CS 261 Notes: Secure Messaging

Scribe: Vy-An Phan

29 Aug. 2018

1 Introduction

Secure messaging is the task of communicating between two parties in a private and safe manner. Some of the properties of secure messaging include:

- **Confidentiality**: only the intended parties, i.e. the sender/receiver, should be able to read the data.
- **Integrity**: the intended parties should be able to ensure that their messages have arrived intact and unchanged, or be able to tell if it has been manipulated.
- **Authentication**: the intended parties should be able to ensure that they are communicating with the intended parties, and only each other.
- **Non-repudiation**: if a party sends a message, they should not be able to claim otherwise later.

In addition to security properties, our methods must take human factors into account:

- **Usability**: if an application is secure, but too slow, most people will simply give up and take their chances with a faster but less secure method.
- **Ease of adoption**: being forced to install extra software or have in-person meetings for key exchanges might cause potential users to give up on the idea of using secure messaging before they even start.

Performance is a common issue covered under both of these properties.

Aside from the goals of secure messaging, we must also consider the threat model. Potential threats can be divided into two main types:

- **Definition 1** *Passive Attacker*

  Also known as "honest but curious" or "semi-honest", these attackers can only read unauthorized data or eavesdrop but not do anything else to it. They otherwise follow protocol.

- **Definition 2** *Active Attacker*

  Also known as "malicious" or "fully untrusted", these attackers can intercept, delete, add, spoof, or change unauthorized data in addition to reading it. They do not necessarily follow protocol.

Active attackers are stronger but also more easily exposed than passive attackers. If one is unable to defend against an active attacker, one should at least be able to detect them. When characterizing threats, anything other than the sender and receiver should be considered an attack surface. Each section under threat should be characterized with its own threat model, though it is acceptable to consider multiple related components a single section should they share the same threat model (e.g. multiple machines and their connections might be abstracted into just one "server" section).

In these notes, we refer to the authorized parties as Alice and Bob, a passive attacker as Eve (for eavesdropper), and an active attacker as Mallory. The term "attacker" used without specification refers to both active and passive attackers.

2 Stages of Secure Messaging

There are three main stages of secure messaging:

1. ***Trust establishment***
   Parties must first initiate a secure channel, in order to ensure that they are both speaking to and only to the intended parties. During this phase, some sort of long-term key exchange (public keys) and handshake protocol occurs.

2. ***Conversation security***
   This refers to the actual protection of messages during communication, involving message encryption and often many short-term keys.

3. ***Transport security***
   While the contents of messages themselves may be fully encrypted, metadata such as the identities of the communicating parties and their volume of communication can still be leaked. This is the most difficult layer to implement, and currently most "secure" communication is in fact not 100% anonymous or otherwise very very slow.

There can be some overlap in responsibilities between each stage.

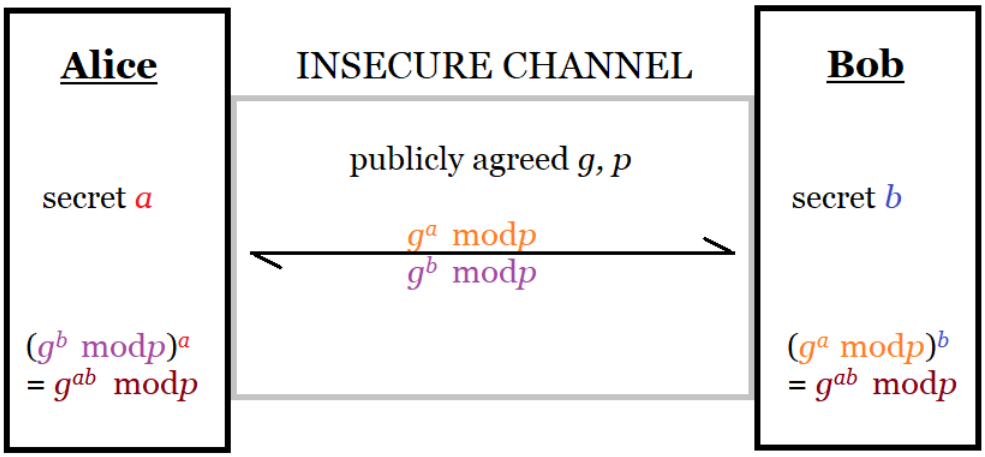
3 Trust Establishment

There are many methods of trust establishment in use today. Here we will review a few and examine their advantages and flaws.

