan önce, protokolün sistem parametresi olan g ve p değerlerinin belirlenmesi gerekmektedir. Aşağıda protokolün işleyişi adım adım anlatılmıştır. Aşağıdaki adımlardan herhangi biri tam olarak tamamlanmazsa, protokol derhal durdurulur.

1. Alice rasgele bir x sayısı üretir ve $g^x \bmod p$ değerini hesaplayıp Bob'a gönderir.
2. Bob rasgele bir y sayısı üretir ve $g^y \bmod p$ değerini hesaplar.
3. Bob, Alice'den aldığı $g^x \bmod p$ değerini kullanarak paylaşımlı simetrik anahtarı K'yi üretir, $K = (g^x \bmod p)^y \bmod p$.
4. Bob iki değeri yan yana yazarak birleştirir, ($g^y \bmod p$, $g^x \bmod p$). Daha sonra birleştirilmiş bu değeri kendi private anahtarını kullanarak imzalar. İmzalanmış değeri ortak K simetrik anahtarı ile şifreler. Üretmiş olduğu bu şifrelenmiş değeri, önceden hesaplamış olduğu $g^y \bmod p$ değeri ile Alice'e gönderir.
5. Alice, Bob'dan aldığı $g^y \bmod p$ değerini kullanarak paylaşımlı simetrik anahtarı K'yi üretir, $K = (g^y \bmod p)^x \bmod p$.
6. Alice üretmiş olduğu K anahtarı ile Bob'un gönderdiği şifrelenmiş değeri çözer. Alice, Bob'un public anahtarını kullanarak, simetrik anahtar K ile şifresini çözerek elde ettiği değerin Bob tarafından imzalanıp imzalanmadığını kontrol eder.
7. Alice iki değeri yan yana yazarak birleştirir, ($g^x \bmod p$, $g^y \bmod p$). Daha sonra birleştirilmiş bu değeri kendi private anahtarını kullanarak imzalar. İmzalanmış değeri ortak K simetrik anahtarı işe şifreler. Üretmiş olduğu bu şifrelenmiş değeri Bob'a gönderir.
8. Bob ortak K simetrik anahtarı ile Alice'in gönderdiği şifrelenmiş değeri çözer. Bob, Alice'in public anahtarını kullanarak, simetrik anahtar K ile şifresini çözerek elde ettiği değerin Alice tarafından imzalanıp imzalanmadığını kontrol eder.

IoT dünyasında alt yapıdan kimlik doğrulamaya, iletişimden veri güvenliğine birçok protokol kullanılmaktadır. Bizim ilgilendiğimiz protokoller kimlik doğrulama, anahtar paylaşımı ve veri şifrelemeyi içinde barındıran protokollerdir. Bu gereklilikleri sağlayan, örnek verilebilecek en güzel protokollerden biri **TLS (Transport Layer Security)** protokolüdür.

### TLS Protokolü

TLS protokolü günümüzde sıklıkla kullanılan bir protokoldür ve adından da anlaşılacağı gibi transport katmanında kullanılır. Bu nedenle uygulamaların ayrıyeten şifrelemeyi kendilerinin geliştirmelerine gerek yoktur. TLS protokolü tam station-to-station protokolünü kullanır. İletişime geçecek tarafların birbirlerinin public anahtarlarını bilmediğini varsayar. Bu nedenle protokol içerisinde sertifika oluşturulması ve paylaşılması da eklenmiştir. Bizim yapmak istediğimiz StoS protokolünün basit varyantı olduğu için, tarafların birbirlerinin public anahtarlarını doğru bildiklerini varsayacağız.

TLS protokolünde birçok farklı versiyon bulunmaktadır. TLS "Handshake" adı verilen protokolü kullanarak iletişimde kullanılacak olan protokollerin belirlenmesini sağlar. TLS'de sertifika server'ın kimliğinin doğrulanmasında ve asimetrik şifreleme ise ortak simetrik anahtarın paylaşılmasında kullanılır.

