COMx501: Computer Security and Forensics

Achim D. Brucker
a.brucker@sheffield.ac.uk https://www.brucker.ch/

Software Assurance & Security Research
Department of Computer Science, The University of Sheffield, Sheffield, UK
https://logicalhacking.com/

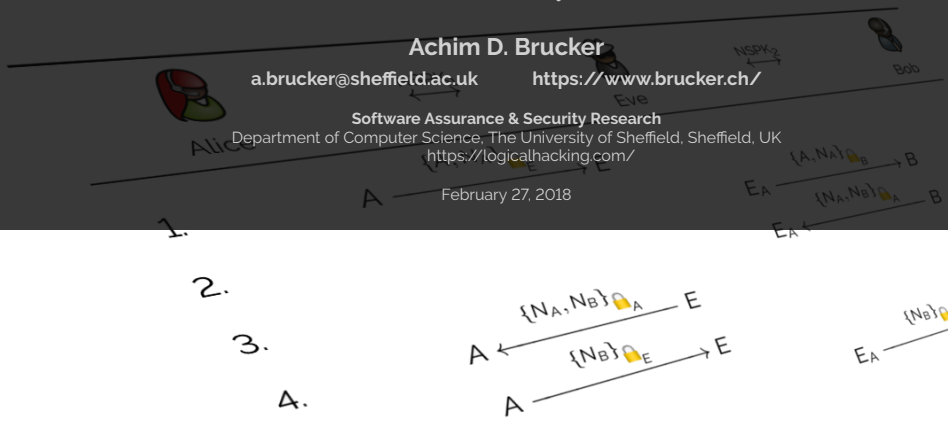
February 25, 2018

COMx501: Computer Security and Forensics
Part 7: Security Protocols

Achim D. Brucker
a.brucker@sheffield.ac.uk https://www.brucker.ch/

Software Assurance & Security Research
Department of Computer Science, The University of Sheffield, Sheffield, UK
https://logicalhacking.com/

February 27, 2018



Outline

Motivation

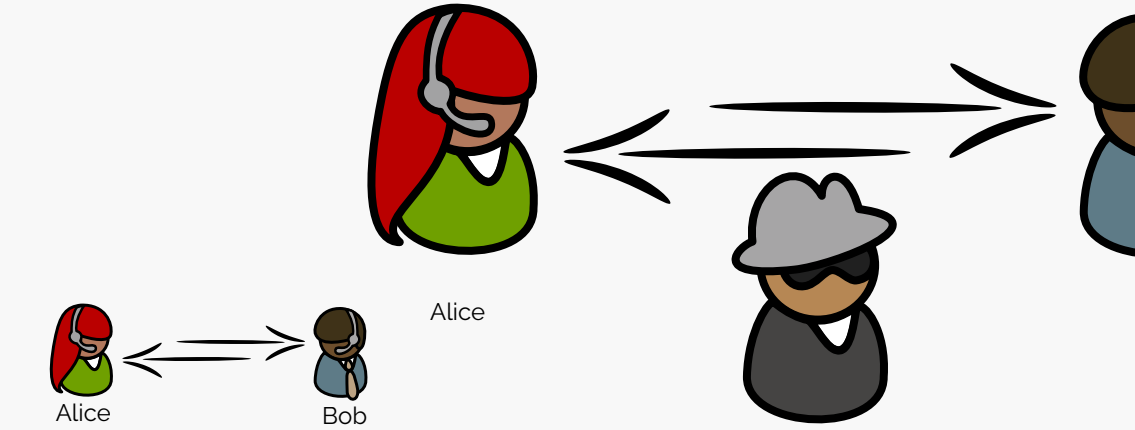
- RSA, AES, etc. provide (probably) very good cryptographic primitives
- How can we construct secure distributed applications with these primitives?
  - Securing Internet connections
  - E-commerce
  - E-banking
  - E-voting
  - Mobile communication
  - Digital contract signing

We will learn:

Even if cryptography is hard to break, this is not a trivial task

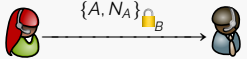
Establishing an Authentic Channel: NSPK

- Alice wants to be sure that she talks to Bob (authenticity)

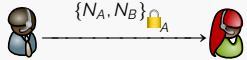


Goal: Mutual Authenticity (Two-way Authentication) After executing the protocol successfully,
Alice and Bob can be sure to talk to each other (and not to somebody else).

