Network Security

Ramakrishna Thurimella
Colorado Research Institute for Security and Privacy
University of Denver, Denver, CO 80208. USA

Leemon C. Baird III
Department of Computer Science
United States Air Force Academy, CO 80840. USA

Abstract

Three pillars of security—confidentiality, integrity, and availability—are examined in the context of networks. Each is explained with known practical attacks and possible defenses against them, demonstrating that strong mathematical techniques are necessary but not sufficient to build practical systems that are secure. We illustrate how adversaries commonly side-step cryptographic protections. In addition, we contend that effective key management techniques, along with privacy concerns must be taken into account during the design of any secure online system. We conclude with a discussion of open problems for which fundamentally new methods are needed.
Keywords: Network Security, ARP Spoofing, Man-In-The-Middle Attack, Jam Resistance, Tor, Traffic Analysis, Key Distribution, DDoS, Quantum Cryptography

INTRODUCTION

Confidentiality, integrity and availability, often abbreviated CIA, are key security requirements in any risk analysis. In short, confidentiality is the privacy of an object, integrity is the trustworthiness and dependability (accuracy and consistency of information), and availability refers to the fact that a resource can reliably be used when desired. Stamp (2006) contains more detailed definitions of these concepts.

The most common use of cryptography online is to provide confidential and authenticated communication between two parties, either in the context of web transactions or for remote access. In order to accomplish this, one needs an effective key management scheme. As a way of demonstrating that many security concepts are intertwined, we present keyless jam resistance, a method that can broadcast messages using radio frequency communication without any prior secret shared between the sender and receiver.

Possibly the most difficult to achieve form of confidentiality is privacy of the identity of an individual performing some action, more commonly referred to as anonymity. While a common security goal is non-repudiation—the assurance that an individual can not retract his responsibility for an action—it's dual, the ability to disclaim responsibility for an action can be equally desirable. Modern mechanisms for generating anonymity combine the use of large groups of operators with a public-key infrastructure and data encryption to decouple an individual's action from their identity.

The remainder of this chapter is organized as follows. The following section presents the necessary background material for this chapter. Next we discuss confidentiality and integrity. After that, a key aspect of privacy, online anonymity, is discussed. Availability is described throughout the chapter and discussed briefly in a separate section. Key Management section presents a comprehensive list of methods to distribute secret keys. Wireless Availability section shows how to eliminate the need for keys by presenting a novel algorithm to do jam resistance communication. We conclude with a discussion of open problems in the last section.

BACKGROUND

In this section, we begin with the basics of cryptography, pointing out the difference between symmetric and asymmetric encryption, followed by a description of the Diffie-Hellman key exchange protocol. Next, we present an abstract description of the man-in-the-middle attack. After that, we give some networking details that are necessary to understand a concrete man-in-the-middle attack on modern local-area networks.

Cryptography

We first begin with a general discussion on cryptography. Figure 1 shows the process of encryption followed by a description. First, the plaintext is transformed into cipher text by applying a key K_e. Applying another key K_d, possibly different from K_e, retrieves the original.

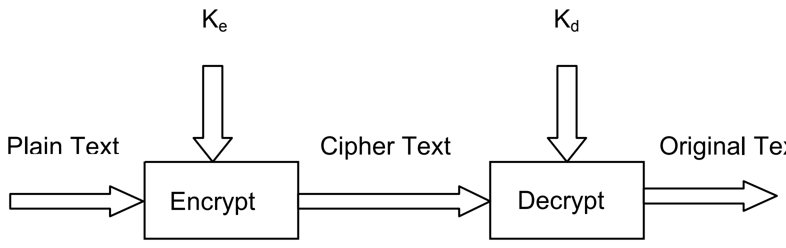


Figure 1 Process of encryption and decryption.

In symbols, this process is shown as $P = D(K_d,E(K_e,P))$.

The encryption and decryption methods, when combined, are known as a *cipher*. When the decryption key is the same as the encryption key, or efficiently derivable from it, the process is known as *symmetric* encryption; otherwise, it is called *asymmetric* encryption. Two popular symmetric encryption methods are Advanced Encryption Standard (AES) (Daemen & Rijmen, 2002) and Triple Data Encryption Standard (3DES) (“Data Encryption Standard,” (2009)). The main difficulty with symmetric encryption is *key distribution*—getting the communicating parties to agree upon a common key. This problem is discussed at length later in the Chapter.

In public key cryptography, each communicating entity maintains one private key and one public key, K_{priv} and K_{pub} respectively. Extending the previous notation, asymmetric encryption can be shown as $P = D(K_{priv},E(K_{pub},P))$.

As the names imply, the public key is made available freely to anyone who wishes to use it, but the private key is kept secret. So, if Alice wishes to communicate with Bob, she encrypts the message with Bob's public key (which is openly available) and sends the encrypted message to Bob. Anyone eavesdropping on this communication cannot decrypt the message unless they have Bob's secret key. Since anyone who wants to communicate with Bob can easily get access to his public key, public key cryptography does not suffer from the key distribution problem. However, public key cryptography does have a different drawback. It entails performing modular arithmetic over large integers (few hundred digits long) which is computationally expensive. In practice, a hybrid method is used: public key cryptography is used initially to exchange a random symmetric key, and this random key is used for the remainder of the session. Two popular public key methods are RSA (Rivest et al., 1978) and ElGamal (1985).

Public key cryptography has another very desirable property. The public and private keys can be applied in the reverse order: $P = D(K_{pub},E(K_{priv},P))$.

If Bob sends Alice $E(K_{priv},P)$, then Alice can be assured that the message P came from Bob as only Bob has access to K_{priv}. In this case, P is said to be *digitally signed* by Bob.

Diffie-Hellman Key Exchange Protocol

The Diffie-Hellman (DH) protocol allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communication channel (Diffie & Hellman, 1976). This protocol is also known as Diffie-Hellman-Merkle ("Diffie-Hellman," 2009). In short, DH is based on the fact that

$$(g^{a} \bmod p)^{b} \bmod p = (g^{b} \bmod p)^{a} \bmod p$$

where all computations are performed over a group of integers modulo p for some large prime p. Its cryptographic strength comes from the fact that it is easy to compute powers modulo a prime but hard to reverse the process when large integers are involved. This intractable problem is known as the *discrete log* problem. For example, if p were a prime of at least 300 digits, and a and b were at least 100 digits long, then even the best algorithms known today could not find a given only g, p, and $g^a \bmod p$, even using all of mankind's computing power ("Diffie-Hellman," 2009). In practice is g usually either 2 or 5.

Alice and Bob can agree on a shared secret by perform the following steps (all arithmetic is modulo p):

1. Alice and Bob agree on a large prime p and a generator g.
2. Alice picks a random number a, $0<a<p$, sends g^a to Bob, and keeps a secret.
3. Bob picks a random number b, $0<b<p$, sends g^b to Alice, and keeps b secret.
4. Alice computes $(g^b)^a$.
5. Bob computes $(g^a)^b$.

Both Alice and Bob are now in possession of the group element g^{ab}, which can serve as the shared secret key. The values of $(g^b)^a$ and $(g^a)^b$ are the same because multiplication in groups is associative. Only a, b and $g^{ab} = g^{ba} \bmod p$ are kept secret. All the other values—p, g, $g^a \bmod p$, and $g^b \bmod p$—are sent in the clear.

Trust, Certificates, and Man-In-The-Middle (MITM) Attack

Say Alice wishes to communicate with Bob using public-key cryptography. In this attack, Mallory, the attacker can participate actively or passively. In the latter role, she faithfully proxies the communication between Alice and Bob, while eavesdropping on their conversation—a breach of confidentiality. In the active mode, Mallory can choose to edit, delete, or inject packets.

If Alice requests Bob's public key and Mallory is able to intercept it, then Mallory can mount a man-in-the-middle attack. Mallory responds back to Alice with her public key K_m. Alice is under the impression that she is talking to Bob and encrypts all her messages with K_m which Mallory can decrypt.

Meanwhile, Mallory, pretending to be Alice, sends K_m to Bob, telling him that it is Alice's public key and requests his public key. Bob, like Alice, encrypts all his messages with K_m which Mallory can decrypt.

Both Bob and Alice are under the impression that they are talking to each other, but all communication passes through Mallory and is completely controlled by Mallory. The attack mounted by Mallory is known as the *man-in-the-middle attack*.

This problem arose because the public keys are sent directly by their owners. The solution is to exchange public keys through a trusted third party. This is accomplished by using *digital certificates* that contain the public key for an entity and an assurance from a trusted third party that the public key belongs to that entity. The trusted third party that issues digital certificates is called a *Certification Authority* (CA). As these certificates are digitally signed by CAs, the certificates provide protection against impersonation. Authenticity of certificates is easily verified since a CA's public key is "universally" available (e.g. embedded in browsers). When a certificate is for an individual entity (resp. Certification Authority), the certificate is a *personal (resp. root) certificate*.

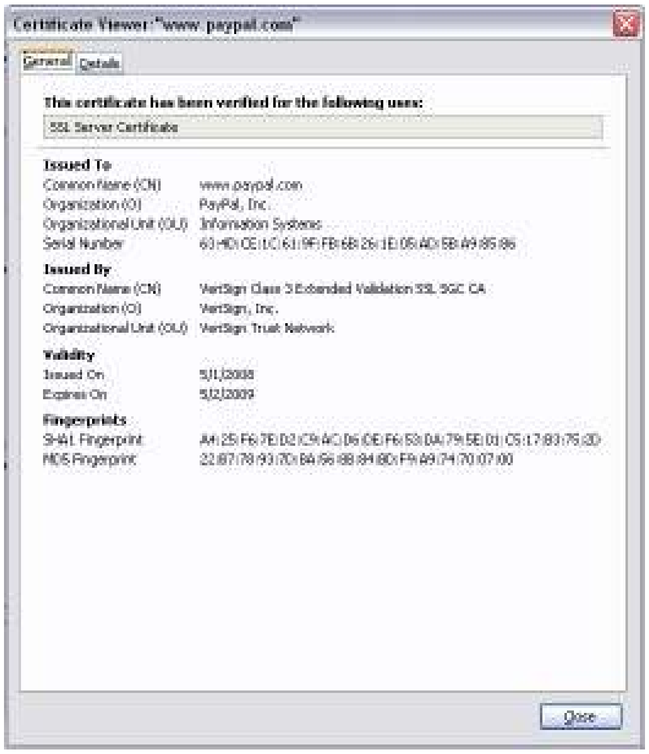
Digital certificates contain at least the following information about the entity being certified:

- The public key of the certificate holder
- The common name of the certificate holder
- The common name of the CA that is issuing the certificate
- The date certificate was issued on
- The expiration date of the certificate
- The serial number of the certificate

For obvious reasons, digital certificates do not contain the private key of the owner because it must be kept secret by the owner. See an example certificate in Figure 2.

A *Public-Key Infrastructure* (PKI) is a system of facilities, policies, and services that support the use of public-key cryptography for authenticating the parties involved in a transaction ("Public Key Infrastructure", 2009). There is no single standard that defines the components of a PKI, but it typically comprises of CAs and Registration Authorities (RAs) that provide the following services:

- Issuing digital certificates
- Validating digital certificates
- Revoking digital certificates
- Distributing public keys



Certificate Viewer:"www.paypal.com"

This certificate has been verified for the following uses:

SSL Server Certificate

Issued To

Common Name (CN)	www.paypal.com
Organization (O)	PayPal, Inc.
Organizational Unit (OU)	Information Systems
Serial Number	63:HD:CE:1C:61:9F:FB:6B:26:1E:05:AD:5B:A9:85:86

Issued By

Common Name (CN)	VeriSign Class 3 Extended Validation SSL SGC CA
Organization (O)	VeriSign, Inc.
Organizational Unit (OU)	VeriSign Trust Network

Validity

Issued On	5/1/2008
Expires On	5/2/2009

Fingerprints

SHA1 Fingerprint	A4:25:F6:7E:D2:C9:AC:D6:DE:F6:53:DA:79:5E:01:C5:17:B3:75:2D
MD5 Fingerprint	22:B7:78:93:7D:BA:56:8B:84:BD:F9:A9:74:70:07:00

Figure 2 Digital certificate received from PayPal web server as viewed from a browser.

The X.509 is an International Telecommunication Union standard for a Public Key Infrastructure (Cooper, 2008). RAs verify the information provided at the time when digital certificates are requested. If the information is verified successfully by the RA, the CA can issue a digital certificate to the requester.

Networking Basics

The Transmission Control Protocol (TCP) and the Internet Protocol (IP) together are at the heart of communication protocols used for the Internet. These protocols resulted from years of research funded by Defense Advanced Research Projects Agency (DARPA). The TCP/IP suite defines a set of rules that enable computers to communicate over a network. The rules specify data formatting, addressing, shipping, routing and delivery to the correct destination.