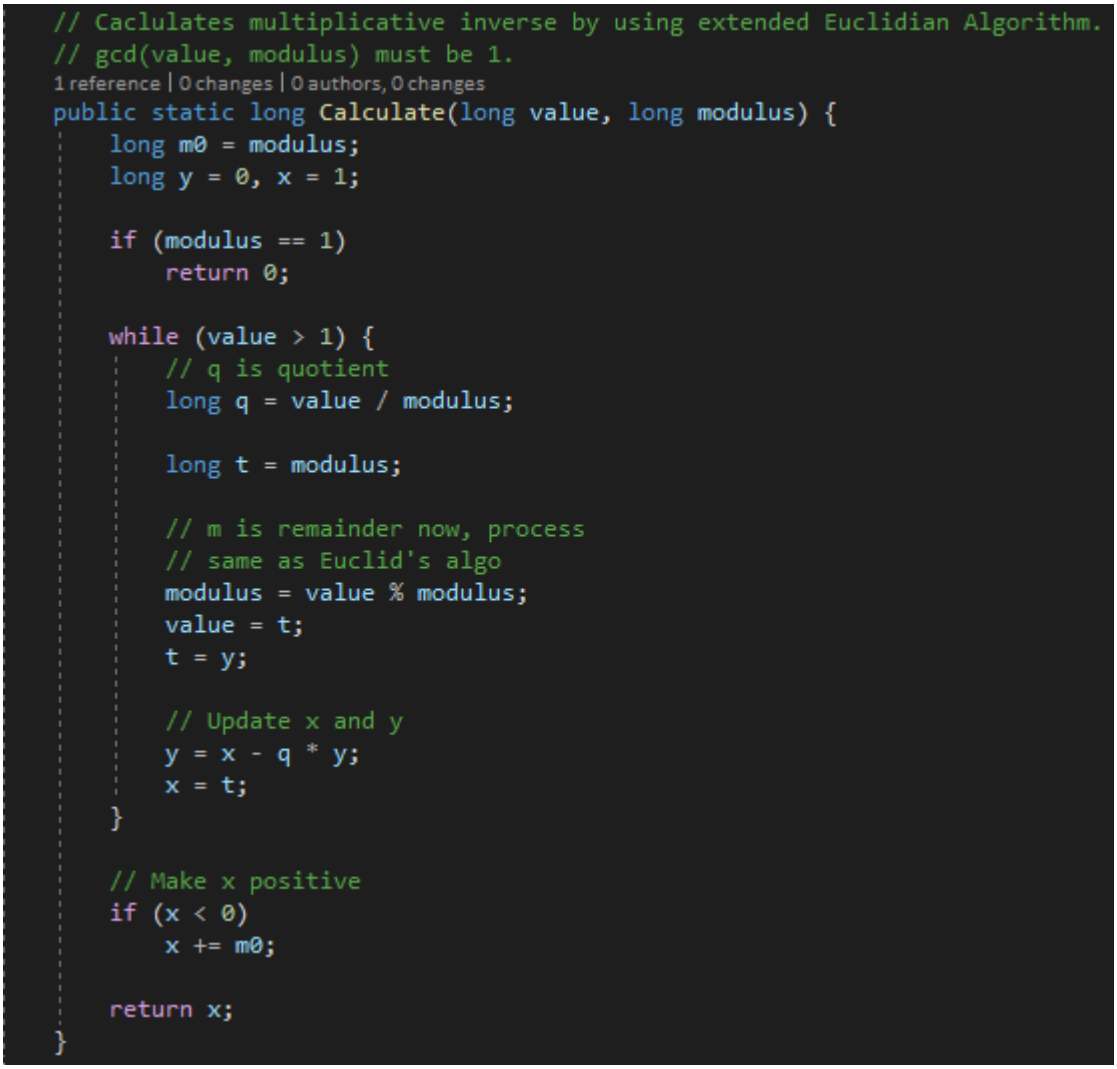
### Implementasyon

Bu projede basit StoS protokolünün çalışması için tüm aşamalar harfiyen yerine getirilmiştir. Proje C# diliyle yazılmıştır. Projede yapılan ilk iş principal'ların protokol gerekliliklerini yerine getirebilmesi için RSA, AES, SHA256 ve Diffie-Hellman kripto sistemlerinin gerçeklenmesi olmuştur. Bu kripto sistemlerde inverse multiplication ve modulo exponentiation kullanılması gerektiği için, Euclid'in genişletilmiş algoritması ve square and multiply algoritması implement edilmiştir.

```
public static int SquareAndMultiply(int generator, int exponent, int modulus) {
    long result = 1;
    for (int i = 31; i >= 0; i--) {
        result = (result * result) % modulus;
        if (((exponent >> i) & 1) == 1) {
            result = (result * generator) % modulus;
        }
    }

    return (int)result;
}
```

