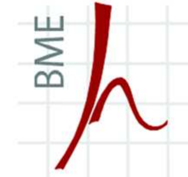
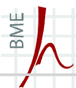
MŰEGYETEM 1782



Session key establishment protocols

Security Protocols (bmevihim132)

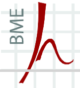
Dr. Levente Buttyán
associate professor
BME Hálózati Rendszerek és Szolgáltatások Tanszék
Lab of Cryptography and System Security (CrySyS)
buttyan@hit.bme.hu, buttyan@crysys.hu



Outline

- Motivations and design objectives
- Basic concepts and techniques (via an example)
- Some examples taken from the literature
- Password based key exchange

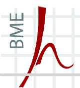
© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



Motivation

- communicating parties must share a secret key in order to use symmetric key cryptographic algorithms (e.g., block ciphers, stream ciphers, and MAC functions)
- it is desired that a different shared key is established for each communication session → session key
  - to ensure independence across sessions
  - to avoid long-term storage of a large number of shared keys
  - to limit the number of ciphertexts available for cryptanalysis
- we need mechanisms that allow two (or more) remote parties to set up a shared secret in a dynamic (on-demand) manner → session key establishment protocols

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

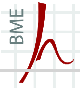


Design objectives

at the end of the protocol

- Alice and Bob should learn the value of the session key K (**effectiveness**)
- no other parties (with the possible exception of a trusted third party) should know the value of K (**implicit key authentication**)
- Alice and Bob should believe that K is freshly generated (**key freshness**)
- optionally, Alice should believe that Bob knows the key K, and vice versa (**key confirmation**)

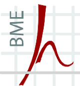
© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



Adversary model

- the underlying cryptographic primitives used in the protocol are secure (the adversary attacks the protocol itself, not the applied crypto primitives)
- the adversary has full control over the communications of the honest parties
  - can eavesdrop, modify, delete, inject, and replay messages
  - can coerce honest parties to engage into protocol runs
- the adversary may be an external party (an outsider), or a legitimate protocol participant (an insider), or the combination of those
- the adversary may obtain old session keys

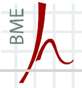
© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



Basic classification of protocols

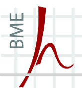
- key transport protocols
  - one party (typically a trusted third party) creates a new session key, and securely transfers it to the other parties
- key agreement protocols
  - the session key is derived by the parties as a function of information contributed by each, such that no party can predetermine the resulting value of the key

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

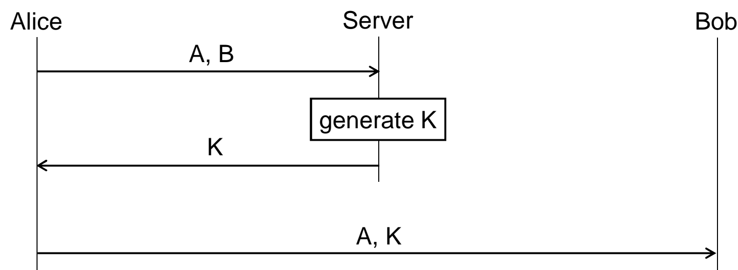


Further protocol characteristics

- reciprocity
  - guarantees are provided unilaterally or mutually
- efficiency
  - number of message exchanges (passes) required
  - total number of bits transmitted (i.e., bandwidth used)
  - complexity of computations by each party
  - possibility of pre-computations to reduce on-line computational complexity
- third party requirements
  - on-line, off-line, or no third party at all
  - degree and type of trust required in the third party
- system setup
  - distribution of initial keying material

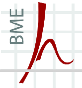


A naïve key transport protocol

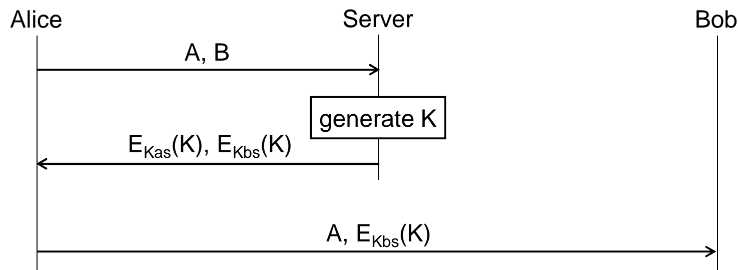


most obvious problem:

- the adversary can eavesdrop K
- implicit key authentication is not provided



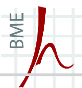
Second attempt



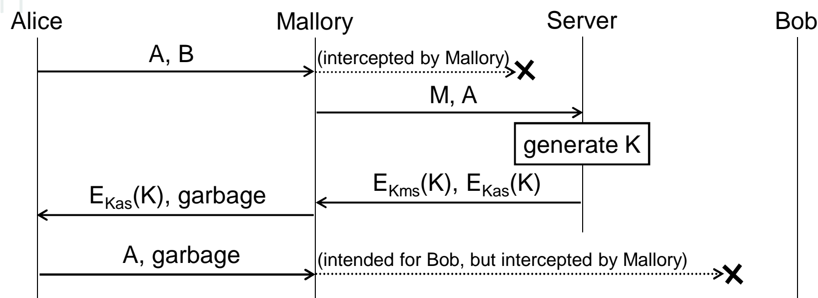
problems:

- Alice cannot be sure that K has been created for the session between herself and Bob
- similarly, Bob cannot be sure that he shares K with Alice
- implicit key authentication is still not provided
- …

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



An attack on the second attempt



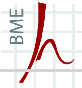
notes:

- typical *man-in-the-middle (MitM) attack*
- Alice believes that she shares K with Bob, but she shares it with the adversary

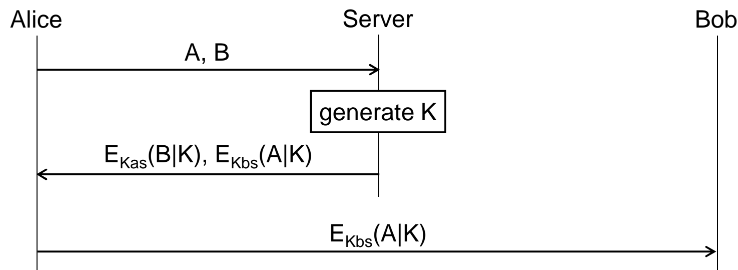
derived design principle:

- ***if the name of a party is essential to the meaning of a message, then it must be mentioned explicitly in the message***

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



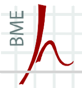
Third attempt



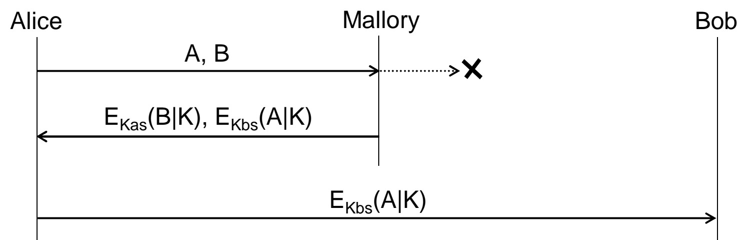
problem:

- neither Alice nor Bob can be sure that K is fresh
- no key freshness is provided

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



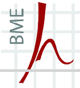
An attack on the third attempt



notes:

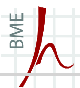
- typical *replay attack*
- if K is compromised by the adversary, then she can decrypt follow-up communications between Alice and Bob
- even if K is not compromised, the adversary can replay encrypted messages to Alice and Bob from the past session where K was used

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



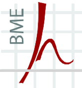
How to achieve key freshness?

- use timestamps
- use random nonces (nonce = number used once)
- don't use counters
- use a key agreement protocol



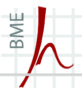
Timestamps

- $E_{Kas}(B \mid K \mid T_s)$, where T_s is the current time on the clock of S
- key is accepted only if the timestamp is within an acceptable window of the current time at the receiver
- can provide strong assurances, but require synchronized clocks
- notes on the importance of clock synchronization:
  - if a party's clock is slow, then (s)he may accept old (possibly replayed) messages
  - if a party's clock is advanced, then (s)he may generate messages that will be considered fresh *in the future* (although they may be dropped near the time of their generation)
- secure clock synchronization usually requires communication with a trusted time source
  - freshness of the time synch protocol messages cannot be ensured by timestamps (the synchronizing party does not know the time and therefore cannot produce and verify timestamps)



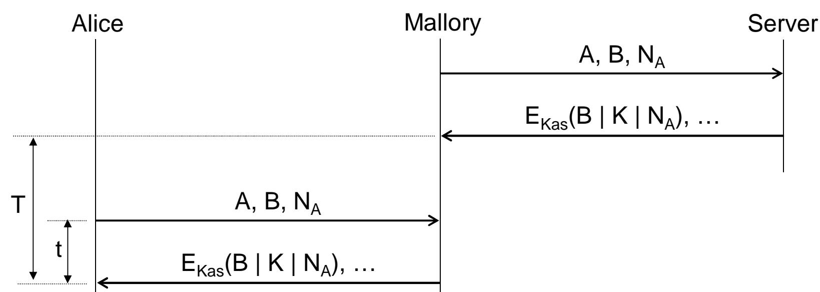
Random nonces

- $E_{Kas}(B | K | N_A)$, where N_A is a fresh and *unpredictable* random number generated by A (and sent to S beforehand)
- key is accepted only if the time that elapsed between sending the nonce and receiving the message containing the nonce is acceptably short
- less precise than a timestamp (exact time of key generation is not known), but it provides sufficient guarantees of freshness in most practical cases
- it requires an extra message to send the nonce, and some temporary state to store the nonce for verification purposes