The TCP/IP stack is an abstraction of four layers as shown in Figure 3. Conceptually similar functions are aggregated into a layer and the resulting layers are stratified based on the services provided. For example, in the 4-layer model, TCP at level 3 provides reliable packet delivery. Building on this, the application layer at level 4 can offer a stateful telnet session to the end user without having to worry about dropping the connection.

7. Application
6. Presentation
5. Session
4. Transport
3. Network
2. Data Link
1. Physical

OSI 7-layer model

4. Application (Telnet,SMTP, HTTP,FTP,…)
3. Transport (TCP,UDP)
2. Internet (IP)
1. Link (Ethernet,Token Ring)

TCP/IP 4-layer model

Figure 3 Seven-layer versus the four-layer networking model

In contrast, the Open Systems Interconnection Reference Model (OSI Reference Model or OSI Model) is a more detailed 7-layer model (Zimmermann, 1980). From top to bottom these are the Application, Presentation, Session, Transport, Network, Data-Link, and Physical Layers.

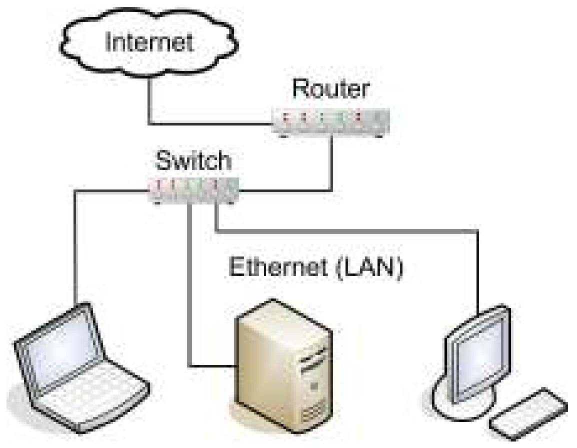


Figure 4 Local area network (LAN) connected to the Internet via a router. A router is seen by the switch as another host on the LAN.

It is useful to understand the role played by various networking components and map their functionality to the services provided in the 4-layer model.

Computers that are in close proximity and connected into the same LAN communicate with each other using Ethernet. This protocol operates at Layer 2 in the OSI model and at the Link Layer in the 4-layer model. In this protocol, frames are sent to a destination Media Access Control (MAC) address, a 48-bit address that is unique to each Network Interface Card (NIC) on the network. The nodes on a LAN are connected using a hub or a switch. The only difference between them is that a hub is less intelligent and cheaper than a switch. It simply broadcasts every packet it receives to every computer on the LAN. For many years, hubs were very common and posed serious security problems for system administrators, as anyone on the LAN can connect to the LAN, put their NIC into “promiscuous” mode and eavesdrop on all data transferred on the

LAN. Switches, on the other hand, direct Ethernet frames to where they need to go instead of broadcasting. In addition to improved security, switches also increase the rate at which data can be transferred.

To connect a LAN to the Internet, one needs a more intelligent device that can route packets to the Internet. This device is called a *router*. This is a Layer 3 device in the OSI model. It is smarter than a switch in the sense that it is programmable and usually includes an interface by which it can be configured. Routers have the ability to communicate with other routers and determine the best way to route network traffic from one point to another on the Internet. For simplicity, let us assume that there is only one router on any given LAN. Then, since all traffic from the LAN must enter and exit through the router, it provides a useful *choke point*. The computers on the LAN can be protected from outside attackers by running a firewall along with an intrusion detection system at this choke point.

A *default gateway* is the node on the LAN that is chosen by the switch when it encounters an IP address that does not belong to any node on the LAN. A router usually assumes the role of a default gateway. In home networks, the functionality of a switch, router, and wireless access point are often combined into one physical unit.

SSL/TLS

Transport Layer Security (TLS) is a security protocol from the Internet Engineering Task Force (IETF) that is based on the Secure Sockets Layer (SSL) 3.0 protocol developed by Netscape. TLS is the successor to SSL. Both protocols include cryptographic frameworks which are intended to provide secure communications on the Internet. SSL is not an industry standard as it was developed by Netscape. TLS is the widely recognized standard issued by the IETF for securing transmitted data. The current version of TLS is 1.1 and is described in RFC 4346 (Dierks & Allen, 1999). It is now supported on most commercial browsers, web and email servers. For the most part, SSL and TLS are interchangeable.

The SSL protocol runs above TCP/IP and below higher-level protocols such as HTTP or SMTP. It uses TCP/IP on behalf of the higher-level protocols, and facilitates the establishment of an encrypted connection between the client and server. See Figure 5.

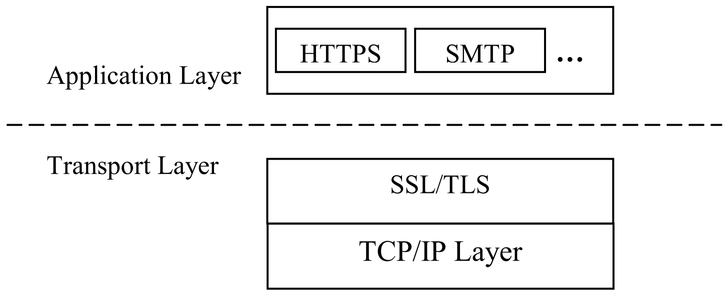


Figure 5 SSL/TLS run above TCP/IP and below the Application Layer that consists of protocols used for accessing the Internet HTTP, send and receive email using SMTP etc.

Both SSL and TLS follow a standard handshake process to establish communication. The handshake prior to an HTTPS session is as follows:

1. The client contacts a server that hosts a secured URL.
2. The server responds to the client's request and sends the server's digital certificate to the browser.
3. The client now verifies that the received certificate is valid. Certificates are issued by well-known authorities (e.g. Thawte or Verisign).
4. The server could *optionally* choose to confirm a user's identity. Using the same techniques as those used for server authentication, SSL-enabled server software can check that the client's certificate is valid and has been issued by a certificate authority (CA) listed in the server's list of trusted CAs. This confirmation might be important if the server is a bank sending confidential financial information to a customer and wants to check the recipient’s identity. (See the benefits of performing this optional step in Possible Defenses against MITM.)
5. Once the certificate is validated, the client generates a random one-time session key, which will be used to encrypt all communication with the server.
6. The client now encrypts the session key with the server's public key, which was transmitted with the digital certificate. Encrypting using the server’s public key ensures that others cannot eavesdrop on this sensitive exchange.

At this point, a secure session is established because the client and server both know the session key. Now, both parties can communicate via a secure channel. See Figure 6.

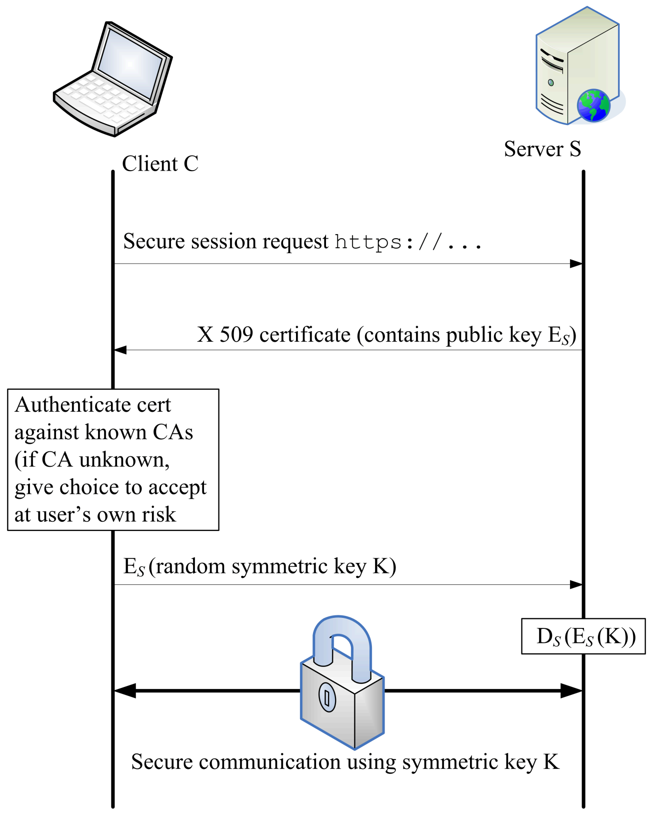

Client C
Server S
Secure session request https://...
X 509 certificate (contains public key E_S)
Authenticate cert against known CAs (if CA unknown, give choice to accept at user’s own risk
E_S(random symmetric key K)
$D_S(E_S(K))$
Secure communication using symmetric key K

Figure 6 SSL/TLS protocol handshake and session key establishment. (Adapted from The SSL Handshake (2009).)

MITM Attack on a Switched LAN using ARP Spoofing

How does Mallory eavesdrop on the exchange between Alice and Bob on modern computer networks? Aren't they built on secure technology? The answer is no, unfortunately. The problem is that Ethernet, upon which virtually all modern LANs are based, was designed without *any* sort of authentication mechanism. An attack known as *ARP spoofing* takes advantage of this weakness and can intercept communications on a LAN running the Ethernet protocol (Wagner, 2001).This attack works against most networks that are in use at the time of this writing.

The attack works as follows. Recall our discussion from the Networking Basics section on how two computers communicate on a LAN using Ethernet frames. To connect to a LAN, each host must be equipped with a Network Interface Card (NIC). Each NIC is assigned a unique Media Access Control (MAC) address by the manufacturer. Communication on Ethernet takes place by sending frames to destination MAC addresses. If a MAC address is unknown, the source node broadcasts an ARP request. This request specifies an IP address and asks the host with this IP address to reply back with its physical address. In other words, A*ddress Resolution Protocol* (ARP) finds MAC address given an IP address.

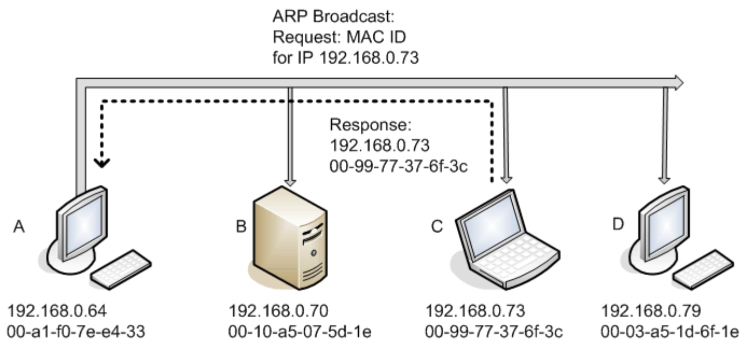
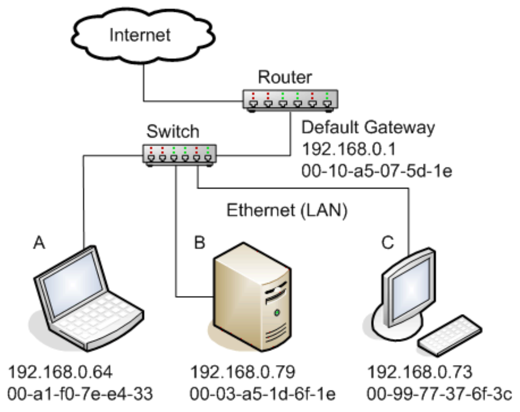


Figure 7 ARP request broadcast and response. Here host A is requesting a MAC (physical) address that corresponds to IP# 192.168.0.73 (host C).

Every node on the LAN receives every ARP request, but only the host with the matching IP address replies back with its physical address; the rest simply ignore it. The response is sent back using an ARP Reply that contains the requested IP number and the corresponding MAC address. When the source node receives this information, it stores it in a table of IP and MAC address pairs. This table is known as the *ARP cache* and the mappings are considered valid for a fixed amount time, after which they expire and are removed. Every node on the LAN maintains such a cache. Note that the source node enters the IP-MAC address pair contained in the ARP Reply into its cache without any validation or further checks. Put differently, there is *total trust* between the nodes on a LAN. To make the matters worse ARP is a *stateless protocol*, i.e. an ARP Reply is not matched to see if there are outstanding ARP Requests. Therefore, any malicious node can takeover a LAN and route all traffic through itself by sending unsolicited ARP Reply messages to various hosts—the only requirement that needs to be met is that the malicious node is a host on that LAN. One easy way is to accomplish this is by connecting to an insecure wireless access point. Many corporations, hospitals, and retail outlets still use easily breakable WEP encryption (Tews et al., 2007). This weakness exists within the TCP/IP stack. Hence, it is a multi-platform vulnerability.

By injecting merely two ARP Reply packets into a LAN, any malicious node M can control all traffic going back and forth between any two nodes on that LAN, e.g. between an unsuspecting victim node A and the default gateway G. First, M sends G a spoofed ARP Reply <IP_A, MAC_M> claiming that it was assigned IP_A (which really belongs to A) but gives its own MAC address MAC_M. The gateway would blindly replace its current correct entry with the spoofed one. At the same time M would send a similar spoofed ARP Reply <IP_G, MAC_M> to A, replacing the correct ARP cache entry for the gateway computer at A with the spoofed one. From this point on, any traffic from A bound for the default gateway would instead go to the attacking computer M. Similarly, all traffic from G destined to A is routed instead to M. Neither A nor G would be aware of the intermediary that is relaying the traffic in the middle. See Figure 8.