*Resim 1. Square and Multiply Algoritması*

Diffie-Hellman anahtar paylaşımı asal sayılar ile çalıştığından dolayı, protokolde kullanılmak üzere asal sayı üreten bir mekanizmanın geliştirilmesine ihtiyaç duyulmuştur. Asal sayıların hesaplanarak bulunması yerine, program başlatıldığında 200.000'e kadar olan asal sayılar dosyadan okunarak bir listeye aktarılmıştır. Bu asal sayıların hafıza da tutulması nedeniyle programın kullandığı bellek artmıştır. Fakat öte yandan asal sayı bulmak için hesaplama yapılmadığından CPU kullanımı daha iyidir ve daha hızlı çalışmaktadır. Sistemin kısıtlamalarına göre asal sayıların listeden okunması veya istendiğinde hesaplanması yöntemleri seçilebilir.



```
// Caclulates multiplicative inverse by using extended Euclidian Algorithm.
// gcd(value, modulus) must be 1.
1 reference | 0 changes | 0 authors, 0 changes
public static long Calculate(long value, long modulus) {
    long m0 = modulus;
    long y = 0, x = 1;

    if (modulus == 1)
        return 0;

    while (value > 1) {
        // q is quotient
        long q = value / modulus;

        long t = modulus;

        // m is remainder now, process
        // same as Euclid's algo
        modulus = value % modulus;
        value = t;
        t = y;

        // Update x and y
        y = x - q * y;
        x = t;
    }

    // Make x positive
    if (x < 0)
        x += m0;

    return x;
}
```

*Resim 2. İki sayının çarpmaya göre tersini bulmak için Euclid'in Genişletilmiş Algoritması*

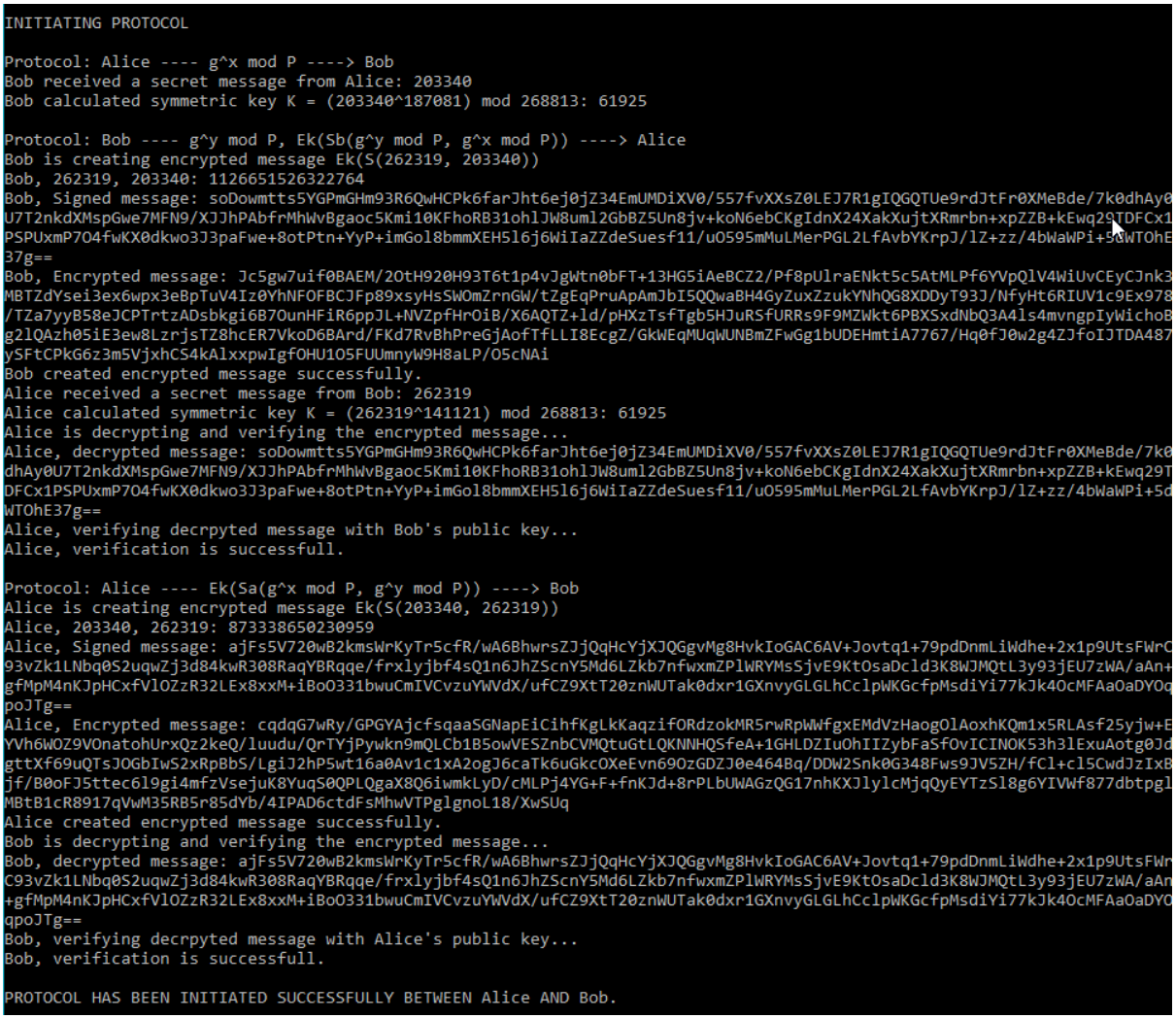
Protokollerde kullanılan RSA ve SHA, Microsoft'un C# ile geliştirme ortamına sunduğu hazır kütüphaneler ile kullanılmıştır. RSA protokolü geliştirilmeye çalışılmış olsa da kullanılan bit sayısı fazla olduğundan hazır çözüm yollarına başvurulmuştur. Öte yandan 128 bitlik AES bazı hazır kütüphaneler kullanılsa bile ayrıyeten geliştirilmiştir.

