



Lecture Notes on Feb/01/2023

Chapter 3: Security Protocols and Algorithms

CYENG 351: Embedded Secure Networking

Instructor: Dr. Shayan (Sean) Taheri
Gannon University (GU)



Chapter 3 Overview

- **Network and Internet security** are dominated by security protocols.
- Covering security protocols in here, including **SSL, SSH, and IPSEC**.
- The **Internet** works on the same principles as your typical **Local Area Network (LAN)**, just at a much larger scale.
- We can look at network and Internet security as being essentially the same.
- **Similar problems for them**: Weather and the unpredictability of end-users.
- The problems may be concerted efforts from malicious individuals or organizations that have something to gain from stealing information or making use of the network difficult.
- Both purposeful and unintended failures in the system compromise the security of your application.
- Almost every communications protocol has some type of built-in security, but it is up to the application developer whether or not to use those options.
- This obviously leads to security problems, since lazy or hurried engineers are likely to skip robust security features in an effort to get the product to market.
- Various protocols designed explicitly for secure communications:
 - Some of them, such as the **Secure Sockets Layer (SSL)** are simply enhancements to existing, less-secure protocols (such as **TCP**).
 - Others are stand-alone protocols that perform a specific service for the application - such as in the case of the Secure Shell protocol, or **SSH**, which provides a secure form of remote console (similar to Telnet).



Chapter 3 Overview (Cont.)

- In order to be generally useful, a secure protocol should not be limited to only supporting connections with known systems, but should allow for anonymous yet secure communications with any system.
- The reason for this is that it is extremely impractical to set up pre-defined trusted networks for some applications.
- Sometimes this is necessary, as when setting up a **Virtual Private Network (VPN)**, but most applications generally use some type of authentication, such as a password, to trust the remote machine.
- For this reason, most dedicated protocols utilize some form of public-key encryption, which allows for the sharing of cryptographic keys with unknown systems.
- The problem with using public-key encryption is that the algorithms are extremely slow - sometimes thousands of times slower than their simpler, symmetric counterparts.
- In order to deal with this problem, many of the protocols utilizing public-key cryptography use a hybrid approach that uses the public-key algorithm to securely exchange keys for a faster symmetric algorithm.
- **Just remember - DO NOT try to optimize the algorithms - but instead, look for extra features that can be removed without damaging the integrity of the security functionality you need.**
- **Protocol Madness**
 - From the lowest levels of the hardware layer, to high level abstractions, protocols are everywhere.
 - Logically, it follows that security is a concern for all these protocols, and even the security itself is defined by other protocols.



Standardizing Security - A Brief History

- In the beginning, security, like the machines being protected, was crude and often fatally flawed in execution.
- In the 1970s academics began to collect information on security and began analyzing the mathematical foundations of those ideas.
- They found **various holes** in the assumed security of preexisting systems and began a rigorous mathematical discipline to produce algorithms that would provide security and stand up to mathematical analysis.
- They abandoned the existing idea of “security by obfuscation,” which is essentially the practice relying on the hiding of security algorithms from attackers.
- This turns out to be a very insecure practice, as there are many ways to attack the algorithms used for security without actually knowing the algorithm.
- Keeping the algorithm secret can help with security - at least on a superficial level.
- It is now generally assumed by security experts that allowing rigorous public scrutiny of algorithms is better than keeping the algorithms secret, since there is a higher chance of finding problems in the algorithms before an attacker finds them and uses them for malicious purposes.
- Since ancient times, there had been only one basic type of security - the shared-key, or symmetric, cryptography where both parties must have knowledge of a secret key shared ahead of the communications.
- Some pioneers thought about how they could protect data without having to share a secret ahead of time. → The basic idea was to develop algorithms that have a two-part key, with one half being public to share, and the other half being secret to provide the security.



Standardizing Security - A Brief History (Cont.)

