UNIVERSITY OF AMSTERDAM

Jaap van Ginkel

Security of Systems and Networks

October 10, 2019 SSH SSL TLS

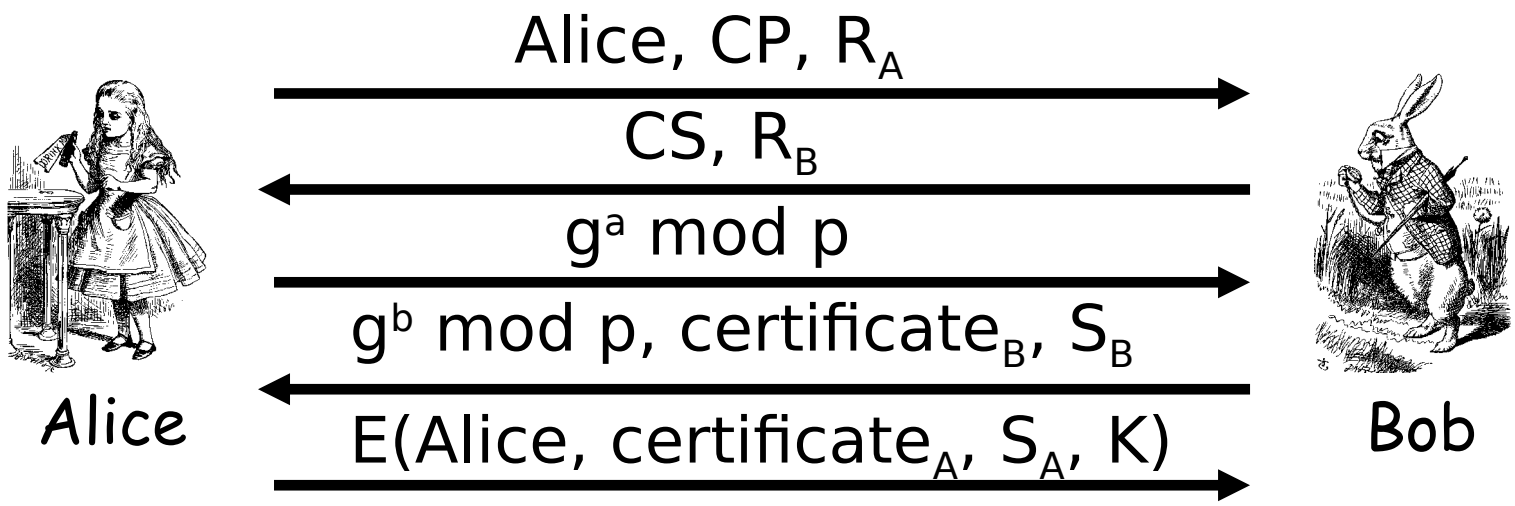
SSH

- Creates a "secure tunnel"
- Insecure command sent thru SSH tunnel are then secure
- SSH used with things like rlogin
  - Why is rlogin insecure without SSH?
  - Why is rlogin secure with SSH?
- SSH is a relatively simple protocol

SSH

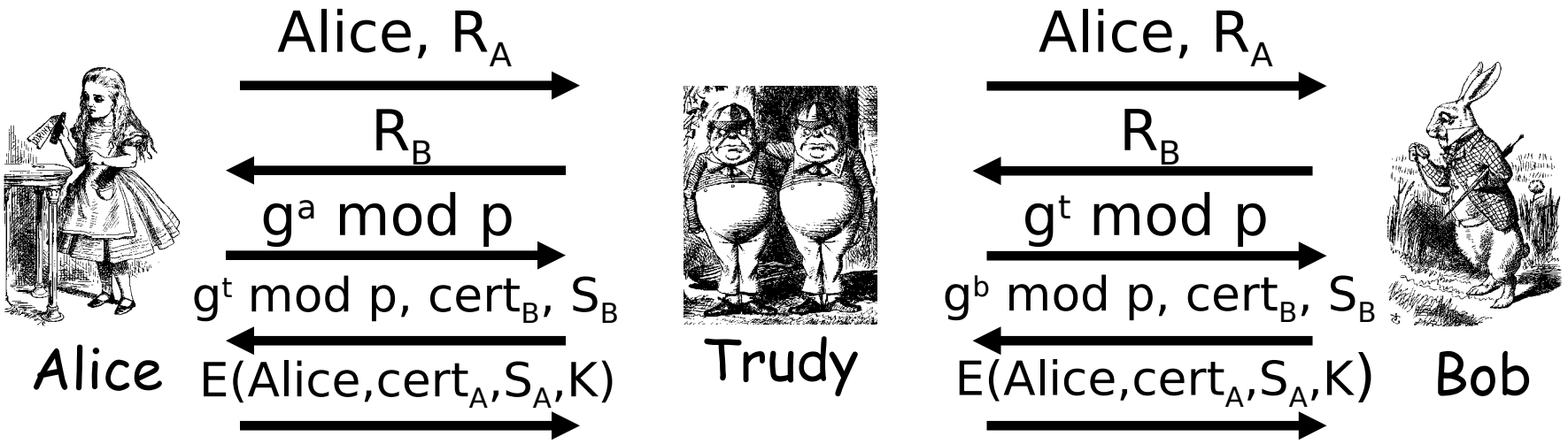
- SSH authentication can be based on:
  - Public keys, or
  - Digital certificates, or
  - Passwords
- Here, we consider ***certificate*** mode
  - Other modes, see homework problems
- We consider slightly simplified SSH…