On a LAN with *n* nodes, that consists of (*n*-2) nodes, 1 router, and 1 attacker, by inserting 2(*n*-2) spoofed ARP Replies, the attacker can take full control of the traffic destined to the Internet from that LAN. This process of inserting false entries into an ARP cache is also referred to *ARP poisoning*. It is worth noting that cache entries are purged after a timeout period. Therefore, to keep control of the network, the attacker must periodically poison each host for the duration of the hijacked session.



After C inserts itself between B and the Default Gateway

At the Default Gateway	
192.168.0.1	00-10-a5-07-5d-1e
192.168.0.64	00-a1-f0-7e-e4-33
192.168.0.73	00-99-77-37-6f-3c
192.168.0.79	***00-99-77-37-6f-3c***

ARP Cache (at every node)	
192.168.0.1	00-10-a5-07-5d-1e
192.168.0.64	00-a1-f0-7e-e4-33
192.168.0.73	00-99-77-37-6f-3c
192.168.0.79	00-03-a5-1d-6f-1e

At B	
192.168.0.1	***00-99-77-37-6f-3c***
192.168.0.64	00-a1-f0-7e-e4-33
192.168.0.73	00-99-77-37-6f-3c
192.168.0.79	00-03-a5-1d-6f-1e

Figure 8 ARP cache values before and after poisoning by node C to insert itself between B and the Default Gateway (Router). The second column shows after ARP poisoning. The two spoofed entries are shown in bold.

In addition to compromising the confidentiality and the integrity of the data as it passes through the local network (as described in detail in the next section), MITM attacks can also adversely affect availability by simply slowing down or completely dropping the network communication by associating a nonexistent MAC address to the IP address of the victim's default gateway. Refer to the Availability section on other ways of affecting availability.

CONFIDENTIALITY AND INTEGRITY

Other than insider attacks, a man-in-the-middle (MITM) attack is probably the easiest and most common attack on network connections secured with SSL. This section presents one such attack. During this attack, in a 24-hour period, the author of the attack, Marlinspike, managed to collect a few hundred user ID/passwords of accounts at popular web email servers, financial institutions, social networking sites, etc.

Futile Defenses against MITM

It has become fashionable at many financial institutions in the United States to present the online user with a set of "secret" questions, in addition to their login credentials. After a successful login, the session might proceed along the following lines:

> To protect the security of your account, please answer the following questions:
>
> **Note:** Your answers are NOT case sensitive.
>
> What is the name of the school where you went to kindergarten?

Or questions such as

> What is the last name of your favorite actor?
>
> What is your favorite color?

Sometimes this "extra" security comes in the form of storing your favorite picture which is transmitted during the beginning of an encrypted session.

The purpose of this section is to demonstrate that these so called additional security measures are *totally ineffective* against MITM attacks. The attack described here is due to Marlinspike (2009).

MITM Attack on SSL Using Bogus Certificates

The *certificate chain* is a list of certificates used to authenticate an entity. Certificate chaining is a process by which *root* certificate authorities delegate the certificate issuing authority to intermediate CAs for efficiency and scalability reasons. This mechanism is part of the trusted computing paradigm. When certificate chains are involved in verification, to check authenticity of a certificate for an entity, the certificate chain is used to reach the *root CA* certificate. The root CA certificate is self-signed. However, the signatures of the intermediate CAs must be verified.

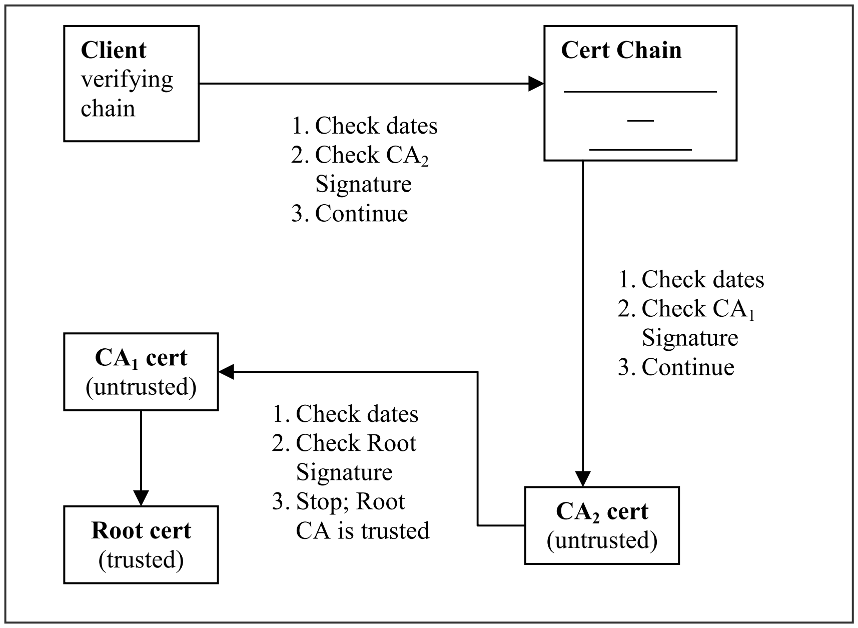


Figure 9 Certificate chain verification process by a client program. (Adapted from Figure 1 of Certificate Chain Verification (2009).)

Chains can be longer than three. Most browsers verify certificate chains as follows:

1. Verify that the name on the certificate matches the name of the entity the client wishes to connect.
2. Check the certificate's expiration date.
3. Check the signature. If the signing certificate is in the list of root CAs in the client, stop, otherwise, move up the chain one link and repeat.

Assume an attacker is in possession of the domain `attacker.com` and a certificate is issued to it by CA_2. Consider the following certificate chain:

Root CA→CA_1→CA_2→attacker.com→victim.com

Anyone connecting to `victim.com`, first checks its name and expiration, and then verifies its signature by applying the public key of `attacker.com`. Assuming that this is successful, the process is repeated with `attacker.com`, CA_2, and CA_1, until `Root CA` is reached. In this example, all signatures and dates pass the validity test, and the `Root CA` would be reached successfully. Since the `Root CA` is always trusted, the whole chain is considered to be intact. Unfortunately there is a problem; `attacker.com` should not have the authority to issue certificates to other domains. This restriction is imposed in the Basic Constraints Extension of the X.509 specification (Cooper et al., 2008). It identifies whether the subject of the certificate is a CA and length of a certificate chain, including itself. The intent in the Standard is to prevent non-CAs from issuing certificates. For non-CAs, this field should be `CA:FALSE` indicating that the entity to which this certificate was issued is not a CA. Unfortunately many CAs did not explicitly set this field and most browsers simply ignored it. The implication of this careless practice is that any entity with a valid certificate could create a certificate for any domain.

In 2002, Marlinspike released a software tool, *sslsniff* that took advantage of this weakness. This tool has the capability to dynamically generate certificates for domains that are being accessed on the fly. The new certificate becomes part of a certificate chain that is signed by any certificate provided to sslsniff.

Using sslsniff, one can perform MITM attack on an HTTPS session as follows. First, an HTTPS request from victimClient trying to connect to victimServer is intercepted using standard techniques such as ARP poisoning. The attacker then sends a bogus certificate in the name of victimServer. Unsuspecting, victimClient authenticates the certificate chain and sends a symmetric key, encrypted using the public key supplied by the attacker. The attacker decrypts the symmetric key, which is used as a session key. Simultaneously, the attacker opens an HTTPS session with victimServer and proxies the traffic between victimClient and victimServer, relaying the set "secret" questions and answers back and forth. All the data that is in transmitted between the client and the server is available to the attacker *in the clear* including sensitive information such as credit card numbers.

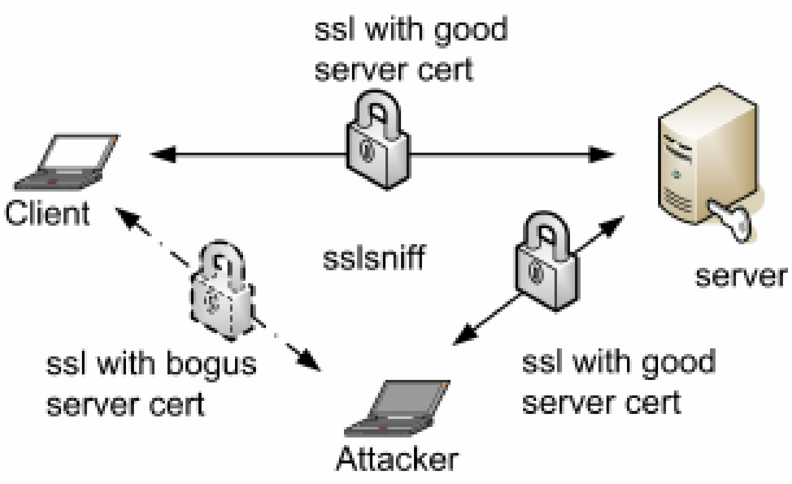


Figure 10 MITM attack on secure web sessions using bogus certificates

This weakness in the Basic Constraints field of X.509 has since been addressed by the CAs and the newer generation of popular browsers are no longer susceptible to this attack.

MITM Attack Using Other Means

Even though one may not be able to carry out MITM attacks using bogus certificates against newer web technology without raising too many red flags, there are a variety of other techniques that one can employ to launch an MITM attack and breach the confidentiality of secure web transactions. The techniques presented here are browser independent and are effective against web sites of some leading financial institutions.

Since it now appears as if HTTPS has been secured, what is the best way to hijack a web session? Marlinspike (2009) provides an answer to this question by asking the following questions related to human-computer interaction (HCI):

1. How do people start an HTTPS session?
2. How are people assured that they are using a secured session?
3. How are people warned that there maybe a problem with the security of the session?

Most often, the answer to question 1 is either

1. User clicking on a button that posts to HTTPS, or
2. Through rerouting from the web server (HTTP response code 302).When the user types `victimServer.com`, the browser resolves it to `http://www.victimServer.com`. For example, the exchange might look like

   ```
   GET /index.html HTTP/1.1
   Host: www.victimServer.com
   ```

   When `victimServer` receives the above request, it reroutes the client as

   ```
   HTTP/1.1 302 Found
   Location: https://www.victimServer.com/index.html
   ```

That is, no one really types `https://` before starting an online transaction. In other words, *access to HTTPS is via HTTP*. The strategy of the attacker becomes, attack HTTP if HTTPS is secure.

Questions 2 and 3 can be best understood by studying how browsers have evolved over the years. Seven years ago, when sslsniff was released, excessive positive feedback was given by the browser that a user was using a secure connection. There were many lock icons, the address bar or uniform resource locator (URL) bar changed color, and a number of other indicators were deployed to give a "warm-and-fuzzy" feeling to the user that the page was secure. A *favicon*, short for favorites icon, is a 16x16 pixel square icon associated with a particular website that is displayed in URL bar. A popular favicon in the older browsers during secure sessions was a small padlock (see Figure 11).

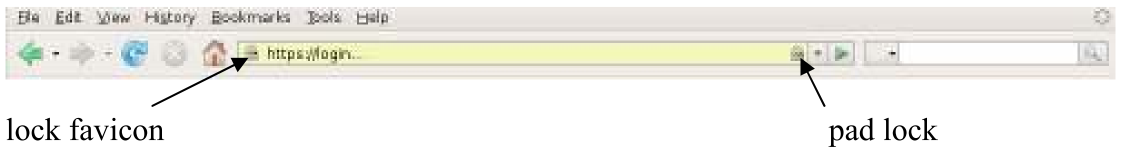
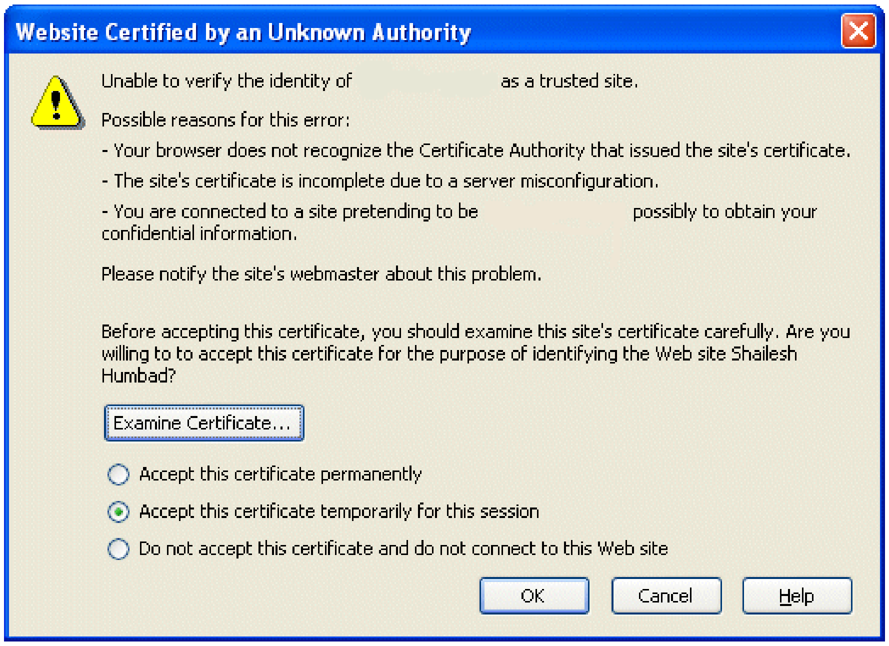


Figure 11 Positive feedback to the user by changing URL bar color and lock icons

Another example of positive feedback is as follows. When a bogus certificate is detected by the browser, a dialog similar to the one shown in Figure 12 is presented to the user. Notice that by default, the certificate chain would be accepted for the session. According to Marlinspike (2009), users typically click through these warning dialogs as they don't completely understand the meaning of the warning.



