Advanced Network Security

Foundations of Network Security: Part II

Thomas Martin

t.martin@mmu.ac.uk

March 6, 2019

Outline

Session Key Agreement

When two parties wish to communicate, even if they already have a shared key it is customary to establish a key just for the particular communication session. This provides several security benefits.

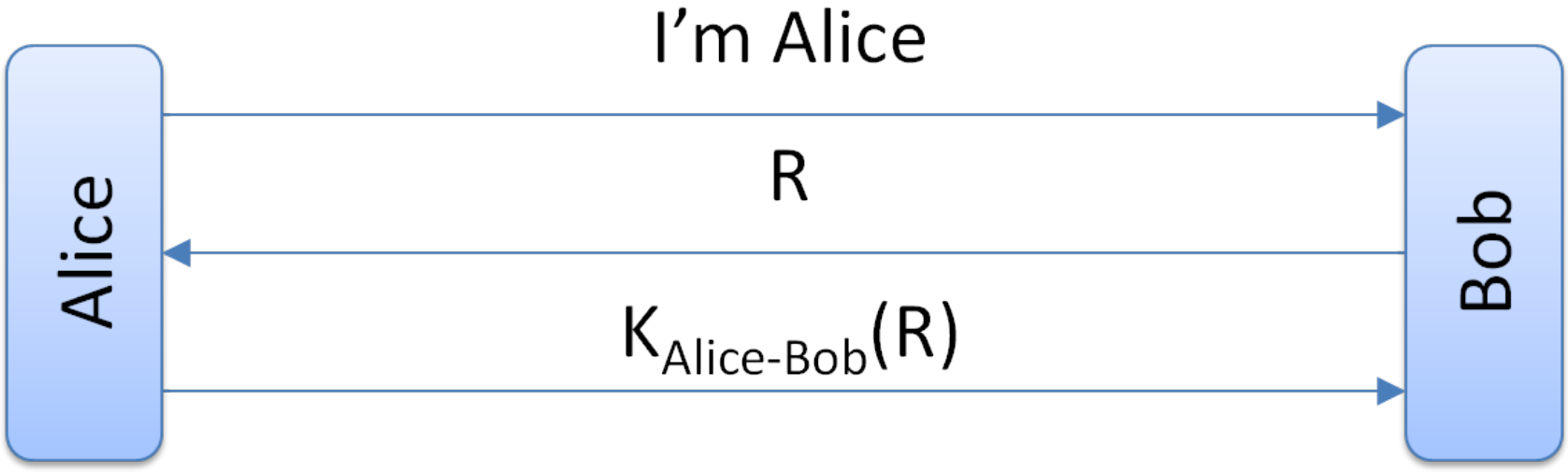
Cryptographic attacks usually require large amounts of encrypted text. The longer a single key is used, the easier this is to obtain.

Using the same keys for integrity-protection in multiple sessions may mean that traffic from old sessions can be replayed.

Limiting the use of keys limits the impact of any disclosure. And with forward secrecy, disclosure of the long term keys does not even disclose the past session keys.

Ideally, after an authentication exchange, Alice and Bob will share a session key and no eavesdropper will be able to figure it out.

Session Key



After the above exchange, $K_{Alice-Bob}(R+1)$ might seem like a good session key, but can be discovered later on by imitating Bob. $(K_{Alice-Bob}+1)(R)$ can be used instead. Any modification of just the random nonce can be discovered later by an imitator. A simple modification of the key encrypting the nonce avoids this problem.

Session Key using Public Keys

If public keys are used, there are a number of options to add in order to agree a session key. The simplest is where Alice attaches $PK_{Bob}(R)$ to any authentication exchange (where R is different from any of the challenges in the protocol). This can simply be replaced by Malory by a different random value encrypted with Bob's public key.

If instead Alice includes a random value that is signed as well as encrypted $Sign_{Alice}(PK_{Bob}(R))$, then the above cannot happen. Malory does not have Alice's private key so cannot create signatures. Any modification will be detected.

However, this does not protect the communications if the private key of Alice is later discovered.

Session Key using Public Keys

Suppose Alice and Bob choose two separate random values, R_1, R_2, and exchange them encrypted with each others public keys ($PK_{Bob}(R_1)$ and $PK_{Alice}(R_2)$). If the session key is defined to be $R_1 \oplus R_2$, then an attacker later learning one of Alice or Bob's private key is not enough to determine the session key.