Gerçeklenen protokolün çalıştırılması için yazılan test metodu Resim 3'te gösterilmiştir. Yazılan koda detaylı yorumlar eklenmiştir ve takip etmesi yeterince kolaydır.

```
Stopwatch watch = new Stopwatch();
watch.Start();

// SYSTEM PARAMETERS FOR DIFFIE HELLMAN KEY EXCHANGE
const int generator = 32941;     // denoted as g or alpha
const int modulus = 268813;      // denoted as p

// Create two parties, both parties create their own RSA public and private keys.
Principal alice = new Principal("Alice", generator, modulus);
Principal bob = new Principal("Bob", generator, modulus);

Console.WriteLine("INITIATING PROTOCOL");
Console.WriteLine("\r\nProtocol: Alice ---- g^x mod P ----> Bob");
#region Alice ---- g^x mod P ----> Bob
// Get calculated g^x mod P secret message.
int secretMessageOfAlice = alice.SecretMessage;
// Send g^x mod P secret message to Bob,
// Bob stores Alice's secret message, calculates its own g^y mod P secret message.
// Bob then calculates and stores K = (g^x)^y mod P symmetric key for communication with Alice
bob.ReceiveSecretMessage(alice.Name, secretMessageOfAlice);
#endregion

Console.WriteLine("\r\nProtocol: Bob ---- g^y mod P, Ek(Sb(g^y mod P, g^x mod P)) ----> Alice");
#region Bob ---- g^y mod P, Ek(Sb(g^y mod P, g^x mod P)) ----> Alice
// Bob concatenates g^y mod P and g^x mod P.
// Bob signs this concatenated secret messages with its private RSA key.
// Bob encrypts this signed message with symmetric key K
Tuple<int, string> encryptedMessageOfBob = bob.CreateSecretMessage(alice.Name);
// Alice calculates and stores K = (g^y)^x mod P symmetric key for communication with Bob.
// Alice decrypts the encrypted message with this symmetric key K.
// Alice verifies signed message with Bob's public key.
alice.ReceiveEncryptedMessage(bob.Name, encryptedMessageOfBob.Item1, encryptedMessageOfBob.Item2);
#endregion

Console.WriteLine("\r\nProtocol: Alice ---- Ek(Sa(g^x mod P, g^y mod P)) ----> Bob");
#region Alice ---- Ek(Sa(g^x mod P, g^y mod P)) ----> Bob
// Alice concatenates g^x mod P and g^y mod P.
// Alice signs this concatenated secret messages with its private RSA key.
// Alice encryptes this signed message with symmetric key K
string encryptedMessageOfAlice = alice.CreateEncryptedMessage(bob.Name);
// Bob decrypts the encrypted message with symmetric key K
// Bob verifies signed message with Alice's public key.
bob.ReceiveEncryptedMessage(alice.Name, encryptedMessageOfAlice);
#endregion
Console.WriteLine($"\r\nPROTOCOL HAS BEEN INITIATED SUCCESSFULLY BETWEEN {alice.Name} AND {bob.Name}.\r\n");

string messageFromAliceToBob = "Hi Bob, It's me Alice!";
Tuple<string, string> encryptedMessageFromAlice = alice.CreateEncryptedMessage(messageFromAliceToBob, bob.Name);
bob.GetEncryptedMessage(encryptedMessageFromAlice, alice.Name);

string messageFromBobToAlice = "Hi Alice, it is really you!";
Tuple<string, string> encryptedMessageFromBob = bob.CreateEncryptedMessage(messageFromBobToAlice, alice.Name);
alice.GetEncryptedMessage(encryptedMessageFromBob, bob.Name);

watch.Stop();
Console.WriteLine("Elapsed ms: " + watch.ElapsedMilliseconds);
```