Website Certified by an Unknown Authority

Unable to verify the identity of as a trusted site.

Possible reasons for this error:

- Your browser does not recognize the Certificate Authority that issued the site's certificate.

- The site's certificate is incomplete due to a server misconfiguration.

- You are connected to a site pretending to be possibly to obtain your confidential information.

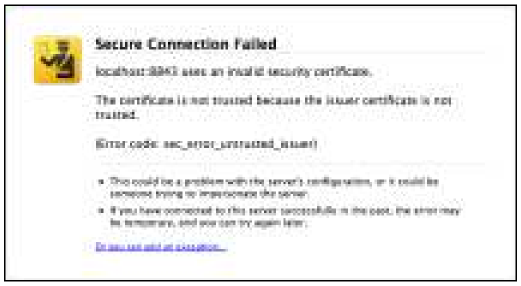
Please notify the site's webmaster about this problem.

Before accepting this certificate, you should examine this site's certificate carefully. Are you willing to to accept this certificate for the purpose of identifying the Web site Shailesh Humbad?

Accept this certificate permanently [✓ Accept this certificate temporarily for this session] Do not accept this certificate and do not connect to this Web site

Figure 12 Warning dialogs that are routinely ignored by most online users.

The trend in the newer browsers is to scale back the positive feedback while emphasizing the negative. For instance, instead of encouraging the user to simply click through the dialog as shown in Figure 12, more ominous looking dialogs like the ones shown in Figure 13 are generated when an invalid certificate is found in the certificate chain. In addition, newer browsers control the proliferation of lock icons, use plain colors for the URL bar, and employ normal favicons.



Secure Connection Failed

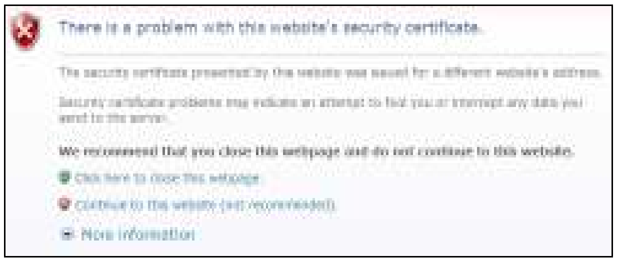
localhost:8443 uses an invalid security certificate.

The certificate is not trusted because the issuer certificate is not trusted.

(Error code: sec_error_untrusted_issuer)

- This could be a problem with the server's configuration, or it could be someone trying to impersonate the server.
- If you have connected to this server successfully in the past, the error may be temporary, and you can try again later.

Or you can add an exception…



There is a problem with this website's security certificate.

The security certificate presented by this website was issued for a different website's address.

Security certificate problems may indicate an attempt to fool you or intercept any data you send to the server.

We recommend that you close this webpage and do not continue to this website.

Click here to close this webpage.
Continue to this website (not recommended).
More information

Figure 13 Negative Feedback

This shift in HCI with respect to online security has been referred by Marlinspike as going from giving the user *positive feedback* to *negative feedback*. His recent attack is based on the observation that any attack that triggers negative feedback is bound to fail, but the absence positive feedback during the attack is not so bad.

The attack proceeds as follows:

1. Intercept all web (HTTP) traffic and replace
   a. `<a href=https://...>` by `<a href=http://...>`
   b. `Location: href=https://...>` by `Location: href=http://...`

And keep a map of all replacements.
2. If there is an HTTP request from the client for a resource for which there was replacement in the previous step, issue an HTTPS connection to the server for the same resource, and
3. Relay the response from the server to the client using HTTP.

The key difference between this MITM attack and the attack using bogus certificates is that in the previous attack, the attacker uses HTTPS to connect to both the client and the server. By comparison, in this new MITM attack, the attacker only communicates with the server in encrypted mode. From the point of view of the server, this would appear like a normal secure online transaction. Compare Figures 10 and 14.

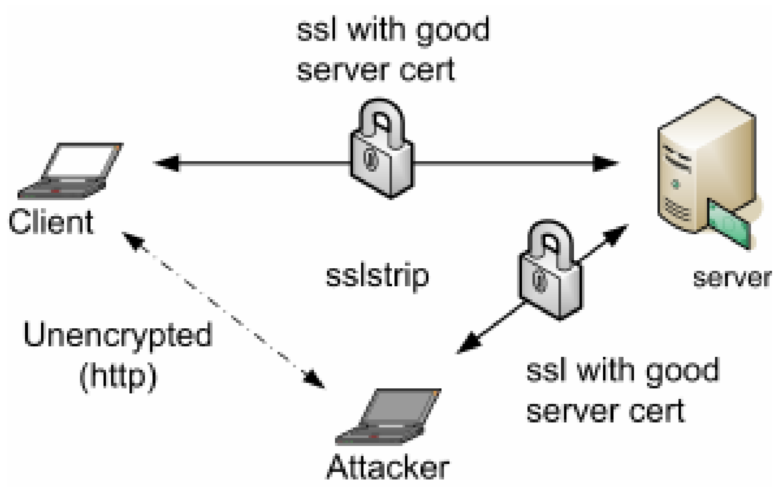


Figure 14 Hijacking secure online transactions

On the client side, there are no tell-tale signs of a breach since the attack suppresses nasty dialogs from popping up. This accomplishes the goal of not triggering any negative feedback. To complete the attack, Marlinspike adds some positive feedback. This is done by adding a lock favicon in the URL bar. That is, whenever a favicon request is noticed for a URL that is in the map, a lock favicon is returned. The only difference a security savvy user would notice is the absence of a lock icon in the status bar and `http` instead of `https` in the address bar.

The results from this experiment are remarkable. The security of over a hundred email accounts, a few credit card numbers, and a few hundred secure logins was breached in a matter of a single *24-hour period*. Another surprising aspect of this test was that *not a single user* attempting to initiate a secure transaction aborted it because the user became suspicious.

Marlinspike also showed how to extend the *homograph* attack (attack that attempts to deceive remote users about what server they are communicating with, by taking advantage of the fact that many different characters have nearly indistinguishable glyphs) to mount MITM against SSL. We omit the details of this attack and the technical problems posed by cached pages. The interested reader is referred to Marlinspike (2009).

Possible Defenses against MITM

We conclude this section by presenting some effective measures an online user can take to defend against MITM attacks. First and foremost is to educate oneself to look for signs of a breach. It is also important to understand the meaning of different warning dialogs presented by the browser.

If a web server offers its services only over HTTPS (on TCP port 443) and routinely redirects all HTTP (port 80) requests to the secure port 443, then sessions can still be hijacked. As long as HTTPS depends on HTTP, it is vulnerable because HTTP is not secure. Why not just turn off port 80? Unfortunately this would cause many `Server Not Found` errors for the users and it would not be good for business. One work around is to have the user type in `https://...` in the address bar. Alternately, the user could bookmark the secure site and issue an HTTPS request by selecting the bookmark. It is tempting to think that if browsers always try to connect over port 443 first, and only connect only to port 80 as a last resort, we can avoid the MITM attacks mentioned here. Unfortunately, the attacker can simply drop the requests to connect to port 443 and make the browsers think that the web server does not offer HTTPS. While this defense might not help in all cases, by including into browsers a select set of sites for which service over HTTPS is known to exist, one can reduce the risk of MITM attacks. The only long term solution is to secure everything, i.e. run only HTTPS.

Another measure that could improve security, that is not currently popular, is the verification of client certificates. By having servers verify the identity of the client, one can achieve better security. But, this requires significant changes to the existing PKI and is not immediately applicable.

ONLINE ANONYMITY

The notion of privacy and anonymity are closely related. When an element from a well-defined set is not identifiable within that set, then that element is said to be *anonymous*. This element could be a human being, a computer, or an email. One way to remain private is to stay anonymous. While encryption guarantees confidentiality, it provides no privacy; an attacker can observe communication patterns and deanonymize the users. For example, if the attacker notices that there are packets flowing between your home computer and a particular bank's web server, then he can reasonably conclude that you have an account at this institution and you are performing a transaction. Since public networks do not hide routing information, this is a real concern. This way of identifying information is known as *traffic analysis*. On the Internet, the main goal of anonymity is to make the communicating parties *unlinkable* by building defenses against traffic analysis. Chaum (1981) is widely credited for introducing and making a case for anonymous communication. He was the first to propose the *mix* as an essential unit for anonymity.

Why is online anonymity important? Who and what needs to be protected? These questions can be answered by considering the following scenarios:

1. **Censorship resistant publishing** The following paragraph from the Publius (2009) homepage, an online censorship resistant publishing system, motivates the importance of such a system:

The publication of written words has long been a tool for spreading new (and sometimes controversial) ideas, often with the goal of bringing about social change. Thus the printing press, and more recently, the World Wide Web, are powerful revolutionary tools. But those who seek to suppress revolutions possess powerful tools of their own. These tools give them the ability to stop publication, destroy published materials, or prevent the distribution of publications. And even if they cannot successfully censor the publication, they may intimidate and physically or financially harm the author or publisher in order to send a message to other would-be-revolutionaries that they would be well advised to consider an alternative occupation. Even without a threat of personal harm, authors may wish to publish their works anonymously or pseudonymously because they believe they will be more readily accepted if not associated with a person of their gender, race, ethnic background, or other characteristics.

2. **Socially sensitive communication** The fact that a person visits certain websites related to a disease with the goal of educating himself, and frequents online support groups for a disorder should be kept private. Otherwise, this person could be denied insurance coverage or be subjected to workplace discrimination.
3. **Law enforcement** In many crime reporting situations, witnesses will not come forward unless they are assured of anonymity. Also when police conduct surveillance, including sting operations, they must remain unidentifiable.
4. **Whistleblower protection** *Whistleblowers* are insiders who reveal questionable practices at their workplace to the public. They need to be protected from retaliation by the management.
5. **Personal information** The websites an individual visits, the set of people she communicates with, the doctors she sees, or the medicines she takes, are all examples of personal information that should remain private.

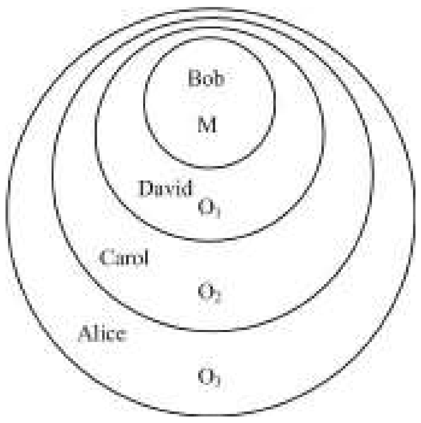
There are many other cases including open-source intelligence gathering (the Secret Service might want to visit news websites of rogue nations anonymously), elections and voting where anonymity is indispensable.

Anonymizing networks are not without their detractors. The main criticism leveled against these networks is that online criminals can hide behind them and carry out their nefarious activities. Law enforcement would have a hard time convicting these criminals as their illegal acts cannot be easily linked back to them. As with any technology, the pros and cons of online anonymity must be carefully weighed before judging its merit. Most people in the security community are of the opinion that the benefits of anonymizing networks far outweigh the risks.

Onion Routing

Goldschlag et al. (1999) introduced the idea of Onion Routing to provide unlinkable communication. It is based on *mix cascades* (or *mixes* for short) (Chaum, 1981): messages travel from source to destination via a sequence of proxies randomly chosen by the sender. To prevent the adversary from eavesdropping on the message content, it is encrypted between routers.

To keep the discussion at the conceptual level, we omit many important practical considerations and introduce Onion Routing with an example.



K_x: X's secret symmetric key with Alice

Alice sends message M to Bob as follows:

- Alice constructs & sends O_3 to Carol where $O_3 = K_c$ (nexthop = David, O_2)
- Carol decrypts O_3, retrieves & sends O_2 to David where $O_2 = K_d$ (nexthop = Bob, O_1)
- David decrypts O_2, retrieves & sends O_1 to Bob where $O_1 = K_b$ (M)
- Bob decrypts O_1 & retrieves M

Figure 15 Onion routing. Intermediate onion routers are only aware of the predecessor and successor nodes, but unaware of the contents of the data or the path the data follows.