An attacker cannot cause a known session key to be used by injecting forged messages (but can cause an unknown session key to be established).

Perfect Forward Secrecy is the property where discovering the long term private keys is not enough to work out any session key derived with them. The Diffie-Hellman protocol has this property.

One-Way Public Key Based Authentication

Where only one side has a public-private keypair, it is typically the server. The strength of the authentication (and agreed key) is limited in this case.

1. Alice can encrypt a random number with Bob's public key. If Malory later overruns Bob, this will be decrypted
2. Alice and Bob can perform a Diffie-Hellman exchange

In both cases, only Alice is assured she is communicating with Bob (assuming she knows Bob's public key). Bob is only assured that the entire conversation is with a single party, and authenticating Alice requires an extra measure (i.e. password authentication).

Privacy and Integrity

There are a number of ways to ensure the integrity of message contents. Block ciphers can be modified to create MACs and keyed hash functions exist. It is generally considered good practice for two separate keys to be generated for the purposes of privacy and integrity.
From the attacks we have seen so far, there is much more than the contents of messages that should be protected. Replaying an old message later in a session can cause problems. Sequence numbers can prevent this.
We have also seen reflection attacks, where the message from one party is used to forge a reply back. If each message contains information about which direction that message is supposed to go (and this information cannot be modified), then this can also be prevented.
To prevent reuse of sequence numbers, *key rollover* (updating the session key to a new value) is often done.

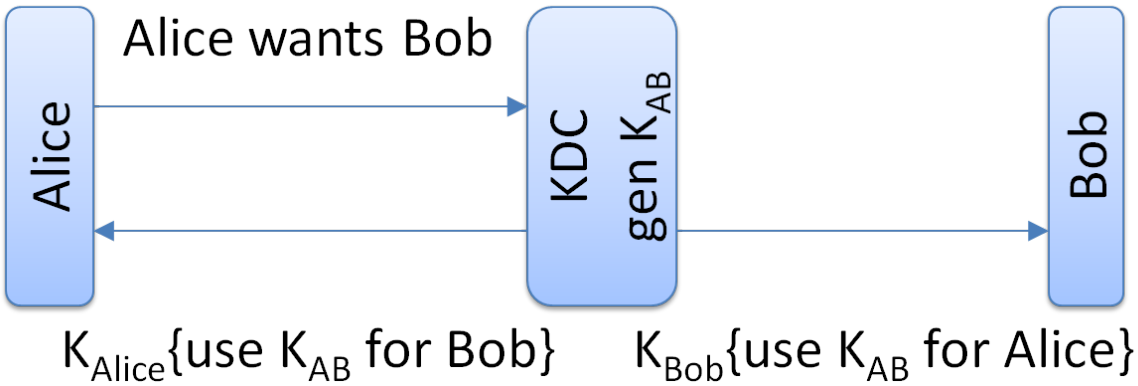
Key Distribution Centers

Assuming a shared key exists between two parties is reasonable. However, if there is a large number of people who need to communicate in private, it can become problematic. For n people in a network, everyone needs $n - 1$ keys. People joining and leaving can require complex updates.

A solution to this is to use a **Key Distribution Center** (KDC). A KDC shares a key with each entity in the network. Authentication between any two parties is mediated by the KDC. Users can be added to the network by giving them a single key.

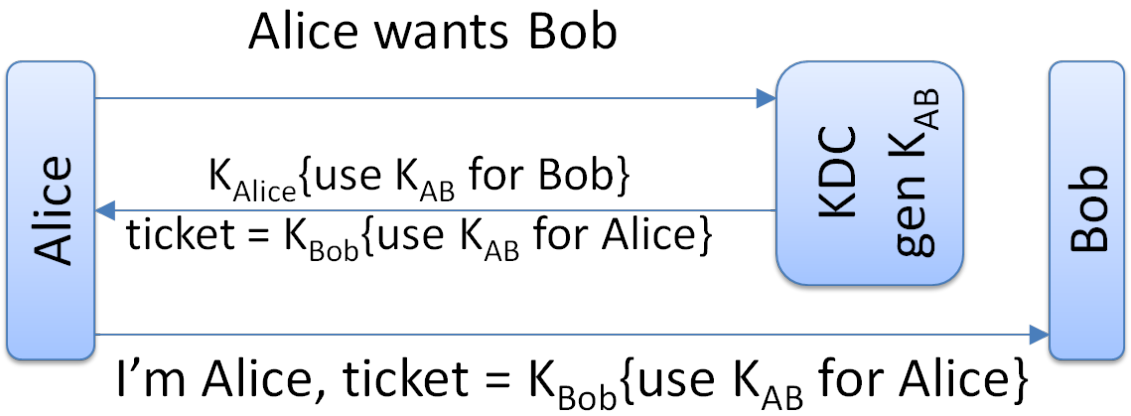
The downsides of a KDC is that they can impersonate anyone, they are a single point of failure and a performance bottleneck.

