CLİENT – SERVER UYGULAMASI

Numara: 436590

Ad-Soyad: Bünyamin Öter

GitHub Repo: https://github.com/bunyaminoter/python_client_server

1. Projenin Amacı ve Kapsamı

Bu projenin temel amacı, Kriptoloji dersi kapsamında öğrenilen şifreleme algoritmalarının, gerçek bir ağ uygulaması üzerinde pratik uygulamasını göstermektir. Proje, Python programlama dili kullanılarak geliştirilen bir İstemci-Sunucu (Client-Server) mimarisine dayanmaktadır. Uygulama, iki uç nokta arasındaki metin tabanlı iletişimi, seçilen kriptografik yöntemle şifreleyerek ağ üzerindeki dinlemelere (packet sniffing) karşı verinin gizliliğini sağlamayı hedefler.

2. Kullanılan Teknolojiler ve Kütüphaneler

- **Python 3.7+ gereklidir.**
- **Tüm şifrelemeler standart kütüphaneleri kullanır. Herhangi bir kütüphane eklemenize gerek yoktur.**
- **pycryptodome kütüphanesini import ederek karşılaştırma yapabilirsiniz.**

2.1 Sistemin Genel Çalışma Akışı

Geliştirilen uygulama, TCP/IP protokolü üzerinde çalışan, çoklu iş parçacığı (multithreading) destekli ve grafik arayüze (GUI) sahip bir İstemci-Sunucu haberleşme sistemidir. Sistemin çalışma akışı şu adımlardan oluşur:

- **Başlatma ve Anahtar Üretimi:**
  - Sunucu (server.py) başlatıldığında, RSACipher sınıfı aracılığıyla 1024-bitlik bir RSA anahtar çifti (Public ve Private Key) üretir. Bu işlem, güvenli anahtar dağıtımı için gereklidir.
  - İstemci (client.py) başlatıldığında, arayüz yüklenir ve connect_to_server fonksiyonu tetiklenir.
- **El Sıkışma (Handshake):**
  - İstemci sunucuya bağlandığında ilk olarak "PUB_KEY_REQ" mesajını göndererek sunucunun açık anahtarını (Public Key) talep eder.
  - Sunucu bu isteği alır ve "PUB_KEY_RES" başlığı altında kendi Public Key'ini (modulus n ve üs e) JSON formatında istemciye iletir.
  - İstemci bu anahtarı alır ve oturum boyunca yapacağı şifreli gönderimler için hafızasında saklar.

- **Hibrit Şifreleme ve Veri İletimi:**
    - Kullanıcı bir mesaj yazıp AES veya DES yöntemini seçtiğinde "Hibrit Şifreleme" devreye girer.
    - Simetrik Anahtar Üretimi: İstemci, her mesaj için rastgele (random) bir oturum anahtarı (session key) üretir (AES için 16 byte, DES için 8 byte).
    - Veri Şifreleme: Mesaj metni, seçilen moda (Manuel veya Kütüphane) ve üretilen oturum anahtarına göre şifrelenir.
    - Anahtar Şifreleme (Kapsülleme): Üretilen oturum anahtarı, sunucunun RSA Public Key'i kullanılarak asimetrik olarak şifrelenir.
    - Paketleme: Şifreli mesaj, RSA ile şifrelenmiş anahtar, kullanılan algoritma, parametreler (IV vb.) ve implementasyon modu (impl_mode) bir JSON nesnesi haline getirilir ve sunucuya gönderilir.
- **Veri Alımı ve Deşifreleme:**
    - Sunucu gelen JSON paketini ayrıştırır. Eğer pakette encrypted_aes_key varsa, önce bunu kendi RSA Private Key'i ile çözerek oturum anahtarını elde eder.
    - Elde edilen oturum anahtarı arayüzdeki ilgili alana otomatik olarak işlenir.
    - Ardından, pakette belirtilen mod (Manuel/Kütüphane) dikkate alınarak asıl mesaj bu anahtarla çözülür ve ekrana "Çözüldü" etiketiyle basılır.

3. Yazılım Mimarisi ve Kod Yapısı

3.1. Şifreleme Yönetimi (Encryption)

Uygulamanın kalbini chipers.py dosyası oluşturmaktadır. Bu sınıf, farklı şifreleme algoritmalarını (AES, DES, Vigenere, Hill, vb.) tek bir çatı altında toplar ve istemci/sunucu kodunun karmaşıklığını azaltır.