Say Alice wants to send a message M to Bob. If anonymity is not a concern, she can simply establish a session key K_B as in SSL, use it to encrypt M and send the encrypted message. But, anyone watching the packet flow can link Alice and Bob.

To make the path taken by the encrypted message unidentifiable, Alice first picks a random path to Bob. Assume that the path goes through Carol and David. Next, she establishes symmetric keys with every Onion Router on the path, in this case with Carol, David and Bob, denoted K_c, K_d, and K_b respectively. The process of establishing these keys must be done in a manner so that it does not give away the path. This is described in the next subsection.

The communication between Alice and Bob starts with Alice finding O_3 and sending it to Carol where

$O_3 = K_c$ (nexthop=David, K_d (nexthop=Bob, K_b(M)))

Carol decrypts O_3, discovers the next hop (David in this case), retrieves O_2 passes it to David where

$O_2 = K_d$ (nexthop=Bob, K_b (M))

David in turn "peels" another layer and sends O_1 to Bob where

$O_1 = K_b$ (M)

Bob decrypts O_1 with K_b and retrieves M (see Figure 15).

Unless Carol and David collude, Carol (resp. David) does not know David's successor (resp. Carol's predecessor). Equivalently, Carol cannot link O_2 to O_1 without the secret key David shares with Alice K_d. Similarly, David cannot link O_2 to O_3 without Carol's secret key K_c. The only person who knows the entire path is the person who chose the path—the sender, Alice; not even the receiver knows the path. Furthermore, the receiver cannot infer the sender's identity from the header information unless the message somehow identifies the sender.

Notice that as the data moves from the source to destination, it gets smaller in size because it has fewer and fewer routing instructions. As an attacker could infer routing information from this monotonically decreasing packet size, the intermediate Onion Routers pad the data with random bits (equal to the number of bits peeled off at that router) so that the size of data remains constant between hops.

Once the path chosen by the sender is established, it remains active for some period of time, i.e. a session. This path is suitable for two-way communication as Bob can reply back to Alice along the same path, i.e. every node on the path would simply do the opposite: encrypt with its session key and the send the data upstream, one step closer to the sender. In our example, when Alice finally receives the response she decrypts it using K_c first, K_d next, and K_b last. Routing information is not included for the reverse path and for all subsequent two-way communication between Alice and Bob as each intermediate node is aware of its two neighbors on the path.

Note that Onion Routing does not provide complete anonymity. A local eavesdropper can observe that Alice is sending and receiving messages, but he cannot infer that the receiver of the messages is Bob.

The Onion Router

The remainder of our discussion on Onion Routing is specific to the way it is implemented in *The onion router (Tor)*, a widely used anonymizing network. Tor is a free software product distributed under GNU General Public License (GPL). Its low latency, high-bandwidth, stream-level anonymous communication ability makes it suitable for common TCP-based applications such as web browsing and instant messaging (Dingledine et al., 2004). Tor's popularity can be attributed to its ease of use and *forward security* (protection of past network activity in the event the current secret key is exposed).

Two potential risks exist for a client who is directly interacting with Tor: 1) Domain Name Service requests (translation service that is needed to resolve URLs to IP numbers) can give away the sites that a client wishes to visit, and 2) web servers typically leave cookies that invade the privacy of the client. To prevent problems of this kind Tor is commonly used with a web proxy such as *Privoxy*, another free program released under GPL. See Figure 16.

The Tor network is a distributed overlay network that is comprised of a set of nodes that act as relays. Anyone who meets certain bandwidth requirements can volunteer to be a relay and Tor server software runs in user space, i.e. no need to have root/administrator privileges. It is the responsibility of each relay to ensure that the correspondence between the incoming streams and outgoing streams is hidden from the attacker.

The *threat model* of Tor assumes that the adversary is not global, i.e. she can observe and control only part of the network, but not the entire network. This is a common assumption in all practical low-latency systems. The threat model of Tor does allow for an adversary who can observe and control (add, delete, delay packets) some fraction of the Tor nodes, and operate their own Tor nodes.

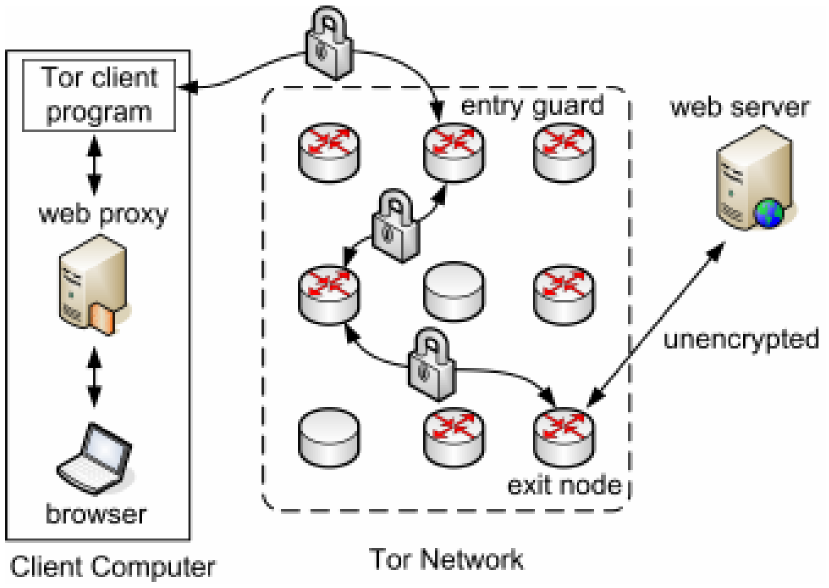


Figure 16 Different components involved when routing through the Tor network. (Adapted from Tor: Overview (2009).)

There are many similarities between the way Tor routes its traffic and circuit switched networks from telecommunications. In fact, the random path chosen by the sender is referred to as the *circuit*. The discussion on path construction so far glossed over one important detail—establishment of secret keys with the Onion Routers along the path without compromising anonymity. The next subsection discusses the manner in which Tor preserves anonymity when creating circuits.

Circuit Creation and Destruction

Circuits are constructed incrementally, one relay at a time. Symmetric keys with each Onion Router (OR) on the circuit are negotiated using the Diffie-Hellman key exchange protocol. Returning to our example from Figure 15, we show the step-by-step process Alice follows to create a circuit between her and Bob.

Alice starts out by sending the `Create Cell` message which contains the first step of the Diffie-Hellman (DH) handshake g^i encrypted with Carol's public key. Carol responds with the second step of DH by sending g^j, along with a hash of the negotiated key $K_c = g^{ij}$. This key K_c is used for all subsequent communication with Carol for this session, and computationally expensive public encryption is not used for the remainder of the session with Carol. This completes the construction of the first segment of the circuit. Alice and Carol refer to this segment of the circuit as C_{ac}.

Next, Alice sends an `Extend Cell` message to Carol. This message contains the address of the next OR (David in this example) and g^k encrypted with David's public key. Carol creates a new Circuit ID C_{cd} and associates C_{ac} with it. The association is known only to Carol; neither Alice nor David is aware of it. David completes the DH handshake initiated by Alice by responding with g^l, along with a hash of the negotiated key $K_d = g^{kl}$ and sends it to Carol who relays the response back to Alice with the `Extended Cell` message containing this information.

With this, building of the second segment is done. Alice and David now share the symmetric key $K_d = g^{kl}$.

This process continues with Alice sending David an `Extend Cell` request, resulting in Alice establishing a symmetric key K_b with Bob.

Once the entire circuit is established two-way communication takes place between the end nodes as described in the Onion Routing section.

There are two points about this construction that are noteworthy: 1) though Alice knows that she is handshaking with David and Bob, they have no idea that it is Alice on the other end, and 2) according to protocol analysis performed by Dingledine et al. (2004) this method of constructing circuits is secure and achieves perfect forward secrecy under the Dolev &Yao (1981) model.

Circuits are torn down at the request of the initiator with the `Destroy` message. Each OR in the circuit that receives a `Destroy` message

- closes all streams on that circuit and
- forwards the `Destroy` message.

There is another mechanism to take down a circuit—the `Relay Truncate` message that is directed at a single OR on a circuit. When an OR receives `Relay Truncate` message from the initiator, it

- sends out a `Destroy` message forward
- responds back with the `Relay Truncated` message to the initiator
- the initiator then sends the `Extend Cell` message to form a modified circuit.

This is also useful when one of the ORs goes down—the neighboring OR can send the `Relay Truncated` message to the initiator.

Attacks on Tor

Tor, and Onion Routing networks in general, are vulnerable to several attacks. In this section, we consider some that are theoretical in nature and some that are very practical.

There is a broad category of attacks called *path selection attacks*. It is important in Tor that the initiator pick the nodes on the circuit so that that the end nodes cannot collude. If they do, the whole circuit can be inferred as the default circuit length is three. The default circuit length value is chosen so that the latency is kept to a minimum. It appears that an immediate fix for this problem is to choose a longer path length. That may not always work; if an adversary-controlled OR cooperates to extend the circuit *only* to other adversary-controlled ORs while refusing to extend the path to any other ORs, then even a longer path does not help. A related attack involves overloading the legitimate ORs to a point where they cannot respond to requests for new circuit construction, introducing a new set of adversary-controlled ORs into the mix, and steering new circuits to choose a path through them.

Next consider *intersection* attacks. These attacks are based on the assumption that ORs that are not continuously present on the network could not have been part of any circuits. Hence the attacker can eliminate them from consideration and narrow the set of ORs that might have participated.

The Tor exit nodes pose a threat to confidentiality. Since anyone can volunteer to run a Tor node, an attacker would have total access to the data that is being routed if the attacker happens to run the exit node of a circuit. Zetter (2007) reports how Dan Egerstad, a security researcher, collected several hundred email account passwords by sniffing on an exit node. He reportedly collected thousands of private e-mail messages sent by foreign embassies and human rights groups around the world. The exit nodes can also carry out an MITM attack by sending back a bogus certificate for the website the initiator wishes to connect. In fact the attack described in the MITM Attack Using Other Means section that netted Marlinspike several hundred email account credentials was mounted from Tor exit nodes.

The most powerful attacks on Tor, and onion routing in general, are statistical attacks based on time measurements and correlations in traffic patterns. These are usually carried out using *congestion* attacks. In these attacks, the adversary monitors the connection between two nodes, creates a path through the network and clogs it to see if that affects the speed of the connection. If one of the nodes is on the path being monitored, the speed should change. In the next subsection, we demonstrate one such practical attack.

Concrete Attack

This section presents a powerful attack that is due to Murdoch & Danezis (2005). It takes advantage of Tor's overly simplistic round-robin policy of relaying cells from the input queues to the output buffer at ORs. The implication of this policy is that a higher load, even due to one extra connection, on a Tor node will result in higher latency of all other connections routed through it. Their attack is particularly powerful because it proves that adversaries with even with modest capabilities can deanonymize Tor users. Our presentation here follows a slight modification to the original attack proposed recently by Evans et al. (2009). The modified attack is effective against the current Tor system with hundreds of ORs. Even though the attack works only for HTTP connections, it is conceptually simple to describe.

In order to describe the attack, we must include a brief description of the scheduling policy each relay implements to forward data from the input queues to the output queue. Tor data is packaged into fixed-size cells (512 bytes), which upon arrival at a relay are buffered before forwarding. Cells from each circuit are queued separately. Each relay simply iterates over all input queues, removes the first cell from every nonempty queue and places it in the output buffer. While this simple round-robin forwarding scheme is fair, it makes the flow pattern through the relay very predictable. As a result, relays become susceptible to congestion attacks.

Three design features of Tor are necessary for this attack to work: 1) the round robin policy at every OR without any addition of random delays, 2) free availability of the addresses of all Tor routers, and 3) no restriction on users from creating paths of arbitrary length.

The attack begins by the attacker running an exit node and attempting to deanonymize the Tor users accessing HTTP servers through the node. For each path that he attempts to deanonymize, he already knows the middle node—he only needs to find out the entry node.

