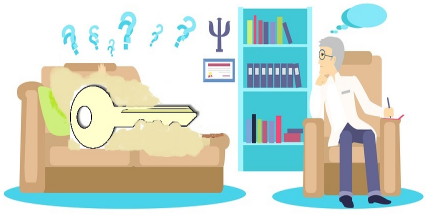
Crypto 13

Note: this material is not intended to replace the live lecture for students.

Contents



13.1 Key Establishment, Distribution and Authentication in networks

Key establishing deals with establishing a shared secret between two or more parties. So is going to be handled by protocols.

Methods are of two types: *key transport* and *key agreement*.

To keep in mind:

Key establishing is strongly related to identification.

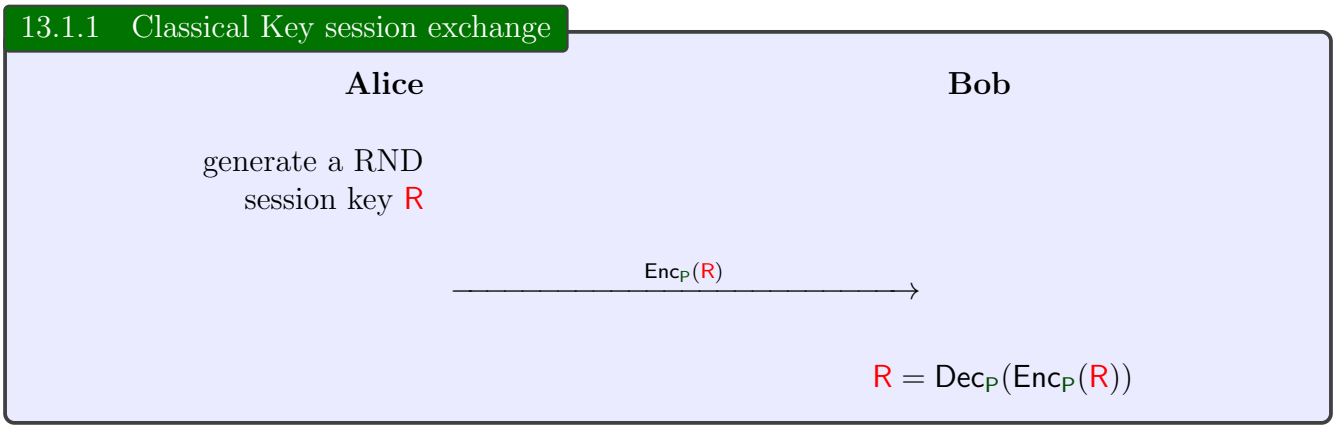
That people pick bad passwords usually words of their language. So to protect users with weak passwords is a very good goal.

Compromised keys due to: negligence, design flaw of the system i.e. an intruder break into the parties computers and steal a key.

13.1.1 Classical Key negotiation

Alice and Bob share a common password P. Using it they want to establish a secure key session.

13.1.1 Classical Key session exchange



Now they share R and can use it as a session key. For example Bob can replies to Alice

$$\mathsf{Enc}_R(\text{Terminal type:})$$

These protocol above is insecure and have flaws. Here some of them:

A eavesdropper could record messages and run a dictionary attack against P. Namely, he can try a candidate P' to decipher $\mathsf{Enc}_P(R)$. So he get the result R' and use it to decipher Enc_R(Terminal type:) looking for expected english words (redundancy).

NOTE 13.1.2

Thus here the main weakness: the above can be done off-line e.g. by brute-force.

Secondary weakness: replay attacks (the attacker may use old messages). Thus protocols must incorporate safeguards i.e. random challenges.

13.1.2 Key Freshness and KDF

In many situations we need keys which are valid for a limited time.
Such keys are called session keys or ephemeral keys.

Limiting the time has several vantages:

1) Less damage if the key is exposed.

2) Less material for crypto attacks e.g. differential cryptanalysis.

3) The attacker is forced to recover several keys for decipher large messages.

NOTE 13.1.3

Applications to: voice encryption in GSM cell phones and video encryption in pay-TV satellite systems. Keys generated within minutes or even seconds.

