CPSC 467: Cryptography and Computer Security

Michael J. Fischer

Lecture 23
November 19, 2014

Elliptic Curve ElGamal Revisited

A Better Elliptic Curve ElGamal Algorithm

Lecture 12 slide 45 presented an elliptic curve cryptosystem based on ElGamal encryption.

A difficulty with the method was how to encode a message value $m \in \mathbf{Z}_p^*$ as a point on the elliptic curve.

Koblitz's method is one approach, but it has the possibility of failing on some message values.

Below is an improved algorithm (from Stinson's book) based on the idea of "blinding" that allows any message in $\mathbf{Z}_p^*$ to be encrypted.

EC ElGamal (improved version)

1. Alice wants to send a message $m \in \mathbf{Z}_p^*$ to Bob.
2. Bob chooses an elliptic curve E mod p. He chooses a point α on E and a secret integer a. He computes $\beta = a \times \alpha$.
3. The points α and β are made public, while a is kept secret.
4. Alice chooses a random k and computes $\gamma = k \times \beta$. She then computes $Y_1 = k \times \alpha$ and $Y_2 = mx_0 \bmod p$, where x_0 is the x-coordinate of γ. She sends the pair (Y_1, Y_2) to Bob.
5. Bob decrypts by calculating $\gamma = a \times Y_1$, letting x_0 be the x-coordinate of γ, and then calculating $m = Y_2 x_0^{-1} \bmod p$.

The green formulas show where this algorithm differs from the version in lecture 12.

Use of blinding

Both versions compute a point $\gamma = k \times \beta$ for a randomly chosen k. In both versions, γ is used as a blinding factor. The difference is how it is used.

In the algorithm from lecture 12, the point M, chosen to represent the real message m, is blinded by adding the point γ to it, giving $Y_2 = M + \gamma$.

In the improved algorithm, the message m is directly blinded with x_0, which is the x-coordinate of the point γ. This gives $Y_2 = mx_0 \pmod{p}$. Note that Y_2 is now a number in $\mathbf{Z}_p^*$, not a point on the elliptic curve.

Bitcoins

Bitcoins in the news

Bitcoins are frequently in the news these days. Depending on who is talking, they:

- Are the transaction medium of the future.
- Are a threat to national security.
- Are of use primarily to drug dealers and criminals.
- Are a good investment.
- Are the analog of cash on the internet.
- Are secure because they don't rely on trust in a government.
- Are a kind of Ponzi scheme that will sooner or later collapse.

How to understand bitcoins

To evaluate these claims, one must first understand how they work.

- Physically, a bitcoin is an entry in a distributed database called the *block chain*.
- The database is a linked list of transaction *blocks*, each containing a set of transactions.
- A bitcoin is identified by an *address*, which is a public key. The owner holds the corresponding private key in his *bitcoin wallet*.
- From a user's perspective, bitcoin is like a mobile app that provides a personal bitcoin wallet.

Bitcoin database

- The database is created and managed by a large number of *miners* operating in a consensus network.
- The consensus network is claimed to be the first decentralized peer-to-peer payment system that is powered by its users with no central authority or middlemen.

Bitcoin transactions

- A transaction takes one or more bitcoins as inputs and produces one or more new bitcoins as outputs.
- The total value of the input bitcoins equals the total value of the output bitcoins.
- The transaction specifies the address(es) at which the output bitcoin(s) are stored.
- The parties involved in a bitcoin transaction work together to create the transaction record. The owners of the input bitcoins must all sign the transaction. The receiving parties must create addresses for the new coins.

Committing a transaction

Once a transaction record has been created, it must be entered into the database.

- The transaction creators broadcast the transaction to all miners.
- Each miner enters the transaction into a list of pending transactions.
- Every now and then, a miner incorporates all valid pending transactions into a new block and adds the new block to the block chain.
- The new block is broadcast all other miners, who then add it to their copies of the old block chain.

Reaching consensus

A transaction can only be committed to a new block if the source bitcoins have not already been spent.

This is how bitcoins cope with the problem of double spending.