- **Rivest, Shamir, and Adelman (RSA)**, was reversible, unlike the other primary algorithm of the day, Diffie-Hellman (also named for its inventors).
- **RSA** could be used for basic **public-key cryptography**, using the public-key to send an encrypted message to the owner of its matching private key.
- The invention of the **World Wide Web** led to an explosion of users, and it became more and more evident that security was needed badly.
- Netscape Company → Created Secure Sockets Layer protocol, or SSL.
- **SSL** was finally standardized by the **Internet Engineering Task Force (IETF)**, one of the standards organizations for the Internet, as Transport Layer Security, or TLS.
- **TLS** was introduced in the late 1990s and is available in essentially all Web browsers and servers.
- Today, security is needed more than ever.
- There are ways to adapt security mechanisms and protocols to resource-constrained systems so that even the most modest devices can have some modicum of security.



Cryptography in Practice - DES

- The **Data Encryption Standard (DES)** was developed in the early 1970s by a team from IBM with some help from the **National Security Association (NSA)**.
- DES by itself is a fairly quick algorithm, but its standard 56-bit key is too small to be practical anymore (a 56-bit key can be broken using brute force methods over a distributed network in very little time).
- **Triple-DES, or 3DES**, uses three **56-bit keys** to achieve a higher level of security, and is more standard today.
- The implementation of DES has some small challenges in software, as a couple of operations are trivial to implement in hardware but require some additional work to implement programmatically.
- The math behind DES is pretty heavy-duty (you can read all about it in [**Applied Cryptography**](#)), but the implementation is pretty straightforward, since most of the tricky math is actually wrapped up in a collection of constants that are used essentially to randomize the data being encrypted.
- These constants are put into what are called substitution boxes, or s-boxes, and permutation boxes (p-boxes).
- The real security of DES resides in the s-boxes and in the key.
- **How to make DES work for an embedded application?**
- The algorithm predates **32-bit** and even **16-bit** buses and is specifically designed to function in an 8-bit world. → Suitable for modern embedded developers, since the operations are designed for 8-bit computing.
- DES is also designed to be implemented directly in hardware, which can be a big advantage for performance.
- It would be very easy to add a **DES coprocessor** to an embedded design and not add too much to the cost.
- Original DES should not be used at all, since it is considered too weak. → **Solution: 3DES.**



Cryptography in Practice – RC4

- **RC4** is probably the simplest algorithm for securing information in existence.
- The entire operation is essentially an **XOR** and a swap in a **256-entry table**.
- The operation is completely reversible (the same operation is used both to encrypt and decrypt), and extremely fast.
- RC4 was originally a trade secret of RSA Security, and was invented by Ron Rivest (the “R” in RSA).
- The public version of RC4 was referred to as **Alleged RC4 (ARC4)** for many years.
- Essentially every web browser and SSL implementation supports it.
- It can be implemented in just a few lines of C code, and is blazingly fast. The real downside to RC4 is that it has not been as rigorously studied as other algorithms, and this makes many security experts a little uncomfortable. Beyond that, the fact that
- RC4 is a stream cipher and has some inherent drawbacks to implementation. → If an RC4 key is used twice to decrypt different messages, then the encrypted messages can be combined to retrieve the plaintext message without knowing the key. → As long as you never reuse the same key (there are a lot of keys in 128 bits), then you should be OK.
- As an embedded designer, it is very hard to overlook RC4 for a secure application, since it is so small and fast.
- **One major caveat to point out with RC4 is that it is a stream cipher** (as opposed to a block cipher like AES or DES).



Cryptography in Practice – RC4 (Cont.)

- A major problem in **using stream ciphers** (or using block ciphers in a stream-cipher mode) is what is called a substitution attack. → If the attacker knows the format and portions of the plaintext message, he/she can anticipate where parts of the message will be and actually modify the encrypted message without knowing the key.
- Protocols that utilize stream ciphers effectively use a checksum or hash of the message included in the encrypted payload to check for any tampering.
- Another problem with stream ciphers is that you must never use the same key twice. → The problem here is a mathematical property of stream ciphers—if an attacker has two different encrypted messages that use the exact same key then he/she can retrieve the message rather trivially.
- The solution to this problem is to use what is called an initialization vector, which is essentially a one-time random string of data that is combined with the secret key to produce a relatively unique key for transferring data (SSL and other protocols that use stream ciphers do this).
- The initialization vector must be large enough to ensure that the probability of using the same vector twice is extremely low. → This was one of the issues with **WEP** (**Wired Equivalent Privacy**, a broken Wi-Fi security protocol).