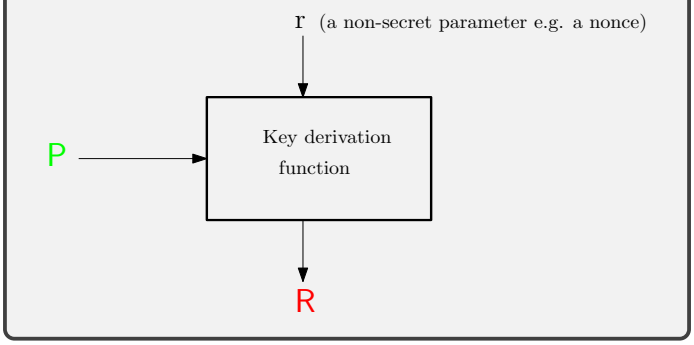
How keys can be updated ?

Two ways:

a) to run over and over again a protocol

b) to use a password (an already established joint secret) to derive fresh keys through a Key Derivation Function (KDF).

Figure 13.1.4: Key Derivation Function



a KDF function should be one-way to prevent the attacker to deduce P in case the session keys R become compromised.

Here is a protocol using a symmetric cipher (Enc, Dec) and a nonce.

13.1.5 Key Derivation with Nonces

Alice		Bob
		generate a nonce r
	$\xleftarrow{\quad r \quad}$	
derive session key $R = \mathsf{Enc}_P(r)$		derive session key $R = \mathsf{Enc}_P(r)$

NOTE 13.1.6

Alternatives:
$\mathsf{R} = \mathsf{HMAC}_{\mathsf{P}}(r)$
or with a counter cnt in the place of the nonce r: $\mathsf{R} = \mathsf{Enc}_{\mathsf{P}}(\mathsf{cnt})$.

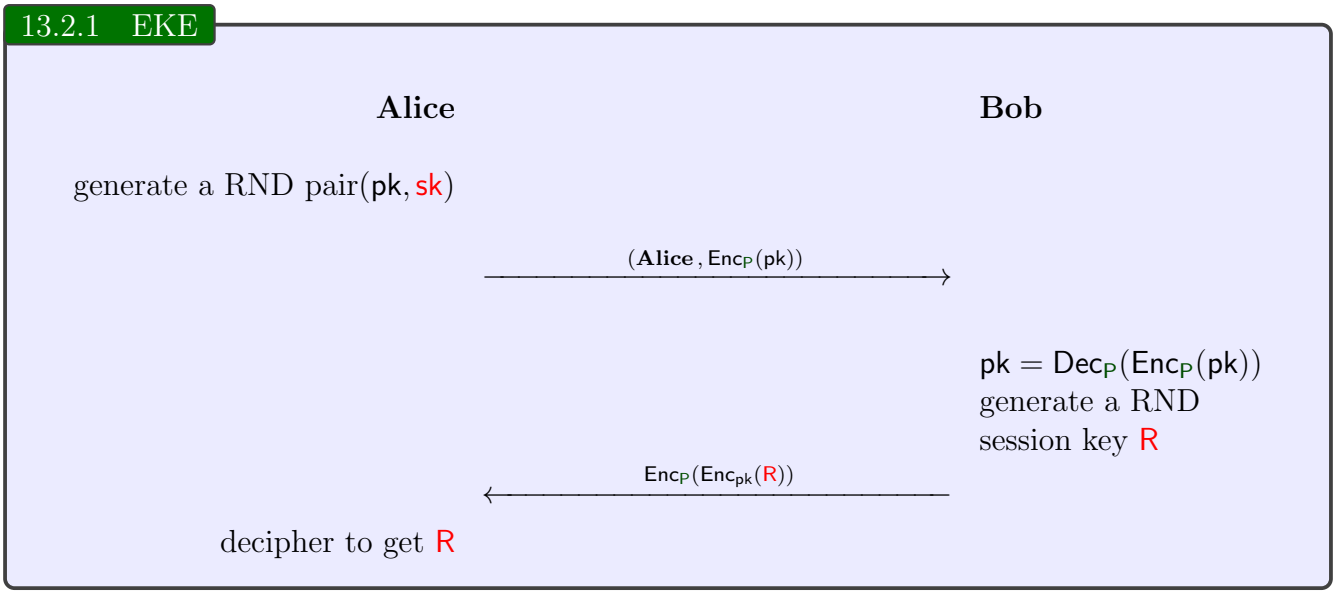
Using the counter can save Alice and Bob one communication session, since no r needs to be transmitted. But this holds when both knows when the next key derivation need to take place. Otherwise the counter must be synchronized.