Because of decentralization, it is possible for the same bitcoins to be used in multiple transactions and those conflicting transactions sent to multiple miners.

It is also possible for two miners to extend the block chain in inconsistent ways.

Bitcoin contains a number of mechanisms designed to make such conflicts rare events and to allow for consensus to be reached in case they co occur.

Puzzles

To add a new block to the block chain, a miner must first solve a time-consuming puzzle.

The difficulty of the puzzles is adjusted dynamically so that among all the miners, it takes approximately 10 minutes for the puzzle to be solved.

The first miner to solve the puzzle extends the block chain and sends the new block to the other miners.

A miner who receives a cetificate block chain that is longer than the one he is currently working on discards is current chain and accepts the new one.

In this way, most miners agree most of the time on what is the current block chain.

SHA-256 puzzles

The puzzle that bitcoin uses is based on the SHA-256 hash function.

Each block chain has an associated hash value.

The puzzle is to find a nonce which, when hashed together with the old hash value, yields an output that begins with a specified number of 0's. The more 0's required, the harder the problem.

Bitcoin economics

Whenever a miner solves a puzzle and extends the block chain, a new bitcoin is created as a bonus and inserted into the block at the miner's address.

- The size of the bonus is algorithmically determined.
- It decreases over time and eventually goes to zero.
- These bonuses are a major incentive for the miners.
- As bonuses get smaller and mining gets more expensive, miners will have to turn to transaction fees to pay their costs.

When is my transaction fully committed?

The answer is, “never”.

A dishonest miner can go back in time to an earlier block and try to extend the block chain in a new direction, perhaps one that alters or excludes an earlier transaction.

To do so, he must solve a sequence of puzzles before any other miner solves the “curent” puzzle.

The further back in time, the more unlikely it is that the dishonest miner can succeed.

So a given transaction becomes more and more secure as more and more new blocks are successfully added to the block chain.

Trust

Trust depends on no small group of miners having the majority of computational power.

The bonus reward structure provided the initial incentives for many miners to enter the competition.

As mining becomes more expensive and less lucrative, one can expect the number of miners to decrease, possibly leading to a total collapse of the system.

Anonymity

Bitcoin transactions are often said to be anonymous.

It's true that they contain only a public key, not the user's personal information.

However, if the key can be linked to an individual, the anonymity is lose.

There are many ways for such linking to take place, e.g., by confiscating the user's wallet, or by monitoring a user's transaction in progress.

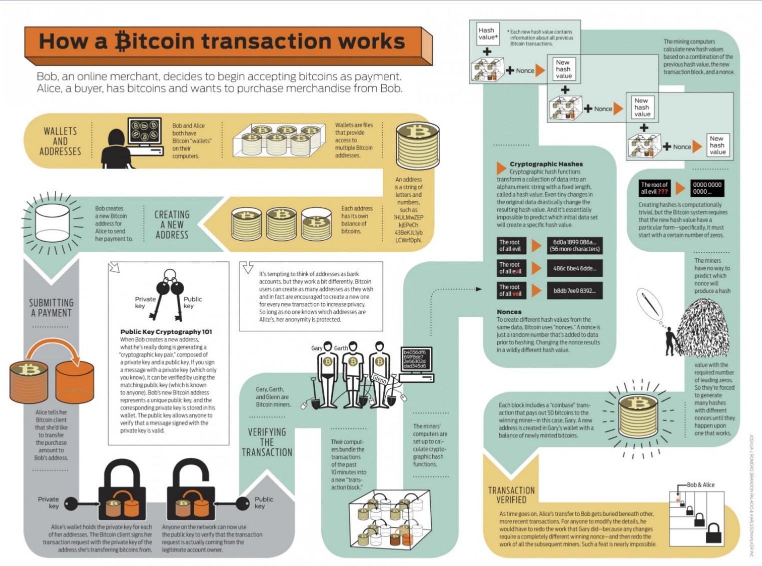
Summary of the transfer protocol

Here's how Alice transfers a Bitcoin to Bob:

1. Alice creates and signs a transaction request giving the coin to Bob.
2. The transaction is then broadcast to all of the miners.
3. Each miner first verifies the validity of the transaction by using its current most-recent copy of the database.
4. If valid, the miner attempts to create a new certified database incorporating the new transaction (along possibly with others) into the current database.
5. To certify a database requires solving a computationally-intensive puzzle.