Simplified SSH



- CP = “crypto proposed”, and CS = “crypto selected”
- H = h(Alice,Bob,CP,CS,R_A,R_B,g^a mod p,g^b mod p,g^{ab} mod p)
- $S_B = [H]_{Bob}$
- $S_A = [H, Alice, certificate_A]_{Alice}$
- K = g^{ab} mod p

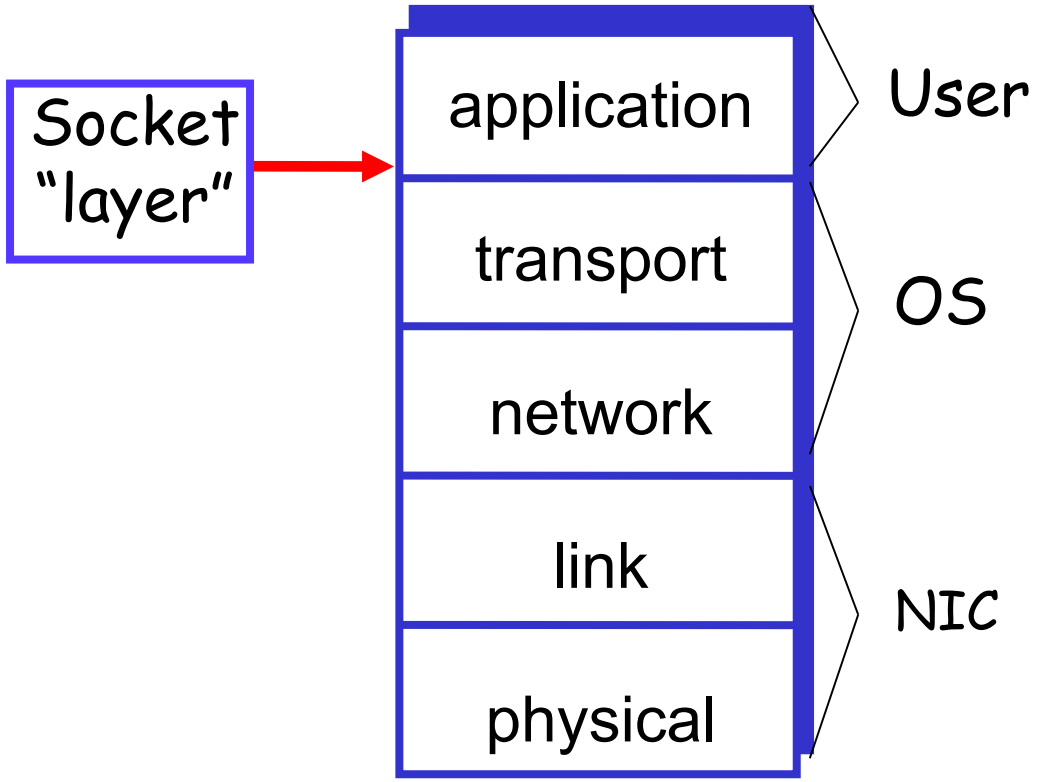
MiM Attack on SSH?