13.2 Encrypted Key Exchange (EKE)

This protocol is a combination of symmetric and asymmetric cryptography.

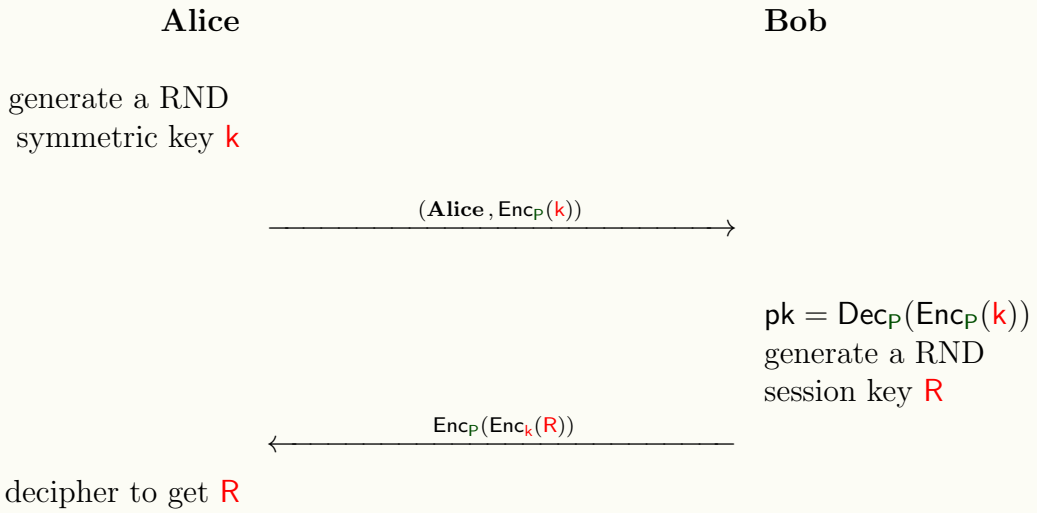
Alice and Bob share a common password P. Using it they want to establish a secure key session.