KDC Operation



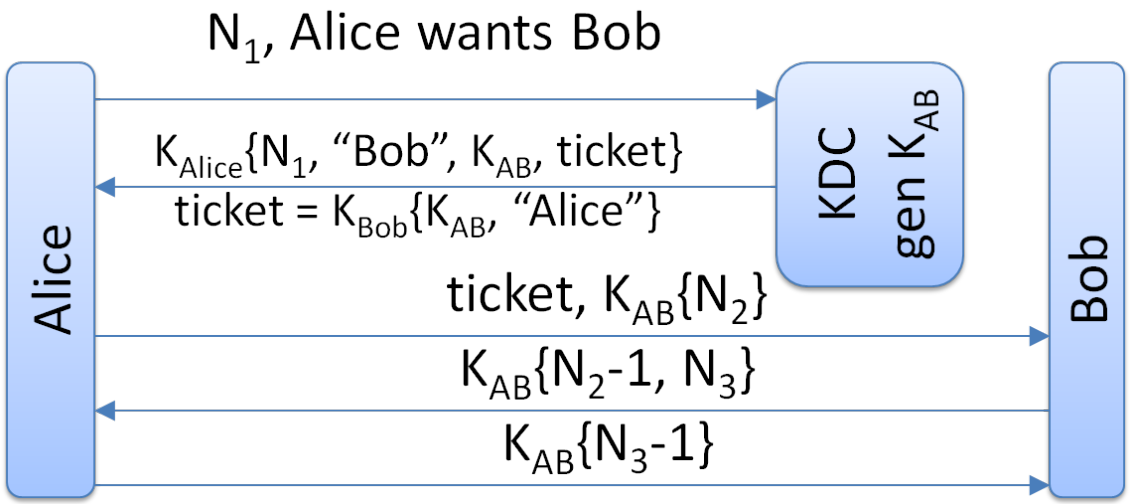
One method of operation of a KDC is described above. The KDC receives a request for setting up secure communication between Alice and Bob. A random key K_{AB} is created for this purpose, encrypted and sent to both parties. Since Alice and the KDC share a secret key, they can authenticate each other with one of the previously described protocols. But even without this, if the request is accepted, it will not benefit an eavesdropper as the session key is encrypted.

KDC Operation



One problem with the previous exchange is that Bob may start receiving encrypted messages from Alice before he gets the session key, and so will not be able to decrypt. Also, it can be problematic for the KDC to initiate a connection to Bob. Instead, the KDC can give Alice a ticket to be forwarded to Bob. This is still incomplete, as both Alice and Bob need to prove to each other that they know K_{AB}.

Needham-Schroeder



A classic authentication protocol using a KDC was designed by Needham and Schroeder in the 70's. It has been used as the basis of many protocols, including Kerberos. It uses a **nonce**, or number used once.

Needham-Schroeder

The inclusion of a nonce in the first message allows Alice to prove that she is really talking to the KDC. Without it, an attacker who had stolen Bob's key, would be able to mount a replay attack to force Alice to accept an old key.

In message 2, the KDC provides Alice with the encrypted session key K_{AB}, and the ticket for Bob. The fact that the names are included and encrypted means they cannot be modified. If Malory tried to change the recipient name in message 1, then Alice would discover this in message 2.

Message 3 allows Bob to obtain the key K_{AB}, and since Alice's name is in the ticket, he knows it is her he is communicating with. In message 4, Bob proves to Alice that he knows K_{AB} since he was able to obtain N_2 and re-encrypt (note how it is decremented). In message 5 Alice proves the same.