- Where does this attack fail?
- Alice computes:
  - $H_a = h(Alice,Bob,CP,CS,R_A,R_B,g^a \bmod p,g^t \bmod p,g^{at} \bmod p)$
- But Bob signs:
  - $H_b = h(Alice,Bob,CP,CS,R_A,R_B,g^t \bmod p,g^b \bmod p,g^{bt} \bmod p)$

Socket layer

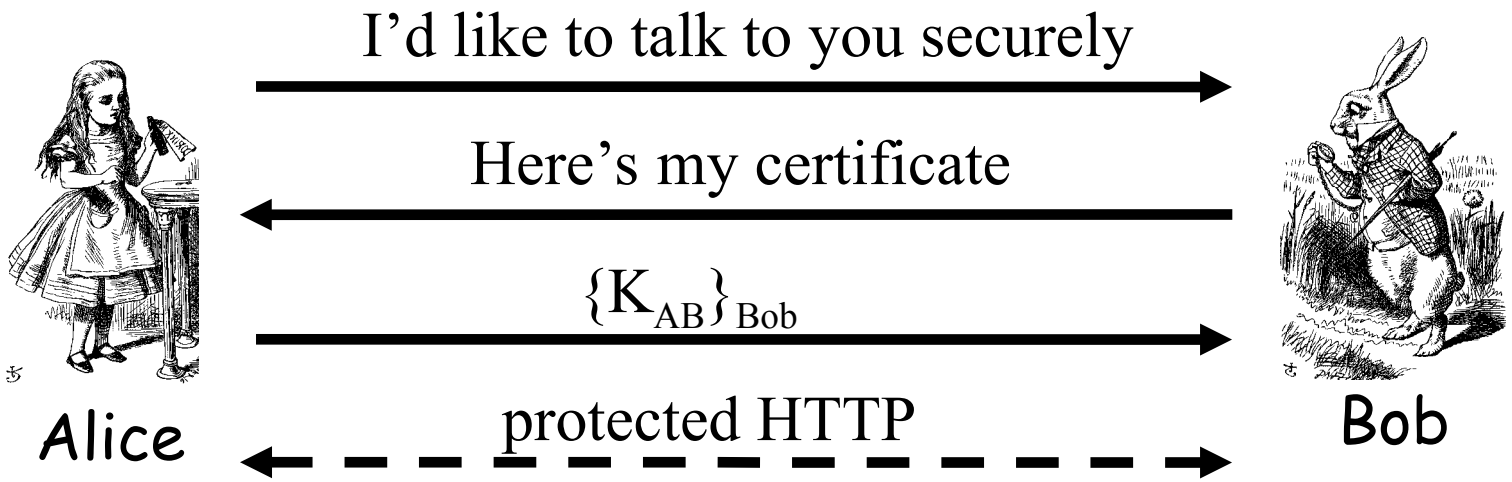
- "Socket layer" lives between application and transport layers
- SSL usually lies between HTTP and TCP



What is SSL?

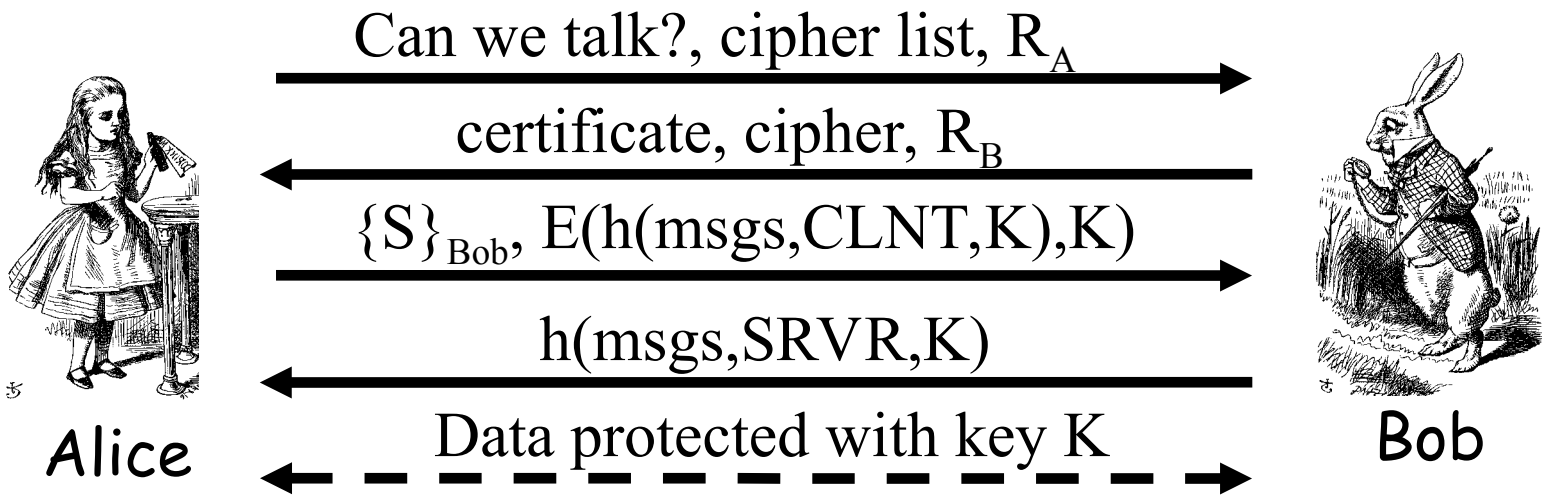
- SSL is the protocol used for most secure transactions over the Internet
- For example, if you want to buy a book at amazon.com…
  - You want to be sure you are dealing with Amazon (**authentication**)
  - Your credit card information must be protected in transit (**confidentiality** and/or **integrity**)
  - As long as you have money, Amazon doesn't care who you are (authentication need not be mutual)