Outline of the transfer protocol (cont.)

6. The puzzle consists of finding a nonce y such that the SHA-256 hash of the database together with y yields a hash value beginning with a long string of 0's.
7. A successful miner broadcasts the new database to all other miners.
8. Each miner upon receiving a new certified database discards all older ones and begins working with the newer one.
9. The system never reaches consensus, but the probability of a certified database being discarded in favor of another decreases exponentially over time.



How a ₿itcoin transaction works

Bob, an online merchant, decides to begin accepting bitcoins as payment. Alice, a buyer, has bitcoins and wants to purchase merchandise from Bob.

WALLETS AND ADDRESSES

Bob and Alice both have Bitcoin "wallets" on their computers.

Wallets are files that provide access to multiple Bitcoin addresses.

An address is a string of letters and numbers, such as 1HULMwZEPkjEPeCh43BeKJL1ybLCWrfDpN.

Each address has its own balance of bitcoins.

CREATING A NEW ADDRESS

Bob creates a new Bitcoin address for Alice to send her payment to.

Public Key Cryptography 101

When Bob creates a new address, what he's really doing is generating a "cryptographic key pair," composed of a private key and a public key. If you sign a message with a private key (which only you know), it can be verified by using the matching public key (which is known to anyone). Bob's new Bitcoin address represents a unique public key, and the corresponding private key is stored in his wallet. The public key allows anyone to verify that a message signed with the private key is valid.

It's tempting to think of addresses as bank accounts, but they work a bit differently. Bitcoin users can create as many addresses as they wish and in fact are encouraged to create a new one for every new transaction to increase privacy. So long as no one knows which addresses are Alice's, her anonymity is protected.

SUBMITTING A PAYMENT

Alice tells her Bitcoin client that she'd like to transfer the purchase amount to Bob's address.

Alice's wallet holds the private key for each of her addresses. The Bitcoin client signs her transaction request with the private key of the address she's transferring bitcoins from.

Anyone on the network can now use the public key to verify that the transaction request is actually coming from the legitimate account owner.

VERIFYING THE TRANSACTION

Gary, Garth, and Glenn are Bitcoin miners.

Their computers bundle the transactions of the past 10 minutes into a new "transaction block."

The miners' computers are set up to calculate cryptographic hash functions.

Cryptographic Hashes

Cryptographic hash functions transform a collection of data into an alphanumeric string with a fixed length, called a hash value. Even tiny changes in the original data drastically change the resulting hash value. And it's essentially impossible to predict which initial data set will create a specific hash value.

The root of all evil	6d0a 1899 086a... (56 more characters)
The root of all euil	486c 6be4 6dde...
The root of all veil	b8db 7ee9 8392...

Nonces

To create different hash values from the same data, Bitcoin uses "nonces." A nonce is just a random number that's added to data prior to hashing. Changing the nonce results in a wildly different hash value.

* Each new hash value contains information about all previous Bitcoin transactions.

The mining computers calculate new hash values based on a combination of the previous hash value, the new transaction block, and a nonce.

The root of all evil ???	0000 0000 0000 ...

Creating hashes is computationally trivial, but the Bitcoin system requires that the new hash value have a particular form—specifically, it must start with a certain number of zeros.

The miners have no way to predict which nonce will produce a hash value with the required number of leading zeros. So they're forced to generate many hashes with different nonces until they happen upon one that works.

Each block includes a "coinbase" transaction that pays out 50 bitcoins to the winning miner—in this case, Gary. A new address is created in Gary's wallet with a balance of newly minted bitcoins.

TRANSACTION VERIFIED

As time goes on, Alice's transfer to Bob gets buried beneath other, more recent transactions. For anyone to modify the details, he would have to redo the work that Gary did—because any changes require a completely different winning nonce—and then redo the work of all the subsequent miners. Such a feat is nearly impossible.

