Session key establishment protocols

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

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**)

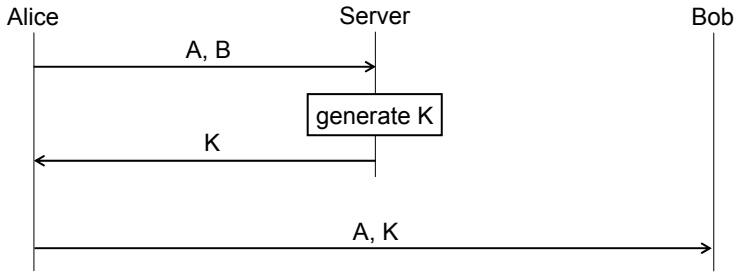
Adversary model (extended Dolev-Yao)

- the underlying cryptographic primitives used in the protocol are secure
- however, the adversary may obtain old session keys
- 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 a legitimate protocol participant (an insider), or an external party (an outsider), or the combination of both

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

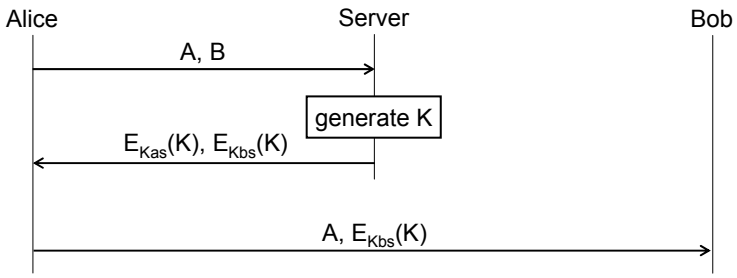
First attempt for a key transport protocol



most obvious problem:

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

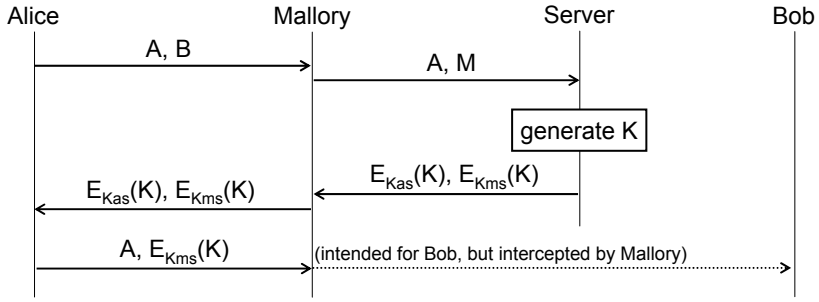
Second attempt



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
- …

An attack against the second attempt



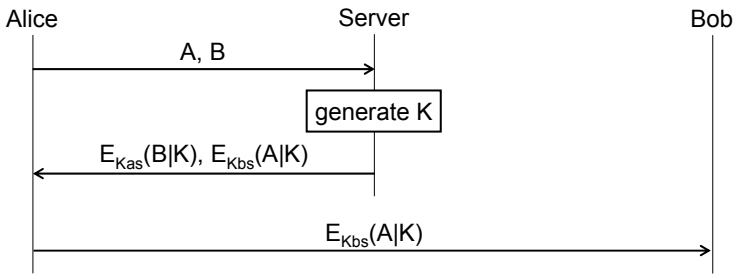
notes:

- 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***

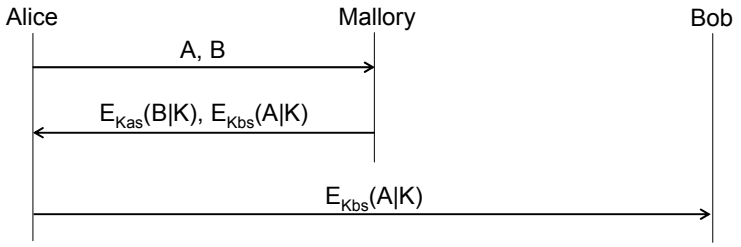
Third attempt



problem:

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

An attack against 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

How to achieve freshness?

- use timestamps
- use nonces
- use counters
- use a key agreement protocol

Timestamps

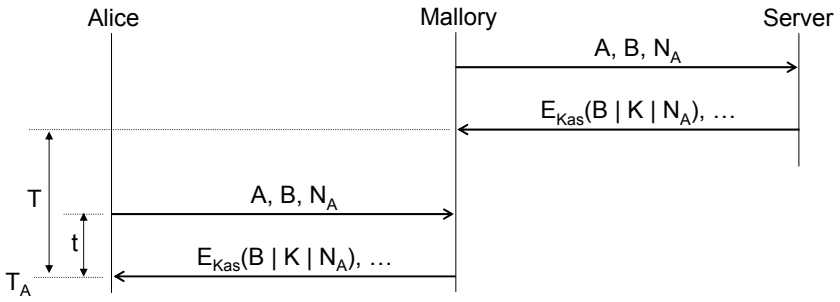
- $E_{Kas}(B \mid K \mid T_s)$, where T_s is the current time on the clock of S
- key is accepted only if the timestamp is within an acceptable window of the current time at the receiver
- can provide strong assurances, but require synchronized clocks
- important warning:
  if a party's clock is advanced, 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)