Simple SSL-like Protocol



- Is Alice sure she's talking to Bob?
- Is Bob sure he's talking to Alice?

Simplified SSL Protocol



- S is **pre-master secret**
- $K = h(S,R_A,R_B)$
- msgs = all previous messages
- CLNT and SRVR are constants

SSL Keys

- 6 "keys" derived from $K = \text{hash}(S, R_A, R_B)$
  - 2 encryption keys: send and receive
  - 2 integrity keys: send and receive
  - 2 IVs: send and receive
  - Why different keys in each direction?
- **Q:** Why is $h(\text{msgs}, \text{CLNT}, K)$ encrypted (and integrity protected)?
- ***A:*** It adds no security…

SSL Authentication

- Alice authenticates Bob, not vice-versa
  - How does client authenticate server?
  - Why does server not authenticate client?
- Mutual authentication is possible: Bob sends **certificate request** in message 2
  - This requires client to have certificate
  - If server wants to authenticate client, server could instead require (encrypted) password

SSL MiM Attack

Alice

R_A

$certificate_T, R_B$

$\{S_1\}_{Trudy}, E(X_1,K_1)$

$h(Y_1,K_1)$

$E(data,K_1)$

Trudy

R_A

$certificate_B, R_B$

$\{S_2\}_{Bob}, E(X_2,K_2)$

$h(Y_2,K_2)$

$E(data,K_2)$

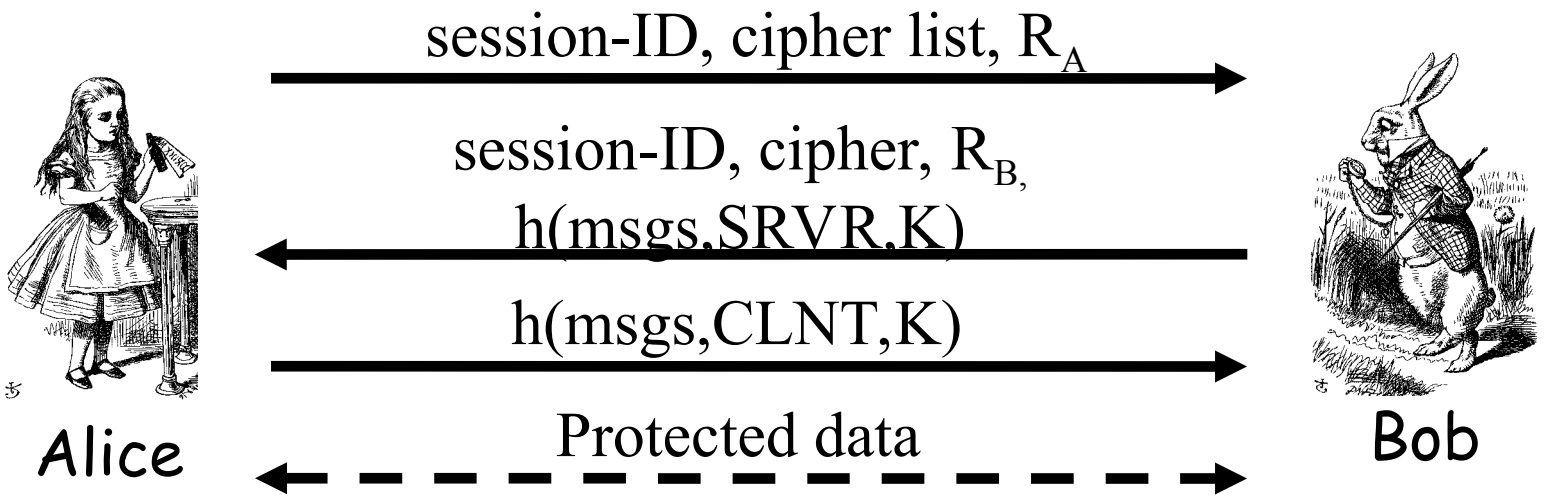
Bob

- **Q:** What prevents this MiM attack?
- **A:** Bob's certificate must be signed by a certificate authority (such as Verisign)
- What does Web browser do if sig. not valid?
- What does user do if signature is not valid?