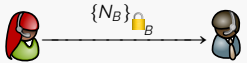
- Correctness argument (informal):



"This is Alice and I have chosen a nonce N_A."



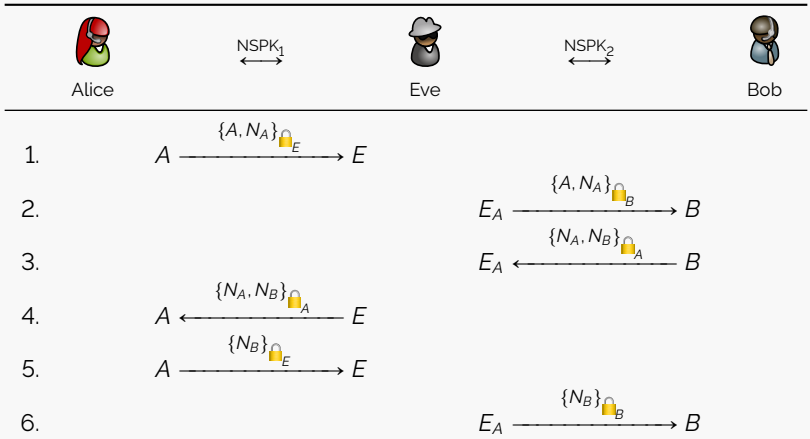
"Here is your nonce N_A. Since I could read it, I must be Bob.
I also have a challenge N_B for you."



"You sent me N_B.
Since only Alice can read this and I sent it back, you must be Alice."

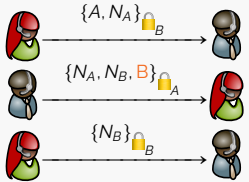
- Protocols are typically *small* and *convincing* … and often wrong!

NSPK: Lowe's attack (1996)



Bob believes he is speaking with Alice (but talks to Eve)!

Needham-Schroeder with Lowe's fix:



"I am Alice and here is my Nonce N_A."

"Here is your Nonce N_A and I have one for you and I am Bob."

"I got it! It is N_B."

Definitions

- A **protocol** consists of a set of rules (conventions) that determine the exchange of messages between two or more principals.
  In short, a distributed algorithm with emphasis on communication.
- **Security (or cryptographic) protocols** use cryptographic mechanisms to achieve security objectives.
  Examples: Entity or message authentication, key establishment, integrity, timeliness, fair exchange, non-repudiation, ...
- Analogous to programming Satan's computer

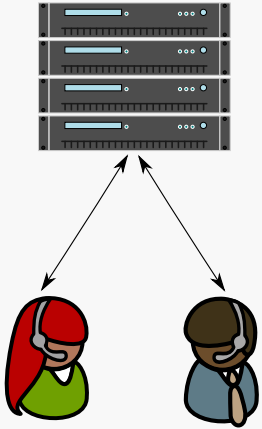
Questions

- Is it secure now?
- How can we detect (and fix) such flaws?
- We work on these questions in the next two weeks (and in the first lab)

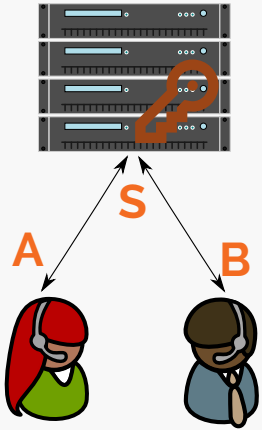
Outline

- An attempt to design a good protocol (from first principles)
- First step: establish the communications architecture

  We choose a common scenario (among many):
  - A set of users, any two of whom may wish to establish a new session key for subsequent secure communications.
    - successful completion of key establishment (& entity authentication) is only the beginning of a secure communications session. Further communication (often also through protocols) may be based on this key
    - Users are not necessarily honest! (More later)
  - There is an honest server.
    - Often called "trusted server", but trust ≠ honesty! We assume that an honest server never cheats and never gives out user secrets



- We thus consider in this scenario protocol with three roles:
  1. initiator role A (Alice)
  2. responder role B (Bob)
  3. server role S
- In a concrete execution of a protocol, the roles are played by agents a.k.a. principals: a, b, c (charly), s, i (intruder), ….
- We use i as the name of the intruder.
  Important: No agent in our model knows that i is not honest.
- Aims of the protocol:
  - At the end of the protocol, K_{AB} should be known to A and B, and possibly S (who forgets it immediately), but to no other parties.
  - A and B can assume that K_{AB} is newly generated.
- Formalization questions (that we will consider later):
  - How do we formalize the protocol steps and goals?
  - How do we formalize “knowledge”, “secrecy”, “newly”, …