Kod yapısı incelendiğinde **Strategy Design Pattern** (Strateji Tasarım Deseni) benzeri bir yapı kurulduğu görülmektedir. encrypt ve decrypt metotları, method parametresine göre ilgili algoritmayı dinamik olarak seçer.

Desteklenen Algoritmalar:

- **Modern Şifreleme:** AES (Advanced Encryption Standard), DES.
- **Klasik Şifreleme:** Caesar, Vigenere, Affine, Rail Fence, Route, Columnar Transposition, Polybius, Pigpen, Hill, Substitution.

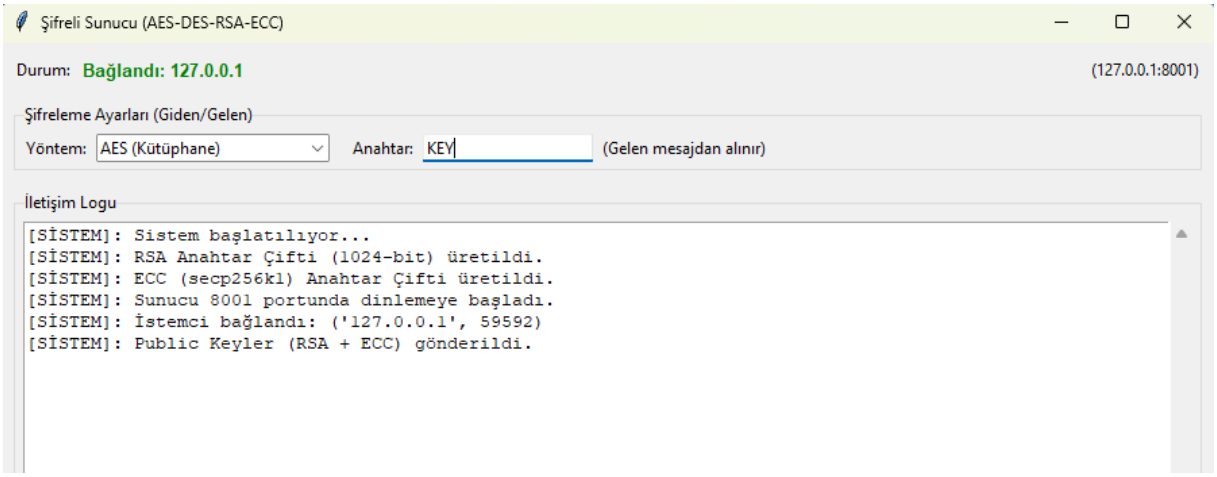
3.2. Haberleşme Protokolü (JSON Yapısı)

Wireshark analizlerinden görüldüğü üzere (Diğer sayfalarda mevcuttur), sistem ham metin göndermek yerine yapılandırılmış bir JSON formatı kullanmaktadır. Bu format şunları içerir:

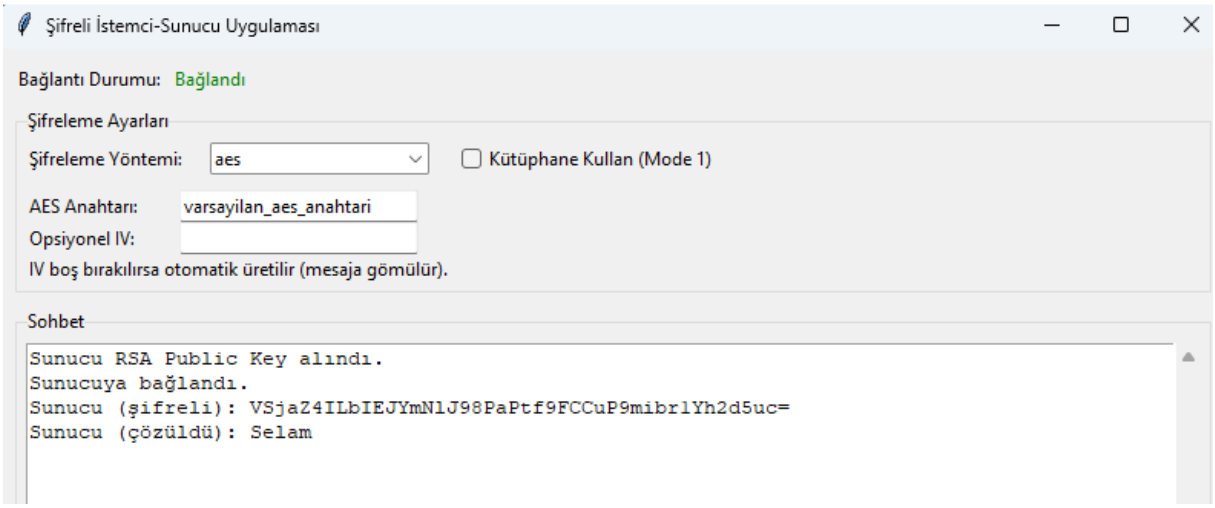
- message: Şifrelenmiş metin (Ciphertext).
- method: Kullanılan şifreleme algoritması (örn: "aes", "vigenere").
- params: Şifre çözme için gerekli parametreler (örn: "key", "iv").
- impl_mode: "manual or library"