SSL Sessions vs Connections

- SSL **session** is established as shown on previous slides
- SSL designed for use with HTTP 1.0
- HTTP 1.0 usually opens multiple simultaneous (parallel) **connections**
- SSL session establishment is costly
  - Due to public key operations
- SSL has an efficient protocol for opening new connections given an existing session

SSL Connection



- Assuming SSL **session** exists
- So S is already known to Alice and Bob
- Both sides must remember session-ID
- Again, $K = h(S,R_A,R_B)$
- **No public key operations!** (relies on known S)

SSL vs IPSec

- IPSec – discussed in next section
  - Lives at the network layer (part of the OS)
  - Has encryption, integrity, authentication, etc.
  - Is overly complex (including serious flaws)
- SSL (and IEEE standard known as TLS)
  - Lives at socket layer (part of user space)
  - Has encryption, integrity, authentication, etc.
  - Has a simpler specification

SSL vs IPSec

- IPSec implementation
  - Requires changes to OS, but no changes to applications
- SSL implementation
  - Requires changes to applications, but no changes to OS
- SSL built into Web application early on (Netscape)
- IPSec used in VPN applications (secure tunnel)
- Reluctance to retrofit applications for SSL
- Reluctance to use IPSec due to complexity and interoperability issues
- Result? **Internet less secure than it should be!**

Secure Network Programming API

- Early research efforts toward transport layer security included the Secure Network Programming (SNP) application programming interface (API), which in 1993 explored the approach of having a secure transport layer API closely resembling Berkeley sockets, to facilitate retrofitting preexisting network applications with security measures.[4]

SSL 1.0, 2.0 and 3.0

- Developed by Netscape engineers
- Phil Karlton, Alan Freier

- Version 1.0 never released

- Version 2.0 February 1995
- some (serious) security flaws

- Version 3.0 1996
- Complete redesign
- Basis for current TLS versions

-

TLS 1.0 (SSL 3.1)

- TLS 1.0 January 1999
- RFC 2246
- Based on SSL Version 3.0
- "the differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough that TLS 1.0 and SSL 3.0 do not interoperate."
-

TLS 1.1 (SSL 3.2)

-
- RFC 4346 April 2006

- CBC attack protection
- Better IV
- Better padding
-
-

TLS 1.2 (SSL 3.3)

- RFC 5246 August 2008
- RFC 6176 March 2011

- MD5-SHA-1 replaced with SHA-256
- Downgrade protections
  Galois/Counter Mode (GCM)

TLS 1.3

- RFC 8446 in August 2018
- Breaks some TLS interception :-)
- Added
  - ChaCha20 stream cipher with the Poly1305 message authentication code
  - Ed25519 and Ed448 digital signature algorithms
  - x25519 and x448 key exchange protocols
- Removed
  - RSA key transport — Doesn't provide forward secrecy
  - CBC mode ciphers — Responsible for BEAST, and Lucky
  - RC4 stream cipher — Not secure for use in HTTPS
  - SHA-1 hash function — Deprecated in favor of SHA-2
  - Arbitrary Diffie-Hellman groups — CVE-2016-0701
  - Export ciphers — Responsible for FREAK and LogJam

Key exchange/agreement and authentication

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Status
RSA	Yes	Yes	Yes	Yes	Yes	No	
DH-RSA	No	Yes	Yes	Yes	Yes	No	
DHE-RSA (forward secrecy)	No	Yes	Yes	Yes	Yes	Yes	
ECDH-RSA	No	No	Yes	Yes	Yes	No	
ECDHE-RSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
DH-DSS	No	Yes	Yes	Yes	Yes	No	
DHE-DSS (forward secrecy)	No	Yes	Yes	Yes	Yes	No[45]	
ECDH-ECDSA	No	No	Yes	Yes	Yes	No	
ECDHE-ECDSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
PSK	No	No	Yes	Yes	Yes		Defined for TLS 1.2 in RFCs
PSK-RSA	No	No	Yes	Yes	Yes		
DHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes		
ECDHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes		
SRP	No	No	Yes	Yes	Yes		
SRP-DSS	No	No	Yes	Yes	Yes		
SRP-RSA	No	No	Yes	Yes	Yes		
Kerberos	No	No	Yes	Yes	Yes		
DH-ANON (insecure)	No	Yes	Yes	Yes	Yes		
ECDH-ANON (insecure)	No	No	Yes	Yes	Yes		
GOST R 34.10-94 / 34.10-2001[46]	No	No	Yes	Yes	Yes		Proposed in RFC drafts