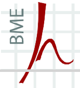


Random nonces

- important warning:
  if nonces were predictable, the adversary could obtain a message containing a future nonce of Alice, which would later be considered as fresh by Alice



Alice believes that K is younger than t, while in fact, it is older than T

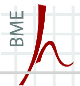


Counters

- two parties A and B can maintain a synchronized counter
  - counter value at A is C_{AB}
  - counter value at B is C_{BA}
- when A receives a message $E_{Kab}(\ldots|C)$,
  - she accepts it only if $C > C_{AB}$
  - if the message is accepted, then C_{AB} is set to C
- this ensures message ordering but no freshness
  - receiver knows that an accepted message has been generated later than the previous accepted messages, but she doesn't know how old the messages are

→ counters alone are <u>not</u> appropriate for providing key freshness

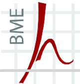
- parties need to maintain permanent state



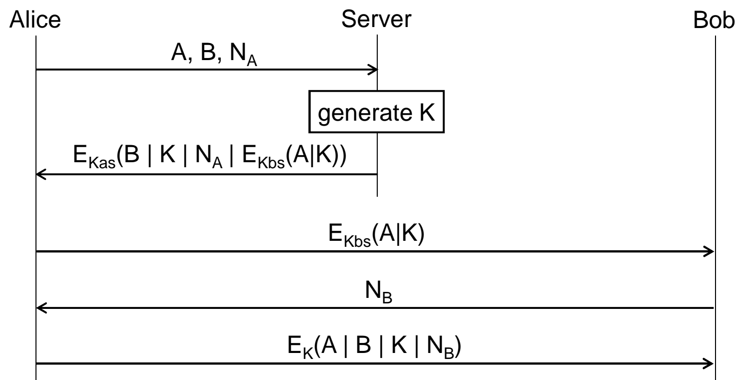
Key freshness with key agreement

- $K = f(k_A, k_B)$, where k_A and k_B are the contributions of Alice and Bob, respectively
- if f(x, .) is a one-way function (for any x), then once Alice has chosen k_A, Bob cannot find any k_B, such that $f(k_A, k_B)$ has a pre-specified value (e.g., an old session key)
- similarly, if f(., y) is a one-way function (for any y), then once Bob has chosen k_B, Alice cannot find any k_A, such that $f(k_A, k_B)$ has a pre-specified value

→ if the contribution of a party is fresh, then (s)he can be sure that the resulting session key is fresh too

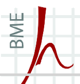


Fourth attempt

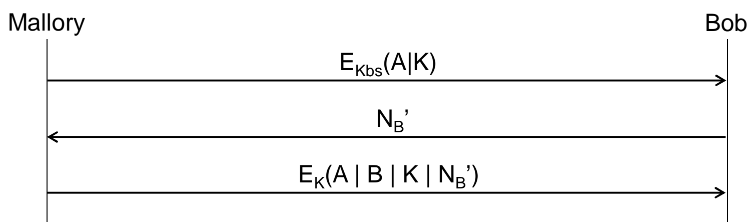


notes:

- nested encryption provides key confirmation for Bob
- this protocol is similar to the well-known Needham-Schroeder protocol (symmetric key)
- seemingly correct, but …



An attack on the fourth attempt

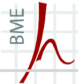


notes:

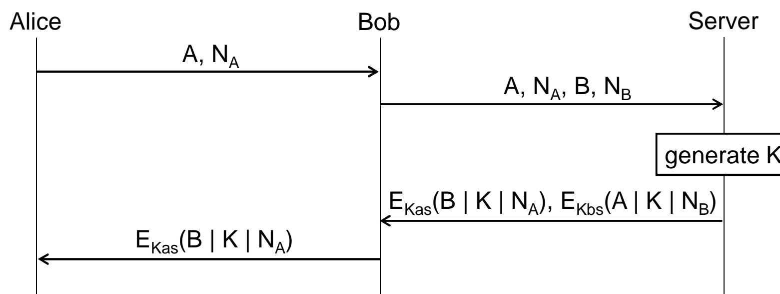
- K is an old session key that is compromised by the adversary
- $E_{Kbs}(A|K)$ is replayed from the old protocol run (where K was established as the session key)
- B will believe that he established a session with A, but A is not present

derived design principles:

- ***the fact that a key K is used recently to encrypt a message does not mean that K is fresh***
- ***when proving the freshness of a key K by binding it to some fresh data (timestamp or nonce), don't use K itself for the binding***

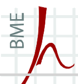


Fifth attempt



- any problems?