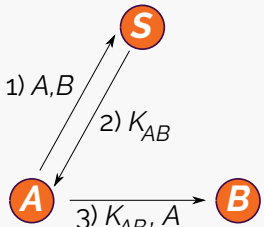


Our first attempt: a protocol that consists of three messages

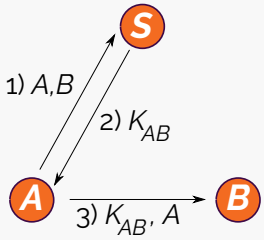
1. A contacts S by sending the identifies of the two parties who are going to share the session key: A, B
2. S sends the key K_{AB} to A: K_{AB}
3. A passes K_{AB} on to B

Note:

- K_{AB} does not contain any "information" about A or B, but is simply a name for the bit-string representing the session key
- before we examine the (lack of) security of this protocol, note that this is a significantly incomplete protocol specification

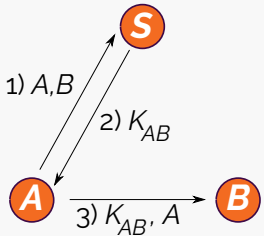


- Only messages passed in a successful run are specified
  - No description of what happens in the case that a message of the wrong format is received or that no message is received at all
  - This is often done for security protocols, since error messages are often not relevant for the security
    (What about Exceptions?)
- No specification of internal actions of principals
  - e.g. "create fresh K_{AB}" and store that it is a key for A and B
- Implicit assumption: A and B "know" that the received messages are part of the protocol.
  - It is common to omit such details which would be required for a networked computer to be able to track the progress of a particular protocol run.
  - This may include details of which key should be used to decrypt a received message which has been encrypted.



- Despite the obvious limitations associated with specifying protocols by showing only the messages of a successful run, it remains the most popular method of describing security protocols
  But there are many works that have addressed these problems to "disambiguate" the notations
- An equivalent representation: Alice & Bob–notation:
  1. $A \longrightarrow S: A, B$
  2. $S \longrightarrow A: K_{AB}$
  3. $A \longrightarrow B: K_{AB}, A$

  Note that sender/receiver names (e.g., "$A \longrightarrow B$") are not part of the message and it is not the case that messages automatically reach their destination (securely).

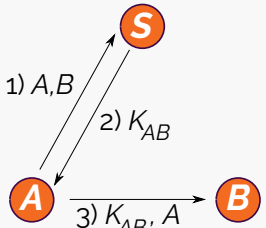


- Problem with this protocol? The session key K AB must be transported to A and B but to no other parties.
- A realistic assumption in typical communication systems such as the Internet and corporate networks:

Security Assumption 1

The intruder is able to eavesdrop on all messages sent in a security protocol.

☞ Use a cryptographic algorithm and associated keys.



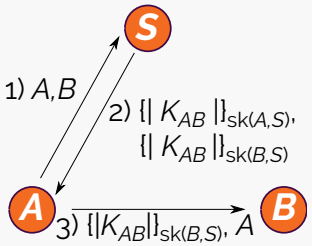
Second attempt:

- assume that S initially shares a secret key $\mathrm{sk}(U,S)$ with each user U of the system:
  - $\mathrm{sk}(A,S)$ with A
  - $\mathrm{sk}(B,S)$ with B

  and S encrypts message 2)
- Problems:
  - Eavesdropping? No.

Perfect Cryptography Assumption

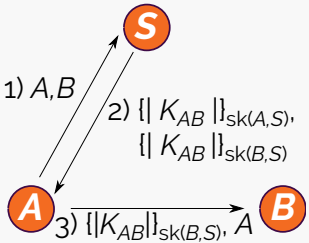
Encrypted messages may only be read by the legitimate recipients who have the keys required to decrypt …



- Problem is not that the protocol gives away the secret key K_{AB}, but that the information about who else has K_{AB} is not protected
- Intruder could not only be able to eavesdrop on messages sent, but also to capture messages and alter them

Security Assumption 2

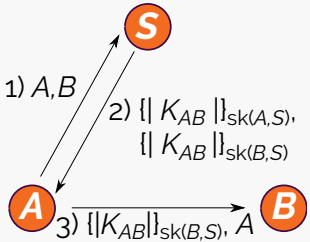
The intruder is able to intercept messages on the network and send messages to anybody (under any sender name).



