# Computer Security – Tutorial 2: Protocols Solutions

School of Informatics

8th February 2014

## Part A: Where is the secret and where is the trust?

1. The **password** is the secret. Demonstrating knowledge of the secret is good enough to authenticate as Alice. The email address (user name) is used in public often, so it has to be considered known to attacker.

   Thus, the starting assumptions are that:

Confidentiality During registration, Alice has transmitted her password to the Gmail server securely (i.e., nobody learned it during registration);

Authenticity The server was talking to Alice during registration

The assumptions during operation are that:

Confidentiality Alice does not tell anyone else the password.

Confidentiality The Gmail server does not ever reveal Alice's password to anyone else.

(a) Noone else (and no machine) can easily guess Alice's password.

Authenticity During login, Alice can be sure to talk to Gmail.

Confidentiality The password transmision to Gmail is secure.

Authenticity In order to convince Gmail to have the password, one needs to know the password

Assuming Alice and Gmail both want to maintain these conditions, they trust one another to do so.

As countermeasures for Alice to protect her password, the common advice is that Alice should not write it down anywhere or store it on a machine, and ideally she should not reuse the same password in lots of places (in case one of them reveals it). However, there is a critical trade-off with **usability**: passwords that are difficult to guess are hard to remember, and having many different passwords compounds this.

As countermeasures for Gmail, the common advice is to use one-way functions to conceal the password when it is stored in a database, along with *salt* which may include a random number and the user id. Furthermore Gmail uses HTTPS with SSL certificates to prove their identity during the communication.

2. Here again, the **password** is secret and the same countermeasures appear to apply.

   However, the challenge of only 3 alphanumeric positions is rather weak, in that it reduces the number of possible passwords considerably, to $36^3$ (about $2^{15.5}$) rather than e.g., $94^8$ for an 8-character ASCII password, which is about 52 bits. This means that it would be easier to guess such a password in a **brute-force** attack.

   As an additional countermeasure, the bank probably applies strong **rate-limiting** to lock-out users after a certain number of login attempts.

   To check the challenge is correct and yet keep a database which is safe from attack, the bank will either have to store hashes which are computed from every possible challenge, or revert to storing plain-text passwords.

   Sometimes the **bank account number** is considered to be secret, and there has been confusion around this in the payments industry and with consumers, as for example Jermey Clarkson discovered in 2008: http://news.bbc.co.uk/1/hi/entertainment/7174760.stm.

3. Plain Diffie-Hellman key agreement is **anonymous**, and establishes a shared secret (which can be used as a key) $g^{xy} \bmod p$. This is created by the messages $g^x \bmod p$ sent by the server and $g^y \bmod p$ sent back by Alice, where $x$ and $y$ are two **private secrets** not revealed to anyone else. The assumptions are that:

confidentiality Alice keeps her secret $y$ private and does not reveal it

confidentiality The Server keeps her secret $x$ private and does not reveal it

   (a) The parameters used for Diffie-Hellman key exchange are appropriately chosen: avoiding certain bad choices, and making sure key sizes are large enough to make the brute force attack infeasible during the lifetime of the key.

integrity The send messages have not been changed.

Plain Diffie-Hellman means that **neither Alice nor the Server is authenticated** to one another. Moreover, **a middleperson attack is possible** in that an encrypted communication may go via an unknown third party.

In practice, anonymous Diffie Hellman is rarely used. Instead, what usually happens is that public Diffie Hellman parameters are contained in a signed package (with a signature that can be verified using the server's *certificate*), thus the Server is authenticated to Alice, but not vice-versa.

## Part B:

1. The **assumptions** are:

   - Keys are shared with the server; Alice, Bob and the Server each believe these are properly kept secret.
   - The timestamps in each run of the protocol are fresh.
   - $A$ believes that $K_{ab}$ is a good key to share with $B$
   - $B$ trusts Alice to generate a good key to use
   - $B$ trusts the server to genuinely relay messages from $A$ (and not an imposter)

   The Wide Mouthed Frog protocol makes the unusual assumption that the principals are trusted to generate good keys (rather than leaving this responsibility to the server).