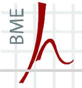
© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



Protocol engineering checklist

- be explicit
  - interpretation of messages shouldn't depend on context information, but it should be based solely on the content of the messages
  - include **names** that are needed to correctly interpret the message
  - consider including protocol type, run identifier, and message number to avoid protocol interference, interleaving, and message reflection attacks, respectively
- think twice about key freshness
  - decide on how you want to ensure key freshness for the different participants
  - consider the advantages and disadvantages of nonces and timestamps in a given application environment
- state assumptions
  - explicitly state all the assumptions on which the security of your protocol depends so that someone who wants to use your protocol can verify if they hold in a given application environment

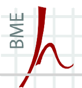
© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



Examples from the literature

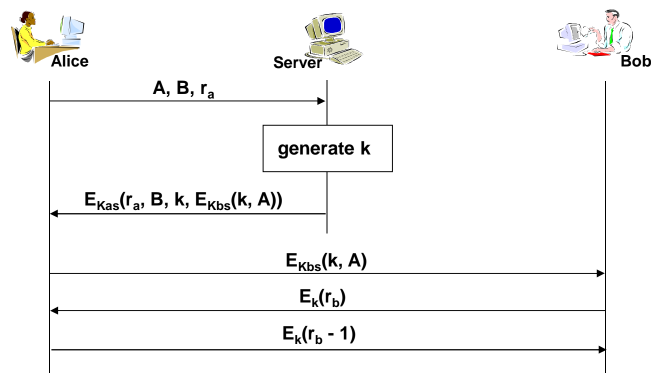
- key transport
  - symmetric-key
    - (symmetric-key) Needham-Schroeder
    - Kerberos
    - Wide-Mouth-Frog
    - Otway-Rees
  - asymmetric-key
    - public-key Needham-Schroeder
    - ISO 11770-3 protocols
- key agreement
    - Diffie-Hellman
    - Station-to-Station

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



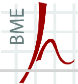
The Needham-Schroeder protocol

summary: Alice requests a session key from the Server; the Server generates the key and sends it to Alice and to Bob via Alice; Alice and Bob performs entity authentication and key confirmation



characteristics: mutual entity authentication (flawed), mutual explicit key authentication (flawed), key freshness with fresh random numbers (flawed), on-line third party trusted for generation of session keys, initial long-term keys between the parties and the server are required

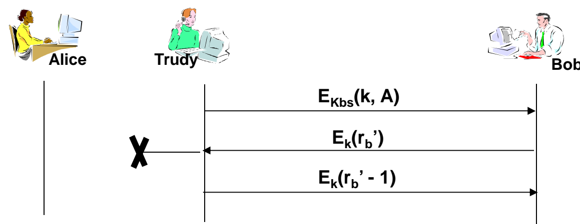
© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



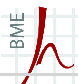
The flaw in the NS protocol

assumption: Trudy recorded a successful run of the protocol and compromised the session key k; thus, she knows k and $E_{Kbs}(k, A)$

summary: Trudy masquerades as Alice to Bob and makes Bob accept the old and compromised session key k

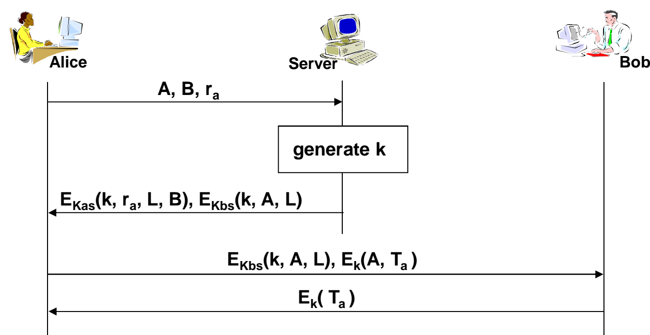


© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



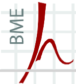
The Kerberos protocol

summary: essentially a correction of the Needham-Schroeder protocol
the protocol is optimized with respect to the original Needham-Schroeder protocol (fewer messages and no double encryption in the second message)



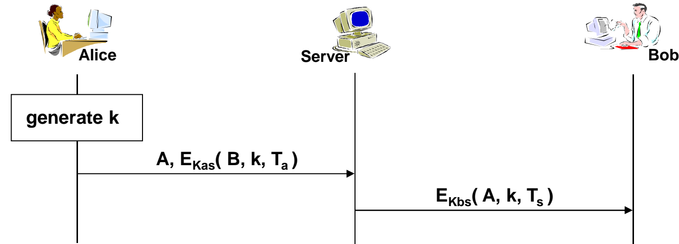
characteristics: mutual entity authentication, mutual explicit key authentication, key freshness with a nonce and with a lifetime value, clock synchronization is required, on-line third party trusted for generation of session keys, initial long-term keys between the parties and the server are required

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

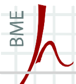


The Wide-Mouth-Frog protocol

summary: a simple key transport protocol that uses a trusted third party
Alice generates the session key and sends it to Bob via the trusted third party