Cryptography in Practice – AES

- **Advanced Encryption Standard (AES)** → A replacement for DES.
- The AES algorithm is a fairly simple design that lends itself to high performance and elegance of implementation.
- Easily implementable in hardware, so AES has some properties that make hardware acceleration a real possibility.
- AES can be implemented completely in software or completely in hardware.
- **Having the hardware acceleration for the AES implementation makes the software smaller and faster.**
- Generally speaking, if you need a symmetric encryption algorithm for an application, you should use AES.
- It is the safest choice because it is the standard used by the US government and it is backed by some of the leading experts in cryptography.



Cryptography in Practice – RSA

- Developed and patented by **Ronald Rivest (“R”), Adi Shamir (“S”), and Leonard Adleman (“A”)** in 1978, RSA is the most well-known and probably most useful public-key algorithm.
- One of the most useful properties of RSA is that it can be used both for the basic public-key operation (I send you my public-key so you can encrypt a message to send back to me), and for authentication (I encrypt a message with my private key which you can verify came from me using my known public-key).
- This property makes **RSA** especially useful for protocols that utilize both a public-key operation and authentication, like **SSL**.
- Essentially, RSA consists of a function that utilizes some unique properties of large prime numbers and modular mathematics.
- The key generation for RSA involves selecting two very large prime numbers and multiplying them together.
- The trick is that if you know the prime factors used to generate the key then the RSA encryption function is simple to reverse (thereby decrypting an encrypted message).
- This prime factoring can be considered similar to **the brute-force search** required for the naïve attack on symmetric algorithms.
- However, factoring a number is inherently easier than searching through all possible symmetric keys represented by a number of the same size.
- It is for this reason that symmetric keys are often 128 bits long, but a similarly-secure RSA key will be 1024 or 2048 bits long.
- Unfortunately, these large numbers are hard to deal with, even on a computer, so RSA is significantly slower than any of the symmetric algorithms we have looked at.



Cryptography in Practice – RSA (Cont.)

- RSA is too slow to be generally useful, but since it does have specific useful properties, it is usually used along with a faster symmetric algorithm, usually to exchange symmetric keys.
- **For embedded systems**, the performance of **RSA** is especially problematic, since an RSA operation on even a moderate-sized key (**1024-bits** is considered moderate as of right now) can take many seconds on slower processors.
- For performance reasons, RSA should be used with some type of hardware acceleration if at all possible.
- The primary part of RSA that benefits from acceleration is **the modular math** - literally millions of operations are done in a typical RSA operation and any improvement in their speed will result in a significant improvement in performance.
- **RSA is an extremely useful algorithm that is employed in thousands of applications.**



Cryptography and Protocols

- Most of the time, cryptographic algorithms are not used on their own, but rather as part of a complete security system or protocol.
- RSA is pretty much always used with some symmetric algorithm as a key exchange mechanism.
- Many protocols are designed to be used for a particular application, such as the Secure Shell (SSH) protocol, which is designed to **provide a remote text-based console**, like Telnet but secure.
- Some protocols are designed to be a general-purpose solution, encrypting everything that is sent between two machines on a network. → Examples of this type of protocol include **SSL and IPSEC**.
- When choosing a protocol for an application, you have to look at not only the features that the protocol provides, but also how the protocol has proven itself in the field.
- **SSL version 2** (the first publicly available version, implemented as part of the Netscape web browser) seemed to be secure, but was later shown to be fatally flawed.
- The replacement, **SSL version 3**, has been in use for almost a decade now, and seems to work pretty well.
- You also need to look at who designed the protocol—was it “**design by committee**” or were there some security experts and cryptographers involved?
- A recent example of why you need to research a protocol before using it is the case of the **Wired-Equivalent Protocol (WEP)**, used by the Wi-Fi protocol suite to provide basic security for wireless transmissions.
- The protocol was designed by a committee that did not include the appropriate experts, and once the protocol went public, it did not take very long for some real experts to show that the protocol was fatally flawed.
- Probably the best defense against improperly implementing a security protocol is to strictly follow good software engineering practices.
- When looking to a protocol for an embedded application, one property to look for is **flexibility**.