Nonces

- $E_{Kas}(B \mid K \mid N_A)$, 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

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 the key is younger than T_A-t, while in fact, it is older than T_A-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 $E_{Kab}(...|C)$,
  - she accepts it only if $C > C_{AB}$
  - if the message is accepted, then C_{AB} is set to C
- 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
- in addition, the parties may be desynchronized

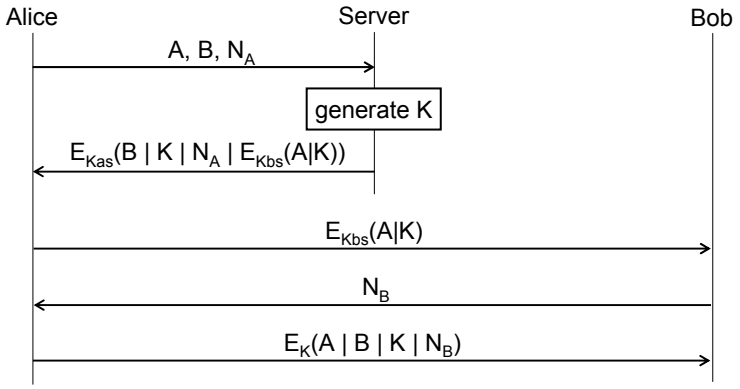
→ counters are not appropriate for providing key freshness

Key freshness in key agreement protocols

- $K = f(k_A, k_B)$, 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

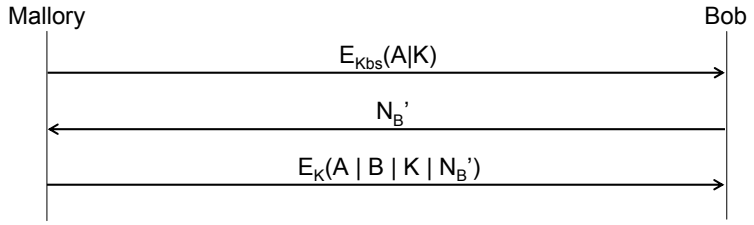
Fourth attempt



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 against the fourth attempt



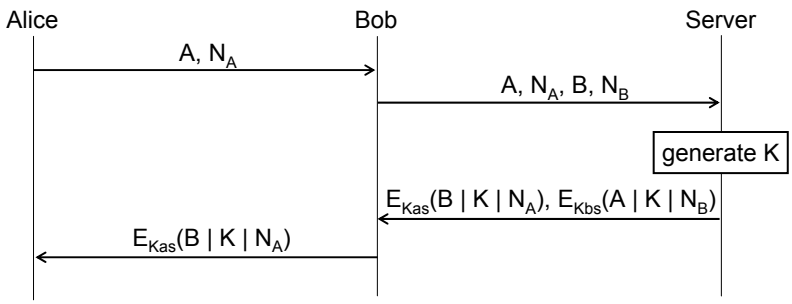
notes:

- 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)
- Bob will believe that he established a session with A, but A is not present

derived design principles:

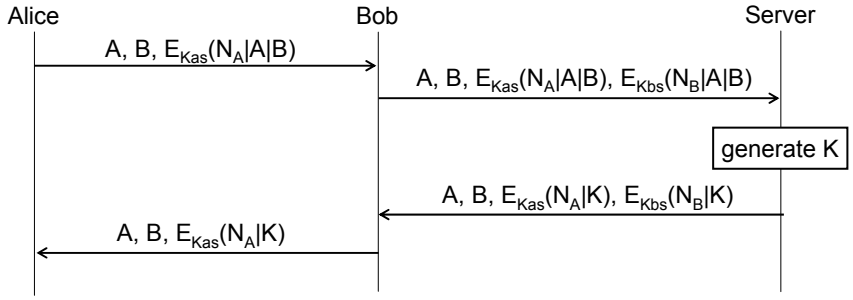
- ***the fact that a key K is used recently to encrypt a message does not mean that K is fresh***
- ***when proving the freshness of a key K by binding it to some fresh data (timestamp or nonce), don't use K itself for the binding***

Fifth attempt



- any problems?