TLS1.3 Adoption

- Slowly in browsers
- Breaks Bluecoat like interception
- Not enabled by default
- Adoption slow

University of Amsterdam

Jaap van Ginkel

Security of Systems and Networks

October 10, 2019 SSH SSL TLS

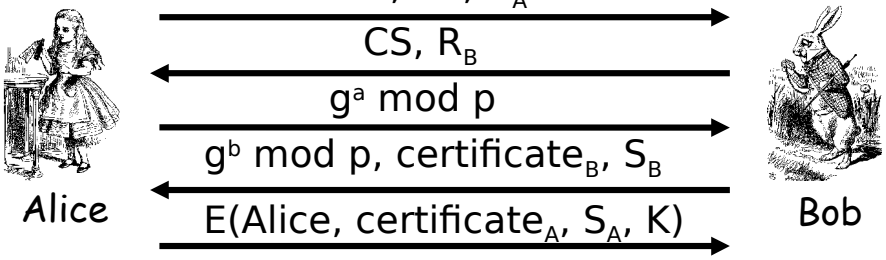
SSH

- Creates a "secure tunnel"
- Insecure command sent thru SSH tunnel are then secure
- SSH used with things like rlogin
  - Why is rlogin insecure without SSH?
  - Why is rlogin secure with SSH?
- SSH is a relatively simple protocol

SSH

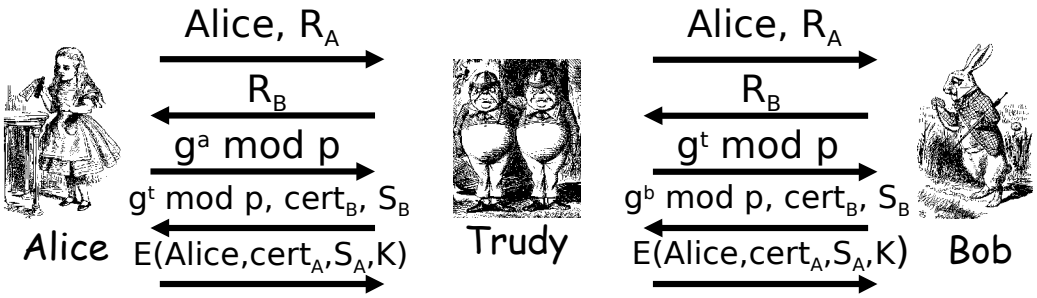
- SSH authentication can be based on:
  - Public keys, or
  - Digital certificates, or
  - Passwords
- Here, we consider ***certificate*** mode
  - Other modes, see homework problems
- We consider slightly simplified SSH…

Simplified SSH



- CP = “crypto proposed”, and CS = “crypto selected”
- H = h(Alice,Bob,CP,CS,R_A,R_B,g^a mod p,g^b mod p,g^{ab} mod p)
- $S_B = [H]_{Bob}$
- $S_A = [H, Alice, certificate_A]_{Alice}$
- K = g^{ab} mod p

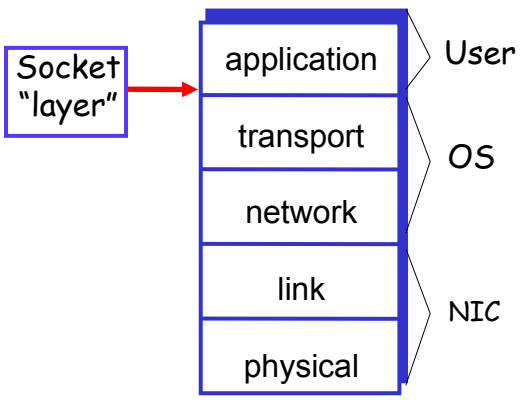
MiM Attack on SSH?



- Where does this attack fail?
- Alice computes:
  - $H_a = h(Alice,Bob,CP,CS,R_A,R_B,g^a \bmod p,g^t \bmod p,g^{at} \bmod p)$
- But Bob signs:
  - $H_b = h(Alice,Bob,CP,CS,R_A,R_B,g^t \bmod p,g^b \bmod p,g^{bt} \bmod p)$

Socket layer

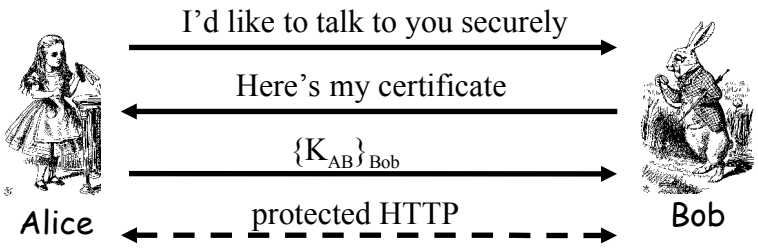
- "Socket layer" lives between application and transport layers
- SSL usually lies between HTTP and TCP



What is SSL?