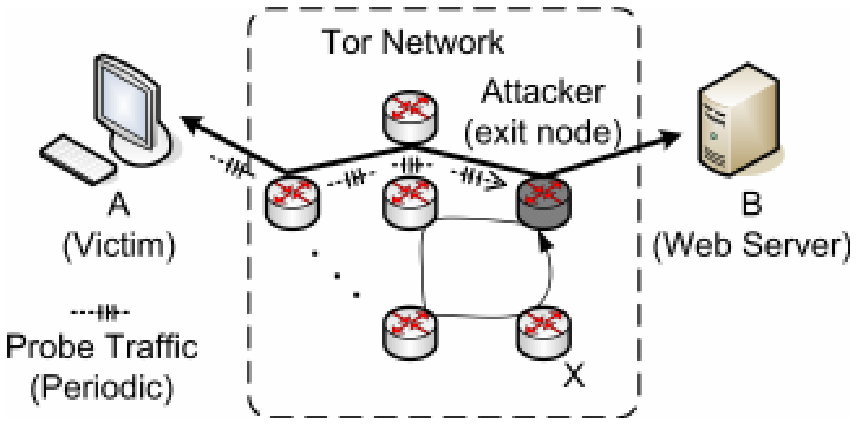


Figure 17 Traffic analysis attack on Tor

For every Tor node X that he suspects to be the entry node of the path between victim A connecting to web site B, he runs the following test. He creates a long circuit that goes through X multiple times and places X under load by clogging the circuit with fake traffic. At the same time, he modifies the HTML response back from the web server B to the client A by inserting a small amount JavaScript code so that the client browser issues a periodic HTTP request to the server. The requests issued by JavaScript are tiny in size, and their sole purpose is to create a steady stream of probe traffic from the client to the attacker under light load conditions. The attacker sends empty responses to these requests which are thrown away by the browser.

With this setup in place, the attacker can determine whether or not X is the entry node of the path in question: if X is not present on the path, the probe traffic should arrive at periodic intervals under a congestion attack through X. But how does the normal traffic interfere with the arrival times of the probe traffic? After all, it is not reasonable to assume that the network is lightly loaded while the attack is taking place. The solution is to establish a baseline for normal traffic load on the circuit before and after the congestion attack.

Newer versions of Tor implement a fix to this attack. The fix involves keeping track of the path lengths and limiting each circuit to at most eight hops long. Unfortunately this is not a satisfactory solution as the attacker can easily defeat this measure by exiting and reentering the network.

Defenses

To defend against the attacks described in the previous section, one could disable cookies, JavaScript, Java, and all plug-ins in the browser. But, these measures result in unacceptable degradation in browsing experience of the end user. Another solution is to use HTTPS. This will prevent sniffing and MITM attacks at the exit node. Also, the JavaScript injection attack fails when HTTPS is used.

The other options include increasing the default path length in Tor at the expense of increasing the latency. This defense has an adverse effect on the responsiveness of the system in general because it creates more traffic on Tor. Simply increasing the default path length from three to four would increase the traffic by 33%. Another solution is to introduce random delays at ORs in place of the simple round robin policy that is currently used. This again increases the latency of the network.

AVAILABILITY

An attack that makes a computer or network resource unavailable is called a *Denial-of-Service* (DoS) attack. When it is a concerted attack by multiple attackers against a single resource, it is termed *Distributed Denial-of-Service (DDoS)* attack. These are generally carried out against high profile targets such as large corporations and financial institutions with intent to disrupt the service provided by them.

A classic DoS example from network security is SYN flooding (Eddy, 2007). To understand this attack, we need to describe the three-way handshake that every client and server must engage in before establishing a TCP connection. In a nutshell, it involves the client initiating a TCP connection request with SYN (synchronize) message. The server acknowledges with SYN-ACK message, which the client acknowledges back with ACK message, completing the handshake.

If the server allocates resources to deal with a new connection request right after receiving a SYN message from the client, then the attacker can exhaust the network resources at the server by flooding the server with SYN requests. Of course, the attacker has no intention of completing any of the initiated handshakes; his only goal is to exhaust server's resource with SYN requests. Once this happens, the server is not in a position to accept any legitimate SYN requests. The attacker has met the objective of disabling the server from establishing any TCP connections with normal clients.

Consider the examples given in the previous sections. During MITM attack, the attacker could simply drop all in-bound and out-bound packets, thus isolating the LAN from the Internet and causing loss of connectivity to all the hosts on the LAN.

Similarly, the adversary can volunteer to run Tor nodes, accept packets, but not relay them. Congestion attacks described in Attacks on Tor section are another form of DoS attack.

As these examples show, it is very easy for an adversary to mount DoS attacks.

KEY MANAGEMENT

The cryptographic techniques discussed in this chapter depend on secure generation, distribution, and management of keys. A number of different approaches have been used for key management, including:

1. Physical key distribution (e.g. couriers)
2. Symmetric key distribution (e.g. Kerberos)
3. Asymmetric key distribution (e.g. PKI certificates)
4. Key agreement protocols (e.g. Diffie-Hellman)
5. Quantum key distribution (also know as. "quantum cryptography")
6. Key elimination (e.g. BBC encoding for jam resistance)

Each of these six approaches has different strengths and weaknesses, so each is typically used in different situation.

Physical Key Distribution

The oldest and simplest method for key distribution is to physically transport the key. This can be slow and cumbersome, but there are several situations where it can be a reasonable way to manage keys.

For example, suppose a small number of banks want to transfer money electronically among them. Security is important, because the ability to modify such messages is equivalent to the ability to counterfeit arbitrarily-large amounts of money. Therefore, the banks might choose to be conservative, using the most thoroughly analyzed cipher available.

By that reasoning, the most conservative cipher would be a symmetric cipher, the Data Encryption Standard (DES). This cipher was approved by the US government three decades ago, and has received more public scrutiny than any other cipher in history. There are no known ways to break it that are significantly faster than a brute force attack of trying all possible keys. The key size of DES is only 56 bits, which allows brute force attacks using modern computers. However, this can problem can be overcome by encrypting the message three times, using two or three different keys. This Triple DES (3DES) gives an effective key size of 112 bits, which cannot be broken by brute force with publically-known algorithms using current technology in a reasonable number of years. Even if someone discovered how to build a large quantum computer, it is not clear that this would allow 3DES to be broken, since the best known algorithm for this, Grover's algorithm, would still require the quantum computer to run for on the order of 2^{56} steps.

Banks commonly use armored courier vehicles to transport cash and other valuables, so it is natural to also use them to distribute 3DES keys. If there are only a handful of major banks in a country that need to communicate this way, then it is possible to establish a separate key for every pair of banks, and distribute these keys by courier. This would be more difficult if new institutions were joining or leaving the network frequently, but banks tend to be stable and new ones are created infrequently.

A system like this can be strengthened by splitting the keys. A bank can protect a key K by generating several random strings of bits $K_1,K_2,\ldots,K_n$, each of which is the same size as K. Then the true key K can be encrypted by XORing it with all of the random keys. The encrypted key and all the random keys are sent by separate armored cars. If all of the cars arrive safely, then the receiver XORs all n+1 keys to obtain K. If any cars are lost, then a new key can be generated, and the process repeated. An attacker can only obtain the key K by stealing the keys from all n+1 armored cars, without the thefts being detected; clearly a difficult task.

Another example is distributing keys to diplomats and spies. In that case, the sender and receiver can meet and physically hand over the key. In this case, the key may be used for only a few short messages, and so it may actually be practical to use a perfect, unbreakable cipher: the *One Time Pad* (OTP).

In the OTP, a message is encrypted by XORing it with the key. The resulting cipher text is decrypted by XORing it with the same key. The sender and receiver both destroy the key after use, so the pad of key material is only used one time. The sender and receiver must both have a large set of key material, at least as long as the combination of all the messages that will ever be sent. The OTP is theoretically unbreakable, if the key is perfectly random, used only once, and kept secret.

The Soviet Union started using this system in the 1930 for diplomats and spies, and continued to use it for decades. Kahn (1996) describes how the keys were printed in small books of many pages which were the size of a postage stamp, or were rolled up into pads the size of a cigarette. One of these tiny books could hold hundreds of characters of key, which would be enough to send a number of short messages. The pads were printed on highly flammable sheets, and the spies carried chemicals that could ignite them quickly, destroying the key and all evidence of its existence.

Although physical key distribution for symmetric keys is sometimes used, it is clearly impractical for most networks. A network of n nodes would require on the order of n^2 separate keys to be transferred securely. Also, it is not clear how the system would work when strangers want to communicate. That is why physical key distribution is much less commonly used than the methods described in the next few sections.

Symmetric Key Distribution

A more efficient way to distribute symmetric keys is by sending them through the network itself, using a trusted third party. This can be done using systems such as Kerberos. Typically, such systems are used for communication between people in a single organization, such as a single company or university.

For example, suppose Alice and Bob are on a network and want to communicate, but have not pre-arranged any shared key between them. If they both know and trust Trent, then they can communicate in the following way.

First, Alice will send a message to Trent using a symmetric key that she and Trent share. This key may have been physically transferred earlier. She will tell Trent that she would like to communicate with Bob. Trent then generates a random key, called a session key, and sends it back to her. Alice's message to Trent and his reply including the new key will both be encrypted with the key Alice and Trent share.

Trent then sends a message to Bob, telling him that Alice would like to communicate with him, and sending him the same session key. This message is encrypted with a key shared by Trent and Bob. After that, Alice and Bob can communicate using the new key, and no further communication with Trent is needed.

With this system, every user has only a single key to manage in the long term: the one shared with Trent. Trent must maintain a list of keys for all users, and must be able to generate new random keys quickly. The simple system described here can be extended by adding various acknowledgement messages, stronger forms of authentication (e.g. message authentication codes), time stamps to avoid replay attacks, and other refinements.

There are several drawbacks to such a system. No two people can ever communicate until they first communicate with Trent. This wastes bandwidth, and also paralyzes the entire system if the Trent server ever goes down or becomes inaccessible. The security of Trent is critical, since an attacker that compromises that one server will be able to eavesdrop on all conversations between all users. Furthermore, there is still the problem of establishing the initial keys shared by Trent and each user. In a company or university, that key can be physically handed to each person when a new employee or student arrives, or physically installed on their computer by a trusted employee. But if this were scaled up to cover all users of the Internet, that could become difficult.

Asymmetric Key Distribution

The last two methods were appropriate for any key, especially symmetric keys. In the case of asymmetric keys, other methods become possible. These methods are typically much easier to scale to large networks, including the entire Internet.

The Background section on cryptography described asymmetric algorithms such as RSA which have a public and private key. This makes key distribution far easier, because the private key doesn't need to be distributed (it is known by only one person), and the public key doesn't need to be kept secret (it's known by everyone).

Suppose Alice wants to be able to receive encrypted messages and send signed messages. To do so, she can generate a public/private key pair. She will keep the private key secret. She could publicize the public key, perhaps by posting it on her website or emailing to other people.

If Bob then wants to communicate with Alice, he would need her public key in order to encrypt his message. But even if he receives her public key from her website or email, he has a problem. How does he know that key truly belongs to Alice? If the key actually belonged to the eavesdropper Eve, and if Bob used that key to encrypt his message to Alice, then Eve could easily decrypt and read the message, re-encrypt it with Alice's true public key, and send it on to Alice. This would allow Eve to perform an MITM attack, as described in the Trust, Certificates, Man-In-The-Middle Attack section.

So for asymmetric keys, the problem is not key distribution, but key authentication. This is usually done by certificates, as mentioned before. There are a number of ways that certificates can be handled.

Programs like Pretty Good Privacy (PGP) work on the principle of a "web of trust". Alice might have several friends sign certificates for her public key. If any of her friends are also friends of Bob, then he might trust them, and accept the key as genuine. Or he might be more cautious, and require at least *n* signatures by people he trusts before he will believe a key is real. Or he might be less cautious and accept Carol's signature on Alice's certificate, because even though he doesn't know Carol personally, he does see that Carol's key is signed by someone he does know. Hence the "web" of trust.

This approach raises an interesting question. Is trust actually transitive? If Bob trusts his friends, does that mean he should trust strangers who have simply had their identity verified by his friends? It also raises the question of what trust means. If Bob trusts his friends to be honest, does that also mean he trusts them to be experts in recognizing fake drivers' licenses? Does it mean he trusts that their computers will never become infected with malware that will sign certificates in their name?

For these reasons, Certificate Authorities (CAs) have become a much more common way of authenticating public keys. If a trusted company signs Alice's certificate, then presumably Bob can trust that the public key truly belongs to Alice.

However, there are still a number of questions that are raised. A modern computer typically comes with a number of different CAs pre-installed as trusted. If even one of them is compromised, then Eve will be able to use it to create false certificates in Alice's name, and launch MITM attacks on her.

Even if the CA is trustworthy, there are still questions about what the certificate means. Some certificates merely say that a given public key is associated with whoever controls a given email account. Some might be more thorough, verifying that the person's claimed name ("Alice

Smith") actually appeared on something that looked like a driver's license or birth certificate. Theoretically, certificates could even include verified DNA measurements to prove identity, but that hasn't been done much in practice.

Key Agreement Protocols

Certificates are a powerful mechanism, but they are not always convenient. Most Internet users do not currently have a certificate. Key agreement protocols can be used to achieve some of the same benefits, with less work on the user's part. A common example of this is the Diffie-Hellman key exchange protocol described in the Background section.

Suppose Alice and Bob want to communicate securely over the Internet, but have never met, have no certificates, and have no trusted friends in common. Clearly, no matter what they do, they will not be able to protect themselves from an active attack by Eve, who cuts their communication wires, and inserts herself in between. She will be able to launch MITM attacks without detection.