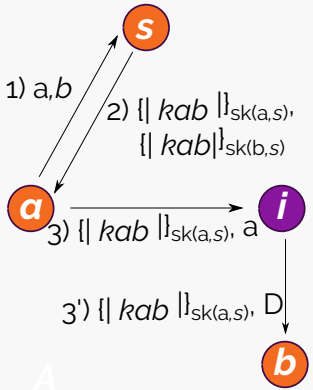
- The intruder has complete control of the channel(s) over which protocol messages flow.

  The intruder has complete control over the network

- In contrast to ordinary communication protocols, we assume the worst-case of a malicious agent
  - Although there may be no more than 4 or 5 messages involved in a legitimate run of the protocol, there are an infinite number of variations in which the intruder can participate
  - These variations involve an unbounded number of messages and each must satisfy the protocol's security requirements



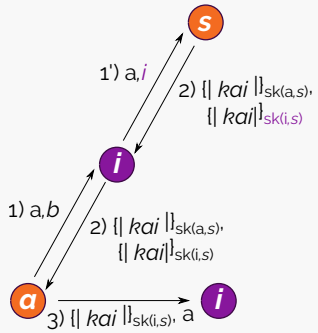
- The intruder *i* simply intercepts the message from *a* to *b* and substitutes *D* for *a*'s identity (*D* is any agent name).
  - → *b* believes that he is sharing the key with *D*, whereas in fact he is sharing it with *a*
- The result of this attack will depend on the scenario in which the protocol is used, but may include such actions as *b* giving away information to *a* which should have been shared only with *D*.
- Although *i* does not obtain *kab*, we can still regard the protocol as broken since it does not satisfy the requirement that the users should know who else knows the session key.
- But there is also another (more serious) attack …



Building a Key Establishment Protocol

Second Attempt: Another Attack

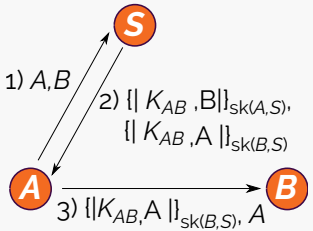
- *i* alters the message from *a* to *s* so that *s* generates *a* key *kai* for *a* and *i* and encrypts it with the key sk(*i*, *s*) of the intruder
- Since *a* cannot distinguish between encrypted messages meant for other principals she will not detect the alteration
  - *kai* is simply *a* formal name for the bit-string representing the session key, so it will be accepted by *a*
  - *i* intercepts the message from a intended for *b* so that *a* will not detect any anomaly
- Hence *a* will believe that the protocol has been successfully completed with *b* whereas *i* knows *kai* and so can masquerade as *b* as well as learn all information that *a* sends intended for *b*
- In contrast to the previous attack, this one will succeed if *i* is *a* legitimate system user known to *s*



Security Assumption 3

The intruder may be a legitimate protocol participant (an insider), or an external party (an outsider), or a combination of both.

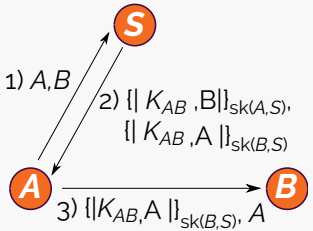
- To overcome these attacks, the names of the principals who are to share K_{AB} need to be bound cryptographically to the key. Neither of two previous attacks succeed on the modified protocol (see right hand side)
- The protocol has improved to the point where an intruder is unable to attack it by eavesdropping or altering the messages sent between honest parties
- However, even now the protocol is not good enough to provide security on normal operating conditions



Building a Key Establishment Protocol

Third Attempt: Problem

- The problem stems from the difference in quality between the long-term key-encrypting keys shared initially with S, and the session keys K_{AB} generated for each protocol run
- Reasons for using session keys:
  - They are expected to be vulnerable to attack (by cryptanalysis)
  - Communications in different sessions should be separated. In particular, it should not be possible to replay messages from previous sessions
- A whole class of attacks becomes possible when old keys (or other security-relevant data) may be replayed in a subsequent session



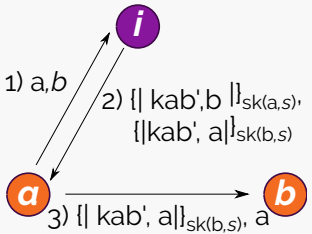
Security Assumption 4

The intruder is able to obtain the value of the session key K_{AB} used in any "sufficiently old" previous run of the protocol.