*Resim 3. Gerçeklenen protokolün denendiği Main metodu*

Yukarıdaki metot çalıştırıldığında programın tam çıktısı Resim 4'te gösterilmiştir. Tüm protokolün işletilmesi ve Alice ile Bob'un birbirlerine mesaj göndermesi yaklaşık olarak 1 saniye sürmektedir. Protokolde en çok zaman alan eylem, RSA anahtarlarının oluşturulması ve RSA ile verinin imzalanmasıdır.




*Resim 4. Protokolde gerçeklenen tüm işlemlerin çıktıları*

Resim 4'te görüleceği gibi, şifrelenmiş metinler, imzalar ve hash değerleri program çıktısının takip edilmesini zorlaştırmaktadır. Bu nedenle protokolün her aşaması aşağıda ayrıca anlatılmıştır. Basit StoS protokolünde olduğu gibi iki tarafın haberleşmesi 3 parçada gösterilmiştir.

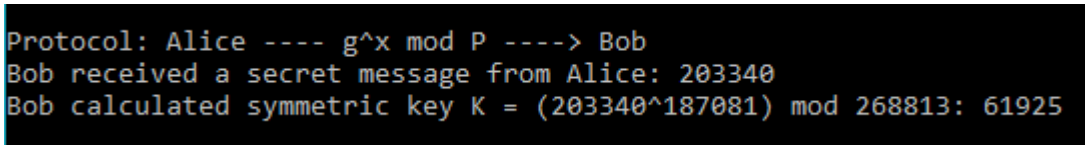
Protokole başlamadan önce yapılacak ilk iş sistem parametrelerinin belirlenmesi ve protokolü kullanacak olan tarafların oluşturulmasıdır. Sistem parametresi olarak generator g değeri 32941 ve modulus p değeri 268813 olarak seçilmiştir. Resim 5'te bu sistem parametreleri kullanılarak principal oluşturma esnasında hesaplanan $g^x$ mod p ve $g^y$ mod p değerleri görülmektedir. Bu değerler kod içerisinde "secret message" olarak adlandırılmıştır.


*Resim 5. Alice ve Bob principallarının oluşturulması, rasgele gizli değerlerinin seçilmesi, $g^{secret}$ mod p değerlerinin hesaplanması ve RSA asimetrik anahtarlarının oluşturulması*

Protokolde kullanılmak üzere Alice ve Bob principalları oluşturulduktan sonra, protokol başlatılmıştır. Protokolün ilk aşamasında;

**Alice → Bob: $g^x$ mod p**

işlevi gerçekleştirilmiştir. Bu kısımda Alice'in Resim 5'te de görüleceği gibi ürettiği secret message değeri olan 203340 Bob'a gönderilmektedir. Bob bu değeri aldıktan sonra veri şifrelemede ortak olarak kullanacakları K simetrik anahtarını üretecektir. Simetrik anahtarı üretmek için $(203340^{187081})$ mod 268813 değerini hesaplayacaktır. Square and Multiply algoritması sayesinde bu değer bir kaç milisaniye içerisinde hesaplanmaktadır. Bu kısma kadar gerçekleşen işlemlerin çıktısı Resim 6'da gösterilmiştir.




*Resim 6. Protokolün 1. kısmı*

Protokolün ikinci aşamasında;