Projenin GUI örnekleri:

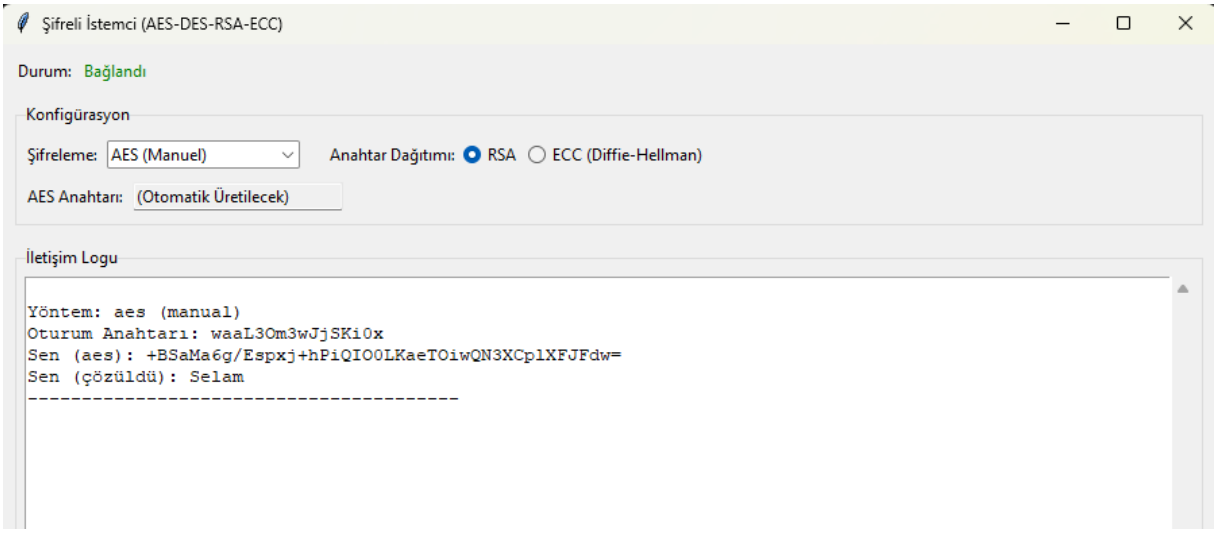
- Sunucu yazılan mesajı seçilen şifreleme metoduna göre şifreleyip istemciye gönderir.