Building a Key Establishment Protocol

Third Attempt: Attack (1/2)

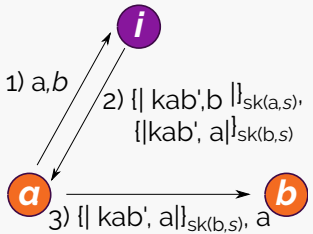
- *i* masquerades as *s*
- Replay:
  kab' is an old key used by *a* and *b* in a previous session
  - By Security Assumption 1, *i* can be expected to know the encrypted messages in which *kab'* was transported to *a* and *b*
  - By Security Assumption 4, *i* can be expected to know the value of *kab'*
  - Thus, when a completes the protocol with *b*, *i* is able to decrypt subsequent information encrypted with *kab'* or insert or alter messages whose integrity is protected by *kab'*



- The replay attack can still be regarded as successful even if *i* has not obtained the value of *kab'*:
  - *i* has succeeded in making *a* and *b* accept an old session key!
  - The attack allows *i* to replay messages protected by *kab'* which were sent in the previous session
- Of course: provided that *a* and *b* don't check the key!
  - "Principals don't think" but just follow the protocol
  - Various techniques may be used to allow principals to check that session keys have not been replayed, e.g. the challenge–response method:

Definition

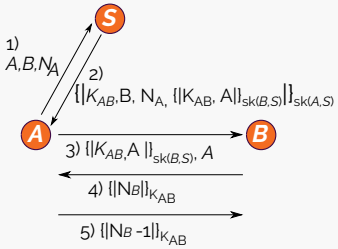
A nonce ("a number used only once") is a random value generated by one principal and returned to that principal to show that a message is newly generated.



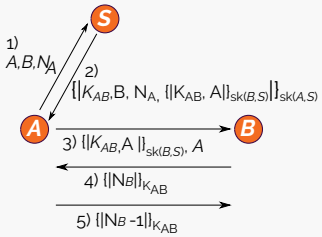
Building a Key Establishment Protocol

Fourth Attempt (1/2)

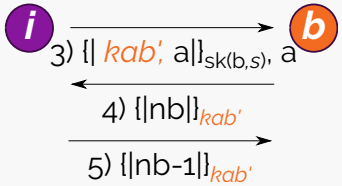
- *A* sends her nonce N_A to *S* with the request for a new key
- If this same value is received with the session key, then *A* can deduce that the key has not been replayed
  (This deduction will be valid as long as session key and nonce are bound together cryptographically in such a way that only S could have formed such a message)
- Note, N_A is just a number
  - There is nothing in N_A that identifies who has created it
  - Hence, we will also write *NA* or even better N_1 or *N*1

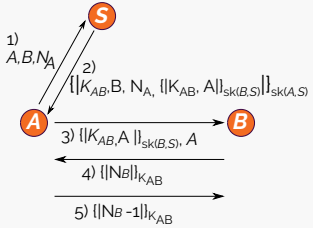


- If the encrypted key for *B* is included in the encrypted part of *A*'s message, then *A* can gain assurance that it is fresh
- It is tempting to believe that *A* may pass this assurance on to *B* in an extra handshake:
  - *B* generates a nonce N_B and sends it to *A* protected by K_{AB} itself
  - *A* uses K_{AB} to reply to *B* ("–1" to avoid replay of message 4)
- This is actually a famous security protocol:
  - Needham Schroeder with Conventional Keys (NSCK)
  - Published by Needham and Schroeder in 1978, it has been the basis for a whole class of related protocols
  - Unfortunately, this protocol is vulnerable to a famous attack due to Denning and Sacco

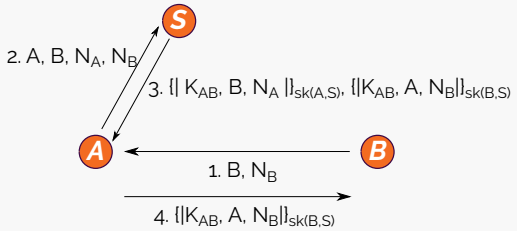


- Problem is assumption that only *A* will be able to form a correct reply to message 4 from *B*
- Since the intruder *i* can be expected to know the value of an old session key, this assumption is unrealistic
- *i* masquerades as *a* and convinces *b* to use old key *kab'*:

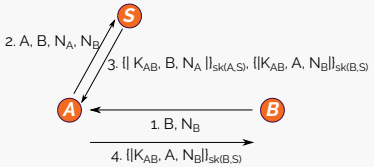




- Idea:
  Let's throw away the assumption that it is inconvenient for both *B* and *A* to send their challenges to *S*
- The protocol is now initiated by *B* who sends his nonce N_B first to *A*
- A adds her nonce N_A and sends both to *S*, who is now able to return K_{AB} in separate messages for *A* and *B*, which can each be verified as fresh by their respective recipients



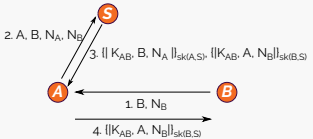
- It may seem that we have achieved more than the previous protocol using fewer messages, but in fact …
  - In the NSCK, *A* could verify that *B* has in fact received the key
  - This property of key confirmation is achieved due to *B*'s use of the key in message 4, assuming that $\{|N_B|\}_{K_{AB}}$ cannot be formed without knowledge of K_{AB}
  - In our final protocol, neither *A* nor *B* can deduce at the end of a successful protocol run that the other has actually received K_{AB}
    - Is this a Problem?



Building a Key Establishment Protocol: Summary

- This protocol avoids all the attacks that we have seen so far, as long as the cryptographic algorithm used provides the properties of both confidentiality and integrity, and the server *S* acts correctly
- It would be rash to claim that this protocol is secure before giving a precise meaning to that term!
- The security of a protocol must always be considered relative to its goals

☞ We need means to formalize protocols and goals



Outline

Notation: Messages

Roles: A, B or *Alice*, *Bob*

Agents: a, b, i

Symmetric Keys: K, K_{AB} , ...; $\mathsf{sk}(A, S)$

Symmetric Encryption: $\{|M|\}_K$

Public Keys: K, $\mathsf{pk}(A)$

Private Keys: $\mathsf{inv}(K)$, $\mathsf{inv}(\mathsf{pk}(A))$

Asymmetric Encryption: $\{M\}_K$

Signing: $\{M\}_{\mathsf{inv}\,K}$

Nonces: NA, $N1$ fresh data items used for challenge/response.

Sometimes, we may use subscripts, e.g. N_A , but it does not mean that principals can find out that N_A was generated by A

Timestamps: T denotes time, e.g. used for key expiration.

Message concatenation: M_1, M_2, M_3

- Fundamental event is communication between principals.

$$A \longrightarrow B : \{A, T_1, K_{AB}\}_{\mathrm{pk}(B)}$$

- A and B name roles
  Can be instantiated by any principal playing in the role
- Communication is asynchronous
- Sender/receiver names "$A \longrightarrow B$" are not part of the message
- Protocol specifies actions of principals.
  Alternatively, protocol defines a set of event sequences (traces).

Notation: Protocols

- A typical protocol description combines prose, data type specifications, various kinds of diagrams, ad hoc notations, and message sequences like
  1. $A \longrightarrow B: \ \{NA, A\}_{\mathrm{pk}(B)}$
  2. $B \longrightarrow A: \ \{NA, NB\}_{\mathrm{pk}(A)}$
  3. $A \longrightarrow B: \ \{NB\}_{\mathrm{pk}(B)}$
- They often include informal statements concerning the properties of the protocol and why they should hold
- What does a message $A \longrightarrow B: \ M$ actually mean?

> “ We assume that an intruder can interpose a computer in all communication paths, and thus can alter or copy parts of messages, replay messages, or emit false material.
>
> We also assume that each principal has a secure environment in which to compute such as is provided by a personal computer …
>
> Needham and Schroeder

Role A:

1. Generate nonce NA, concatenate it with name, and encrypt with $\mathrm{pk}(B)$
   - Send $\{NA, A\}_{\mathrm{pk}(B)}$ to B
2. Receive a message M
   - Decrypt M it with $\mathrm{inv}(\mathrm{pk}(A))$, call it M'. If decryption fails, reject M'
     - Q: how to detect wrong decryption
     - Q: what to do about rejected messages?
   - Split the message into two nonces NA' and NB. If that is not possible, reject M
     - Q: how to check this?
   - Check that $NA' = NA$; if not, reject M
3. Encrypt NB with $\mathrm{pk}(B)$ and send to B