From `http://lsvp.com/2013/03/28/how-bitcoin-works/`

Analysis

Why is consensus almost-certainly reached?

- Suppose two miners solve a puzzle simultaneously.
- Both broadcast their versions of the new database D and D'.
- Perhaps half of the miners work on D and half on D'.
- Most miners are likely attempting to incorporate Alice's transaction into a new database.
- Suppose some miner working on D solves the puzzle and sends out the new database D''.
- All miners receiving D'' discard the old D or D' and begin working on D''.
- Now an overwhelming majority of them believe D'' is the current database. They will only change their minds if a yet-longer certified database shows up.

Where's my money?

A good question to ask is, “Where is my money?”.

It's obviously in the cloud, but where it is exactly is in the miners' computers.

Security relies on there being many honest miners.

Successful miners are currently rewarded with new bitcoins, but as time goes on, the rewards are programmed to diminish.

What happens when miners no longer have the incentive to solve the computationally-intensive puzzles?

Other potential problems

There are other potential problems as well.

- What happens if Alice's private signing key gets compromised?
- What happens to bitcoins that are lost?
- What happens if the puzzle turns out to be not as hard as expected?
- What happens if people turn their attention to a competing digital cash scheme?
- Is this another Ponzi scheme? Why or why not?
- Bitcoins have been compared to gold. Is that comparison valid?

Kerberos

Kerberos

Kerberos is a widely-used authentication system and protocol developed originally by M.I.T.'s Project Athena in the 1980's.

The protocol was named after the character Kerberos (or Cerberus) from Greek mythology which was a monstrous three-headed guard dog of Hades.

http://collectionsonline.lacma.org/mwebcgi/mweb.exe?request=record;id=12845;type=101

Simple authentication protocol

Alice and Bob want to communicate privately.

If they already share a private key K, they can just send encrypted messages to each other.

Problems with this approach:

1. Every time Alice uses K, she exposes it to possible cryptanalysis, so she really only wants to use it to establish a *session key* K_{ab} to encrypt her message to Bob.
2. Alice needs a different key for each different user she might wish to communicate with. In an N-party system, this could require $O(N^2)$ keys and becomes unwieldy.

Kerberos overview

Kerberos overcomes these problems by using a trusted server called the *Key Distribution Center (KDC)*.

Every user shares a key with the KDC.

When Alice wishes to talk to Bob, she asks the KDC to generate a session key K_{ab} for them to use.

The KDC uses Alice and Bob's private keys K_a and K_b for authentication and for the secure distribution of the session key K_{ab} to Alice and Bob.

Problems to overcome

The protocol must overcome several problems to be useful in practice:

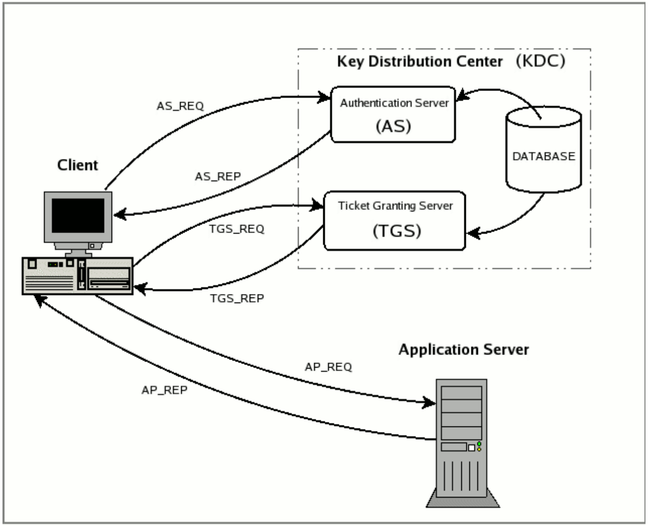
- Network security is not assumed, so uses must never send their private keys over the network.
- Once Alice obtains K_{ab}, she needs a way of verifying that the other party holding K_{ab} is really Bob and not someone else pretending to be Bob.
- Users do not want to be constantly asked to provide their passwords, so a *single sign-on (SSO)* system is desirable.
- In a large system, the KDC could become a bottleneck, so it needs to be scalable.