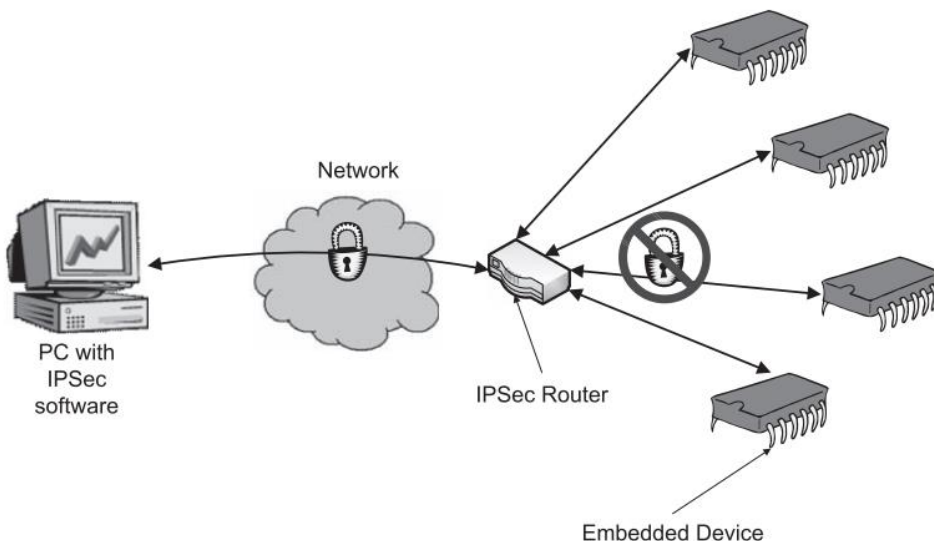
Other Security Protocols

- The more specialized a protocol or standard is, the more likely it is that it can be implemented in a small footprint.
- On the other hand, the specialized nature of the protocols leaves less room for flexibility, but for some applications, this may not be an issue.
- **SSH**
 - Sharing many similarities to a specialized version of SSL, the **Secure Shell (SSH) protocol** provides a secure Telnet-style console interface, as well as several other remote-access services.
 - This is a highly useful tool for creating secure remote configuration interfaces, since it is geared toward remote shell functionality.
 - Due to the similarity in abbreviations, SSH is often confused with SSL, but they are separate protocols developed by separate organizations (the Netscape Corporation created SSL, and SSH was developed by a Finnish researcher).
 - The two protocols do share many similarities, such as the use of public-key encryption to exchange symmetric-key encryption keys and the inclusion of **both authentication and encryption mechanisms**.
- **IPSEC**
 - IPSEC is a protocol designed to work at the IP layer, and is widely used for implementing Virtual Private Networks.
 - IPSEC is a security framework similar to SSL, but with a bit larger scope.
 - The IPSEC protocol, described in RFC 2401, consists of various security mechanisms and policies that are used to build different kinds of secure tunnels between devices.
 - The protocol is quite broad, and implementing it for an embedded system would represent many challenges.