Needham-Schroeder

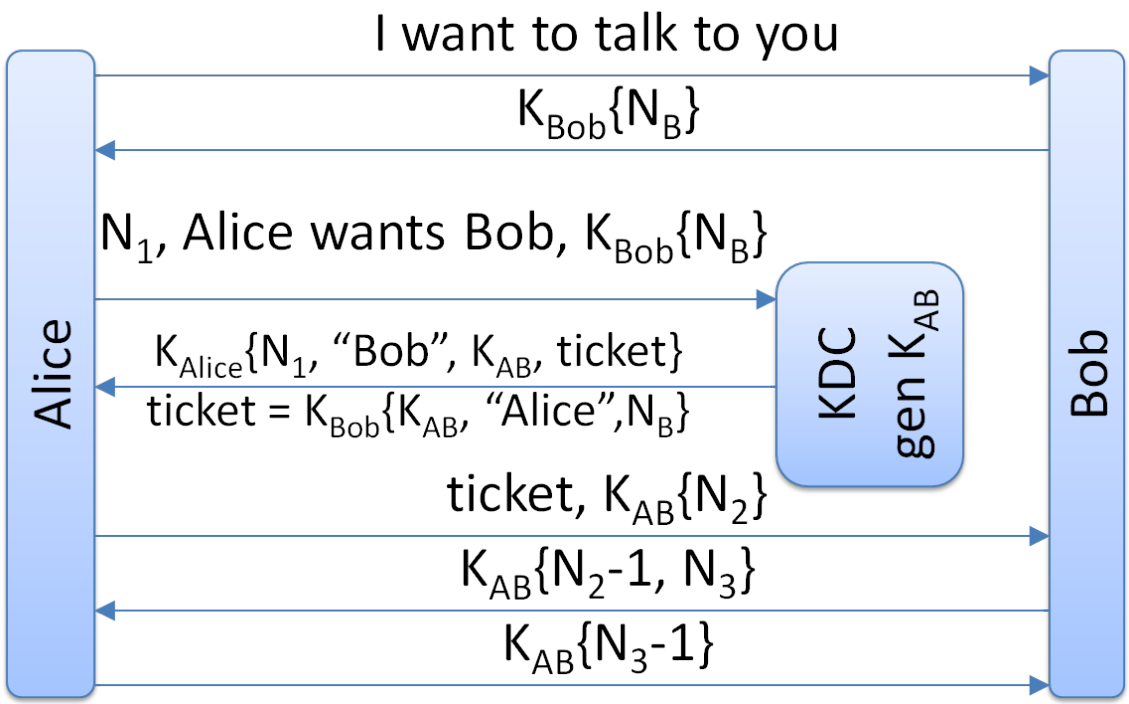
It may seem reasonable to assume that encrypted message cannot be modified. This is not the case, however. Stream-ciphers allow any bits to be flipped at will. Block-ciphers can be used in simple modes (ECB) which allow whole blocks to be rearranged. If the sizes of the nonces in the previous protocol were equal to the block length of the cipher, and such a mode were used, a reflection attack would be possible where an attacker impersonates Alice to Bob.
Malory eavesdrops an entire authentication. Later, she replays message 3 to Bob. She cannot create the required response, but instead starts a new connection splicing in the second half of the old message 4 as the second half of the new message 3. This forces Bob to create the response that Malory needs to complete the old connection.

Needham-Schroeder

If an attacker obtains K_{Alice}, the key she shares with the KDC, then they can get a key and a ticket for Bob from the KDC. This is unavoidable. However, there is a problem with the fact that old tickets still stay valid after Alice changes her key (the ease of key updates is an advantage of using KDC's).