13.2.1 EKE



Exercise 13.2.2

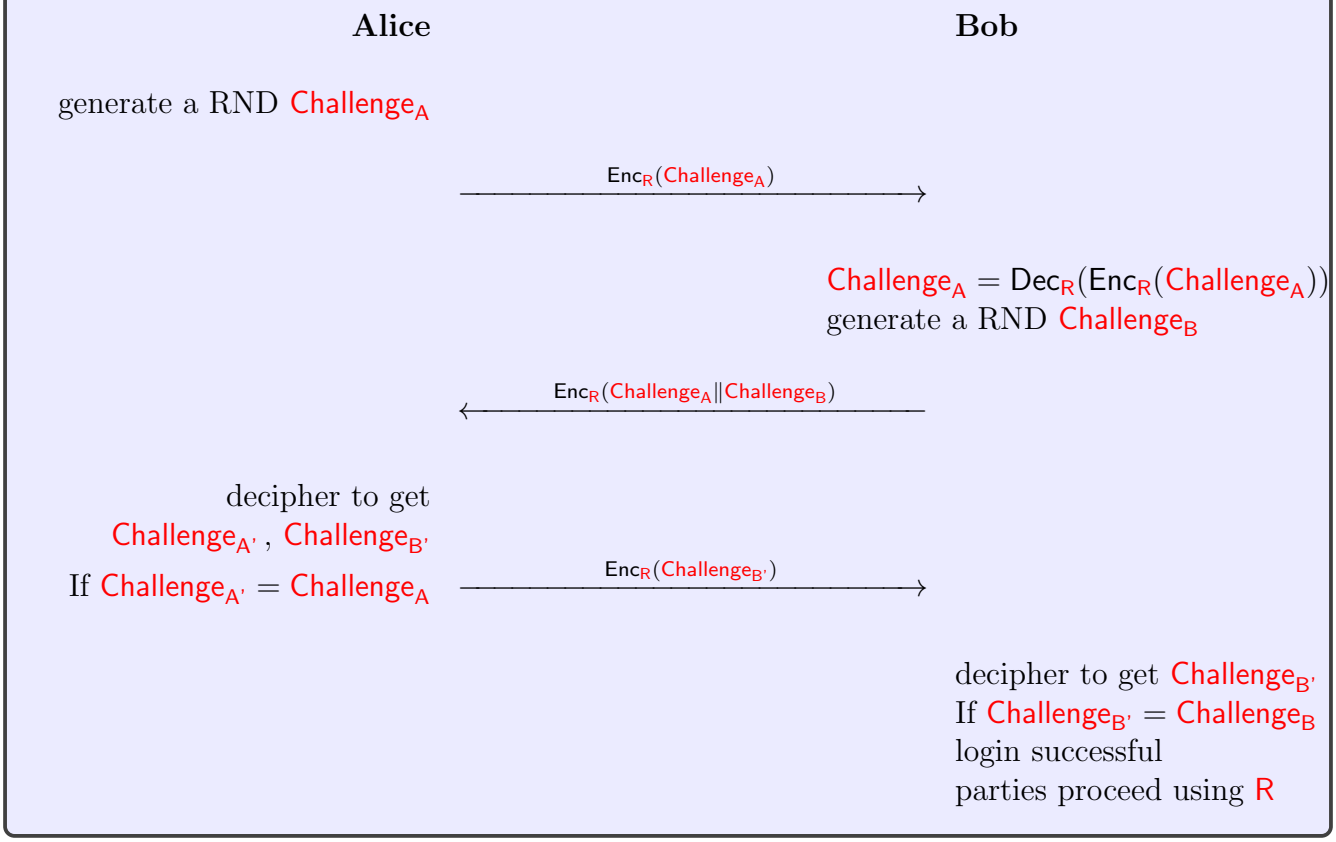
Why it is necessary to use an asymmetric cipher? i.e. why the below protocol does not work:



13.2.1 Validation of keys

Here is a standard technique for validating cryptographical keys:

13.2.3 Challenge-Response: Validation of Crypto-Keys



NOTE 13.2.4

If a party sends $\mathsf{Challenge_C}$ encipher with R, where $\mathsf{Challenge_C}$ was never used before, and receives another encipher message containing $\mathsf{Challenge_C}$ in reply, it follows that the message originator has the ability to encipher messages with R.

13.2.2 Key Exchange Key (KEK)

Suppose that a cryptanalyst has recovered some session key R (e.g. by a trojan or by using the record of an old session protected by the protocol 13.2.1). Then it can attack P e.g. dictionary and off-line attack. Indeed, now for $\mathsf{P'}$ in the dictionary we compute