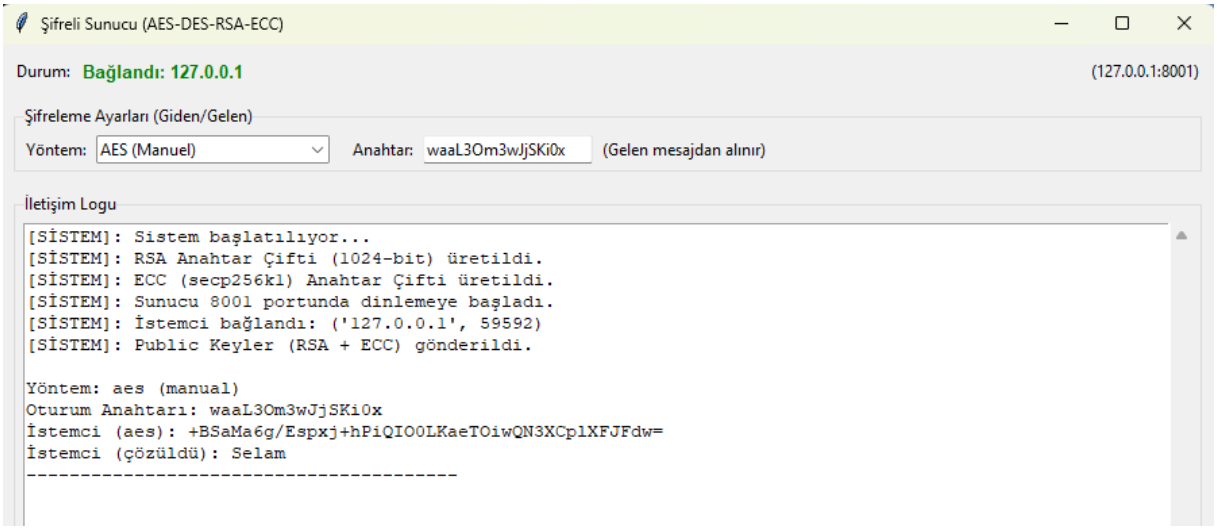
- Şifrelenmiş mesajlar rapor için çıktıları görebilmek amacıyla kullanıcıya verilmiştir.
- AES şifreleme mantığı kod yapısında düzenli bir şekilde yazılmıştır. (S-BOX dahil)



- Mesajı alan istemci gelen ciphertext'i çözer ve kullanıcıya sunar.



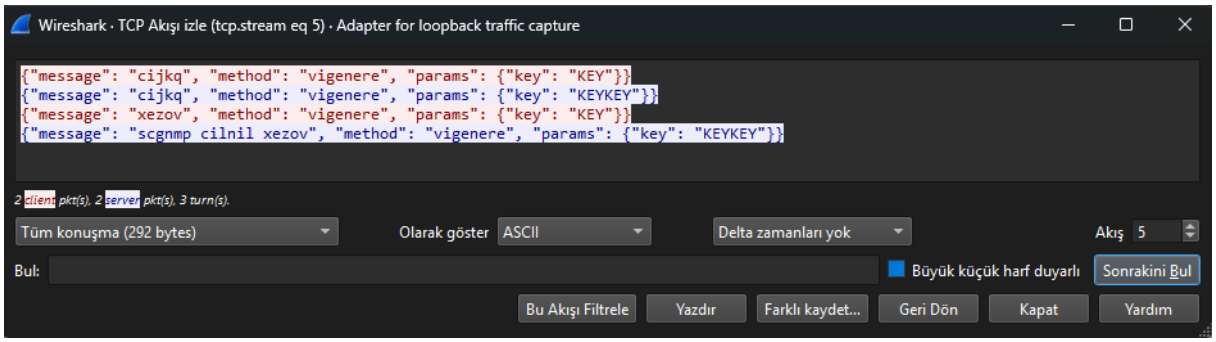
- Kütüphane ile şifrelenen her mesaj için sunucuya benzersiz bir oturum anahtarı gider.



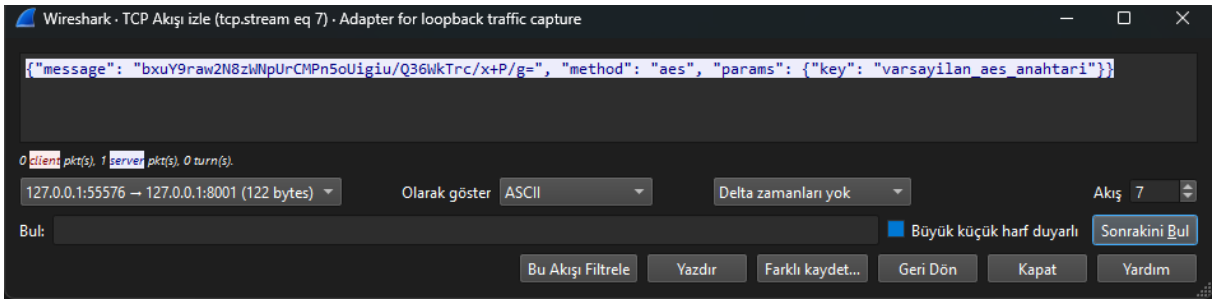
- Sunucu bu anahtar ile gelen mesajı çözer ve kullanıcıya sunar.