Parties to the protocol

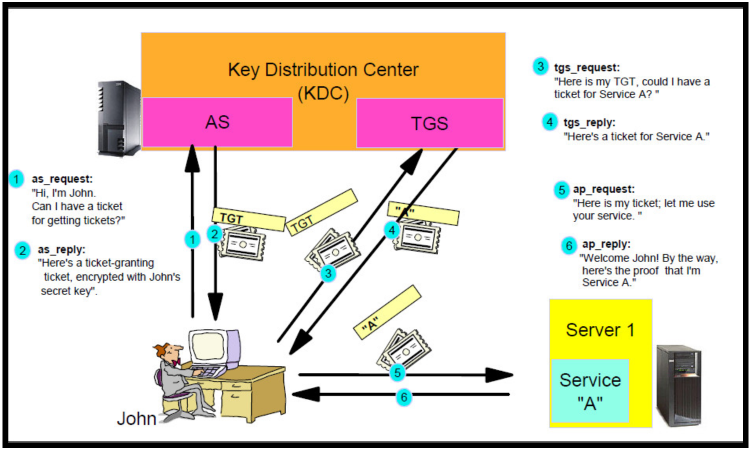
Four parties are involved in the basic protocol:

- The *authentication server (AS)*;
- The *ticket granting server (TGS)*;
- The *client, Alice in our examples*;
- The *service server (SS), Bob in our examples*.

The KDC contains the database of all keys and generally runs both the AS and the TGS.



From `http://www.zeroshell.org/kerberos/Kerberos-operation/`



From `http://www.ibm.com/developerworks/ibmi/library/i-sso/`

Basic protocol

At a high level, the basic protocol consists of three phases:

1. Alice authenticates herself to the AS and receives a *ticket granting ticket (TGT)* in return.
2. Alice presents a TGT to the TGS to obtain an *Alice-to-Bob ticket*.
3. Alice presents the Alice-to-Bob ticket to Bob in order to obtain service.

Alice only uses her private key in step 1. The TGT obtained in step 1 contains a *client/TGS session key* that is used for securely communicating with the TGS in step 2.

Phase 1: Obtaining a TGT

Alice authenticates herself to AS and obtains a TGT.

- Alice sends a cleartext message with her ID "a" to the AS.
- The AS obtains Alice's secret key K_a from the database and sends back two messages:
  1. Message A: A Client/TGS session key $K_{a,\mathrm{TGS}}$, encrypted with K_a.
  2. Message B: A TGT (Alice's ID, her IP address, expiration time, $K_{a,\mathrm{TGS}}$), encrypted with K_{TGS}.
- Alice decrypts message A to obtain $K_{a,\mathrm{TGS}}$. She is unable to decrypt message B.

Phase 2: Obtaining an A-to-B ticket

Alice uses her TGT to obtain an Alice-to-Bob ticket (A-to-B).

- Alice sends two messages to the TGS:
  1. Message C: (Message B, Bob's ID).
  2. Message D: (Alice's ID, timestamp), encrypted with $K_{a,\mathrm{TGS}}$.
- The TGS retrieves message B from message C and decrypts it to get $K_{a,\mathrm{TGS}}$, which it then uses to decrypt message D. It checks Alice's ID and IP address, generates a session key K_{ab} and then sends two messages to Alice:
  1. Message E: A-to-B ticket = (Alice's ID, her IP address, expiration time, $K_{a,b}$), encrypted using K_b.
  2. Message F: $K_{a,b}$, encrypted using $K_{a,\mathrm{TGS}}$.

Phase 3: Authenticating herself to Bob

Alice uses her A-to-B ticket to authorize herself to Bob.

- Alice sends two messages to Bob:
  1. Her A-to-B ticket, which she received from TGS as Message E.
  2. Message G: An authenticator = (Alice ID, timestamp), encrypted with $K_{a,b}$.
- Bob decrypts the ticket to retrieve $K_{a,b}$, which he uses it to decrypt the authenticator. He sends the following message to Alice:
  1. Message H: (1 + timestamp from the authenticator), encrypted with $K_{a,b}$.
- Alice decrypts and checks message H for correctness.