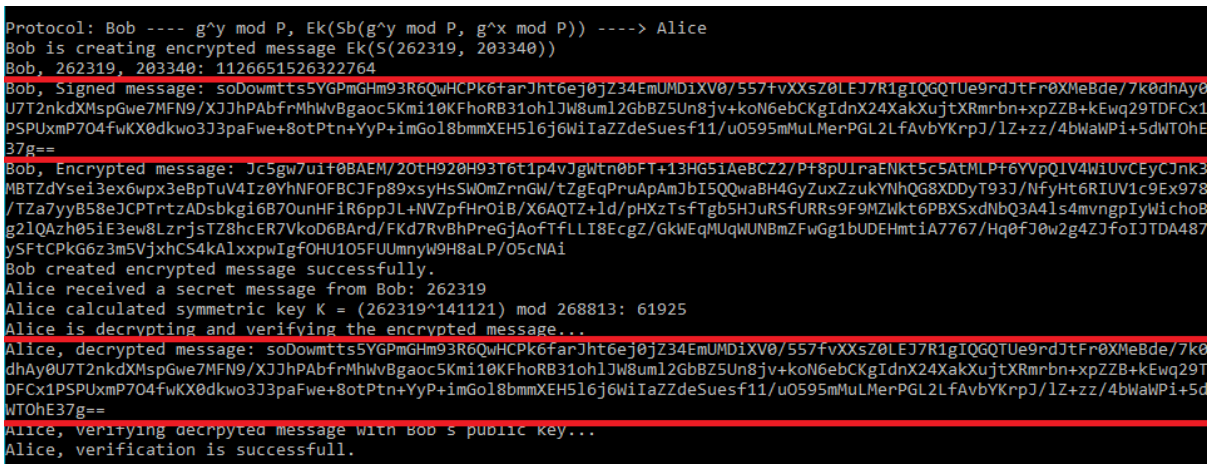
**Bob → Alice: $g^y$ mod p, $E_K(S_B(g^y$ mod , $g^x$ mod p))**

işlevi gerçekleştirilmiştir. Bu kısımda Bob, Alice'ten bir önceki aşamada aldığı 203340 secret message değerini kendi secret message değeri olan 187081 değeri ile birleştirecektir. Burada kullanılan secret message değerleri 32-bitlik integer değerleri olduğu için, bu iki secret message'ın birleştirilmiş uzunluğu 64 bit olacaktır. Birleştirilmiş secret message'ların değeri ((262319 << 32) | 203340) = 1126651526322764 olarak hesaplanmıştır. Bob hesaplanan bu birleştirilmiş değeri kendi private RSA anahtarını kullanarak imzalamıştır ve Resim 7'de görülen "Signed message" değerini elde etmiştir. Daha sonra bu değeri K simetrik anahtarı olan 61925 değerini kullanarak CBC modda çalışan 128 bitlik AES ile şifrelemiştir. Şifrelenmiş değerin çıktısı Resim 7'de gösterildiği gibidir. Oluşturulan bu şifrelenmiş değer ile Bob'un secret message değeri olan 262319, Alice'e gönderilmektedir.

Alice, Bob'dan aldığı 262319 secret message değerini kullanarak K simetrik anahtarını üretmiştir. Simetrik anahtarı üretmek için $(262319^{141121})$ mod 268813 işlemini gerçekleştirmiştir ve Bob'un bulduğu ile aynı K = 61925 değerini bulmuştur. Bu aşamaya

kadar gerçekleşen işlemler ile Diffie-Hellman anahtar paylaşımı tamamlanmıştır fakat taraflar birbirlerinin kimliklerini daha doğrulamamışlardır. Bu kısımdan sonra yürütülecek protokol işlemleri ile kişilerin kimlikleri doğrulanacaktır. Protokolun başarıyla tamamlanması halinde, Alice ve Bob kendi aralarında veri paylaşımı yaparken K ortak anahtarını kullanacaktır.

Alice bulmuş olduğu K simetrik anahtarı kullanarak Bob'dan almış olduğu şifrelenmiş metnin şifresini çözer. Şifresi çözülmüş metin Resim 7'de gösterilmiştir. Resim 7'de kırmızı ile işaretlenen alanların aynı olduğu görülecektir. Bu Alice ve Bob'un ortak olarak kullandığı K simetrik anahtarının doğru bir şekilde paylaşıldığını ve kullanıldığını göstermektedir. Alice şifresini çözdüğü mesajın Bob'tan gelip gelmediğini anlamak adına, Bob'un public anahtarını kullanarak imzasını doğrular. İmza doğrulanma işleminin başarıyla tamamlanması halinde, Alice mesajın Bob'tan geldiğine emindir ve Bob'un kimliğini doğrulamıştır. Bob'un Alice'in kimliğini doğrulaması için ise protokolün 3. ve son kısmına ihtiyaç duyulmaktadır.