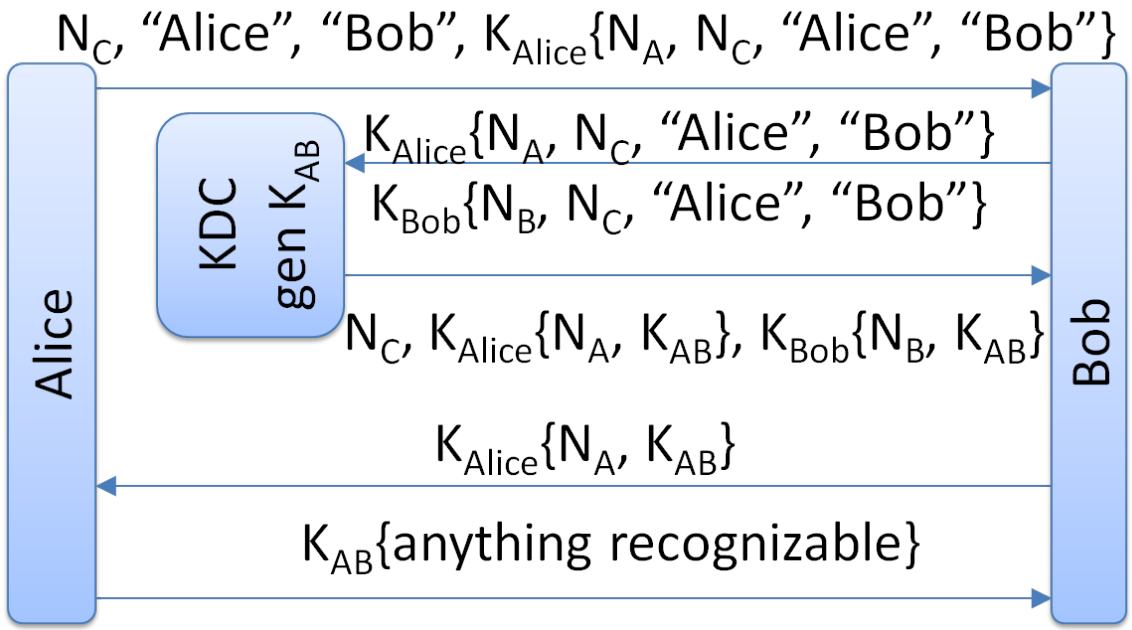
Say Malory eavesdrops an authentication and then later discovers Alice's key. If Alice updates her key with the KDC from K_{Alice} to K'_{Alice}, then Malory can no longer impersonate Alice to the KDC, but she does not need to. Knowing K_{Alice} gives her K_{AB} from the old exchange, which allows her to pickup from message 3 with Bob, and complete a new connection. Since the KDC does not participate in this conversation, the fact that Alice's key has changed does not matter.

Expanded Needham-Schroeder



The problem can be corrected with an additional two messages at the start. The nonce is sent by Bob, but decrypted by the KDC. This assures that once a key is changed, all old tickets are invalid.

Otway-Rees



The Otway-Rees protocol differs from the previous one in that it is the initiator of the communication who generates the nonce, and the second party communicates with the KDC.

Otway-Rees

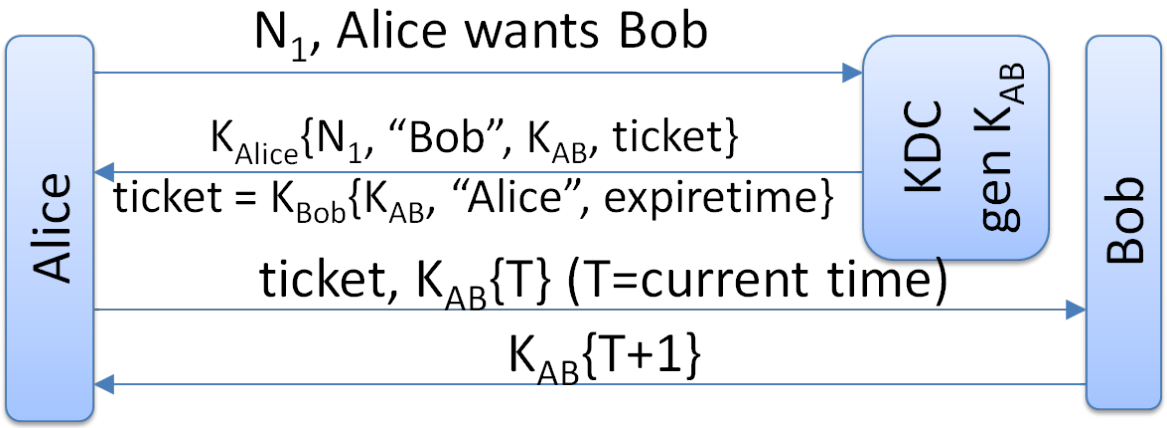
For message 1, Alice generates two nonces. The first is sent in the clear to Bob, and both are encrypted for the KDC.
In message 2, Bob forwards the encrypted message from Alice, and encrypts his nonce and the nonce from Alice for the KDC. Note that both the names of the participants are included in these encrypted messages, which will alert the KDC to any redirection/impersonation.
The KDC makes sure the nonce N_C is the same in both messages, and rejects otherwise.
Messages 3 and 4 are where the session key is distributed to Alice and Bob, but the fact that they are encrypted with the respective nonces is what provides the authentication.
Message 5 is needed to show that Alice knows K_{AB}.

