A demonstration of secure network communication between winsock applications using RSA and AES encryption.

Shawn Andrews (BSc hon.)

saportfolio.ca

Introduction

This report details cryptography techniques that can be used in winsock application with wincrypt to perform secure message communication between winsock applications running in untrusted network while preventing man-in-the-middle attacks and forgery of packets. Secure message transmission is not limited to the techniques used in my demonstration and I will suggest alternative solutions along their trade-offs. Modern cryptographic security is always changing as new standards begin challenging hackers to crack even more sophisticated algorithms, so I will examine the growing complexity of modern-day encryption and hash algorithms used to thwart these hackers' attempts to crack the code. Finally I will answer some security concerns that you may have during your read of this report.

Assumptions

Before we look at the included winsock program, let us of discuss a few caveats pertaining to security concerns that will be discussed later in theory but not included in the demonstration. Certificates are not included in the program, so I must preface my demonstration with the following assumption:

- Assume that public-keys received from a party have been verified as authentic by a certificate signing third-party such as Verisign.

Demonstration

Let us first look at the key-exchange mechanism between two users that will lead us to secure and fast symmetric message encryption. The following series of steps result in a situation wherein both parties, the server and the client, have used slow albeit secure asymmetric encryption to transfer a session key for subsequent symmetric encryption.

I. User $\in\{Server\} \cup \{Client\}$
II. Server: Generate session key (AES-256)
III. Client: Generate private/public key pair (RSA-2024)
IV. Client: Send public key to Server
V. Server: Generate private/public key pair (RSA-1024)
VI. Server: Send public key to Client
VII. Server: Hash session key (SHA-256)

VIII. Server: Sign with Server's private key the hashed session key concatenated alongside a copy of the plaintext session key, then encrypt the signed hash with the Client's public key and send to Client
IX. Client: Decrypt packet with Client's private key then decrypt again using the Server's public key to get the plaintext session key and hash value
X. Client: Verify message integrity by hashing the plaintext session key and comparing it to the hash value
XI. Both parties now hold the session key for fast encrypted communication

The above steps can be implemented by using the following wincrypt functions, not in any particular order:

- CryptAcquireContext
- CryptGetUserKey
- CryptCreateHash
- CryptHashData
- CryptSignHash
- CryptDestroyHash
- CryptImportKey
- CryptVerifySignature
- CertOpenStore
- CertFindCertificateInStore
- CertGetNameString
- CryptAcquireCertificatePrivateKey
- CryptSignAndEncryptMessage
- CryptDecryptAndVerifyMessageSignature
- CertFreeCertificateContext
- CertCloseStore

The key exchange performed by RSA is seen in figure 1.1 where each user can be seen with their held keys and the secure packet:

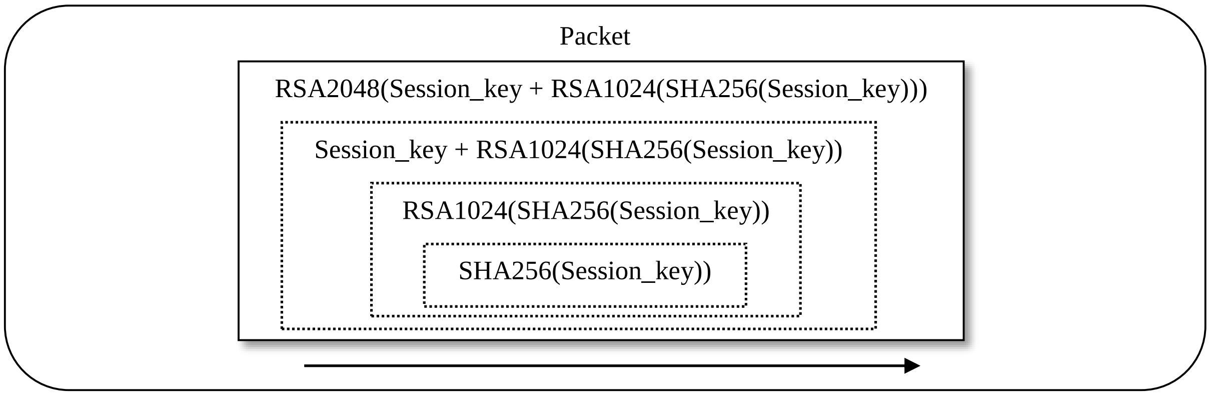
Packet
RSA2048(Session_key + RSA1024(SHA256(Session_key)))
Session_key + RSA1024(SHA256(Session_key))
RSA1024(SHA256(Session_key))
SHA256(Session_key))

Figure 1.1 RSA key exchange

- Asymmetric vs symmetric

Alternatives

- Trade-off

Security

- complexity theory

- purpose of the hash is not for security

- Why does symmetric messages not need to be signed? Because during my RSA the messages were already authenticated as authentic because only 2 people hold the session key so by deduction we know the server sent it.