*Resim 7. Protokolün 2. kısmı*

Protokolün üçüncü ve son aşamasında;

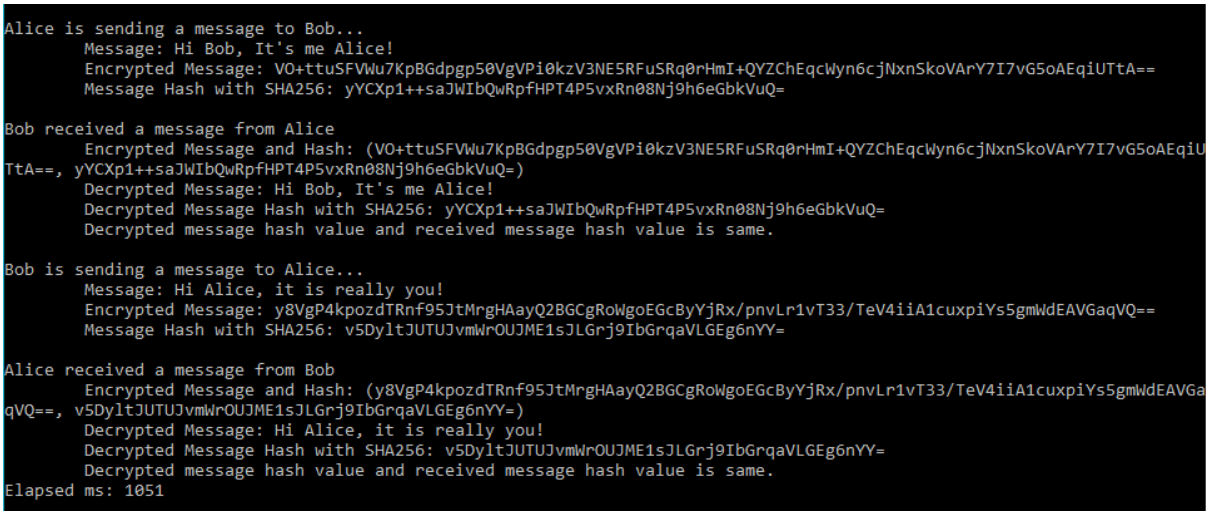
**Alice → Bob: $E_K(S_A(g^x \bmod , g^y \bmod p))$**

işlevi gerçekleştirilmiştir. Bu aşamada Bob, Alice'in kimliğini doğrulayacaktır. Öncelikle Alice, bir önceki aşamada Bob'tan aldığı secret message ile kendi secret message değerini birleştirecektir. Birleştirilmiş secret message'ların değeri ((203340 << 32) | 262319) = 873338650230959 olarak hesaplanmıştır. Alice hesaplanan bu birleştirilmiş değeri kendi private RSA anahtarını kullanarak imzalamıştır ve Resim 8'de görülen "Signed message" değerini elde etmiştir. Daha sonra bu değeri K simetrik anahtarı olan 61925 değerini kullanarak CBC modda çalışan 128 bitlik AES ile şifrelemiştir. Şifrelenmiş değerin çıktısı Resim 8'de gösterildiği gibidir. Oluşturulan bu şifrelenmiş değer Bob'a gönderilmektedir.

Bob, Alice'ten almış olduğu şifreli değeri, daha önceden üzerinde anlaştıkları K simetrik anahtarı ile çözer. Şifresi çözülen metin Resim 8'de gösterilmiştir. Resim 8'de kırmızı ile işaretlenen alanların aynı olduğu görülecektir. Bob şifresini çözdüğü mesajın Alice'den gelip gelmediğini anlamak adına, Alice'in public anahtarını kullanarak imzasını doğrular. İmza doğrulama işleminin başarıyla tamamlanması halinde, Bob da artık Alice ile konuştuğundan emindir. Böylelikle protokol başarıyla tamamlanmış olur ve Alice ile Bob kendi aralarında şifreli veri paylaşımına başlayabilirler.