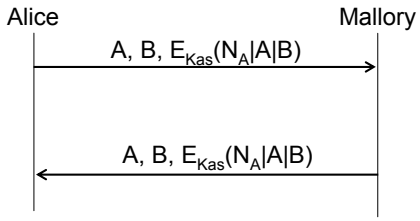
A variant of the Otway-Rees protocol



note:

- 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

A typing attack on Otway-Rees



notes:

- typical *reflection attack*
- Alice may accept A|B as the session key K

derived design principle:

it should be possible to decide for each message which protocol message it is (protocol type, protocol run, and message number)

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

Examples taken from the literature

Classification of protocols

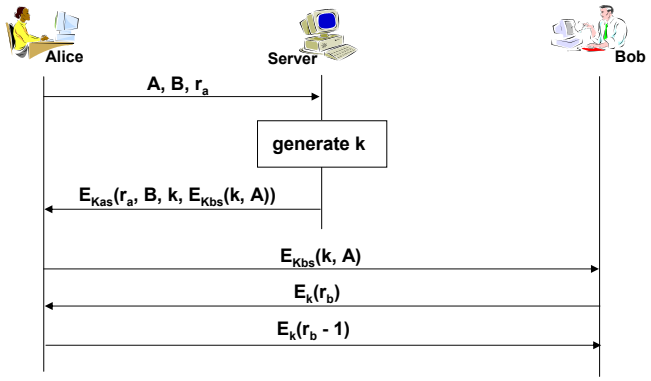
- key control
  - key transport or key agreement
- security services provided
  - implicit key authentication
  - key freshness
  - key confirmation
  - explicit key authentication
    = implicit key authentication + key confirmation

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

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

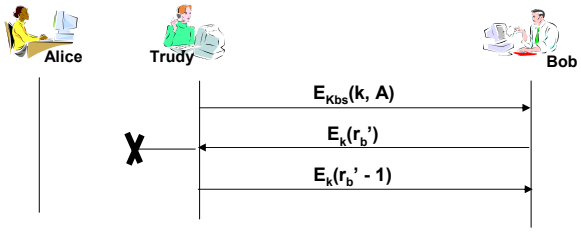


characteristics: mutual entity authentication, mutual explicit key authentication, key freshness with fresh random numbers (flawed), on-line third party trusted for generation of session keys, initial long-term keys between the parties and the server are required

A flaw in the Needham-Schroeder protocol

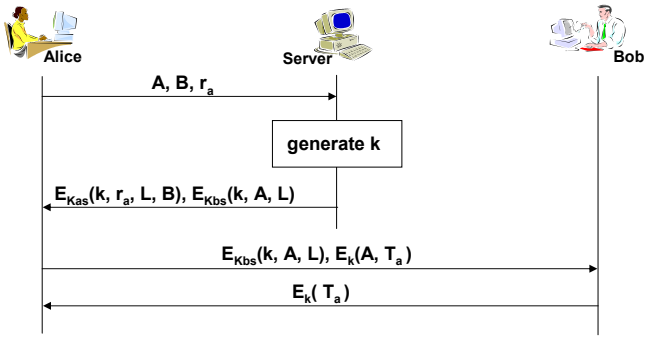
assumption: Trudy recorded a successful run of the protocol and compromised the session key k; thus, she knows k and $E_{Kbs}(k, A)$

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



The Kerberos protocol

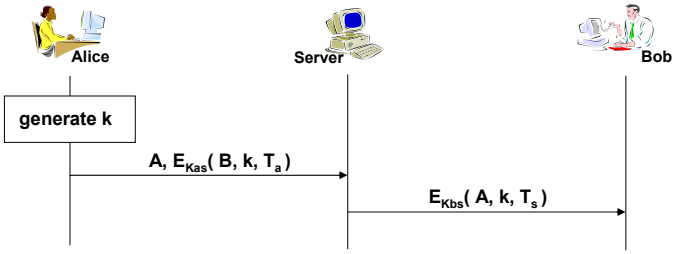
summary: essentially a correction of the Needham-Schroeder protocol
the protocol is optimized with respect to the original Needham-Schroeder protocol
(fewer messages and no double encryption in the second message)



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

The Wide-Mouth-Frog protocol

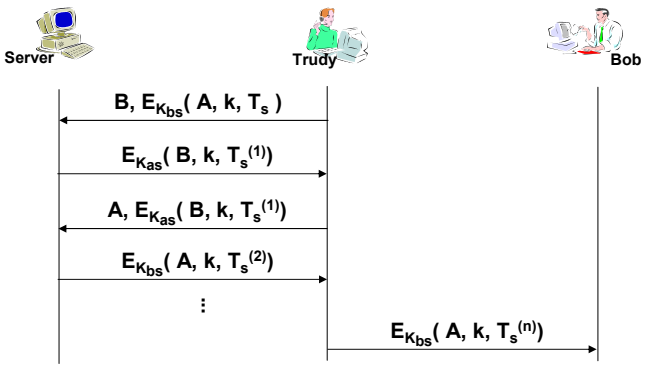
summary: a simple key transport protocol that uses a trusted third party
Alice generates the session key and sends it to Bob via the trusted third party



characteristics: key control for Alice
implicit key authentication for Alice
explicit key authentication for Bob
key freshness for Bob with timestamps (FLAWED!)
unilateral entity authentication of Alice
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 Wide-Mouth-Frog protocol