2. (a) If the lifetime of a key can be extended indefinitely, the key may be compromised (broken or stolen) and so the attacker can read messages (e.g., confidential medical data).

   (b) The intruder masquerades as Bob and Alice alternately, and the time stamp is continually updated:

$$\begin{array}{lll} \text{Message 3.} & M(B) \to S: & B, \{\, T_s, A, K_{ab} \,\}_{K_{bs}} \\ \text{Message 4.} & S \to A \quad : & \{\, T'_s, B, K_{ab} \,\}_{K_{as}} \\ \text{Message 5.} & M(A) \to S: & A, \{\, T'_s, B, K_{ab} \,\}_{K_{as}} \\ \text{Message 6.} & S \to B \quad : & \{\, T''_s, A, K_{ab} \,\}_{K_{bs}} \end{array}$$

   (c) This attack is not possible under the some additional assumptions which are sometimes taken to be implicit, in particular, that principals "recognise and ignore their own messages".

   In message 3, a message from $S$ is reflected.

   An easy fix is to change format of one of the messages, to prevent this reflection attack.

3. (a) In this attack, the intruder can replay messages but not necessarily decrypt them. The replay alone could be damaging if otherwise undetected, e.g. in the case of a money transfer order.

   (b) This attack should not be possible because $B$ should check that $T_s$ is "later than any other time-stamp received from $S$," so the repetition should be recognised. (Anyway, a possible fix is to include a nonce-handshake in extra messages).

## Part C: A Multi-party Key Exchange Protocol

This question tests your ability to read and understand what a protocol is trying to achieve.

1. At the start you should be able to state the assumptions that all parties have concerning the keys, e.g., shared keys are known only to the parties that share them, private keys are known only to their owners, etc. After that, you should consider how beliefs of principals change *after each message is received*. Because messages can be faked by attackers, we only build up authentication beliefs in response to *successfully met fresh challenges*. Challenges relate to demonstrating knowledge of keys or shared secrets. An assumption that is sometimes made is that *participants behave honestly*, but that is sometimes inaccurate!

   In this case, here are how the beliefs change during a run of protocol 1:

   - Message 1.0: A believes card $C_s$ is present
   - Message 1.1: B believes card $C_s$ in in ATM identified by $A_s$
   - Message 1.2: A believes B sent PRN
   - Message 1.3: U believes it is communicating with the real bank
   - Message 1.4: No belief change
   - Message 1.5: B believes U is authorised to access the account.

   And for protocol 2:

   - Message 2.0 No belief change
   - Message 2.1 B believes Card $C_s$ is in ATM identified by $A_s$
   - Message 2.2 No belief change
   - Message 2.3 U believes it is communicating with the real Bank
   - Message 2.4 No belief change
   - Message 2.5 B believes U is authorised to access the account.

   Note that with protocol 2 A doesn't hold any beliefs during a run of the protocol, all A does is pass information between the card and the bank. (while adding some information in message 2.1)

2. In this part of the question you need to think practically. You are the attacker, what do you want and what parts of the system can you gain access too? The goal of the attack is simply "free money" (aka: Convince the back you are authorised to access an account). It would make sense to mount the attack from the users point of view. You will have access to all the information the user sends/receives and can then impersonate the user.

3. In protocol 1 as $C_s$ and the PIN are both sent un-encrypted to the terminal it is simple to intercept them, store them and send them again at a later date (a replay attack). The random numbers are there for the user's benefit, not the bank. The imposter has to send back a new random number with the PIN. If the bank accepts the same random number that it sent then the real user wouldn't notice this attack (apart from the lack of money in her account) though if the imposter has to send back a different number the real user wouldn't be able to access her account the next time she tried.

   With protocol 2 it is almost exactly the same. We need to record message 2.0 and 2.4 then replay them when we are impersonating the user. As there are no random numbers in this protocol the real user would only notice this attack when she checks her balance.

4. How do we create a better protocol? It is fairly obvious that protocol 2 is going to be a better place to start as all communication is encrypted so even with the current replay attack the attacker never discovers $C_s$ or the PIN. All we need to do is implement a defence against a replay attack. The simplest way to do this is for Message 2.0 to include a nonce and make sure that the bank checks that nonce against all previously seen nonces.