4. Test Sonuçları ve Ağ Analizi

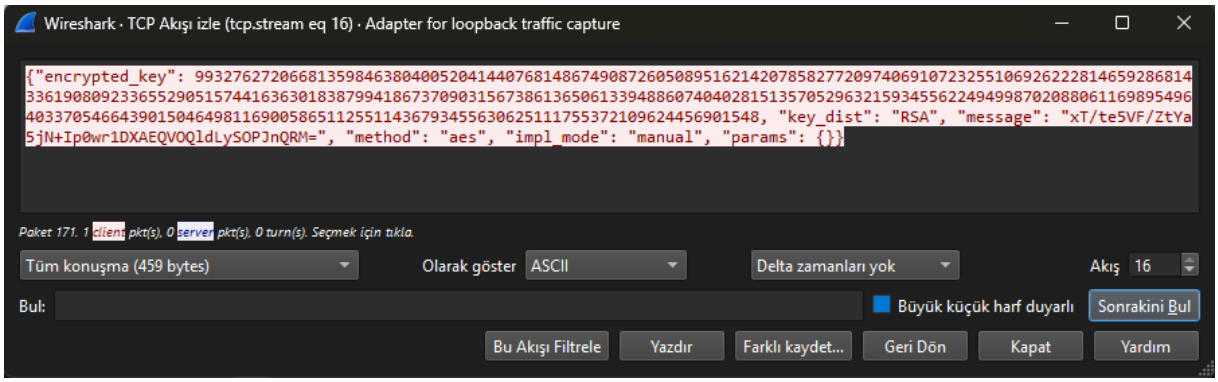
Bu bölümde, uygulamanın çalışır haldeki görüntüleri ve Wireshark ile yapılan ağ trafiği analizleri incelenmiştir.



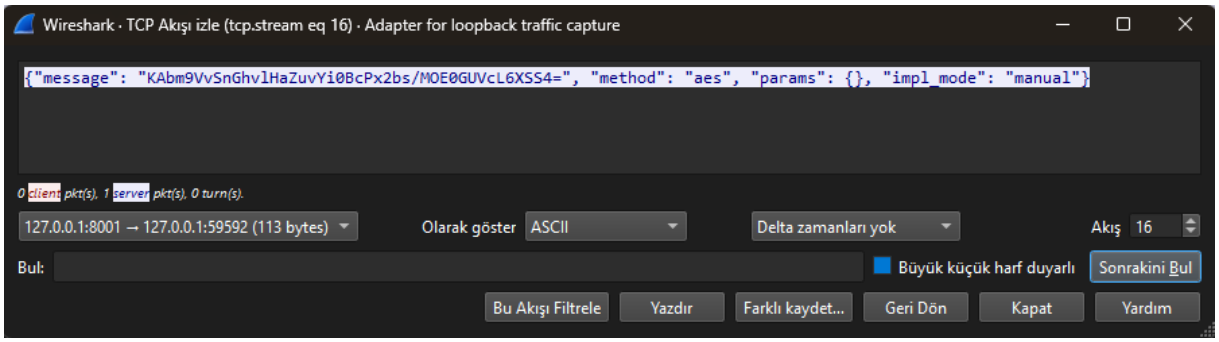
- Bu görüntüde sıradan bir mesajlaşma izlenmiştir.
  Selam – Selam – Naber – İyidir senden naber
- Mesajlar sadece veri gittiğini göstermek için açık gösterilmiştir. Güvenlik için aes ve des bu şekilde tasarlanmamıştır.



- Burada şifrelenmiş dataların { “message”:…} "method": "aes", "params": {"key":…} impl_mode: "manual or library" şeklinde gittiği gösterilmiştir.



- Kütüphane ile şifrelenen her mesaj için sunucuya benzersiz bir oturum anahtarı gider.



- Sunucu ve Server arasındaki iletişim gösterilmiştir.

4.2. Wireshark ile Ağ Trafiği Analizi

Uygulamanın güvenliğini doğrulamak için "Loopback" arayüzü dinlenmiş ve TCP paketleri yakalanmıştır.