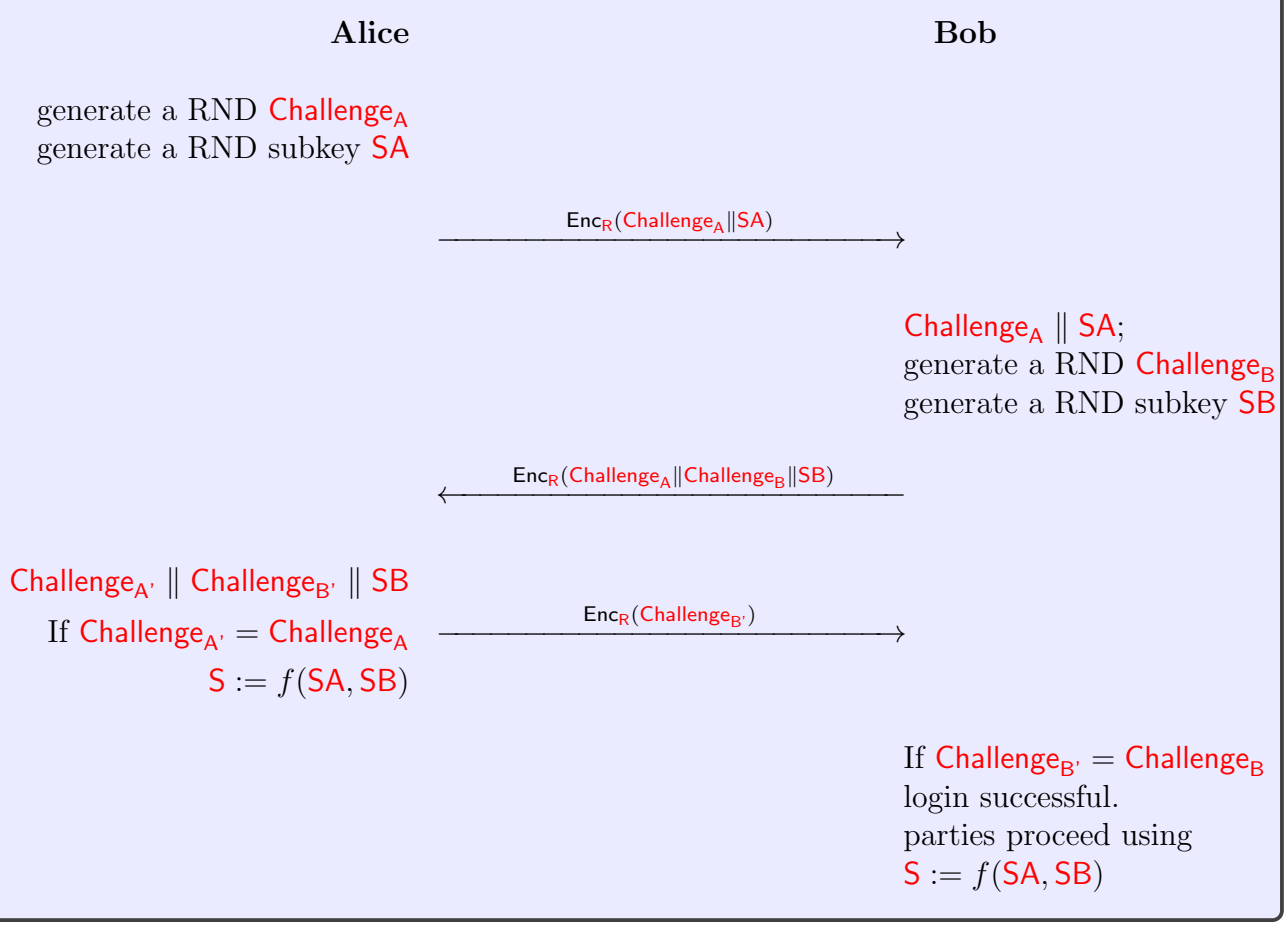
$$\mathsf{pk'} := \mathsf{Dec_{P'}}(\mathsf{Enc_P}(\mathsf{pk}))$$

and check

$$\mathsf{Enc_P}(\mathsf{Enc_{pk}}(\mathsf{R})) \stackrel{?}{=} \mathsf{Enc_{P'}}(\mathsf{Enc_{pk'}}(\mathsf{R}))$$

To prevent this a true session key S can be obtained with a minor variation of the Challenge-Response 13.2.3 portion of the protocol. The R is now called a key exchange key (KEK).

13.2.5 Challenge-Response: KEK



NOTE 13.2.6

Observe that if a cryptanalyst recover some session key S then it can not attack P as explained above.

Exercise 13.2.7

Have a look at [Schneier15, Section 22.5, page 518].

13.2.3 EKE using RSA and Partition Attacks

2.5.1 Partition attacks

We have stated that the encryptions using P must leak no information. This is often quite difficult, simply because of the numerical properties of the cryptosystems used. For example, we noted that public keys in RSA are always odd; if no special precautions are taken, an attacker could rule out half of the candidate values P' if $P'^{-1}(P(e))$ were an even number. At first blush, this is an unimportant reduction in the key space; in fact, if left uncorrected, it can be a fatal flaw.