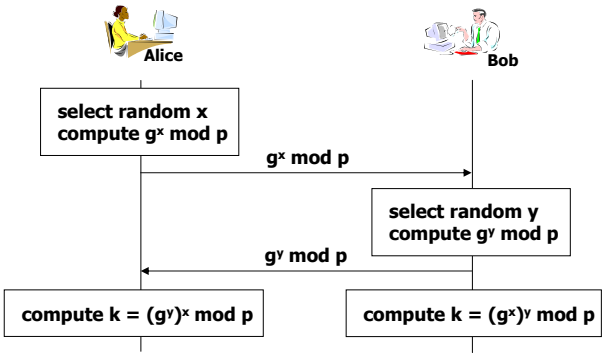
summary: after observing one run of the protocol, Trudy can continuously use the Server as an oracle until she wants to bring about re-authentication between Alice and Bob



The 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

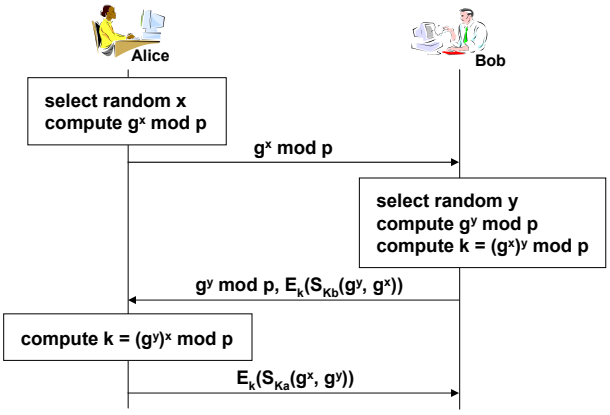
assumptions: p is a large prime, g is a generator of Z_p^*, both are publicly known system parameters



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

The Station-to-Station protocol

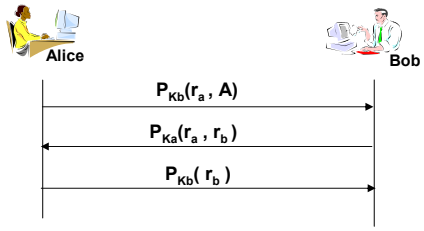
summary: three-pass variation of the basic Diffie-Hellman protocol; it uses digital signatures to provide mutual entity authentication and mutual explicit key authentication



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

The public-key Needham-Schroeder protocol

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



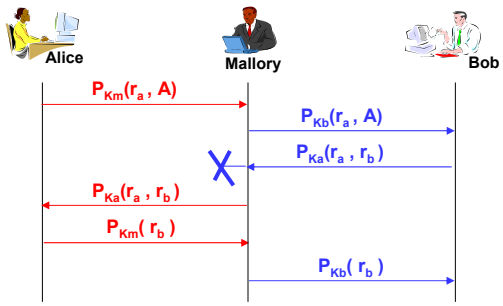
key derivation: both party computes $k = f(r_a , r_b)$

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

Lowe's attack

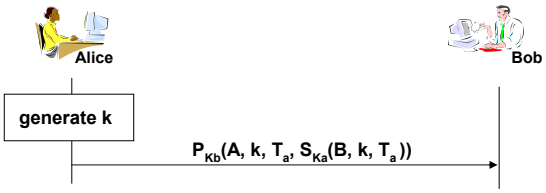
assumption: Mallory is a malicious user, his public key is K_m

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



Encrypting signed keys

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

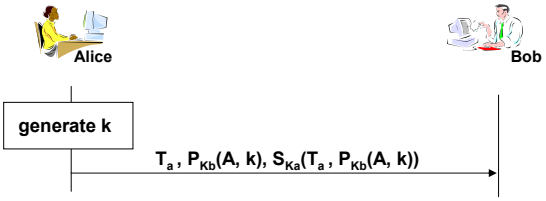


characteristics: unilateral entity authentication (of Alice), mutual implicit key authentication, key confirmation for Bob, key freshness with timestamp, clock synchronization needed, off-line third party for issuing public key certificates may be required, initial exchange of public keys between the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

summary: Alice generates a session key, encrypts it with Bob's public key, then sings 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: 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
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)