However, this can actually be difficult for Eve in some situations. She must be able to not only read the traffic flowing between Alice and Bob, but actively intercept those messages and prevent them from getting through. If the network is the Internet, then Eve can't just passively eavesdrop; she must actively control servers or routers to stop or modify certain packets.

Therefore, Alice and Bob may decide that there is some benefit to having a protocol that protects them from passive eavesdropping, even if it doesn't protect them from active MITM attacks. Fortunately, Diffie-Hellman key exchange can do that. It generates a session key that will securely encrypt all messages during the session. As long as Eve is only a passive eavesdropper, she will not get the session key.

Protocols like Diffie-Hellman can be further strengthened if the messages are signed. This is implemented in SSL. Suppose Alice has no certificate and is a customer, and Bob is an online store and has a certificate signed by a CA that Alice trusts. Then theoretically, Alice should be able to perform Diffie-Hellman with Bob (who digitally signs each message), and be secure even from active attacks by Eve. The MITM Attack on SSL Using Bogus Certificates section gives an example of how this has failed in practice because of flaws in how the system is designed and implemented.

Even when such flaws are fixed, there can still be problems. On current browsers, if Alice goes to a website with an invalid certificate, there will typically be a popup dialog asking her if that is OK. Most users have been trained by long experience to automatically click OK on all such popup boxes. So security can be compromised, even when the cryptography and protocols are flawless.

Quantum Key Distribution

The perfect security of a One Time Pad is very appealing. The only problem is key distribution. There is a form of key distribution based on quantum mechanics that is also perfectly secure. This is called *Quantum Key Distribution* (QKD) and is also known by the name of *quantum cryptography.*

If Alice are in separate buildings that have good security, and if they correctly implement QKD systems, then they can run a fiber optic cable between their buildings and communicate in a way that is perfectly secure. Eve can cut the fiber and manipulate it any way she wants, she will never be able to read or modify any messages, assuming the QKD was implemented perfectly.

The system takes advantage of an interesting property of quantum mechanics. It is possible to force a photon of light to be polarized in one of two orthogonal directions (vertically or horizontally) or in one of two diagonal directions (tilted left 45 degrees or tilted right). A person receiving that photon can measure it in one of two ways: orthogonally or diagonally.

If the photon was polarized vertically or horizontally, then an orthogonal measurement will determine which way it was polarized. If the photon was polarized diagonally to the left or right, then a diagonal measurement will determine which. However, if the photon was polarized orthogonally (vertically or horizontally) and is measured diagonally, the result will be random: there is an equal chance of getting the result “tilted left” or “tilted right”. Similarly, if it was polarized with a tilt of left or right, and it’s measured orthogonally, then the result is random (“vertical” or “horizontal”). Finally, a measurement of either type will destroy the photon’s polarization. So the receiver gets only one chance to measure the photon before all information is lost.

The core idea in QKD is simple. Alice will send Bob a sequence of photons. Alice will randomly choose one of the 4 polarizations for each one. Bob will randomly choose one of two measurements for each photon: orthogonal or diagonal. After many photons have been sent and measured, Alice and Bob will communicate on an ordinary channel, such as through the Internet.

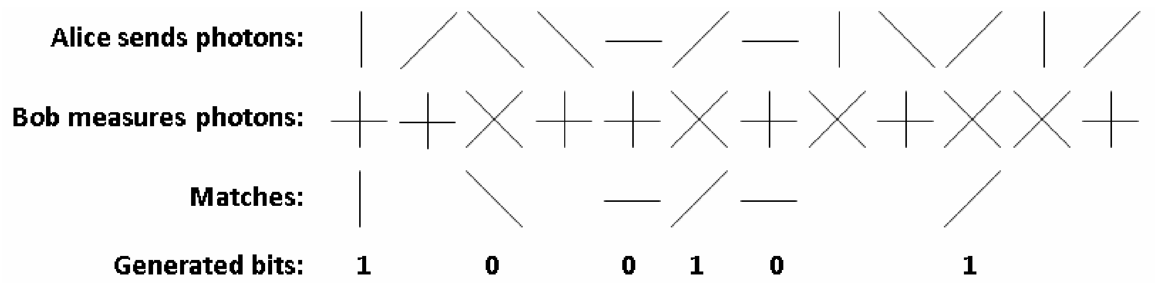


Figure 18 Quantum key distribution

On that ordinary channel, Alice will tell Bob whether each photon she sent was orthogonal or diagonal. Bob will tell Alice which measurement he performed on each photon. For about half the photons, Alice’s choice will be different from his measurement, so the two of them will ignore those. For the rest, each photon will have transmitted one random bit. If Alice and Bob both happened to chose orthogonal for a given photon, then Bob’s measurement will reveal a result that is identical to what Alice chose, either vertical or horizontal. This generates a bit, say 0 for horizontal and 1 for vertical, that becomes a shared secret between Alice and Bob. Given a large number of such bits, Alice and Bob can use them as a One Time Pad to encrypt other messages.

If Eve simply eavesdrops, she’ll learn nothing useful. She’ll know which photons they agreed on, and which were orthogonal and which were diagonal. But she won’t know whether they were vertical/horizontal/tilted left/tilted right. So she won’t know any bits of the pad.

If Eve cuts the fiber, she can intercept all the photons, measure them, and then send them on. But if there is a photon that Alice and Bob happen to measure the same way, but which Eve happens to measure differently, Eve will learn nothing about that bit, and the bit received by Bob will be random, so Alice and Bob can erroneously end up with a different bit in that position in the shared pad that is created.

The description above is simplified. There are a number of details involved in condensing the pad to deal with small errors introduced by Eve, and verifying that the open communication about measurements is not corrupted by Eve. But the description above has the essential elements of quantum key distribution.

This approach has the advantage that it is theoretically perfect. Even an infinite number of computers could not break the resulting cipher in infinite time. There are several drawbacks to it. By its very nature, it requires a direct connection from Alice to Bob. It can’t be done over the existing Internet. Alice and Bob must lay fiber between them, or have a direct line of sight. Fortunately, it has been demonstrated over fairly long distances (hundreds of kilometers). In fact, it should be possible to perform such communication from a ground station to a satellite, and then have the satellite perform the reverse back to the ground. The sender and receiver hardware must be very sensitive to make it work. One might ask whether any given implementation is perfect, or whether some small flaw leaks information that could be useful to an attacker. There is no simple way to test whether the implementation is perfect. So although it is perfect in theory, it is an open question how secure it would be in practice.

Key Elimination

The best way to manage keys is to eliminate the need for them in the first place. This is especially important in areas where there are no asymmetric algorithms, and only symmetric algorithms exist. Perhaps the best example of this is in the assurance of availability for wireless networks.

When combating DoS attacks on wireless networks, one important factor is jam resistance. It should be difficult for an attacker to jam the communication by broadcasting radio frequency noise or other wireless signals. Jam resistant methods have been known for many decades. All of them are based on the use of a symmetric key. There is no equivalent of asymmetric keys for jam resistance.

For this reason, key management can be a problem in large wireless networks that are intended to be resistant to jamming. The problem is even worse for Mobile Ad Hoc Networks (MANETs), which involve many radios that are in motion, and that constantly form new connections to create a constantly-changing network. In that case, the secret key would have to be loaded into every node that might ever connect to the network. For very large MANETs, this can pose a serious problem. If an attacker captures even one of the radios and extracts the key from it, that key could be used to jam the entire network. Therefore the key should be changed frequently. But that can be a challenge when there is a large, distributed network of radios.

This key management problem was solved for the first time in 2007 with the development of concurrent codes and the BBC algorithm (Baird et. al., 2007; Bahn et. al., 2008). With these new techniques, it was finally possible to have jam resistance without any key or secret at all. That eliminated the need for keys for jam resistance, and so eliminated the need to manage the symmetric keys. Messages traversing such a network might still have encryption and digital signatures, to ensure confidentiality and authenticity, but those could be achieved with symmetric keys. As seen in the previous sections, key management tends to be much easier for symmetric keys than for asymmetric.

The next section describes this new algorithm in more detail.

WIRELESS AVAILABILITY—JAM RESISTANCE

Availability is one of the three goals of network security as mentioned in the Introduction. For a wireless network, this includes resistance to jamming. An attacker can launch a DoS attack by broadcasting radio frequency noise, with a large amount of power. This can overwhelm the legitimate signal, and prevent wireless messages from being received. In many cases, the attacker can accomplish the same thing without using much power at all, by crafting special signals designed to disrupt the particular form of wireless communication being used.

The attacker would usually prefer a low-power attack. If an attack requires megawatts of energy, that prevents the attacker from using small, battery-powered devices. It is also much easier for the authorities to track down the attacker (or the attacking device) and shut it down. The attacker would prefer to use cheap, low-power devices to do the jamming. These low-power attacks are foiled by jam resistant systems.

In traditional jam resistance, some form of spread spectrum radio communication is used. In order to be jam resistant, it must create a communication channel that is a function of a secret key, shared by the sender and receiver.

For example, in a frequency hopping system, the sender broadcasts a signal at a particular frequency. Then, it jumps to a new frequency. These jumps occur many times per second. The key is used to choose the sequence of frequencies. The legitimate receiver knows the secret key, and so is able to listen to the correct frequencies in the correct sequence, in order to receive the message. If the attacker does not know the key, then the attacker cannot guess which frequency sequence will be used. So the attacker must jam all of the frequencies (or a large fraction of them), which requires a large amount of power. However, if the attacker discovers the key, then the jammer can jam just the frequency in use at each moment, jumping in synchrony with the sender and receiver, and can accomplish this jamming with very little power.

Another approach is pulse-based systems. The following is a simplified version of it. If a message is to be sent during a one-second period, the sender divides that period into many small time slices. The key is used to select a small number of time slices, perhaps one out of every thousand. For the n^{th} chosen time slice, if the n^{th} bit of the message is a 0, then the sender is silent. If it is a 1, then the sender broadcasts a short, powerful burst of radio frequency noise that spans a very broad part of the spectrum. A receiver who knows the secret key will be able to calculate which time slices to observe, and will easily recover the message. An attacker who knows the secret key will be able to broadcast pulses in all of the chosen time slices, which the receiver will interpret as the message "111111…", and so the jammer is successful with very little power consumption. Conversely, if the attacker does not know the key, then the attacker must fill every time slice with a pulse (or a large fraction of them). That requires 1000 times as much power, in this example.

With the development of concurrent codes and the BBC algorithm (Baird et. al., 2007), it is now possible to send jam-resistant messages without any secret key at all. The algorithms to encode and decode BBC are given in Figures 19 and 21.

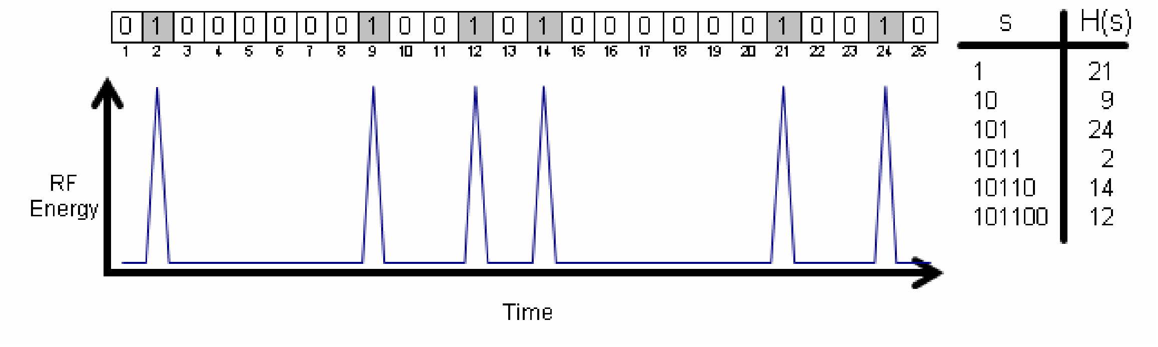
Figure 20 gives an example of BBC encoding the message "1011" without using a key. The encoding is simple. First, several zeros are appended to the message to act as checksum bits. In this example, two zeros are appended, yielding the string "101100". Next, all possible prefixes of that string are found. These are shown in the table in the first column, under the "S". The period of time used for the message is divided up into time slices. In this example, the period is divided up into 25 slices, which are numbered from 1 to 25. Finally, a hash function is used to convert each prefix string into a number from 1 to 25. A hash function is simply a function that scrambles its inputs in a random-looking way. A standard hash function such as SHA-1 or MD5 could be used. It doesn't have to be secret. It is assumed that the sender, receiver, and attacker all know the hash function. In this example the hash function is given in the table. Each of the strings in the "S" column maps to the number in the "H(S)" column. Finally, the sender broadcasts a strong pulse of radio noise in each time slot chosen by the hash function.