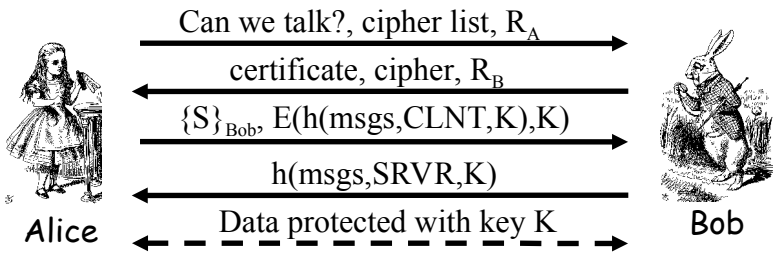
- SSL is the protocol used for most secure transactions over the Internet
- For example, if you want to buy a book at amazon.com…
  - You want to be sure you are dealing with Amazon (**authentication**)
  - Your credit card information must be protected in transit (**confidentiality** and/or **integrity**)
  - As long as you have money, Amazon doesn't care who you are (authentication need not be mutual)

Simple SSL-like Protocol



- Is Alice sure she's talking to Bob?
- Is Bob sure he's talking to Alice?

Simplified SSL Protocol



- S is **pre-master secret**
- K = h(S,R_A,R_B)
- msgs = all previous messages
- CLNT and SRVR are constants

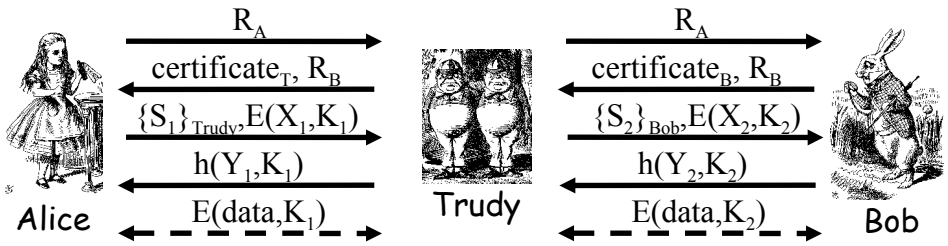
SSL Keys

- 6 "keys" derived from $K = \text{hash}(S, R_A, R_B)$
  - 2 encryption keys: send and receive
  - 2 integrity keys: send and receive
  - 2 IVs: send and receive
  - Why different keys in each direction?
- **Q:** Why is $h(\text{msgs}, \text{CLNT}, K)$ encrypted (and integrity protected)?
- **A:** It adds no security…

SSL Authentication

- Alice authenticates Bob, not vice-versa
  - How does client authenticate server?
  - Why does server not authenticate client?
- Mutual authentication is possible: Bob sends **certificate request** in message 2
  - This requires client to have certificate
  - If server wants to authenticate client, server could instead require (encrypted) password

SSL MiM Attack

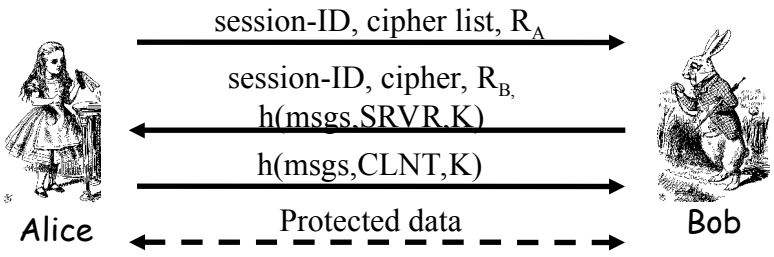


- **Q:** What prevents this MiM attack?
- **A:** Bob's certificate must be signed by a certificate authority (such as Verisign)
- What does Web browser do if sig. not valid?
- What does user do if signature is not valid?