*Resim 8. Protokolün 3. kısmı*

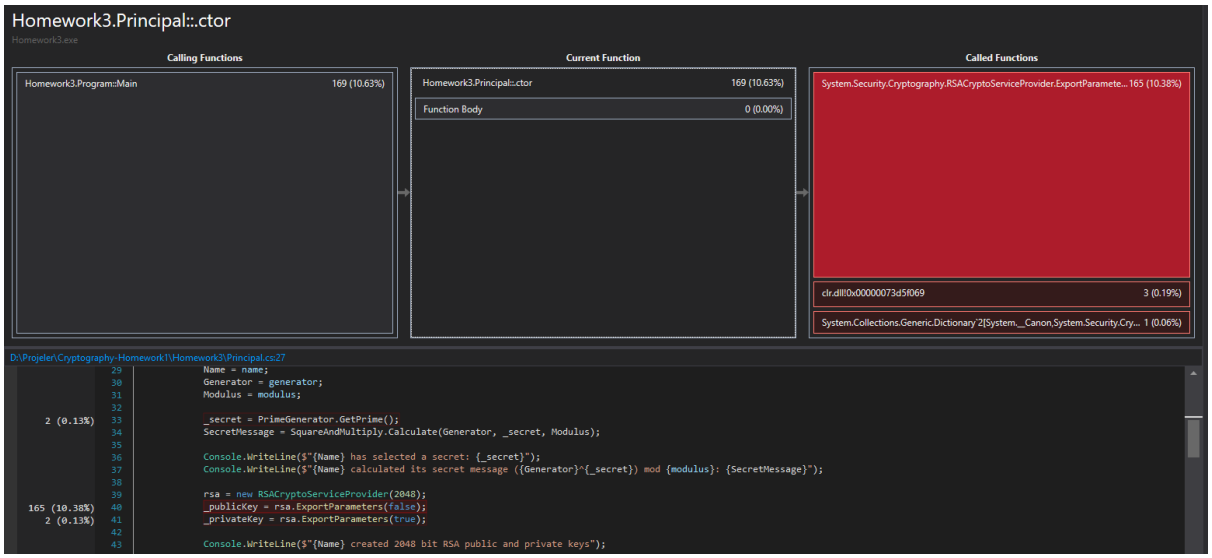
Protokol başarıyla tamamlanmasıyla, Alice ile Bob'un mesaj paylaşmasına iki örnek verilmiştir. Mesaj paylaşma işleminde taraflar arasında gönderilecek mesajlar K ortak anahtarı ile şifrelenir ve mesajın orijinalinin SHA256 ile hash değeri hesaplanır. Bu şifrelenmiş metin ve hash değeri karşı tarafa gönderilir. Böylelikle karşı taraf mesajı aldıktan sonra, mesajın aradaki bir kişi tarafından değiştirilip değiştirilmediğini kontrol edebilir. Bunun için yapması gereken, aldığı şifreli mesajı ortak anahtar K ile çözmeli ve hash değerini hesaplamalıdır. Eğer iki hash değeri birbiriyle aynı ise, gönderilen mesaj değiştirilmemiştir ve doğrudur. Bu durum Resim 9'de gösterilmiştir.




*Resim 9. Alice ile Bob arasında şifreli mesajlaşma örneği*

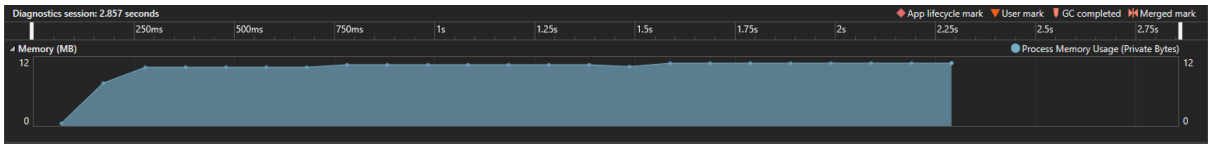
### Performans

Geliştirilen protokol Windows 10 işletim sisteminde, i5-7300HQ işlemcili ve 16GB RAM'e sahip bilgisayarda çalıştırılmıştır. Resim 9'da görüleceği gibi principal'ların oluşturulması, protokolün gerçekleştirilmesi ve örnek mesajlaşmanın yapılması 1051 milisaniye sürmüştür. Bu süre zarfında gerçekleştirilen en maliyetli işlem Resim 10'da gösterildiği gibi RSA anahtarlarının üretilme işlemidir.




*Resim 10. Geliştirilen kodun incelenmesi sonucu RSA anahtarlarının oluşturulmasının sistem performansına genel etkisi*

Geliştirilen program, disk üzerinde 176 kb'lik yer kaplamaktadır. Program çalıştırıldığında, asal sayıların hafıza da tutulması nedeniyle, Resim 11'de görüleceği üzere 12 MB'tan az yer tutmaktadır. Kullanılan maksimum asal sayının arttırılması veya 2den fazla principal üretilmesi halinde bu değer artacaktır. Fakat protokol tamamlandıktan sonra, Alice ve Bob'un tutması gereken değer sadece K ortak anahtarıdır. Dolayısıyla, protokol sonrası mesajlaşmada gereken bellek miktarı çok düşüktür.




*Resim 11. Geliştirilen programın kullandığı bellek miktarı*

Projede geliştirilen kodların tamamının çok uzun olmasından ve her birinin resminin bu rapora koyulmasının, raporun okunabilirliğini düşüreceğinden, yazılan kodların tamamı aşağıdaki github adresine yüklenmiştir.

Github: https://github.com/qua11q7/Cryptography/tree/master/Homework3