1. $A \longrightarrow B : \{NA, A\}_{\mathrm{pk}(B)}$
2. $B \longrightarrow A : \{NA, NB\}_{\mathrm{pk}(A)}$
3. $A \longrightarrow B : \{NB\}_{\mathrm{pk}(B)}$

Outline

Different Kind of Attacks

Man-in-the-middle (or parallel sessions) attack: $A \longleftrightarrow i \longleftrightarrow B$

Replay (or freshness) attack: reuse parts of previous messages

Masquerading attack: pretend to be another principal

Reflection attack: send transmitted information back to originator

Oracle attack: take advantage of normal protocol responses as encryption and decryption "services"

Binding attack: using messages in a different context/for a different purpose than originally intended

Type flaw attack: substitute a different type of message field

These attack types are not formally defined and there may be overlaps between these types

- Diffie-Hellman Key-Exchange
  - A and B agree on a DH group (g, p)
  - A generates large x and sends half-key $X = g^x \mod p$ to B
  - B generates large y and sends half-key $Y = g^y \mod p$ to A
  - A and B compute key $k = Y^x \mod p = X^y \mod p$

  Security depends on the difficulty of computing the discrete logarithm of an exponentiated number modulo a large prime number
- Diffie-Hellman (without authentication of the half-keys) can be attacked:
  1. $a \longrightarrow i(b): \exp(g, x)$
     - 1' $i(a) \longrightarrow b: \exp(g, z)$
     - 2' $b \longrightarrow i(a): \exp(g, y)$
  2. $i(b) \longrightarrow i(a): \exp(g, z)$
  - a believes to share key $\exp(\exp(g, x), z)$ with b, b believes ...
  - i knows both keys ...
- Prevention: authenticate the half-keys, e.g. with digital signatures
  1. $A \longrightarrow B: \{\exp(g, X)\}_{\mathrm{inv}(\mathrm{pk}(A))}$
  2. $B \longrightarrow A: \{\exp(g, Y)\}_{\mathrm{inv}(\mathrm{pk}(B))}$
- Many protocols are based on Diffie-Hellman, which is not a bad idea!

Type Flaw Attacks

- A message consists of a sequence of sub-messages, e.g., a principal's name, a nonce, a key, …
- Real messages are bit strings without type information, e.g., 1011 0110 0010 1110 0011 0111 1010 0000
- Type flaw is when $A \longrightarrow B$: M and B accepts M as valid but parses it differently
  - 👉 B interprets the bits differently than A

Type Flaw Attacks: The Otway-Rees Protocol (1/2)

- Server-based protocol providing authenticated key distribution (with key authentication and key freshness) but without entity authentication or key confirmation
  1. $A \longrightarrow B: \; M, A, B, \{|NA, M, A, B|\}_{sk(A,S)}$
  2. $B \longrightarrow S: \; M, A, B, \{|NA, M, A, B|\}_{sk(A,S)}, \{|NB, M, A, B|\}_{sk(B,S)}$
  3. $S \longrightarrow B: \; M, \{|NA, K_{AB}|\}_{sk(A,S)}, \{|NB, K_{AB}|\}_{sk(B,S)}$
  4. $B \longrightarrow A: \; M, \{|NA, K_{AB}|\}_{sk(A,S)}$

  Server keys already known and M is a session id (e.g., an integer)
- Now suppose $|M, A, B| = |K_{AB}|$ (e.g., $|M| = 32$ Bit, $|A| = 32$ Bit, $|B| = 32$ Bit, and $|K_{AB}| = 64$ Bit
- Attack 1 (reflection/type-flaw):
  i replays parts of message 1 as message 4 (omitting steps 2 and 3)
  1. $a \longrightarrow i(b): \; m, a, b, \{|na, m, a, b|\}_{sk(a,s)}$
  4. $i(b) \longrightarrow a: \; m, \{|na, \underbrace{m, a, b}_{\text{mistaken as } kab} |\}_{sk(a,s)}$

Type Flaw Attacks: The Otway-Rees Protocol (2/2)

- Otway-Rees:
  1. $A \longrightarrow B: \; M, A, B, \{|NA, M, A, B|\}_{sk(A,S)}$
  2. $B \longrightarrow S: \; M, A, B, \{|NA, M, A, B|\}_{sk(A,S)}, \{|NB, M, A, B|\}_{sk(B,S)}$
  3. $S \longrightarrow B: \; M, \{|NA, K_{AB}|\}_{sk(A,S)}, \{|NB, K_{AB}|\}_{sk(B,S)}$
  4. $B \longrightarrow A: \; M, \{|NA, K_{AB}|\}_{sk(A,S)}$