Analiz 1: AES ve RSA ile Hibrit Şifreleme Trafiği

Figür 3 (Wireshark Çıktısı): Ağ üzerinde yakalanan paket içeriği, geliştirilen hibrit protokol gereği JSON formatındadır ve şu yapıdadır:

JSON

{"message": "U2FsdGVkX1+...","method": "aes", "encrypted_aes_key": 48192301... (çok büyük tamsayı), "params": {"iv": "..."}, "impl_mode": "manual"}

Bulgular:

- **Veri Gizliliği (Payload):** "Merhaba" gibi açık metinler (plaintext), AES-CBC modu ile şifrelendiği için anlamsız karakter yığınları (ciphertext) olarak görünmektedir.
- **Anahtar Güvenliği (Key Transport):** Önceki versiyonların aksine, simetrik şifreleme anahtarı (key) açık metin olarak gönderilmemektedir. Bunun yerine, istemci tarafından üretilen rastgele oturum anahtarı, sunucunun RSA Public Key'i ile şifrelenerek encrypted_aes_key alanında taşınmaktadır. Bu, "Man-in-the-Middle" saldırılarına karşı tam koruma sağlar.
- **Protokol Bütünlüğü:** JSON yapısı; şifreli mesajı, şifreli anahtarı ve implementasyon modunu (manual veya library) düzenli bir şekilde taşıyarak sunucunun doğru deşifreleme yapmasını sağlar.

4.3. Asimetrik Anahtar Muhafızları (ECC & RSA)

Asimetrik algoritmalar, simetrik anahtarların ağ üzerinden güvenli bir şekilde karşı tarafa ulaştırılmasını (Digital Envelope) sağlar.

● ECC (Elliptic Curve Cryptography): Projenin en güncel bileşenidir. Secp256r1 eğrisi kullanılarak kurgulanmıştır. RSA'ya göre çok daha küçük anahtar boyutlarıyla (256-bit ECC ≈ 3072-bit RSA) aynı güvenlik seviyesini sunması, ağ trafiğinde paket yükünü minimize etmektedir.

● RSA (Rivest–Shamir–Adleman): 2048-bit anahtar uzunluğuyla kurgulanan RSA, sistemdeki alternatif asimetrik yöntemdir. Anahtar takası (Key Exchange) sırasında simetrik anahtarları kapsüllemek için kullanılır.

5. Sonuç

Bu proje ile, teorik olarak öğrenilen kriptoloji algoritmalarının çalışan bir yazılım sistemine entegrasyonu başarıyla tamamlanmıştır.

1. **Hibrit Mimari:** Simetrik şifrelemenin (AES/DES) hızı ile asimetrik şifrelemenin (RSA) güvenli anahtar dağıtımı yeteneği birleştirilerek, günümüz HTTPS standartlarına benzer güvenli bir **Hibrit Şifreleme** yapısı kurulmuştur.
2. **Eğitimsel Derinlik:** Algoritmalar hem hazır kütüphaneler (pycryptodome) kullanılarak hem de **manuel (kütüphanesiz)** olarak kodlanmıştır. Bu sayede S-Box dönüşümleri, Feistel ağları ve matematiksel modüler aritmetik işlemleri pratik olarak deneyimlenmiştir.
3. **Güvenlik:** Wireshark analizleri, hem mesaj içeriğinin hem de şifreleme anahtarının ağ üzerinde şifreli olarak taşındığını kanıtlamıştır.
4. **Esneklik:** Geliştirilen modüler yapı ve JSON tabanlı haberleşme protokolü sayesinde sisteme yeni algoritmalar kolayca eklenebilir durumdadır.
5. **Modern Asimetrik Yaklaşımlar:** Geleneksel RSA-2048 metoduna alternatif olarak sunulan modern ECC (Secp256r1) implementasyonu ile daha düşük paket boyutlarında üst düzey güvenlik seviyelerine ulaşılmıştır.
6. **Otonom Güvenlik (Handshake):** TCP katmanında kurgulanan otomatik el sıkışma protokolü sayesinde, anahtar takas sürecinin kullanıcı müdahalesi olmaksızın güvenli bir şekilde gerçekleştirilmesi sağlanmıştır.
7. **Ağ Katmanı Doğrulaması:** Wireshark analizleri ile şifreli verinin ağ üzerindeki görünümü, paket boyutlarındaki değişimler ve asimetrik metotların (ECC vs RSA) ağ trafiği üzerindeki performans farkları somut verilerle ispatlanmıştır.