Algorithm: BBCencode(M)

This function broadcasts an m-bit message M[1 . . .m], adding k checksum bits to the end of the message. H is a hash function. The definition of H and the values of m and k are public (not secret).

```
Append k zero bits to the end of M
for  i ← 1 … m + k do
        Send a pulse at the time given by H(M[1 . . . i])
end for
```

Figure 19 Algorithm for BBC encoding



S	H(s)
1	21
10	9
101	24
1011	2
10110	14
101100	12

Figure 20 BBC encoding of the message "1011"

Algorithm: BBCdecode(n)

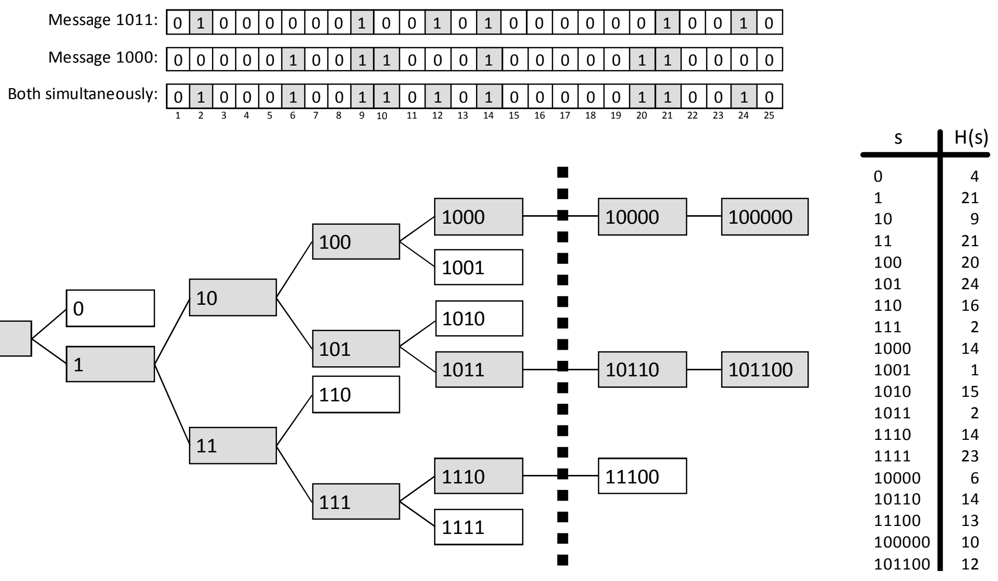
This recursive function can be used to decode all the messages found in a given packet by calling BBCdecode(1). There must be a global M[1 . . .m + k] which is a string of m + k bits. The number of bits in a message is m, and the number of checksum zeros appended to the message is k. H is a hash function. The definition of H and values of m and k are public (not secret).

```
if n = m + k + 1
        output "One of the messages is:" M[1 . . .m]
else
        if n > m
            limit←0
        else
            limit←1
        end if
        for  i ← 0 … limit do
            M[n] ←i
            if there was a pulse at time H(M[1 . . . n])
                    BBCdecode(n + 1)
            end if
        end for
end if
```

Figure 21 Algorithm for BBC decoding

This is a simple algorithm. Any message can be quickly converted to a sequence of pulses. There are no secrets involved. An attacker cannot jam this by broadcasting just a few pulses. If an attacker can broadcast a pulse in every single time slot, then of course it would be jammed, but that would require far more energy. This system has been mathematically proven to be secure, in the sense that no attacker can jam it without using far more energy than the legitimate sender (Baird et. al., 2008).

Figure 22 shows the decoding of two messages sent at the same time. The receiver observes the bitwise OR of the two packets. The timing of pulses observed by the receiver is labeled "Both simultaneously", with a 1 for each pulse detected, and a 0 for radio silence. The receiver must decode those 25 bits to recover the two messages that were sent: "1000" and "1011".



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Message 1011:	0	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	1	0	0	1	0
Message 1000:	0	0	0	0	0	1	0	0	1	1	0	0	0	1	0	0	0	0	0	1	1	0	0	0	0
Both simultaneously:	0	1	0	0	0	1	0	0	1	1	0	1	0	1	0	0	0	0	0	1	1	0	0	1	0

s	H(s)
0	4
1	21
10	9
11	21
100	20
101	24
110	16
111	2
1000	14
1001	1
1010	15
1011	2
1110	14
1111	23
10000	6
10110	14
11100	13
100000	10
101100	12

Figure 22 BBC decoding of two messages sent simultaneously

The packet is decoded by following a tree of possible prefixes. The receiver knows that whatever messages were sent, each one must have started with either a 0 or 1. If a message started with 0, then the shortest prefix would have been simply "0", and there would have been a pulse at time H(0). Similarly, if any messages started with 1, there would be a pulse at time H(1). In this example, H(0) = 4 and H(1) = 21. Note that there is a pulse at time 21, but not at time 4. That tells the receiver that at least one message was sent that started with 1, but none were sent that started with 0. In the tree, the box labeled "0" is white, indicating that there was no pulse at time H(0), and the box for "1" is gray, indicating that there was a pulse at time H(1).

At this point, the first bit of the message has been decoded. It is a 1. The receiver knows that whatever messages were sent that started with 1, the second bit must be 0 or 1. Therefore the first two bits together must be 10 or 11. The receiver therefore checks at time H(10) = 9 and H(11) = 21 and notices a pulse in both locations. Therefore, the receiver concludes there are actually two messages being received, and continues exploring the tree down both of those branches.

When the receiver finishes the fourth bit, the complete messages are now obtained. In this example, the receiver obtained the messages 1000, 1011, and 1110. That last message is actually spurious: it only appeared to exist because each of its prefixes happened to hash to the same location as some prefix of one of the legitimate messages. However, the receiver doesn't stop there. There were several checksum zeros appended to the end of the message (2 of them, in this example). The receiver adds on each of those zeros, and checks to ensure there are pulses at those locations as well. The spurious message is caught this way, and only the two legitimate messages survive.

Of course, an attacker can always read the messages and send additional messages. If this is undesirable, the sender would encrypt and sign messages before doing the BBC encoding. And the receiver would check the signature and decrypt them after doing the BBC decoding. The encryption and signatures can be done with asymmetric keys. The BBC encoding eliminates the symmetric keys required by traditional jam resistance. This greatly simplifies the key distribution and management problem, as discussed in the previous section.

OPEN PROBLEMS

We conclude this chapter with some open problems in network security. Defending anonymizing networks like Tor against the attacks described in this chapter, while minimizing latency and keeping the system usable is a challenging open problem. Unless the system is responsive and useable, not enough users would use it. If there are not enough users on the system, it is not possible to achieve a high degree of anonymity.

Network security is a broad area with issues ranging from identity theft, network intrusions, to session hijacking. The case studies presented here show that without strong mathematical techniques, it is impossible to build practical systems that are secure. The attacks we presented show adversaries commonly side-step cryptographic protections, thus proving that mathematical foundations are necessary but not sufficient. One area that is in need of formal methods is network intrusion detection. The goal is to build a system that can automatically identify attempts by an intruder to compromise the confidentiality, integrity or availability of a resource over the network. The current systems fall significantly short of achieving this goal. The alarm volume from sensors deployed at different client sites can be in the millions at managed security service centers, out of which over 99% are false alarms (Treinen & Thurimella, 2006). Finding true alarms from false ones in this environment is a great challenge. Past attempts using statistical modeling, analysis of traffic patterns, and enumeration of possible attack paths have all met with limited success. We believe fundamentally new techniques based on solid mathematical foundations are required.

Acknowledgements

The authors are grateful to the referee, Christian Grothoff, Will Mitchell, and James Treinen for their helpful comments. Their feedback helped improve the presentation significantly. Dr. Thurimella would also like to gratefully acknowledge the partial support he received from the NSF under Grant No. DUE – 0416969. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect those of the NSF.

REFERENCES

Bahn, W. L., Baird III, L. C., & Collins, M. D. (2008). *Jam resistant communications without shared secrets*, in Proceedings of the 3rd International Conference on Information Warfare and Security, 37-44, Omaha, Nebraska, 2008.

Baird III, L. C., Bahn, W. L., & Collins, M. D. (2007). *Jam-Resistant Communication Without Shared Secrets Through the Use of Concurrent Codes*, Technical Report, U. S. Air Force Academy, USAFA-TR-2007-01, Feb 14.

Baird III, L. C., & Bahn, W. L. (2008). *Security Analysis of BBC Coding*, Technical Report, U. S. Air Force Academy, Academy Center for Cyberspace Research, USAFA-TR-2008-ACCR-01, Dec 8

Certificate chain verification. (2009). Retrieved March 15, 2009, from http://publib.boulder.ibm.com/infocenter/tpfhelp/current/index.jsp?topic=/com.ibm.ztpf-ztpfdf.doc_put.cur/gtps5/s5vctch.html

Chaum., D. (1981). *Untraceable electronic mail, return addresses, and digital pseudonyms*. Communications of the ACM, 24(2):84–88.

Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., & Polk, W. (2008). *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile,* RFC 5280, at http://tools.ietf.org/html/rfc5280

Daemen, J. and Rijmen, V. (2002). *The Design of Rijndael: AES-The Advanced Encryption Standard.* Springer-Verlag Berlin Heidelberg, New York.

Data Encryption Standard (DES). Federal Information Processing Standards Publication (1999). Retrieved May 29, 2009, from http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf

Dierks, T., & Allen, C. (1999). *The TLS Protocol*. RFC 2246, at http://www.ietf.org/rfc/rfc2246.txt

Diffie, W., & Hellman, M. E. (1976). *New Directions in Cryptography*, IEEE Transactions on Information Theory, IT-22(6), 644–654.

Diffie-Hellman key exchange, Retrieved March 15, 2009, from http://en.wikipedia.org/wiki/Diffie-Hellman

Dingledine, R., Mathewson, N., & Syverson, P. (2004). *Tor: The second-generation onion router*. In Proceedings of the 13th USENIX Security Symposium, 303–320.

Dolev, D., & Yao, A.C. (1981). *On the security of public key protocols*. In Proceedings of the IEEE 22nd Annual Symposium on Foundations of Computer Science, 350-357.

Eddy, W. (2007). *TCP SYN Flooding Attacks and Common Mitigations,* RFC 4987, at http://www.ietf.org/rfc/rfc4987.txt

ElGamal, T. (1985). *A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Transactions on Information Theory, IT-31(4), 469–472.

Evans, N.S., Dingledine, R., & Grothoff, C. (2009). *A Practical Congestion Attack on Tor Using Long Paths*. To be presented at the 18th USENIX Security Symposium, Montreal, Canada.

Goldschlag, D. M., Reed, M. G., &. Syverson, P. F. (1999). *Onion routing.* Communications of the ACM, 42(2), 39–41.

Kahn, D. (1996). *The Code-Breakers: The Comprehensive History of Secret Communication from Ancient Times to the Internet*. New York: Scribner, 650-664.

Marlinspike, M. (2009). *New Techniques for Defeating SSL/TLS.* Presented at Black Hat DC Briefings 2009, Crystal City, USA, Feb 16-19, 2009. Slides are at http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf

Murdoch, S.J., & Danezis, G. (2005). *Low-Cost Traffic Analysis of Tor*. In Proceedings of the 2005 IEEE Symposium on Security and Privacy, 183-195, Washington DC.

Public Key Infrastructure, Retrieved March 15, 2009, from http://en.wikipedia.org/wiki/Public_key_infrastructure

Publius Censorship Resistant Publishing System, Retrieved March 16, 2009, from http://cs.nyu.edu/~waldman/publius/

Rivest, R., Shamir, A., Adleman, L., (1978). *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems.* Communications of the ACM 21 (2), 120–126.

The SSL Handshake, Retrieved March 15, 2009, from http://publib.boulder.ibm.com/infocenter/tivihelp/v2r1/index.jsp?topic=/com.ibm.itame2.doc_5.1/ss7aumst18.htm

Stamp, M. (2006). *Information Security: Principles and Practice.* Hoboken, NJ: John Wiley&Sons.

Syverson, P., Goldschlag, D., & Reed, M. (1997). *Anonymous Connections and Onion Routing*, in Proceedings of the IEEE Symposium on Security and Privacy, 44-54.

Tews, E., Weinmann R. P., & Pyshkin, A. (2007). *Breaking 104 bit WEP in less than 60 seconds*. Cryptology ePrint Archive, no. 2007/120, at http://eprint.iacr.org

Tor: Overview, Retrieved March 16, 2009, from http://www.torproject.org/overview.html.en

Treinen, J. J., & Thurimella, R. (2006). *A Framework for the Application of Association Rule Mining in Large Intrusion Detection Infrastructures*. In Proceedings of Recent Advances in Intrusion Detection (RAID), 1-18.

Wagner, R. (2001). *Address Resolution Protocol Spoofing and Man-in-the-Middle Attacks*. http://www.sans.org/rr/whitepapers/threats/474.php

Zetter, K. *Rogue Nodes Turn Tor Anonymizer into Eavesdropper's Paradise*, http://www.wired.com/politics/security/news/2007/09/embassy_hacks

Zimmermann, H. (1980). *OSI Reference Model—The ISO Model of Architecture for Open Systems Interconnection.* IEEE Transactions on Communications, 28(4), 425-432.