3.1 Opportunistic Encryption

Opportunistic encryption assumes a passive attacker only, and the communicating parties do a pure key exchange (Diffie-Hellman, for example). This is the simplest and easiest to implement, as well as the most user-friendly. It is safe against passive attackers. Unfortunately, it is also vulnerable to a man-in-the-middle attack as we cannot make such assumptions about active attackers in the real world.

To demonstrate, we first explain Diffie-Hellman key exchange. Diffie-Hellman key exchange relies on the fact that while the discrete log is easy to compute, it is difficult to reverse. Given some g, a, and p, one can easily find $(g^a \mod p)$, but given $(g^a \mod p)$, g, and p, one cannot easily compute a.

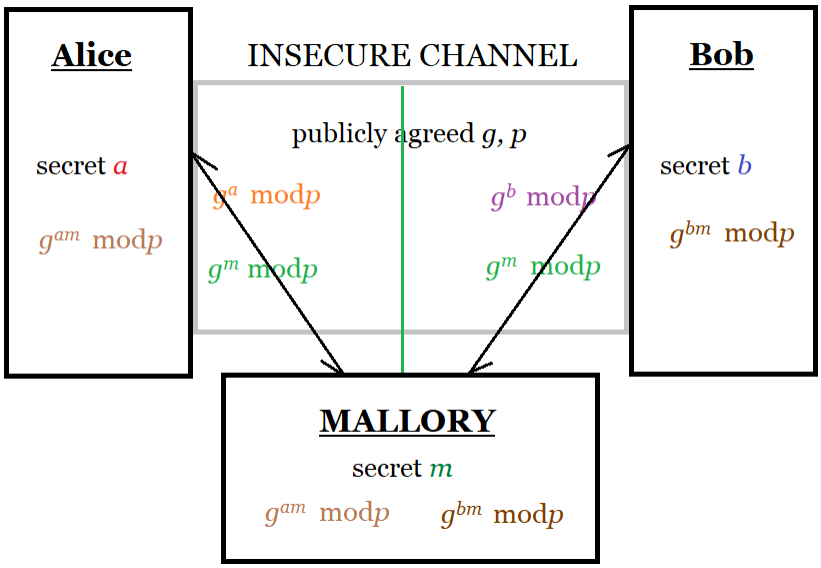


The key exchange protocol proceeds as follows:

1. Together, Alice and Bob both agree upon a large prime p and a number $g \in [2, p-1]$. g and p are both public and so it is also visible over the insecure channel.
2. Alice then picks a private secret a which she does NOT share over the channel. Bob does the same with his own private secret b.
3. Alice then computes $(g^a \mod p)$ and sends this to Bob over the channel. Bob computes $(g^b \mod p)$ and sends that to Alice. Both these values are visible over the insecure channel, but a and b themselves are still unknown.
4. Alice and Bob each privately compute $(g^{ab} \mod p)$ and thus arrive at the same key despite never explicitly sharing their secrets over the insecure channel.

Note that neither a passive nor active attacker can recompute a, b, or $(g^{ab} \mod p)$. They cannot distinguish (g^a, g^b, g^{ab}) from (g^a, g^b, g^r) where r is any random number.

Despite not knowing what the secret key is, however, an active attacker Mallory with the ability to spoof and drop packets.