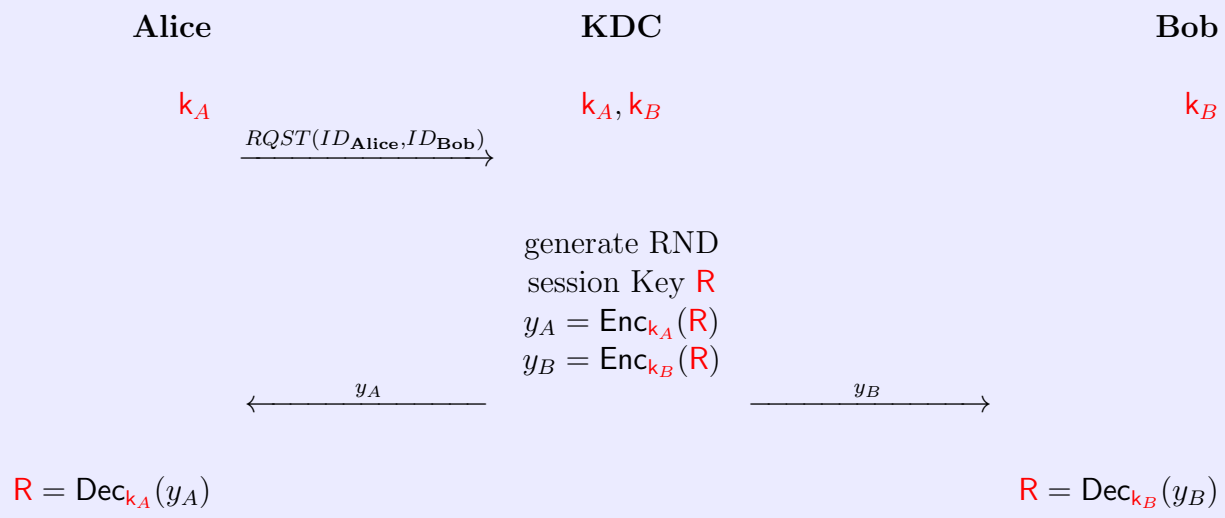
Recall that each session will use a different public key, independent of all others previously used. Thus, trial decryptions resulting in illegal values of e' will exclude different values of P' each time. Put another way, each session will partition the remaining candidate key space into two approximately-equal halves. The decrease in the keyspace is logarithmic; comparatively few intercepted conversations will suffice to reject all invalid guesses at P. We call this attack a *partition attack*.

[BeMe92, Bellovin & Merrit]

13.3 Key Distribution Center (KDC)

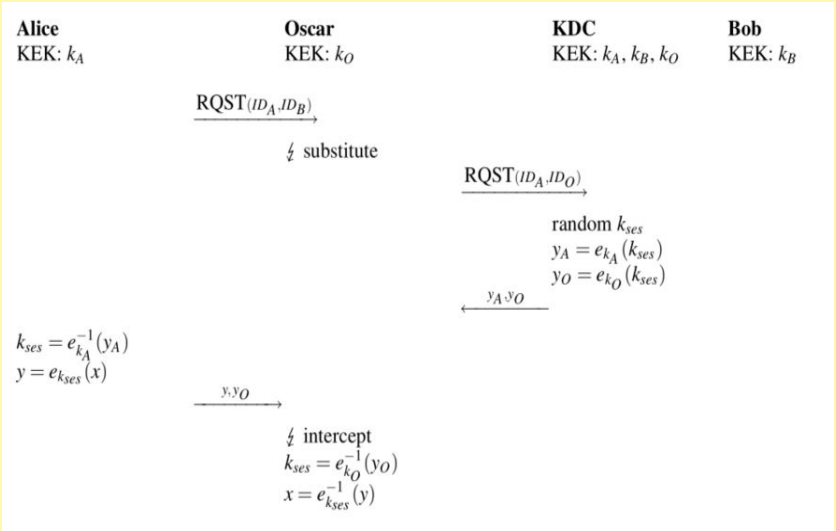
A KDC is a server that is fully trusted by all users and that shares a secret key k_U with each user. k_U is called Key Encryption Key (KEK)).