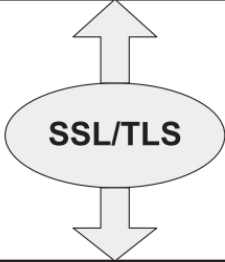
➤ Other Protocols

- There are literally hundreds of protocols available for security, many of them freely available through the IETF RFC standards.
- There are also numerous security packages available from various vendors, and some of them are quite popular, but those products frequently use predefined and proven algorithms and mechanisms.
- For cryptographic algorithms, one need only look at the size of Applied Cryptography and glance through it to learn that those algorithms number in the thousands.
- A few remaining protocols that bear mentioning here are: the **HTTP authentication**, the **HTTPS protocol (HTTP over SSL)**, the **secure FTP**, and the **PGP**.

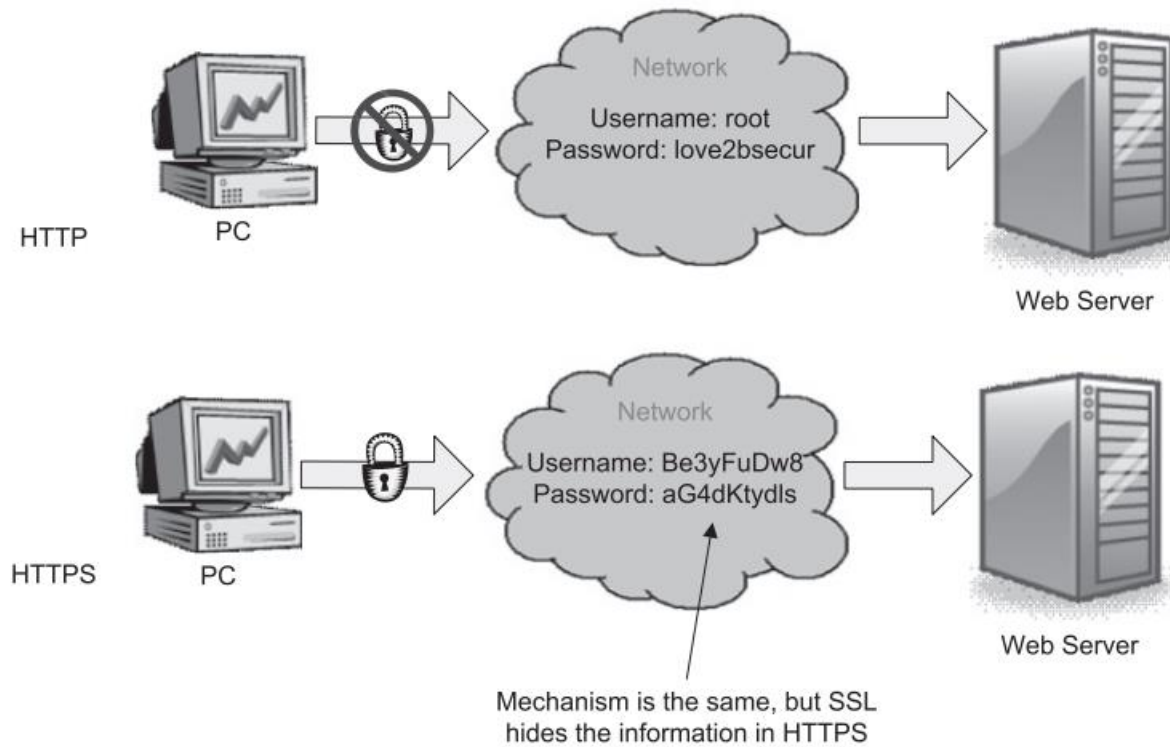
Embedded Devices Behind an IPSEC Router



IPSEC vs. SSL in the Network Stack

Application	HTTP, FTP, SMTP, DHCP, Telnet	
		
Transport	TCP, UDP	
Network	IP, ARP	IPSec
Link	Ethernet (software), 802.11 MAC, PPP	
Physical	Ethernet (physical), 802.11 radios, RS-232	

Other Security Protocols (Cont.)



Authentication in HTTP versus HTTPS



Assignment

➤ Reading Assignment:

- Stapko, T., 2011. **Practical embedded security: building secure resource-constrained systems**. Elsevier.
 - ✓ “Chapter 3: Security Protocols and Algorithms”, Pages 49-66.



Questions?