Mallory's attack on opportunistic encryption proceeds as follows:

1. Mallory comes up with their own secret m and computes $(g^m \mod p)$ from the publicly known g, p which Bob and Alice agreed upon.
2. Mallory intercepts Alice's and Bob's messages to each other. Mallory now knows both $(g^a \mod p)$ and $(g^b \mod p)$, but neither Bob nor Alice received them.
3. Mallory sends $(g^m \mod p)$ to Alice while pretending to be Bob, and the same to Bob while pretending to be Alice. As neither Alice nor Bob received each other's $(g^a \mod p)$ and $(g^b \mod p)$, they have no idea they are actually looking at Mallory's false message.
4. Alice and Bob compute the incorrect secret keys, and enter into secure communication – but with Mallory. Mallory can now decode both Alice and Bob's messages, as well as change them and pass them on, all the while pretending to be Bob and Alice to the other. Alice and Bob, meanwhile, have no way of knowing that any of this is happening in the first place.

3.2 Trust on First Use (TOFU)

An extension of opportunistic encryption, TOFU attempts to mitigate an active attacker's potential by remembering the previous public key used to communicate with a certain party and accepting no other changes. It still assumes that there is no active attacker but only during the first key exchange. Should an active attacker attempt to insert some $(g^m \mod p)$
any later, it would not be accepted as it would not match the information received during the first exchange.

It is still simple to implement and use, but it does not support even legitimate key revocation, which is often extremely necessary.

WhatsApp uses a weaker version of TOFU which warns users when there is a key change and accepts it based on the user's decision. This will work for alert and educated users; however, for a lazy or uneducated user who will always click through the warning message, it is just as bad as TOFU.

3.3 Key Fingerprint Verification

One example using this method is Signal. It still starts with Diffie-Hellman key exchange. Once both Alice and Bob have computed their supposedly mutual secret key, they must validate the hash of the key over a second channel. This second channel is trusted for integrity, not confidentiality. This is fine, as using a hash allows Alice and Bob to verify they have computed the same correct secret key without actually revealing the key itself.

If Alice and Bob are using Mallory's m, their secret keys won't match. In fact, Mallory has no efficient way of generating her own secrets such that both Alice and Bob end up computing the same secret key.

Unfortunately, this method also depends the integrity of that second channel. Once again, if Mallory is able to manipulate the second channel, Alice and Bob can be tricked into thinking their key hashes match when in fact they do not.

There is also the matter of human error, again. If users do not actually check the hash, then it is just as bad as TOFU; if they also ignore warnings for key changes, then it revers to opportunistic encryption. One more secure but less user-friendly mitigation is forcing users to personaly input the expression rather than verifying a match; this protects lazy users and misreading alert users.

3.4 Certificate Authorities

This system stores public keys in central servers. While it allows Alice and Bob to verify one another's identities with asymmetric key cryptography – meaning Mallory's old man-in-the-middle attack methods no longer work – these central servers become single points of failure.

When certificate authorities are breached or coerced, attackers can issue themselves a valid certificate, or change a certificate to match their own. If an attacker can pretend to be someone else with a legitimately issued certificate as "proof", they can once again construct man-in-the-middle attack.

The pitfall of certificate authorities is that they provide both a single point of failure and a multiple point of attack. The DigiNotar hack, for example, only required that one particular CA (out of the many that exist) to be compromised.

3.5 Web of Trust

This attempts to mitigate the single point of failure of certificate authorities by separation of powers. Users verify each other's keys manually by signing those keys with their own key. For example, if Alice meets Bob and trusts him, she verifies his key. If Bob later signs Carol's key, Alice should theoretically be able to trust Carol by association. This trust should be stronger if multiple users trusted by Alice also trust Carol.

Assuming the scheme is implemented perfectly and no good users make mistakes, MiTM attackers and malicious operators or servers are limited in power. PGP is an example of how web-of-trust can be used for key exchange and signing.