13.3.1 Key Distribution Center



NOTE 13.3.2

Even if the above protocol protect against passive attacks it is really weak to active attacks:
1) Replay Attack. As Denning and Sacco: If an attacker uses an older, compromised value for R, he can then replay the either y_A o y_B.
2)(lack of) Key Confirmation Attack: Oscar a legitimate (but malicious) can intercept Alice request to the KDC change Bob identification $ID_{\mathbf{Bob}}$ and "impersonate" the KDC:



NOTE 13.3.3

Even though Kerberos provides strong assurance that the correct keys are being used and that users are authenticated, there are still drawbacks to the protocols discussed so far. We now describe remaining general problems that exist for KDC-based schemes.

Communication requirements One problem in practice is that the KDC needs to be contacted if a new secure session is to be initiated between any two parties in the network. Even though this is a performance rather than a security problem, it can be a serious hindrance in a system with very many users. In Kerberos, one can alleviate this potential problem by increasing the lifetime T of the key. In practice, Kerberos can run with tens of thousands of users. However, it would be a problem to scale such an approach to "all" Internet users.

Secure channel during initialization As discussed earlier, all KDC-based protocols require a secure channel at the time a new user joins the network for transmitting that user's key encryption key.

Single point of failure All KDC-based protocols, including Kerberos, have the security drawback that they have a *single point of failure*, namely the database that contains the key encryption keys, the KEKs. If the KDC becomes compromised, all KEKs in the entire system become invalid and have to be re-established using secure channels between the KDC and each user.

No perfect forward secrecy If any of the KEKs becomes compromised, e.g., through a hacker or Trojan software running on a user's computer, the consequences are serious. First, all future communication can be decrypted by the attacker who eavesdrops. For instance, if Oscar got a hold of Alice's KEK k_A, he can recover the session key from all messages y_A that the KDC sends out. **Even more dramatic is the fact that Oscar can also decrypt past communications if he stored old messages y_A and y.** Even if Alice immediately realizes that her KEK has been compromised and she stops using it right away, there is nothing she can do to prevent Oscar from decrypting her *past* communication. Whether a system is vulnerable if long-term keys are compromised is an important feature of a security system and there is a special terminology used:

13.3.4 Forward Secrecy

A protocol has perfect forward secrecy (PFS) if the compromise of long-term keys does not allow an attacker to obtain past session keys.

To assure PFS is necessary to employ asymmetric techniques.

Exercise 13.3.5

Convince yourself that protocol 13.2.1 has PFS. Keep in mind that the long-term key is P.

Exercise 13.3.6

Read the "Wide-Mouth Frog" protocol at [Schneier15, page 56]. Have a look at "Lessons Learned" at [Schneier15, page 64].

13.3.7 Needham-Schroeder

Here, Alice (A) initiates the communication to Bob (B). S is a server trusted by both parties. In the communication:

- A and B are identities of Alice and Bob respectively
- K_{AS} is a symmetric key known only to A and S
- K_{BS} is a symmetric key known only to B and S
- N_A and N_B are nonces generated by A and B respectively
- K_{AB} is a symmetric, generated key, which will be the session key of the session between A and B

The protocol can be specified as follows in security protocol notation:

$A \to S : A, B, N_A$

Alice sends a message to the server identifying herself and Bob, telling the server she wants to communicate with Bob.

$S \to A : \{N_A, K_{AB}, B, \{K_{AB}, A\}_{K_{BS}}\}_{K_{AS}}$

The server generates K_{AB} and sends back to Alice a copy encrypted under K_{BS} for Alice to forward to Bob and also a copy for Alice. Since Alice may be requesting keys for several different people, the nonce assures Alice that the message is fresh and that the server is replying to that particular message and the inclusion of Bob's name tells Alice who she is to share this key with.

$A \to B : \{K_{AB}, A\}_{K_{BS}}$

Alice forwards the key to Bob who can decrypt it with the key he shares with the server, thus authenticating the data.

$B \to A : \{N_B\}_{K_{AB}}$

Bob sends Alice a nonce encrypted under K_{AB} to show that he has the key.

$A \to B : \{N_B - 1\}_{K_{AB}}$

Alice performs a simple operation on the nonce, re-encrypts it and sends it back verifying that she is still alive and that she holds the key.

The last two steps are the **handshake** Cf. challenge-response 13.2.5.