Use in practice

Tickets have a relatively long lifetime and can be used many times.

Authenticators have a relatively short lifetime and can be used only once.

The latest protocol has additional security enhancements beyond those described here.

Advantages of Kerberos

- Passwords aren't exposed to eavesdropping.
- Password is only typed to the local workstation.
  - It never travels over the network.
  - It is never transmitted to a remote server.
- Password guessing is more difficult.
- Single sign-on.
  - More convenient: only one password, entered once.
  - Users may be less likely to store passwords.
- Stolen tickets hard to reuse.
  - Need authenticator as well, which can't be reused.
- Much easier to effectively secure a small set of limited access machines (the KDC).
- Easier to recover from host compromises.
- Centralized user account administration.

Drawbacks and Limitations

- Kerberos server can impersonate anyone.
- KDC is a single point of failure.
  - Can have replicated KDC's.
- KDC could be a performance bottleneck.
  - Everyone needs to communicate with it frequently.
  - Not a practical concern these days.
  - Having multiple KDC's alleviates the problem.
- If local workstation is compromised, user's password could be stolen.
  - Only use a desktop machine or laptop that you trust.
  - Use hardware token pre-authentication.
- Kerberos vulnerable to password guessing attacks.
  - Choose good passwords!
  - Use hardware pre-authentication.
    - Hardware tokens, Smart cards etc.

Secure Shell (SSH)

Secure Shell (SSH)

SSH is a family of protocols that provide a secure encrypted channel connecting two networked computers over an insecure network.

It was initially designed to allow secure login between a user and a remote computer. This replaced the older `telnet`, `rlogin`, and `rsh` programs that provided unencrypted versions of this service and sent unencrypted passwords over the network.

The first version was designed by Tatu Ylönen at Helsinki University of Technology, Finland, and released as freeware in July 1995.

Open source and version forking

By December 1995, Ylönen created a startup company to market and develop SSH. The original version remained free, but many enhancements were only available in the commercial version.

Over the next five years, the licensing agreement became more and more restrictive as they worked to remove open source code (such as libgmp) from their code base.

Several vulnerabilities in SSH-1.5 were discovered, giving further motivation to create an open source version of SSH that could be more easily vetted for bugs and distributed more widely.

Starting from Ylönen's SSH-1.2.12, the last version released under an open source license, Björn Grönvall developed OSSH. OpenBSD developers then forked Grönvall's code and did further development to create the widely used OpenSSH.

SSH-2

By 2006, an improved but incompatible version 2 of the SSH protocol was adopted as a standard by the Internet Engineering Task Force (IETF).

Development of OpenSSH continues to this day, allowing more and more applications derive the benefits of secure encrypted communications.

SSH protocol outline

SSH is based on public key cryptography. Each machine has a public and private *host key*. Each user also has a public and private *user key*.

When logging onto a remote machine, the client first authenticates the remote host key using the public key for the host with that domain name or IP address found in the local `known_hosts` file.

Host key verification

If the host is not in the file, one gets a prompt

```
The authenticity of host 'vm1.cs.yale.edu (128.36.229.150)' can't be established.
RSA key fingerprint is c9:a5:be:55:af:ab:05:77:b4:30:62:ed:bd:be:50:43.
Are you sure you want to continue connecting (yes/no)?
```

If you say yes, the public key of that host gets entered into the `known_hosts` file and used the next time.

If the host is in the file but the key returned by the contacted host differs from the key in the file, a much sterner warning is produced