However, there are a few problems. Firstly, web-of-trust currently involves complex and non-standardized user interfaces. Secondly, in the case of conflict, it is difficult to identify the cheater, or cheaters in case of collusion. And thirdly, malicious operators can still withhold changed keys, so this scheme only offers partial operator accountability and revocation.

One variation of web-of-trust is SPKI, where if Alice trusts Bob and Bob trusts Carol, Carol will appear to Alice as "Bob's Carol" until she can personally verify Carol's identity for herself.

3.6 Transparency Log

In frameworks such as CONIKS, there is no central point of trust. Public keys are stored in Merkle Trees, which are publicly available and allows auditing of these authorities. If a public key has been changed, or if a server has been compromised or coerced, the Merkle Tree auditing will detect these actions.

The problem with such a system is that it requires a great deal of infrastructure and log management. It can only detect, not prevent a man-in-the-middle attack. Users also have issues with key revocation; a server changing a past public key can be detected, but it is difficult to tell if a server using the correct protocol to add a new public key is a legitimate user or an attacker.

4 Conversation Security

This phase of secure communication refers to keeping the messages themselves secret, unchanged, and arriving at the correct parties. The problem of repudiation (deniability) also comes into play at this point; if Alice sends a message, there needs to be a method to prove she did so if she later claims she did not.

Assuming that trust has been established, and that our public keys are indeed correct, we now concern ourselves with a new set of problems:

- **Definition 3** *Forward secrecy*

  If an attacker steals a long-term (i.e. persistent) key, they cannot compromise earlier messages (as previous ephemeral keys are thrown away).

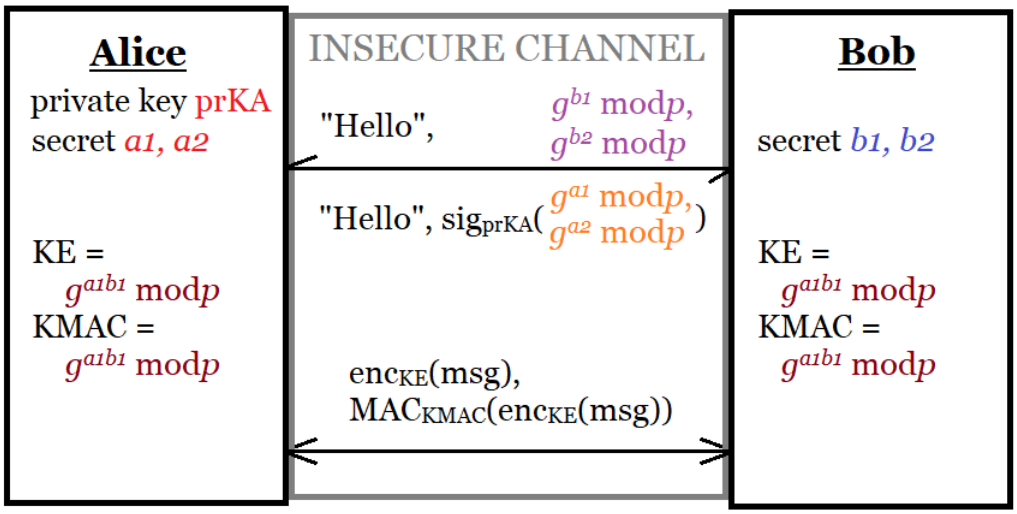
- **Definition 4** *Backward secrecy*

  If a passive attacker steals a long-term key, they cannot decrypt later messages (e.g., they cannot compute future ephemeral keys based on that key).

- **Definition 5** *Future secrecy*

  If an active attacker steals a long-term key, they cannot decrypt later messages. This is the most difficult to implement and achieve.

Using authenticated Diffie-Hellman, as well as signatures and MACs, we can set up a system of ephemeral keys that can protect against forward secrecy and backward secrecy attacks.



The diagram illustrates a client-server relationship, for example, a home computer visiting Amazon. Note that only the server has their public key verified and only the server is signing their messages. We assume that the existing public key infrastructure is safe and no attackers have issued themselves a legitimate certificate (i.e. the only way to impersonate Alice is to specifically find out Alice's private signing key).