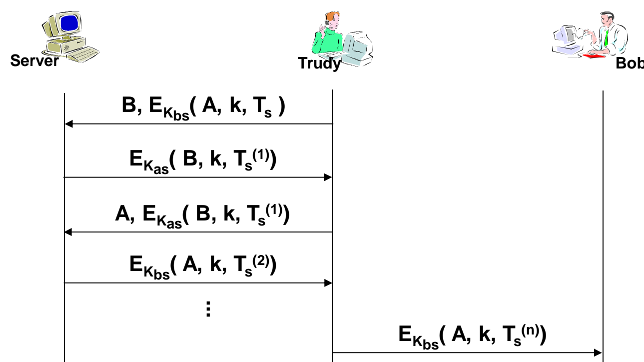


characteristics: key control for Alice
implicit key authentication for Alice
explicit key authentication for Bob (flawed)
key freshness for Bob with timestamps (FLAWED!)
unilateral entity authentication of Alice (flawed)
on-line third party (Server) trusted for secure relaying of keys and
verification of freshness,
in addition A is trusted for generating good keys
initial long-term keys between the parties and the server are required

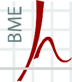


A flaw in the WMF protocol

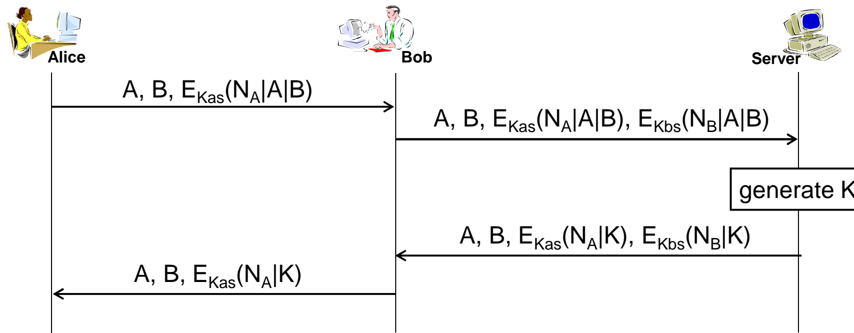
summary: after observing one run of the protocol, Trudy can continuously use the Server as an oracle until she wants to bring about re-authentication between Alice and Bob



The Otway-Rees protocol

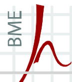


summary: this protocol is similar to our fifth attempt
note that names are omitted in the server's response, because A and B have already been bound to N_A and N_B by the encryption in the first two messages (not a recommendable practice, though)

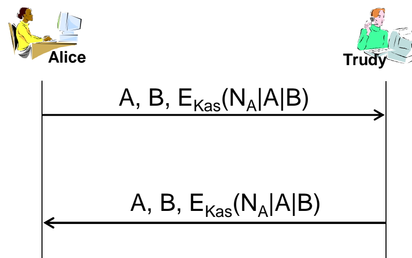


characteristics: mutual implicit key authentication, no entity authentication, key freshness with fresh random numbers, on-line third party trusted for generation of session keys and verification of matching identifiers in the received encrypted blocks, initial long-term keys between the parties and the server are required

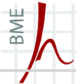
A typing attack on Otway-Rees



summary: Trudy sends back Alice's first message to Alice (reflection attack), Alice accepts this as the fourth message of the protocol, due to the structural similarity between the first and the fourth message, Alice interprets A|B as the new session key, but this is also known to Trudy



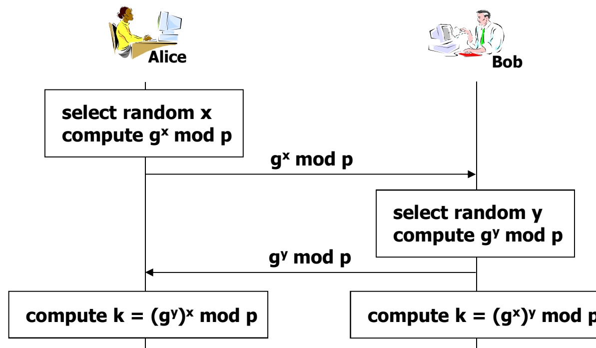
note: reflection attacks can be avoided by using direction bits in messages, even better if the protocol is designed in such a way that it is possible to tell about any message which protocol's which message it is, type identifiers in messages can also be useful, in order to be sure that no typing attack is possible



The Diffie-Hellman protocol

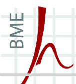
summary: a **key agreement** protocol based on one-way functions; in particular, security of the protocol is based on the hardness of the discrete logarithm problem and that of the Diffie-Hellman problem

assumptions: p is a large prime, g is a generator of Z_p^*, both are publicly known system parameters