  Server keys already known and M is a session id (e.g., an integer)
- Now suppose $|M, A, B| = |K_{AB}|$ (e.g., $|M| = 32$ Bit, $|A| = 32$ Bit, $|B| = 32$ Bit, and $|K_{AB}| = 64$ Bit
- Attack 2 (reflection/type-flaw): The intruder can play the role of S in 2 and 3 by reflecting the encrypted components of 2 back to B
  1. $a \longrightarrow b: \; m, a, b, \{|na, m, a, b|\}_{sk(a,s)}$
  2. $b \longrightarrow i(s): \; m, a, b, \{|na, m, a, b|\}_{sk(a,s)}, \{|na, m, a, b|\}_{sk(b,s)}$
  3. $i(s) \longrightarrow b: \; m, \{|na, \underbrace{m, a, b}_{kab}|\}_{sk(a,s)}, \{|nb, \underbrace{m, a, b}_{kab}|\}_{sk(b,s)}$
  4. $b \longrightarrow a: \; m, \{|na, \underbrace{m, a, b}_{kab}|\}_{sk(a,s)}$

  👉 a and b accept the wrong key and i can decrypt their subsequent communication (key authentication and secrecy fails)

Example of a Binding Attack

- The protocol
  1. $A \longrightarrow S: \; A, B, NA$
  2. $S \longrightarrow A: \; S, \{S, A, NA, K_B\}_{\mathsf{inv}(\mathsf{pk}(S))}$

  admits a binding attack:

  1.1 $a \longrightarrow i(s): \; a, b, na$
  2.1 $i(a) \longrightarrow s: \; a, i, na$
  2.2 $s \longrightarrow i(a): \; s, \{s, a, na, ki\}_{\mathsf{inv}(\mathsf{pk}(S))}$
  1.2 $i(s) \longrightarrow s: \; s, \{s, a, na, ki\}_{\mathsf{inv}(\mathsf{pk}(S))}$
- Fix: include thename of B in 2.

Outline

- Principles proposed by Abadi and Needham (1994, 1995):
  - Every message should say what it means.
  - Specify clearly conditions for a message to be acted on.
  - Mention names explicitly if they are essential to the meaning.
  - Be clear as to why encryption is being done: confidentiality, message authentication, binding of messages, …
    (e.g. $\{X, Y\}_{\mathsf{inv}(K)}$ versus $\{X\}_{\mathsf{inv}(K)}, \{Y\}_{\mathsf{inv}(K)}$)
  - Be clear on what properties you are assuming.
  - Beware of clock variations (for timestamps).
  - etc.
- Good advice, but
  - Is the protocol guaranteed to be secure then?
  - Is it optimal and/or minimal then?
  - Have you considered all types of attacks?
  - etc.

Conclusion

- Theses:
  - A protocol without clear goals (and assumptions) is useless
  - A protocol without a proof of correctness is probably wrong
- Assumptions/Intruder model (following Dolev and Yao)
  - can control the network
  - can participate in the protocol
  - can compose/decompose messages with the keys he has
  - annot break cryptography

  👉 Worst-case assumption of an intruder
- Goals: What the protocol should achieve, e.g.,
  - Authenticate messages, binding them to their originator
  - Ensure timeliness of messages (recent, fresh, …)
  - Guarantee secrecy of certain items (e.g. generated keys)

Thank you for your attention!
Any questions or remarks?

Contact:

Dr. Achim D. Brucker
Department of Computer Science
University of Sheffield
Regent Court
211 Portobello St.
Sheffield S1 4DP, UK

a.brucker@sheffield.ac.uk
@adbrucker
https://de.linkedin.com/in/adbrucker/
https://www.brucker.ch/
https://logicalhacking.com/blog/

Bibliography I

Ross J. Anderson.

Security Engineering: A Guide to Building Dependable Distributed Systems.

John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 2001.

The complete book is available at: http://www.cl.cam.ac.uk/~rja14/book.html.

Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot.

Handbook of Applied Cryptography.

CRC Press, Inc., Boca Raton, FL, USA, 5th edition, 2001.

The complete book is available at: http://cacr.uwaterloo.ca/hac/.

Bruce Schneier.

Applied Cryptography.

John Wiley & Sons, Inc., 2 edition, 1996.

Document Classification and License Information