Otway-Rees

There is an attack that is possible on Otway-Rees, that depends on how the nonce is implemented. A counter is a number that is only used once (increased by 1 each time), but if Alice uses this for N_C, Malory can impersonate Bob to Alice. She can complete the protocol (authenticate), but not learn the session key.

If Malory learns that Alice is currently using $N_C = 007$, then she starts a connection with Bob using $N_C = 008$. Not knowing K_{Alice}, whatever she sends will be rejected by the KDC, however, she will get Bob's reply $K_{Bob}\{N_B, 008, \text{“}Alice\text{”}, \text{“}Bob\text{”}\}$. When Alice next tries to communicate with Bob, Malory intercepts it, and forwards Alice's encrypted part with Bob's encrypted part from the old connection. This will be accepted by the KDC who will provide Malory with the encrypted session keys. When the key is forwarded to Alice, she will be convinced she is communicating with Bob.

Kerberos



The Kerberos protocol (default authentication in many versions of Windows) is based on Needham-Schroeder, but assumes synchronized clocks, and includes expiration times in the tickets. An organization might configure tickets for office machines to have a 10 hour life-span, but public terminals (where people often leave and come back) to only last 10 minutes.

Nonces

A nonce is a value in a protocol that will be different each time the protocol is executed. There are three main types of nonce:

1. Timestamp
2. Sequence Number
3. Large random number

Timestamps require accurate time-keeping across the network. Sequence numbers require some non-volatile storage. And unlike the other two, random numbers can give no guarantees about repeats (with sequence numbers you at least know when you will wrap around). However, random numbers are the most unpredictable.

Nonces

Consider the first two secret-key based authentication protocols we looked at, and how they function if the random values are predictable. Alice is sent a challenge and must encrypt it. Suppose Malory observes previous exchanges and can predict the next challenge. She intercepts the next request from Alice and provides the challenge. Alice dutifully provides the encrypted response. Malory can use this to impersonate Alice to Bob. Unlike other man-in-the-middle attacks, this does not require both parties to be online at the same time.

In the second protocol, where Bob sends an encrypted challenge and Alice must decrypt it, it is easier for Malory. If the challenge is predictable, then there is no need for Malory to try decrypt it or manipulate Alice into providing it. She can just use it to complete the exchange directly.

Generating random values

There are two types of ways to create random values. A pseudo-random number generator creates values from an initial state and a deterministic algorithm. This can create long sequences of apparently random values from a small initial seed. A random number generator is one that generates truly unpredictable numbers. Physical quantities can be good sources for randomness, i.e. the number of radioactive decays detected by a Geiger counter in a given time period. It is rare for computer equipment to have such sources. Most use pseudo-random generators that are seeded with key-stroke timings, mouse movements, disk seek times, counts of packet arrivals. For non-security uses of randomness, e.g. testing software with various inputs, a good distribution of values may be all that is needed. It needs to be uniform, with no bias. However, such randomness is not enough for use in protocols, it needs to be unguessable.

Pitfalls with randomness

One possible pseudo-random generator would be to iteratively hash the seed. A good hash function will give a uniformly random output. But unfortunately, it is predictable. An attacker learning any one random value (nonce that is made public in an exchange) will be able to determine all future outputs. Other possible problems with pseudo-random generators:

- Seeding the generator with a seed that is too small a space - you need to rule out the possibility of someone performing a brute-force attack.
- Using a hash of the current time when the application needs a random value - even if the time is recorded to a fraction of a second and an attacker only knows the hour it happened, the number of possible values is relatively small.
- Divulging the seed value - very possible when using time.

Performance

Besides security, there are a number of different parameters we consider in evaluating authentication protocols:

- number of cryptographic operations using a private key
- number of cryptographic operations using a public key
- number of bytes encrypted or decrypted using a secret key
- number of bytes to be cryptographically hashed
- number of messages transmitted

As network travel can be the source of most delay, the last factor can be the most important. The types of computations can be important as well, e.g. if the encryptions can be done in parallel or not.
Some protocols allow caching of credentials that will speed up the process of subsequent authentications.

In Class Tasks

You find yourself in a foreign country with all your possessions stolen. How would you:

- Prove you are who you say you are to your family/friends and get them to send money to you if you only had access to email?
- Same question, but with access to a social network instead?
- What if you needed to contact your employer?
- What complications might arise and how could they be mitigated?

Thank you

Any Questions?