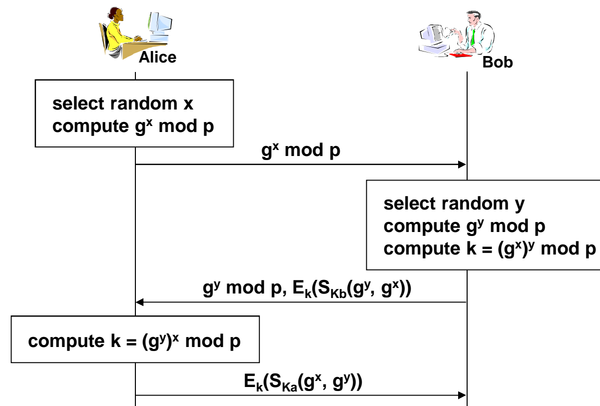
characteristics: NO AUTHENTICATION, key freshness with randomly selected exponents, no party can control the key, no need for a trusted third party

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



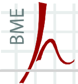
The Station-to-Station protocol

summary: three-pass variation of the basic Diffie-Hellman protocol; it uses digital signatures to provide mutual entity authentication and mutual explicit key authentication



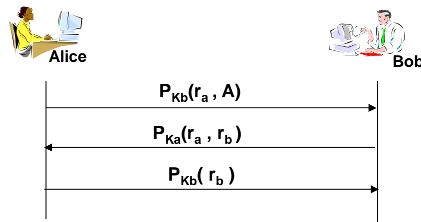
characteristics: mutual entity authentication, mutual explicit key authentication, key freshness with random exponents, no party can control the key, off-line third party for issuing public key certificates may be required, initial exchange of public keys between the parties may be required

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



Public-key Needham-Schroeder

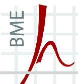
summary: originally a challenge-response type mutual authentication protocol based on public-key encryption only (no signatures); however, since the random numbers exchanged never appear in clear, it was suggested to derive a session key from them



key derivation: both party computes $k = f(r_a, r_b)$

characteristics: mutual entity authentication, mutual implicit key authentication (flawed), no key confirmation, key freshness with random numbers, no party can control the key, off-line third party for issuing public key certificates may be required, initial exchange of public keys between the parties may be required

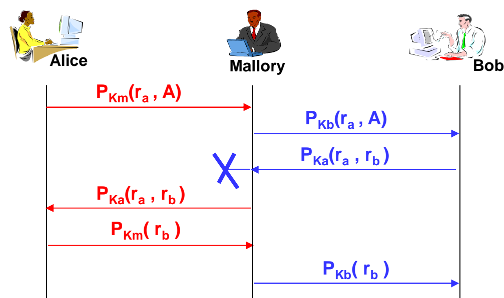
© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



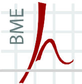
Lowe's attack

assumption: Mallory is a malicious user, his public key is K_m

summary: when Alice starts the protocol with Mallory, he can masquerade as Alice to Bob; Mallory uses Alice as an oracle to decrypt a message received from Bob; if the protocol is used for key establishment, then Bob falsely believes that he shares a secret key with Alice, but indeed he shares it with Mallory

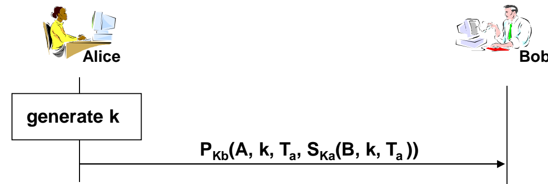


© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



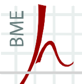
Encrypting signed keys (ISO 11770-3/3)

summary: Alice generates a session key, signs it, then encrypts it with Bob's public key, and sends it to Bob



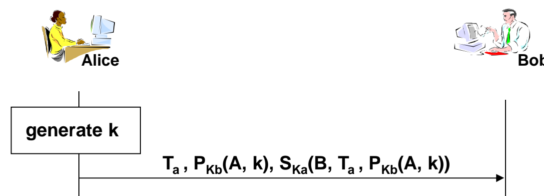
characteristics: unilateral entity authentication (of Alice), mutual implicit key authentication, key confirmation for Bob, key freshness with timestamp, clock synchronization needed, off-line third party for issuing public key certificates may be required, initial exchange of public keys between the parties may be required, Alice is trusted to generate keys, non-repudiation guarantee for Bob

notes: the ID of Bob in the signature prevents Bob from sending the signed key on to another party and impersonating Alice;
the ID of Alice in the encrypted message is a hint for Bob that helps him to choose the right key for verification of the signature.