NOTE 13.3.8

The Denning-Sacco Attack occurs when an intruder captures the session key K from an eavesdropped session and uses it either to gain the ability to impersonate the user directly or to conduct a brute-force search against the user's password.

Exercise 13.3.9

Explain a replay attack to the Needham-Schroeder protocol. That is to say, show that the protocol is not secure if when session keys are compromised.

Exercise 13.3.10

Read [Schneier15, 24.7 SESAME, page 572].

13.4 Hopper-Blum's Identification Protocols

The scenario is the following:

-The user and computer share a secret.
-The computer randomly challenge the user.
-The user responds in such a way that an adversary cannot easily learn the secret.

3.1 Learning Parity in the Presence of Noise

Suppose the secret shared between the human and the computer is a vector $\mathbf{x}$ of length n over $GF(2)$. Authentication proceeds as follows: The computer, C, generates a random n-vector $\mathbf{c}$ over $GF(2)$ and sends it to the human, H, as a challenge. H responds with the bit $r = \mathbf{c} \cdot \mathbf{x}$, the inner product over $GF(2)$. C accepts if $r = \mathbf{c} \cdot \mathbf{x}$. Clearly on a single authentication, C accepts a legitimate user H with probability 1, and an impostor with probability $\frac{1}{2}$; iteration k times results in accepting an impostor with probability 2^{-k}. Unfortunately, after observing $O(n)$ challenge-response pairs between C and H, the adversary M can use Gaussian elimination to discover the secret $\mathbf{x}$ and masquerade as H.

Suppose we introduce a parameter $\eta \in (0, \frac{1}{2})$ and allow H to respond incorrectly with probability η; in that case the adversary can no longer simply use Gaussian elimination to learn the secret $\mathbf{x}$. This is an instance of the problem of *learning parity with noise* (LPN). In fact the problem of learning $\mathbf{x}$ becomes NP-Hard in the presence of errors; it is NP-Hard to even find an $\mathbf{x}$ satisfying more than half of the challenge-response pairs collected by M [8]. Of course,

[HB01, page 56]

https://towardsdatascience.com/where-machine-learning-meets-cryptography-b4a23ef54c9e

13.5 Bibliography

Books I used to prepare this note:

[Paar10] Paar, Christof, Pelzl, Jan, *Understanding Cryptography, A Textbook for Students and Practitioners*, Springer-Verlag, 2010.

[Schneier15] Bruce Schneier, *Applied Cryptography: Protocols, Algorithms and Source Code in C* ,Wiley; 20th Anniversary edition,2015.

Papers I used to prepare this note:

[BeMe92] S. Bellovin and M. Merritt; *Encrypted Key Exchange: Password-based Protocols Secure Against Dictionary Attacks*, Proc of the Symposium on Security and Privacy, pages 72–84, IEEE, 1992 https://www.cs.columbia.edu/~smb/papers/neke.pdf

[DeSa81] D.E. Denning and G. M. Sacco; *Timestamps in key distribution protocols*, Communications of the ACM. 24 (8): 533-535. (1981) http://pages.cs.wisc.edu/~remzi/Classes/736/Spring2005/Papers/data-encryption-denning.pdf

[HB01] Nicholas J. Hopper and Manuel Blum; *Secure Human Identification Protocols*, Advances in Cryptology — ASIACRYPT 2001, 52–66,Springer Berlin Heidelberg. https://link.springer.com/chapter/10.1007/3-540-45682-1_4

and some interesting links:

https://www.youtube.com/watch?v=iaH8UG2yMg4

https://en.wikipedia.org/wiki/Kerberos_(protocol)

https://en.wikipedia.org/wiki/Security_protocol_notation

https://en.wikipedia.org/wiki/Simultaneous_Authentication_of_Equals

http://homes.di.unimi.it/visconti/PBKDF2.pdf

https://en.wikipedia.org/wiki/Needham%E2%80%93Schroeder_protocol

http://www.cs.unc.edu/~fabian/course_papers/needham.pdf

http://pages.cs.wisc.edu/~remzi/Classes/736/Spring2005/Papers/data-encryption-denning.pdf

http://srp.stanford.edu/