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
ed:42:05:a0:13:b2:af:32:57:d1:80:77:28:42:e2:2e.
Please contact your system administrator.
Add correct host key in /home/fischer/.ssh/known_hosts to get rid of this message.
Offending RSA key in /home/fischer/.ssh/known_hosts:337
RSA host key for killdeer.homelinux.net has changed and you have requested strict checking.
Host key verification failed.
```

User authentication

SSH supports several authentication methods. I'll describe the `publickey` method.

Here, the host checks that the user's public key is in its `authorized_keys` file. If it is, it verifies that the user has the matching private key and accepts the authentication if it does.

Actual protocol

The actual protocol is much more complicated than this.

It include negotiations for which cipher to use, how to generate the shared session, what an encrypted packet looks like, and so forth.

Transport Layer Security (TLS)

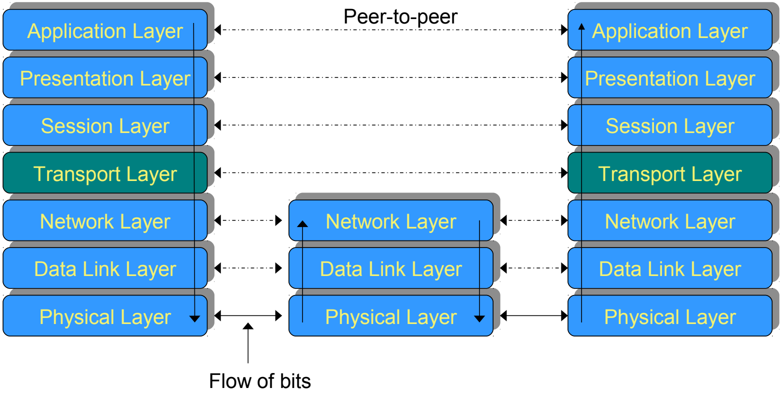
TLS protocol

Transport Layer Security (TLS) is a protocol to secure and encrypt network traffic at the transport layer.

Like SSH, it provides authentication and encryption.

It is used to implement secure web traffic (`https:`) but applies much more generally.

ISO/OSI Model SSL: Security at Transport Layer



History

The TLS protocol has gone through several versions. It was originally called *Secure Socket Layer (SSL)*.

TLS 1.0 was first defined in 1999. It did not interoperate with the existing SSL 3.0 and so was renamed.

Key management

TLS uses X.509 certificates for its key management as described in lecture 18.

They allow the client to authenticate the server as follows:

1. The client obtains the server's certificate, generally from the server itself.
2. The client checks the validity of the certificate by verifying that it is properly signed by a trusted certificate authority.
3. The client then runs a simple authentication protocol whereby the server shows that it has the private key corresponding to its certificate.
4. Finally, client and server establish a shared symmetric key and use it to encrypt traffic between themselves.

The actual protocol

The actual protocol has all of the complications of the other practical protocols we've mentioned.

- It begins with a handshaking phase where client and server agree on the protocol level, cipher suite, and other parameters.
- Generally the server is authenticated by the client.
- The protocol allows for the server to require client authentication. However, this is not usually done for two reasons:
    1. Most clients lack certificates.
    2. Most servers use other means such as passwords to authenticate clients. This occurs after the encrypted connection has been established, so the passwords are not sent in the clear.

Preventing man-in-the-middle attacks

TLS is secure against man-in-the-middle attacks, even with only one-way authentication.

This is possible because the certificate gives the client a reliable means of obtaining its public key (providing the certificate authorities are trustworthy).

At the end of the handshaking protocol, after the session key has been established, the client sends the server the hash of the complete transcript between client and server as seen by the client, secured with the server's public key.

The server checks that hash value against the hash of its view of the same handshake. If they don't agree, it indicates the presence of a man-in-the-middle or other network error and the protocol does not continue.

Digital Rights Management and Trusted Computing Platform

Control of information

Another attempted use of cryptography has been to control the use of information.

Digital Rights Management (DRM) is the term for a class of encryption schemes to disallow certain kinds of use of data, for example, copying and modifying.

Trusted Computing Platform (TCP) is a hardware architecture that requires authorization tokens to perform certain operations. These tokens are cryptographically produced using keys that are securely stored inside of a special crypto module.

A different paradigm

In most uses of cryptography studied so far, the owner of the secret keys also possesses and protects them.

With DRM and TCP, the party controlling the keys is not the same as the owner of the machine that uses them.

This means that the keys must be hidden from the owner of the device on which they are used.

While it is easy to prevent casual users from looking inside their box, protecting embedded secrets against sophisticated attacks has proved to be very difficult, and many DRM schemes have been broken soon after being introduced.