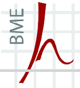
Signing encrypted keys (ISO 11770-3/2)

summary: Alice generates a session key, encrypts it with Bob's public key, then signs it, and sends it to Bob



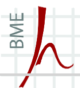
characteristics: unilateral entity authentication (of Alice), mutual implicit key authentication, no key confirmation, key freshness with timestamp, clock synchronization, off-line third party for issuing public key certificates may be required, initial exchange of public keys between the parties may be required, Alice is trusted to generate keys, non-repudiation guarantee for Bob

notes: an advantage of this protocol over the "encrypting signed keys" protocol is that here less data is encrypted (almost surely fits in the block size)



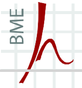
Lessons learnt

- many protocols were proposed in the literature, but most of them have been found flawed later on (sometimes years after the publication of the protocol)
- flaws are often very subtle and hard to find
- adherence to the protocol engineering principles of slide 22 would have prevented many of these flaws
  - see also the Abadi-Needham paper on Prudent Engineering Practice for Cryptographic Protocols (→ last slide)
- considerable amount of research was done on applying formal methods to model and verify key establishment protocols
  - logics, theorem provers, process calculi, model checking, …
  - formal methods are less error-prone, because they allow for systematic analysis
  - however, the modeling step still requires human creativity (and hence prone to errors)



Password based key exchange

- assume that two parties (e.g., a user and a server) share a password (relatively weak secret)
- how to set up a cryptographic key (strong secret) with the help of this password?



A naïve solution

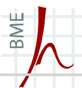
- Alice can generate a key K and encrypt it with the password pwd (or its hash value):

$$A \rightarrow B: \ A, E_{H(pwd)}(K)$$

- Bob can use the hash of the password to obtain K from $E_{H(pwd)}(K)$, and then use K to encrypt messages for Alice
- for example:

$$B \rightarrow A: \ E_K(\text{“Last login at 16:34, Monday”})$$

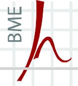
© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



The problem

- (key freshness is not provided by the naïve protocol, but it could be added by including a timestamp)
- if a weak password is used, then the naïve solution is vulnerable to an **off-line dictionary attack**:
  - assume that the attacker eavesdropped a protocol run
  - for each candidate password pwd?, compute the candidate key K? = $D_{H(pwd?)}(E_{H(pwd)}(K))$
  - test K? by checking if $D_{K?}(E_K(\text{“Last login …”}))$ is a meaningful message
  - if so, then pwd? is Alice's password, otherwise throw away pwd? and try a new candidate password from the dictionary

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



Encrypted Key Exchange (EKE)

- Alice generates a public key / private key pair K^+ and K^-, and encrypts K^+ with the (hash of the) password pwd:

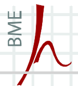
$$A \rightarrow B: \; A, E_{H(pwd)}(K^+)$$

- Bob uses the (hash of the) password to obtain K^+, then generates a (symmetric) key K, and encrypts it with K^+ in the public key cryptosystem; the result is further encrypted with the (hash of the) password:

$$B \rightarrow A: \; E_{H(pwd)}(AE_{K+}(K))$$

- Alice uses the (hash of the) password and K^- to obtain K from $E_{H(pwd)}(AE_{K+}(K))$; then she can use K to send messages to Bob:

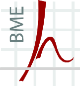
$$A \rightarrow B: \; E_K(\text{“Last login at 16:34, Monday”})$$



Why is this good?

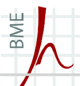
- for a candidate password pwd?, the attacker can compute a candidate public key $K^+?$ as $D_{H(pwd?)}(E_{H(pwd)}(K^+))$
- but $K^+?$ cannot really be tested
  - the attacker needs to find a key K? such that
    - $AE_{K+?}(K?) = D_{H(pwd?)}(E_{H(pwd)}(AE_{K+}(K)))$
    - $D_{K?}(E_K(\text{“Last login …”}))$ makes sense
  - both would require an exhaustive search over the key space from which K is chosen (or breaking the symmetric or the asymmetric cipher)

→ the relatively small space of passwords is thus multiplied by the large key space from which K is chosen (privacy amplification effect)



What about key freshness?

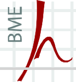
- as Bob generates K, key freshness is provided for Bob
- for Alice K^+ is fresh, and this guarantees freshness of K through the encryption $AE_{K+}(K)$ (assuming that Alice trusts Bob for generating fresh session keys)
  - Alice can conclude that someone who knows the password (which can only be Bob) has recently sent K to the other holder of the password (which can only be Alice)



Implementing EKE with RSA

- reminder on RSA
  - public key: (e, n), where n = pq and e is relatively prime to (p-1)(q-1)
  - private key: $d = e^{-1}$ (mod (p-1)(q-1))
  - encryption: $c = m^e \bmod n$
  - decryption: $m = c^d \bmod n$

- problem with encrypting (e, n) in the first message
  - attacker can compute (e?, n?) = $D_{H(pwd?)}(E_{H(pwd)}((e, n)))$, and check if n? has a small prime factor
  - if pwd? is not the correct password, then n? is random and it has a small prime factor with high probability
  - thus, the attacker can throw away wrong password candidates easily

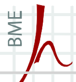


Implementing EKE with RSA

- how about encrypting only e ?
  - note that e must be odd (as (p-1)(q-1) is even)
  - the attacker can compute e? = $D_{H(pwd?)}(E_{H(pwd)}(e))$, and if e? is even, then she can throw away pwd?
  - doing this for ~log(dictionary size) protocol runs narrows down the set of possible passwords to a singleton