SSL Sessions vs Connections

- SSL **session** is established as shown on previous slides
- SSL designed for use with HTTP 1.0
- HTTP 1.0 usually opens multiple simultaneous (parallel) **connections**
- SSL session establishment is costly
  - Due to public key operations
- SSL has an efficient protocol for opening new connections given an existing session

SSL Connection



- Assuming SSL **session** exists
- So S is already known to Alice and Bob
- Both sides must remember session-ID
- Again, $K = h(S,R_A,R_B)$
- **No public key operations!** (relies on known S)

SSL vs IPSec

- IPSec – discussed in next section
  - Lives at the network layer (part of the OS)
  - Has encryption, integrity, authentication, etc.
  - Is overly complex (including serious flaws)
- SSL (and IEEE standard known as TLS)
  - Lives at socket layer (part of user space)
  - Has encryption, integrity, authentication, etc.
  - Has a simpler specification

SSL vs IPSec

- IPSec implementation
  - Requires changes to OS, but no changes to applications
- SSL implementation
  - Requires changes to applications, but no changes to OS
- SSL built into Web application early on (Netscape)
- IPSec used in VPN applications (secure tunnel)
- Reluctance to retrofit applications for SSL
- Reluctance to use IPSec due to complexity and interoperability issues
- Result? **Internet less secure than it should be!**

Secure Network Programming API

- Early research efforts toward transport layer security included the Secure Network Programming (SNP) application programming interface (API), which in 1993 explored the approach of having a secure transport layer API closely resembling Berkeley sockets, to facilitate retrofitting preexisting network applications with security measures.[4]

SSL 1.0, 2.0 and 3.0

- Developed by Netscape engineers
  - Phil Karlton, Alan Freier
- Version 1.0 never released
- Version 2.0 February 1995
  - some (serious) security flaws
- Version 3.0 1996
  - Complete redesign
  - Basis for current TLS versions

TLS 1.0 (SSL 3.1)

- TLS 1.0 January 1999
- RFC 2246
- Based on SSL Version 3.0
- "the differences between this protocol and SSL 3.0 are not dramatic, but they are significant enough that TLS 1.0 and SSL 3.0 do not interoperate."
-

TLS 1.1 (SSL 3.2)

-
- RFC 4346 April 2006

- CBC attack protection
- Better IV
- Better padding
-
-

TLS 1.2 (SSL 3.3)

- RFC 5246 August 2008
- RFC 6176 March 2011

- MD5-SHA-1 replaced with SHA-256
- Downgrade protections
  Galois/Counter Mode (GCM)

TLS 1.3

- RFC 8446 in August 2018
- Breaks some TLS interception :-)
- Added
  - ChaCha20 stream cipher with the Poly1305 message authentication code
  - Ed25519 and Ed448 digital signature algorithms
  - x25519 and x448 key exchange protocols
- Removed
  - RSA key transport — Doesn't provide forward secrecy
  - CBC mode ciphers — Responsible for BEAST, and Lucky
  - RC4 stream cipher — Not secure for use in HTTPS
  - SHA-1 hash function — Deprecated in favor of SHA-2
  - Arbitrary Diffie-Hellman groups — CVE-2016-0701
  - Export ciphers — Responsible for FREAK and LogJam

Key exchange/agreement and authentication

Algorithm	SSL 2.0	SSL 3.0	TLS 1.0	TLS 1.1	TLS 1.2	TLS 1.3	Status
RSA	Yes	Yes	Yes	Yes	Yes	No	
DH-RSA	No	Yes	Yes	Yes	Yes	No	
DHE-RSA (forward secrecy)	No	Yes	Yes	Yes	Yes	Yes	
ECDH-RSA	No	No	Yes	Yes	Yes	No	
ECDHE-RSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
DH-DSS	No	Yes	Yes	Yes	Yes	No	
DHE-DSS (forward secrecy)	No	Yes	Yes	Yes	Yes	No[45]	
ECDH-ECDSA	No	No	Yes	Yes	Yes	No	
ECDHE-ECDSA (forward secrecy)	No	No	Yes	Yes	Yes	Yes	
PSK	No	No	Yes	Yes	Yes		Defined for TLS 1.2 in RFCs
PSK-RSA	No	No	Yes	Yes	Yes		
DHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes		
ECDHE-PSK (forward secrecy)	No	No	Yes	Yes	Yes		
SRP	No	No	Yes	Yes	Yes		
SRP-DSS	No	No	Yes	Yes	Yes		
SRP-RSA	No	No	Yes	Yes	Yes		
Kerberos	No	No	Yes	Yes	Yes		
DH-ANON (insecure)	No	Yes	Yes	Yes	Yes		
ECDH-ANON (insecure)	No	No	Yes	Yes	Yes		
GOST R 34.10-94 / 34.10-2001[46]	No	No	Yes	Yes	Yes		Proposed in RFC drafts

TLS1.3 Adoption

- Slowly in browsers
- Breaks Bluecoat like interception
- Not enabled by default
- Adoption slow