MODULE 2.3 - Protocols

Overview

Protocols define the sequence of events between parties and the rules that are to be followed for the purpose of accomplishing some set of goals. Protocols can range from incredibly complex to incredibly simple. The key to evaluating protocols is do they meet their stated goals. A protocol to provide integrity may not provide confidentiality, which may be ok, if confidentiality is not important. When designing or selecting protocols it's crucial to be aware of your assumptions and to identify the protocol's goals correctly and completely.

A Simple Protocol

Consider this simple protocol.

Trustworthy Trent operates a timestamping service that provides proof of when a document was created. For example, suppose Alice has a document with supporting evidence for her patent application. Alice wants her document timestamped to prove when her designs existed.

1. Alice transmits a copy of the document to Trent
2. Trent records the date and time he received the document and stores that along with a copy of the document for reference. If anyone questions when the document was created, they could contact Trent who could verify the timestamp and that the document presented matches the document stored.

What are some security or operational concerns with this protocol?

Confidentiality of the document - The protocol does not address the confidentiality of the document being time stamped. Integrity of the document - The protocol does not address any measures to ensure the integrity of the document being timestamped. If the document is modified or tampered with after it were timestamped, the timestamp would no longer tbe valid.

Some concerns could include:

- No privacy.
- Database of documents would be huge.
- No integrity checking of what was timestamped.
- Loss of Trent's database would invalidate all timestamps.
- It could be used by persons' not as trustworthy as Trent.

What are some strengths?

Simplicity - Simple and straight forward protocol, which is easy and low cost to implement. Non-repudiation - The protocol provides non-repudiation because the timestamp of the document cannot be denied or repudiated by the sender of the receiver. This is because Trent is the trusted third party and can be verified independently.

No keys needed to be exchanged.

What could be some attacks on this protocol that would invalidate the service?

Timestamp tampering - An attacker could intercept the document and modify the timestamp. Man-in-the-middle attack - An attacker could intercept communication and impersonate. Document substition - Intercepting and changing out the document. Denial-of-service attack - A DoS attack could render Trent's timestamping business unavailable, preventing Alice from getting her document timestamped.

- Hack the clock
- Hack Trent’s database
- Destroy Trent’s database

How could this protocol be improved? State any particular goals or assumptions.

Use a digital signature, transfer file over an encryted channel, hash the files in the database (to ensure integrity).

If instead of sending her document to Trent, Alice sends the hash of her document to timestamp, in what ways does this improve the protocol?

Ensure the integrity of the file while in transit / in storage & prevents attackers from breaching confidentiality.

What problems remain?

Communication channels need to be encrpyted.

Message Exchange

Alice uses the following protocol to send a document to Bob.

1. Alice and Bob exchange public keys
2. Alice and Bob agree on a protocol for message exchange
3. With M being Alice's document, Alice generates $ C = S_{A}(T||H(M) || E_{B}(M)) $ and sends $ C $ to Bob.

What steps would Bob take to recover and validate Alice's document?

Bob verifies the message using Alices's PK. Bob separates the message into a timestamp, message digest, and the encrypted message. Bob decrypts the message. Bob generates the digest and compares it will Alices'.

1. *Bob Verifies the message using Alice's Public Key*
2. *Bob separates the message into, a timestamp $ T,$ a message digest $ H(M) $ and the encrypted message, $ E*{B}(M) $_
3. *Bob decrypts the message $ D*{B}(E_{A}(M)) = M $_
4. *Bob generates the digest of $ M' = H(M) $*
5. *Bob compares $ M' $ and $ M $ and if the same, he knows he has the message came from Alice and it was as she sent it.*

Why is Bob sure it came from Alice? What assumptions are you making?

It was signed with Alice's private key (PK) that only she would know.

It was signed with Alice's private key. We assume that only Alice has access to that key.

Why is Bob sure the document received is the document Alice sent?

The hash of her unencrpyte document is included in the message. And the message matches the hash of the decrypted document.

The hash of her unencrypted document included in the message matches the hash of the decrypted document.

In practice documents would not be sent being encrypted using asymmetric encryption. What would be done instead, and why?

Because asymmetric encryption is slow, Alice would generate and send a session key to be used with symmetric encryption. The document would be encrypted with that key and the encrypted document sent to Bob.

Asymetric encrpytion is a lot slower. Hybrid cryptography encryption would be more efficient (using asymetric encrpytion with symetric encrpytion).

Alice's Robots

Alice is broadcasting encrypted messages to her team of 16 robots. The messages include the robot's ID number (0-15), and a command move forward, move back, move right, move left, turn right, turn left, pick up, or put down). The message format is M = [ID || Command].

Alice encrypts the messages using the shared public key for the robot team.

Bob wants to know what Alice is sending to her robots. And its pretty simple.

How can he do this?

Hint: using the above method, how many possible different messages will Alice send?

There are 128 possible messages. It would not prevent Bob from keeping track of the messages and associating them with what the robots are doing.

If instead of sending M, would sending H(M) prevent Bob from knowing what she was sending.

No, because there would still be 128 possible messages, just will look different.

How would (or would not) Alice signing the message, either S(A,M) or S(A,H(M)), improve security?

It would not change anything because there would still only be 128 different possible messages.

What simple change would you make to M to make this communication secure?

Alice could salt the hashes to make the robot communication more secure.

Describe your solution using standard notation.

EA(M,H(salt,M))