1. The long-term key in this case is Alice's private key, $prKA$. It is only used for authentication, e.g. signing messages sent during the ephemeral key exchange. It is never used for encryption.

2. Alice (the server) and Bob (the client) now use Diffie-Hellman to agree on an ephemeral session key. Again, they each pick random secrets a, b and send each other $(g^a \mod p), (g^b \mod p)$. For trust establishment, Alice also signs her $(g^a \mod p)$ with $prKA$.

3. Bob can verify that $(g^a \mod p)$ indeed came from Alice and not someone else because it is signed with her private key. They both locally compute a common, correct $(g^{ab} \mod p)$ as in Diffie-Hellman.

4. This common key will be used as ephemeral keys. The diagram displays a_1, a_2 and b_1, b_2 because in this case we actually compute two ephemeral keys per period P: a K_E for encryption and a K_{MAC} for message authentication. The former prevents an attacker from reading the message; the latter prevents an attacker from changing the message without detection.

5. K_E and K_{MAC} are recomputed every set period P. Each time, Alice signs her side of the exchange with $prKA$. Both Alice and Bob throw away their random secrets a, b once the common secret has been computed.

Without $prKA$, Mallory cannot impersonate Alice. Thus, Alice can send her messages safely and securely. Since Bob does not sign his messages, however, Mallory can still attempt to impersonate Bob, dropping all their packets and sending $(g^m \mod p)$ instead.

However, as this is a client-server setup, it is likely that there is some other login and password system set up. Mallory impersonating Bob would therefore be useless without knowing Bob's password. Mallory cannot steal Bob's password by pretending to be Alice (we ignore other methods of attack such as phishing), whose messages are signed.

If this were a user-user setup, for example, Alice and Bob are individuals texting each other directly, then both of them would have to sign their messages to prevent either one of them from being impersonated.

Assuming any MiTM attacks fail, the ephemeral keys should be secure, i.e. neither a passive nor active attacker can figure out $g^{ab} \mod p$. Without the encryption key, no attacker can read the messages. Nor can any attacker change the messages undetected. If we were just using hashes, Mallory could compute a hash on her own message and replace both message and hash with the forgeries; the hash would be valid even if the message did not decrypt properly. However, Mallory cannot do the same to a MAC without K_{MAC}.

If an attacker to compromise the long-term private signing key, the previous ephemeral keys (and thus the previous messages) are safe because those keys were computed independently of the long-term key. Just knowing how those exchanges were signed does not aid in finding $g^{ab} \mod p$, especially since a, b were thrown away. Thus forward secrecy is achieved.

Eve would not be able to determine any future keys. Remember that Eve can only try to decrypt and read messages, and not actively send her own. Knowing Alice's long-term signature key should give her no information about a, b, and thus no information on the ephemeral keys. Thus backward secrecy is achieved.

However, knowing the long-term key will allow Mallory to man-in-the-middle future ephemeral keys. Unlike Eve, Mallory can drop and send messages; thus, knowing $prKA$ allows her to impersonate Alice. The next time Alice and Bob try to exchange an ephemeral key, Mallory can generate her own ephemeral secret m and send $sig_{prKA}(g^m \mod p)$, and trick Bob into using $g^{mb} \mod p$ (which she actually knows) as the ephemeral key. Thus future secrecy is NOT achieved.

5 Transport Security

Efficient obfuscation and full anonymity are currently extremely difficult, as proper confidentiality and authentication rely on some form of public agreement and identity verification. Even services such as Tor are not completely perfect. This will be discussed in a later lecture.

References

[1] Unger, Nik, et al., "SoK: Secure Messaging." *SP '15 Proceedings of the 2015 IEEE Symposium on Security and Privacy*, 2015, pp. 232–249., doi:10.1109/SP.2015.22.