  → e needs special encoding:
    - before encrypting add 1 to e with probability ½ → obtain e'
    - when receiver decrypts, she should subtract 1 from the result if it is even
    - attacker can get even and odd numbers with the same probability no matter if she used the right or a wrong password candidate
  - can e? = $D_{H(pwd?)}(E_{H(pwd)}(e'))$ be distingushed from a random odd number?
    - if p = 2p' + 1 and q = 2q' + 1, then the overwhelming majority of the odd integers (mod n) are relatively prime to (p-1)(q-1)
    - thus, almost all e? could be the good exponent, and the attacker cannot throw away wrong password candidates

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

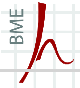


Implementing EKE with RSA

- how about not encrypting (e, n) at all?
  - then, an attacker can impersonate Alice, and select p, q, compute n, and choose an e that is **not** relatively prime to (p-1)(q-1)
  - Bob cannot verify if e is correct, because he does not know (p-1)(q-1)
  - in this case, the space C of possible cryptograms is just a fraction of [0, n-1], and with the knowledge of p and q, the attacker can tell from any value if it is in C
  - Bob chooses K, and computes $E_{H(pwd)}(K^e \bmod n)$
  - the attacker computes, for each candidate password pwd?, $D_{H(pwd?)}(E_{H(pwd)}(K^e \bmod n))$ and checks if the result is in C
  - if not, then pwd? can be thrown away

- summary:
  - encrypt only the exponent e in the first message
  - no need to encrypt (with the password) the second message

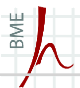
© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



Implementing EKE with ElGamal

- reminder on ElGamal
  - private key: a
  - public key: $A = g^a \bmod p$
  - encryption: $(R = g^r \bmod p, C = mA^r \bmod p = mg^{ar} \bmod p)$
  - decryption: $C/R^a \bmod p = mg^{ar}/g^{ra} \bmod p = m$
- if a and r are random, then A and (R,C) are random
  - →fits nicely to EKE
  - →seems that either encryption with the password can be omitted
  - however, an attack may be possible if the second message is sent in clear

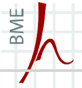
© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



Implementing EKE with ElGamal

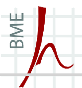
- assuming that the second message is sent in the clear
  - Alice sends in the first message $E_{H(pwd)}(A)$
  - the attacker chooses r', and sends (R', X) back to Alice, where X is just a random string
  - Alice computes $K' = X/R'^a$, and sends $E_{K'}$(“Last login at 16:34, Monday”)
  - note that $K' = X/R'^a = X/A^{r'}$, however, the attacker does not know A
  - still, for any candidate password pwd?, she has a candidate $A? = D_{H(pwd?)}(E_{H(pwd)}(A))$, and hence a candidate $K'? = X/A?^{r'}$
  - the attacker can test K'? by trying to decrypt the third message $E_{K'}$(“Last login at 16:34, Monday”)
  - if decryption fails, then the password candidate pwd? can be thrown away
- summary: unlike in the case of an RSA based implementation, the second message needs to be encrypted, while the first one may not be encrypted

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem



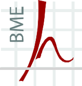
Selecting the symmetric-key cipher

- if a block cipher is used in CBC mode, then using non-random padding is vulnerable
  - for any candidate password pwd?, the attacker can try to decrypt with H(pwd?) and check if the padding obtained is valid or not
- even weak encryption could be used (e.g., encrypt the public key by XORing H(pwd) to it)
  - public key can be correctly recovered only by some one who knows the password → authentication
  - the public key XORed with the password is random → confidentiality of the password (~one-time pad)



Lessons learnt

- passwords are very widely used, but they are often weak (coming from a limited "dictionary"); still cleverly designed protocols can use them to set up strong cryptographic secrets in a secure way
- encrypting a public-key with a symmetric key may seem to be counter-intuitive for the first sight
  - note that the reverse (encryption of a symmetric key with a public key) is the standard hybrid approach for efficient public-key encryption
- there are many pitfalls in the implementation of the basic idea
  - one cannot just consider the public key cryptosystem as a black box, but needs to understand the details of its operation



Recommended reading

- R. Anderson and R. Needham, Programming Satan's computer, In *Computer Science Today*, Springer LNCS 1000, 1995.

- M. Abadi and R. Needham, Prudent engineering practice for cryptographic protocols, In *IEEE Transactions on Software Engineering*, 22(1), 1996.

- R. Anderson and R. Needham, Robustness principles for public-key protocols, In *Advances in Cryptology -- CRYPTO'95*, Springer, 1995.

- S. Bellovin, M. Merritt, Encrypted Key Exchange: Password-based Protocols Secure Against Dictionary Attacks, In *IEEE Symposium on Security and Privacy (Oakland)*, 1992.

  → and follow-up papers

© Buttyán Levente, Híradástechnikai Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem