Cryptanalysis

Protocols and Random Numbers

John Manferdelli
JohnManferdelli@hotmail.com

Random Numbers

- Critical in Cryptographic Algorithms
- No single test
  - Unpredictability
  - Statistical Tests
- Random Number weaknesses and Key management are greatest points of attack for otherwise “safe” cryptosystem.
- Can’t generate enough Random bits so use Pseudo Random Number Generators

- Reference
  - J. Kelsey, B. Schneier, D. Wagner, and C. Hall, *“Cryptanalytic Attacks on Pseudorandom Number Generators”*, Fast Software Encryption, Fifth International Workshop Proceedings (March 1998), Springer-Verlag, 1998, pp. 168-188.

Random Numbers

- Requirements
- Attacks
- Entropy
- Mixing
- PRNG
- 800-90

Cryptographic Random Numbers

- Requirements
  - $\Pr([x_1,x_2,\ldots,x_n]= [a_1,a_2,\ldots,a_n])=2^{-n}$. $\Pr([x_1,x_2,\ldots,x_n]=$
  - $H(\mathbf{x}=[x_1,x_2,\ldots,x_n])= n$
  - $\Pr([x_1,x_2,\ldots,x_n]= [a_1,a_2,\ldots,a_n])= \Pr(x_1=a_1)\cdot\Pr(x_2=a_2) \cdot\ldots\cdot\Pr(x_n=a_{n\cdot})$
  - $\Pr([x_1,x_2,\ldots,x_n] | [,x_2,\ldots,x_n])=\Pr(x_1)$
  - Guessing values at random with equal probability is as well as you can do

- Failure tests
  - Frequency tests
  - Hidden Markov modeling

Remember: H for the key distributions

- Distribution A: $H(X)= ¼ \lg(4) + ¼ \lg(4) + ¼ \lg(4) +1/4 \lg(4) = 2$ bits
- Distribution B: $H(X)= 16x(1/16) \lg(16)= 4$ bits
- Distribution C: $H(X)= 2^n x(1/2^n) \lg(2^n) = n$ bits
  - Expected time for key search is ~ 2^n.

- Distribution A’: $H(X) = ½ \lg(2) + 3 x(1/6 \lg(6))= 1.79$ bits
- Distribution B’: $H(X) = ½ \lg(2) + 15 x(1/30 \lg(30))= 2.95$ bits
- Distribution C’: $H(X) = ½ \lg(2) + ½(2^n-1)x(1/(2^n-1) \lg(2^n-1))= n/2+1$ bits
  - Expected time for key search is ~ $2^{n/2}+1$.

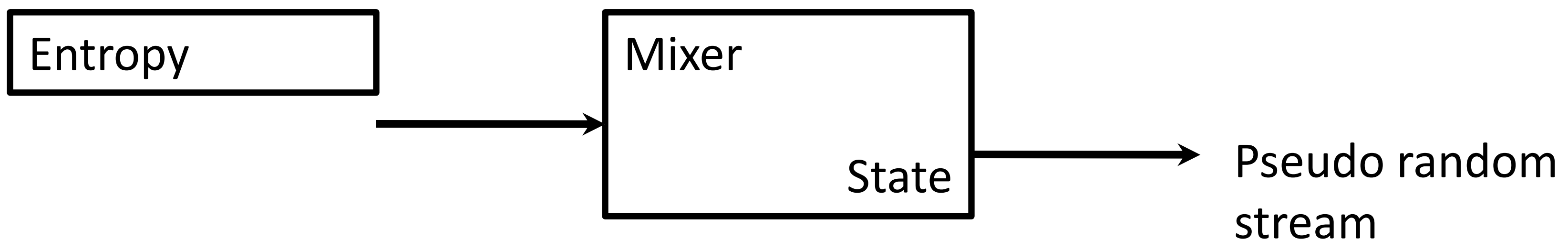
Sources of Entropy

- Coin Tosses
- Radioactive decay
- Typing Speed
- Thermal noise
- Ring Oscillator
- Lava Lamps
- Noisy diode
- Disk arm speed variation
- Process id, thread id
- Drift between clock and timer interrupts
- Ticks since boot
- Memory stats
- Disk Free
- Cursor
- Counters
- Execution time (Jitter)

Some entropy source calculations

- Fair coin toss:
  - Each coin toss adds 1 bit of entropy
- Biased (but independent) coin tosses
  - Pr(x=1)= 1/4, Pr(x=0)= 3/4.
  - Entropy: $-1/4 \lg(1/4)-3/4 \lg(3/4)= 1/2 + 1/4 \lg(3) \approx .85$ bit
- If John wears red shoes, $x_i=1$ otherwise $x_i=0$. $x_{i+1}=x_i \oplus 1$
  - Even if John wears red shoes randomly with p=1/2, every 2n bits only have n bits of entropy.
  - Calculate entropy with a different “wear red shoes” distribution”

Pseudo random number generation



- Smooth and stretch entropy
- Must first estimate entropy input and maintain sufficient entropy
- Idea is to generate n bit key state should maintain n bits of entropy

Pseudo-Random Generators (PRNGs)

- "Anyone discussing deterministic generation of random number is, strictly speaking, already in a state of sin" – von Neuman.
- Output of pseudo-random number generators must produce output that looks random
  - Start with a fixed state S and collect inputs with high entropy
- Generators can be built using
  - Block ciphers
  - Hash functions
  - Stream Cipher

Guidelines for PRNG

- Base the PRNG on something strong.
- Make sure the whole PRNG state changes over time..
- Do “catastrophic reseeding” of the PRNG.
- Resist backtracking.
- Resist Chosen-Input Attacks.
- Recover from Compromises Quickly
- Use a hash function to protect vulnerable PRNG outputs and entropy mixing.
- Hash PRNG inputs with a counter or timestamp before use.
- Occasionally generate a new starting PRNG state.

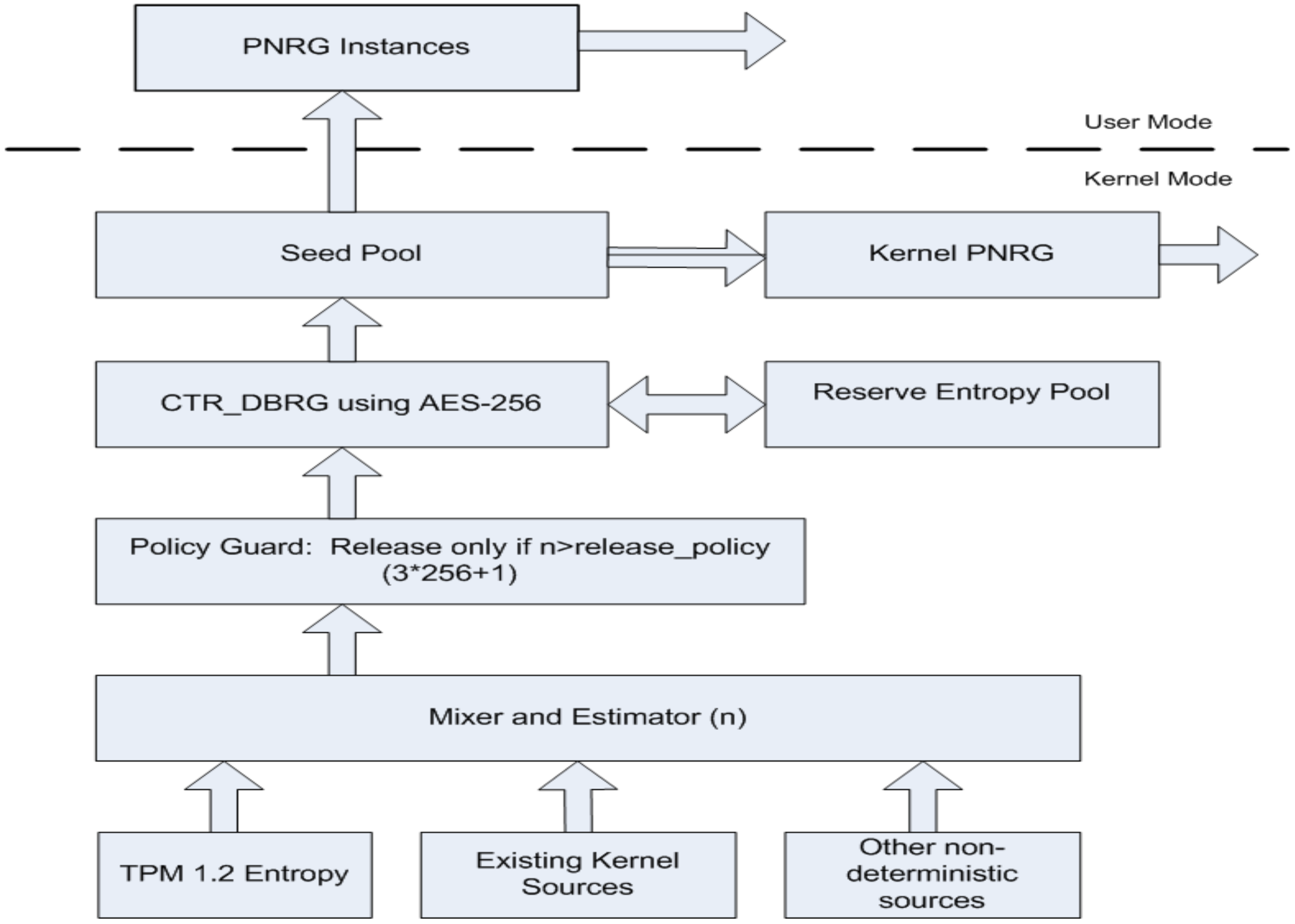
RNG Attacks

- Direct Cryptanalytic attack
- Entropy Input Guessing
- Input-based attack
  - Known input
  - Replayed input
  - Chosen input
- State compromise extension attacks
  - Backtracking attacks (forward immunity)
  - Permanent compromise attacks (backward immunity)
  - Iterative guessing attacks
  - Meet-in-the-middle attacks
- Oversampling

Popular PRNGs

- FIPS 186
  - t, c 160 bits
  - H= $t_1 \mid\mid t_2 \mid\mid \ldots \mid\mid t_5$
  - Pad c with 0s to get 512 bit giving M
  - Apply SHA-1 step
- ANSI 9.17
  - I= $E_k(D)$. D= timestamp.
  - $x_i = E_k(I \oplus s)$, s= State
  - s= $E_k(x_i \oplus s)$
- Current NIST 800-90
  - HASH-256
  - CTR-AES-256
  - Dual Elliptic Curve

Sample 800-90 RNG System



HASH-256

Initiate

```
1.    seed_material = entropy_input || nonce || personalization_string.
2.    seed = Hash_df (seed_material, seedlen).
3.    V = seed.
4.    C = Hash_df ((0x00 || V), seedlen).
5.    reseed_counter = 1.
6.    Return V, C, and reseed_counter as initial_working_state.
```

HASH-256

Generate

```
1.  If reseed_counter > reseed_interval, then return reseed required.
2.  If (additional_input != Null), then do
        w = Hash (0x02 || V || additional_input).
      V = (V+w) mod $2^{seedlen}$.
1.  (returned_bits) = Hashgen (requested_number_of_bits, V).
2.  H = Hash(0x03 || V).
3.  V = (V+H+C+reseed_counter) mod $2^{seedlen}$.
4.  reseed_counter= reseed_counter+1.
5.  Return SUCCESS, returned_bits, and the new values of V, C, and
    reseed_counter for the new_working_state.
```

HASH-256

Hash_df

```
1.  temp = the Null string.
2.  len =no_of _bits_to_return/outlen□
3.  counter = 8-bit binary value representing 1.
4.  for i = 1 to len do
        temp= temp||Hash(counter||no_of_bits_to_return||input_string).
        counter= counter+1.
5.  requested_bits= Leftmost (no_of_bits_to_return) of temp.
6.  Return SUCCESS and requested_bits.
```

HASH-256

Hashgen

```
1.  m =requested_no_of _bits□/outlen
2.  data = V.
3.  W = the Null string.
4.  For i = 1 to m
        $w_i$ = Hash (data).
        W = W || $w_i$.
        data = (data + 1) mod $2^{seedlen}$.
5.  returned_bits = Leftmost (requested_no_of_bits) bits of W.
6.  Return returned_bits.
```

CTR-AES-256

Initiate

```
1. $temp$= len ($personalization\_string$).
2. If ($temp<seedlen$), then
       $personalization\_string = personalization\_string \,||\, 0^{seedlen-temp}$.
3. $seed\_material = entropy\_input \oplus personalization\_string$
4. $Key = 0^{keylen}$.
5. $V = 0^{outlen}$.
6. ($Key$, $V$)= CTR_DRBG_Update($seed\_material$, $Key$, $V$).
7. $reseed\_counter$= 1.
8. Return $V$, $Key$, and $reseed\_counter$ as the $initial\_working\_state$.
```

CTR-AES-256

Generate

```
1.  If reseed_counter > reseed_interval, then return reseed required.
2.  If (additional_input ≠ Null), then
        temp = len (additional_input).
        If (temp<seedlen) then additional_input= additional_input||$0^{seedlen - temp}$.
        (Key, V) = CTR_DRBG_Update (additional_input, Key, V).
3.  temp= Null.
4.  While (len (temp) < requested_number_of_bits)
        V = (V+1) mod $2^{outlen}$.
        output_block = Block_Encrypt(Key, V).
        temp = temp || output_block.
5.  returned_bits = Leftmost requested_number_of_bits of temp.
6.  (Key, V) = CTR_DRBG_Update(additional_input, Key, V).
7.  reseed_counter = reseed_counter + 1.
8.  Return SUCCESS and returned_bits; also return Key, V, and reseed_counteras the
    new_working_state.
```

CTR-AES-256

Update

```
1.  temp = Null.
2.  While(len (temp)<seedlen)
      $V = (V+1) \bmod 2^{outlen}$.
      output_block = Block_Encrypt(Key, V).
      temp = temp || ouput_block.
3.  temp = Leftmost seedlen bits of temp.
4.  temp = temp $\oplus$ provided_data.
5.  Key = Leftmost keylen bits of temp.
6.  V = Rightmost outlen bits of temp.
7.  Return the new values of Key and V.
```

Preliminaries: Elliptic Curves

- Elliptic curves are the set of points (x,y) with coordinates in a field F that are solutions to an equation:

  $$y^2 = x^3 + ax + b$$

- These points (plus an identity) form a group.
- All of the curves that we will be discussing are over finite fields (characteristic p) and will have prime order q.

The Dual EC PRNG

- φ : prime curve → integers
  $\varphi\,(x,y) = x$
- P, Q points on the curve (per SP800-90)

$$\begin{tikzcd}
 & \varphi(r_i*P) \arrow[dl] & & & & \\
s_{i+1} \,/\, s_i \arrow[rr] & & r_i \arrow[ul] \arrow[r] & \varphi(r_i*Q) \arrow[r] & t_i \arrow[r] & \mathrm{LSB}_{\text{bitlen-16}}(t_i)
\end{tikzcd}$$

Equations:

$r_i = \varphi(s_i*P)$ $\qquad$ $t_i = \varphi(r_i*Q)$ $\qquad$ $s_{i+1} = \varphi(r_i*P)$

Protocols

Unless otherwise noted, remaining slides courtesy of Mark Stamp, SJSU
See: Information Security: Principles and Practice, Mark Stamp

Protocol

- Human protocols — the rules followed in human interactions
  - Example: Asking a question in class
- Networking protocols — rules followed in networked communication systems
  - Examples: HTTP, FTP, etc.
- Security protocol — the (communication) rules followed in a security application
  - Examples: SSL, IPSec, Kerberos, etc.

Protocols

- Protocol flaws can be very subtle
- Several well-known security protocols have serious flaws
  - Including IPSec, GSM and WEP
- Common to find implementation errors
  - Such as IE implementation of SSL
- Difficult to get protocols right…

Ideal Security Protocol

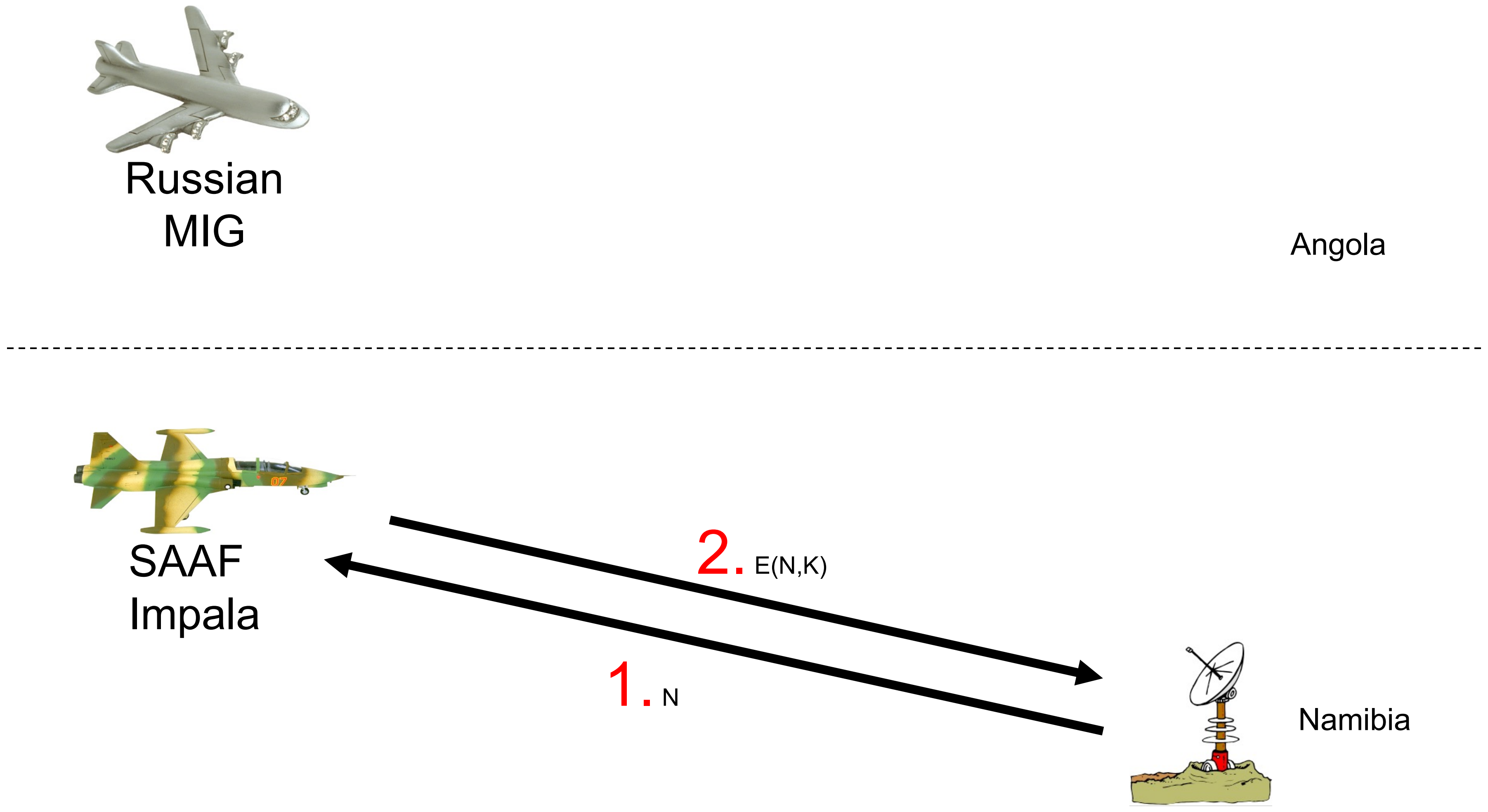
- Satisfies security requirements
  - Requirements must be precise
- Efficient
  - Minimize computational requirement — in particular, costly public key operations
  - Minimize delays/bandwidth
- Not fragile
  - Must work when attacker tries to break it
  - Works even if environment changes
- Easy to use and implement, flexible, etc.
- Very difficult to satisfy all of these!

Simple Security Protocols

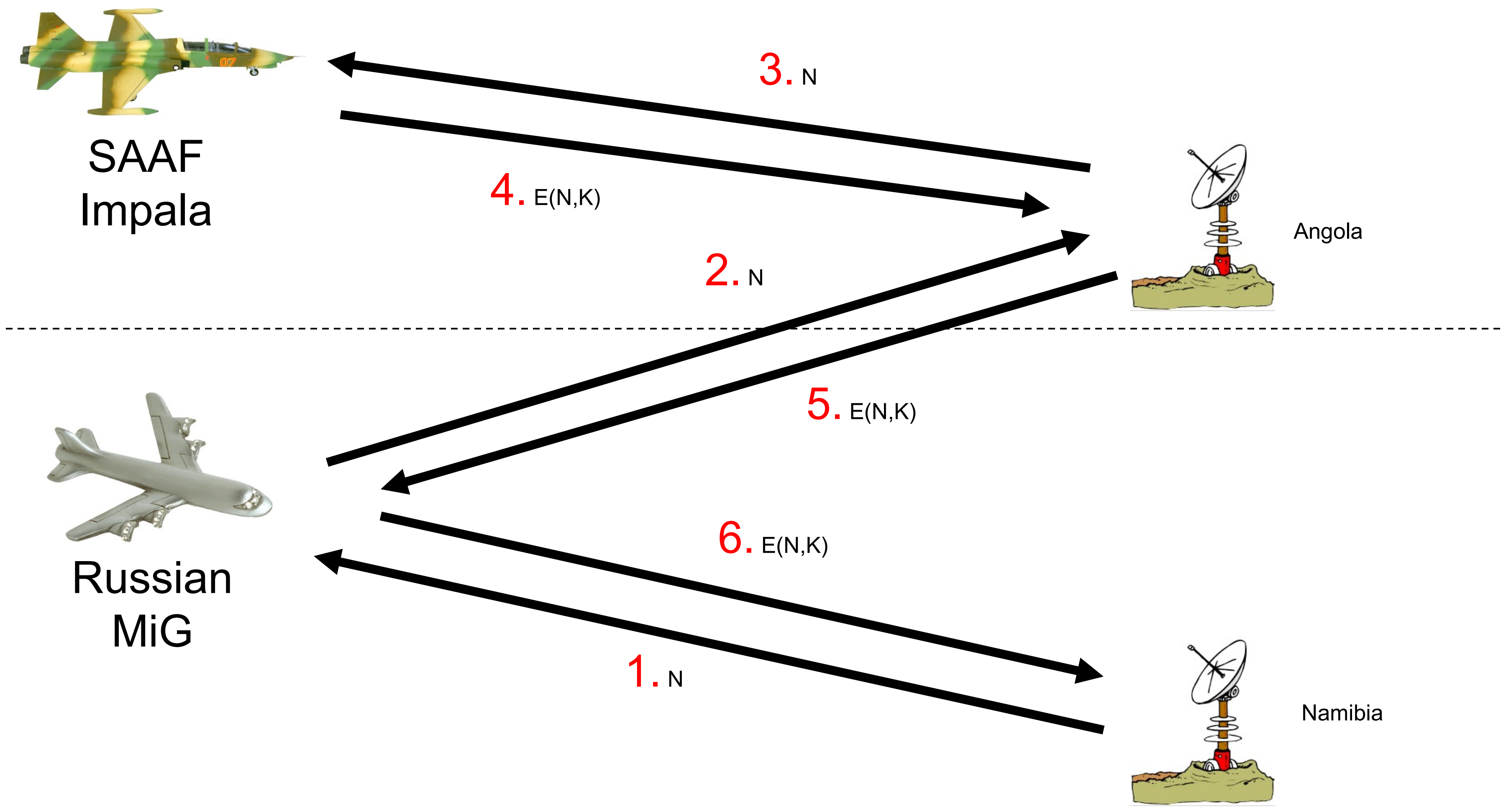
ATM Machine Protocol

1. Insert ATM card
2. Enter PIN
3. Correct PIN?
    - **Yes?** Conduct your transaction(s)
    - **No?** Machine eats card

Identify Friend or Foe (IFF)



MIG in the Middle



Authentication Protocols

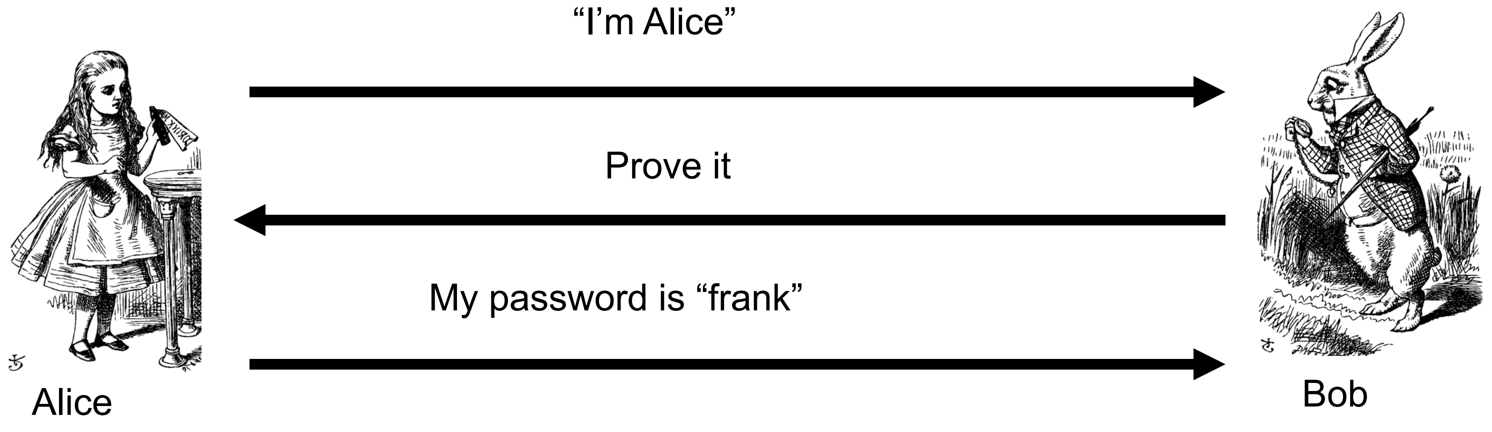
Authentication

- Alice must prove her identity to Bob
  - Alice and Bob can be humans or computers
- May also require Bob to prove he’s Bob (mutual authentication)
- May also need to establish a session key
- May have other requirements, such as
  - Use only public keys
  - Use only symmetric keys
  - Use only a hash function
  - Anonymity, plausible deniability, etc., etc.

Authentication

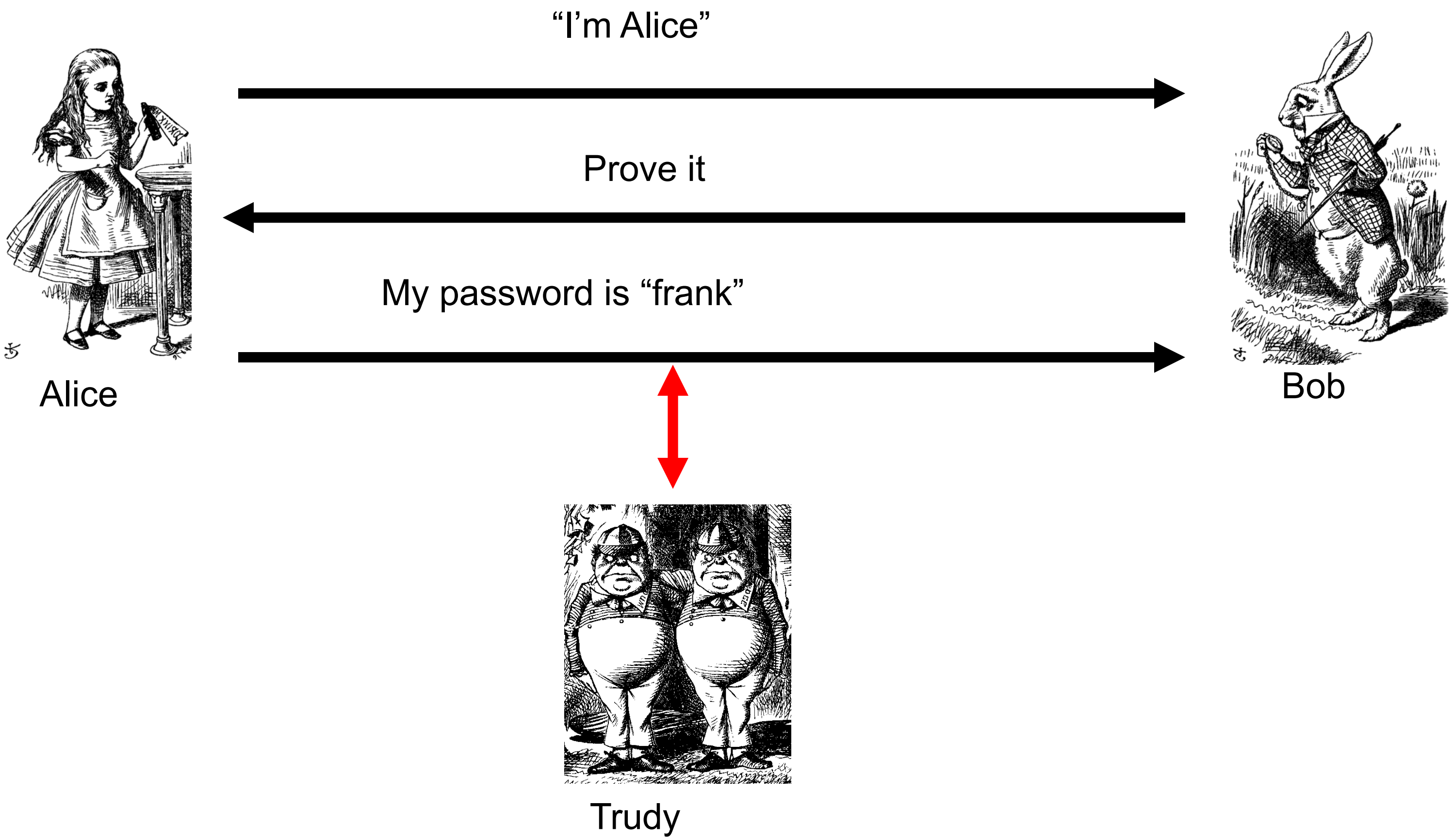
- Authentication on a stand-alone computer is relatively simple
  - "Secure path" is the primary issue
  - Main concern is an attack on authentication software (we discuss software attacks later)
- Authentication over a network is much more complex
  - Attacker can passively observe messages
  - Attacker can replay messages
  - Active attacks may be possible (insert, delete, change messages)

Simple Authentication

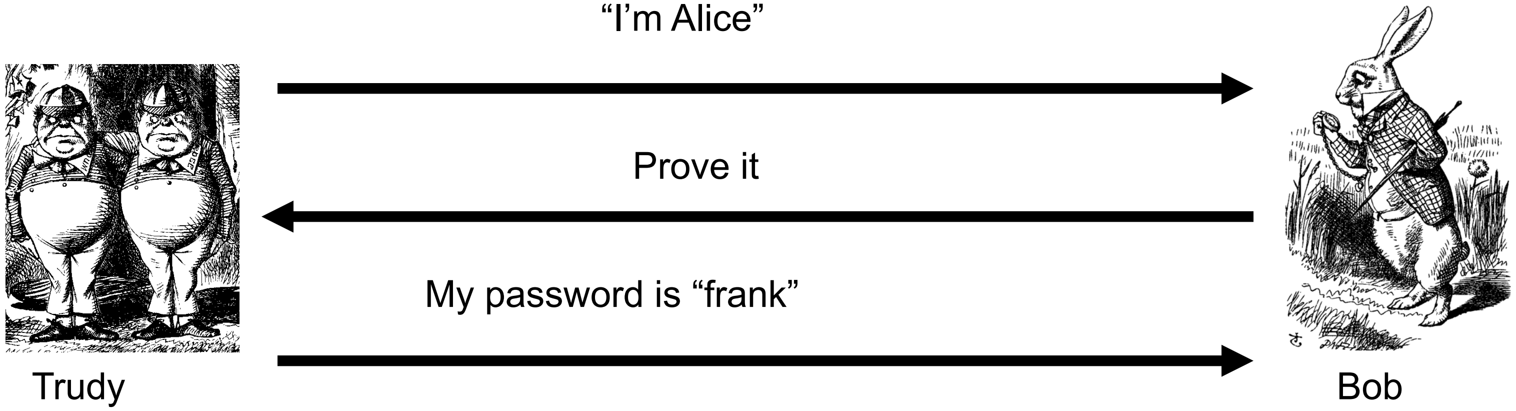


- Simple and may be OK for standalone system
- But insecure for networked system
  - Subject to a replay attack (next 2 slides)
  - Bob must know Alice's password

Authentication Attack

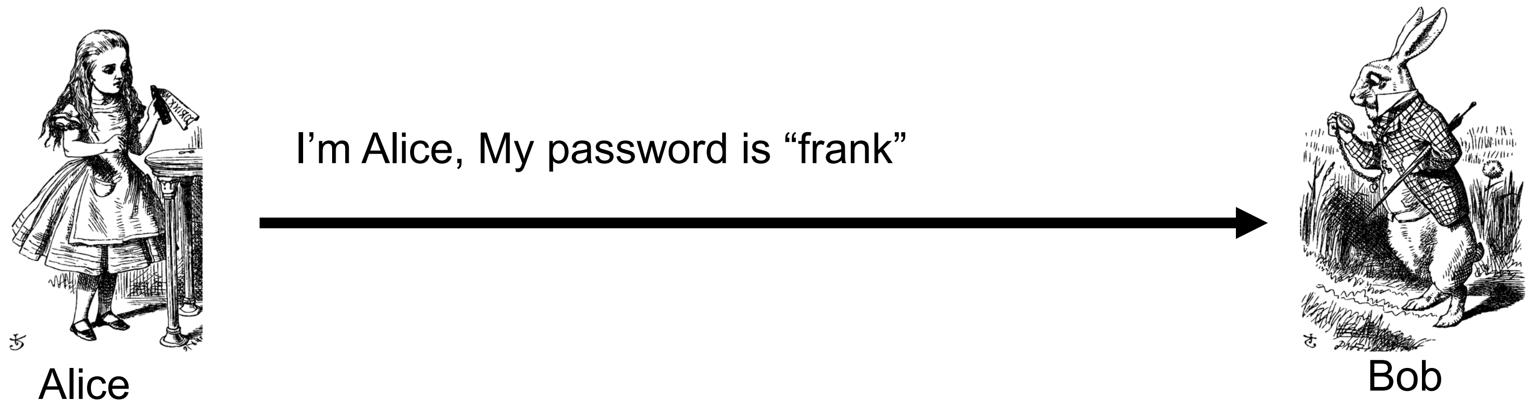


Authentication Attack



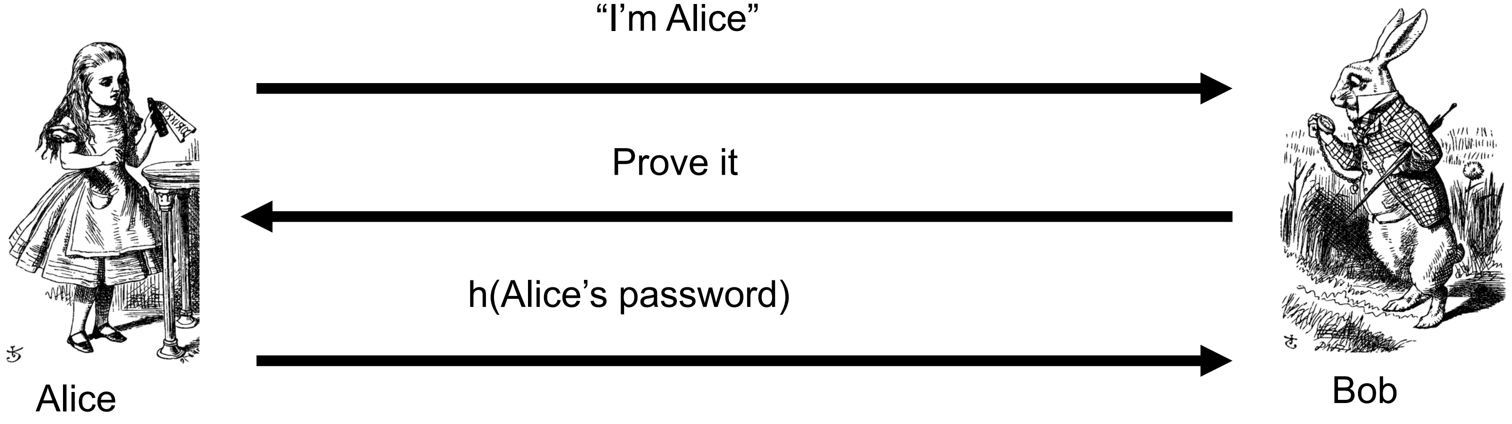
- This is a **replay** attack
- How can we prevent a replay?

Simple Authentication



- More efficient…
- But same problem as previous version

Better Authentication

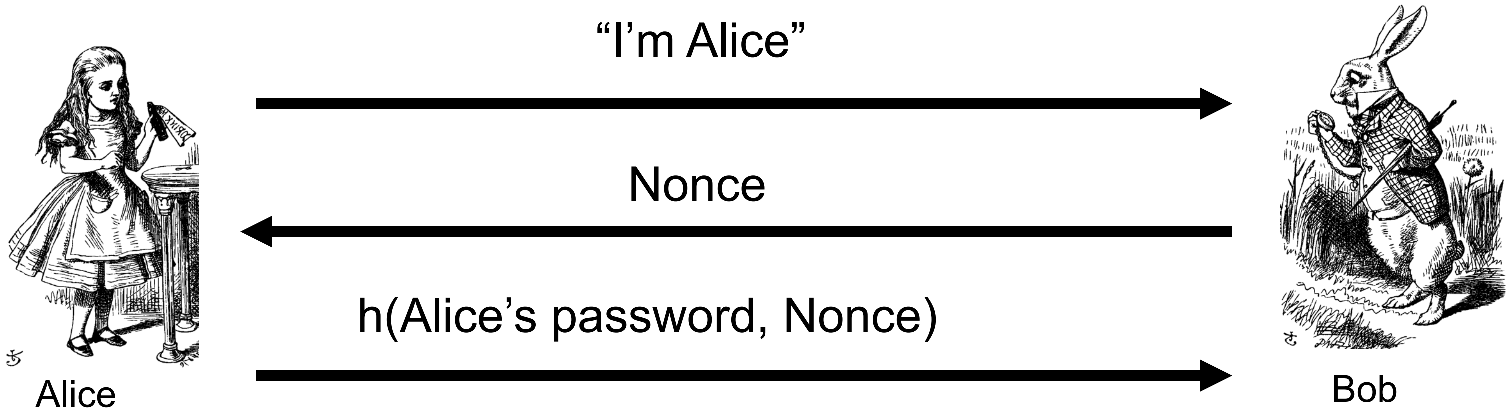


- Better since it hides Alice's password
  - From both Bob and attackers
- But still subject to replay

Challenge-Response

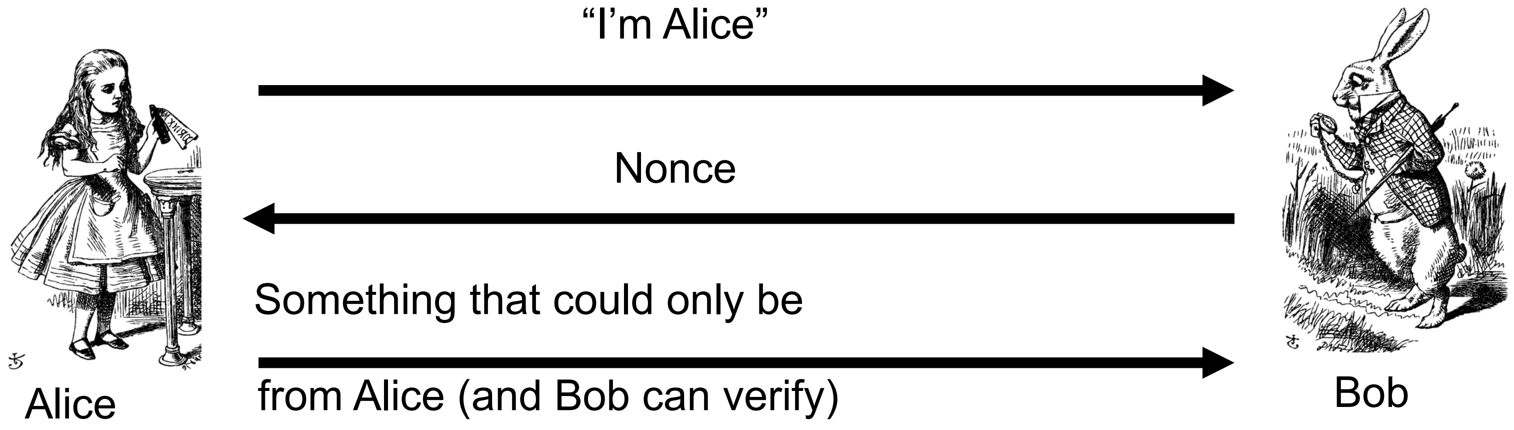
- To prevent replay, challenge-response used
- Suppose Bob wants to authenticate Alice
  - Challenge sent from Bob to Alice
  - Only Alice can provide the correct response
  - Challenge chosen so that replay is not possible
- How to accomplish this?
  - Password is something only Alice should know…
  - For freshness, a "number used once" or **nonce**

Challenge-Response



- Nonce is the challenge
- The hash is the response
- Nonce prevents replay, insures freshness
- Password is something Alice knows
- Note that Bob must know Alice's password

Challenge-Response



- What can we use to achieve this?
- Hashed passwords works, crypto might be better

Symmetric Key Notation

- Encrypt plaintext P with key K

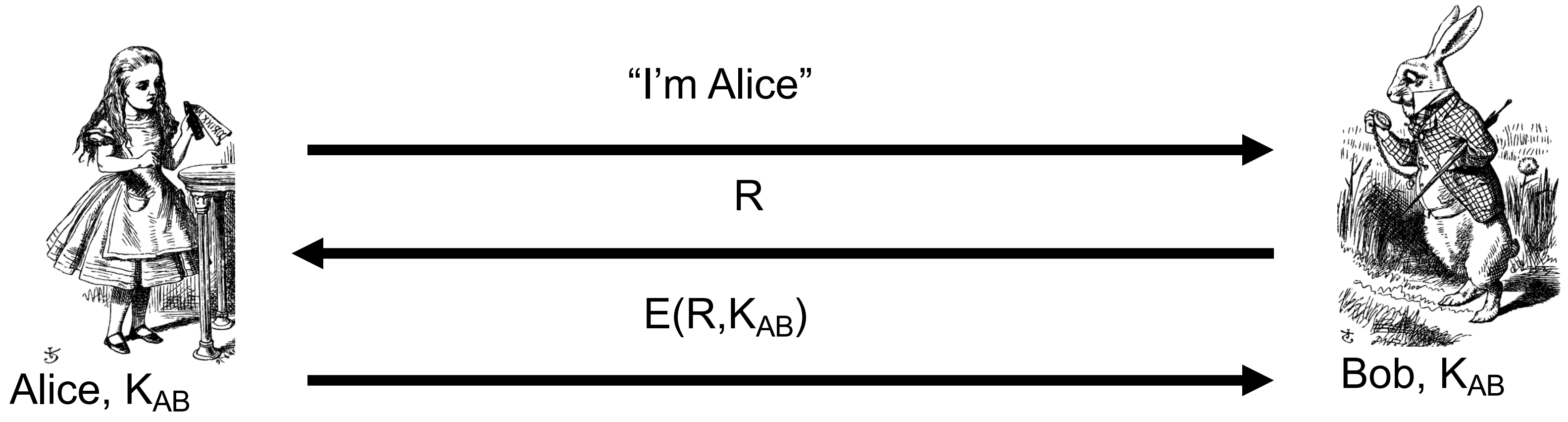
  C = E(P,K)
- Decrypt ciphertext C with key K

  P = D(C,K)
- Here, we are concerned with attacks on **protocols**, not directly on the crypto
- We assume that crypto algorithm is secure

Symmetric Key Authentication

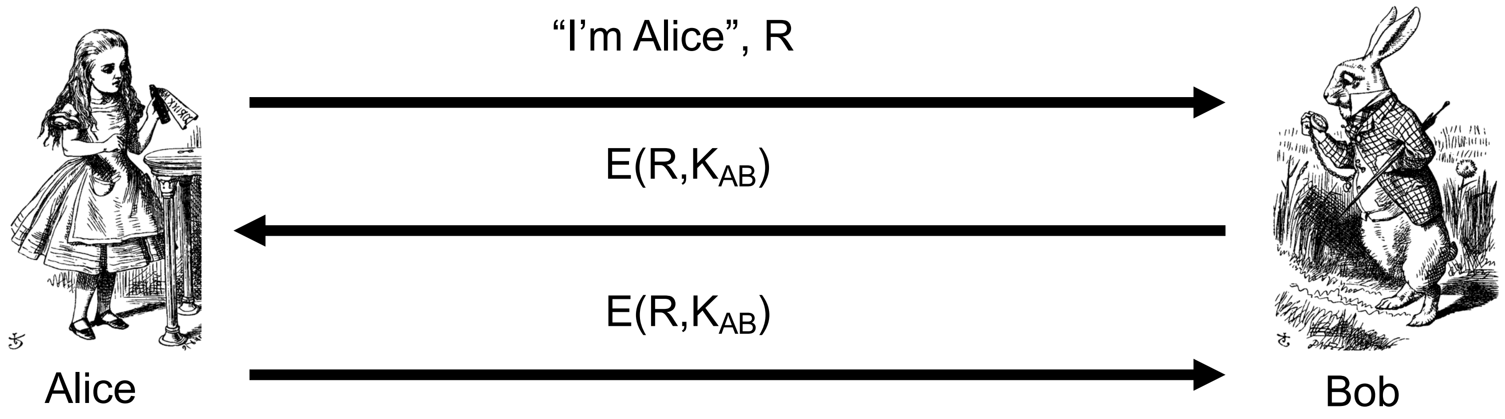
- Alice and Bob share symmetric key K_{AB}
- Key K_{AB} known only to Alice and Bob
- Authenticate by proving knowledge of shared symmetric key
- How to accomplish this?
  - Must not reveal key
  - Must not allow replay attack

Authentication with Symmetric Key



- Secure method for Bob to authenticate Alice
- Alice does not authenticate Bob
- Can we achieve mutual authentication?

Mutual Authentication?

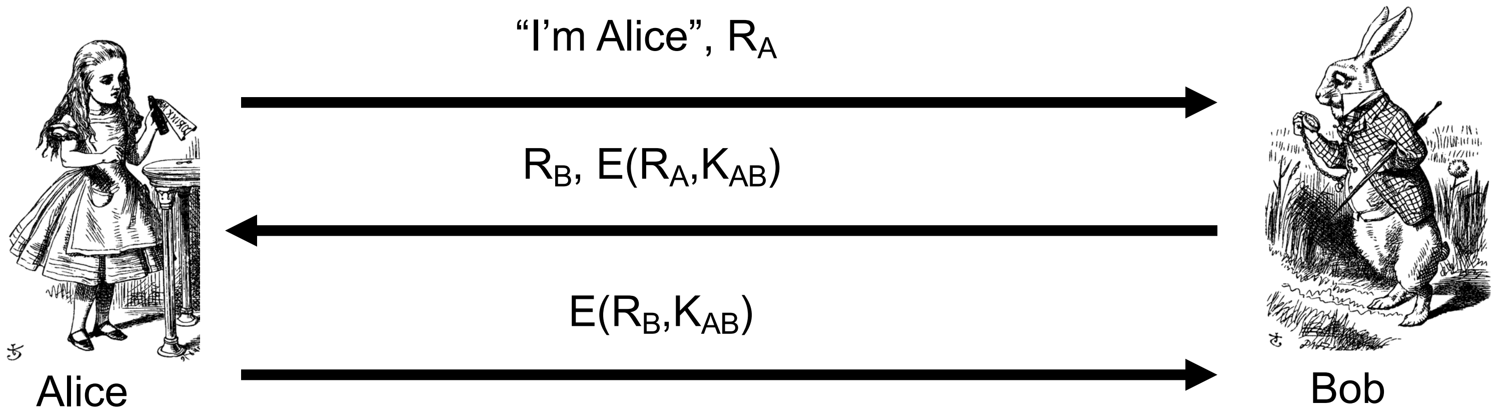


- What's wrong with this picture?
- "Alice" could be Trudy (or anybody else)!

Mutual Authentication

- Since we have a secure one-way authentication protocol…
- The obvious thing to do is to use the protocol twice
  - Once for Bob to authenticate Alice
  - Once for Alice to authenticate Bob
- This has to work…

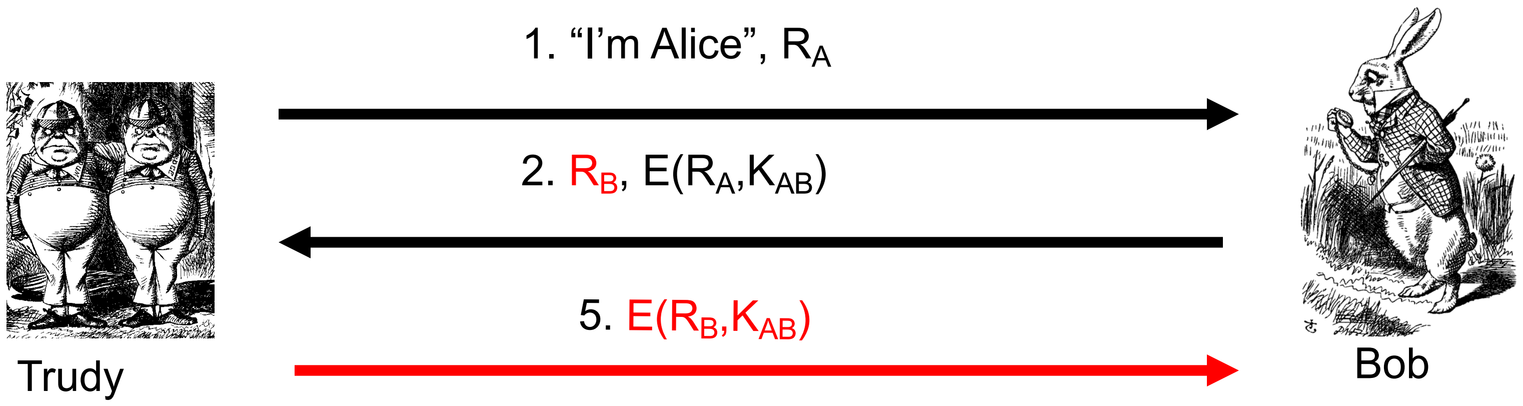
Mutual Authentication

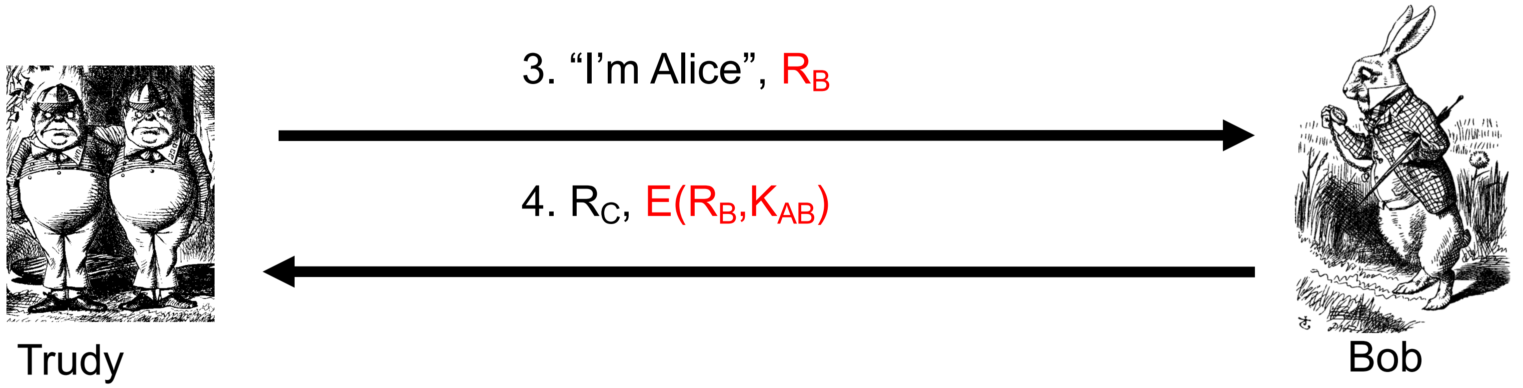


- This provides mutual authentication…

  …or does it? See the next slide

Mutual Authentication Attack

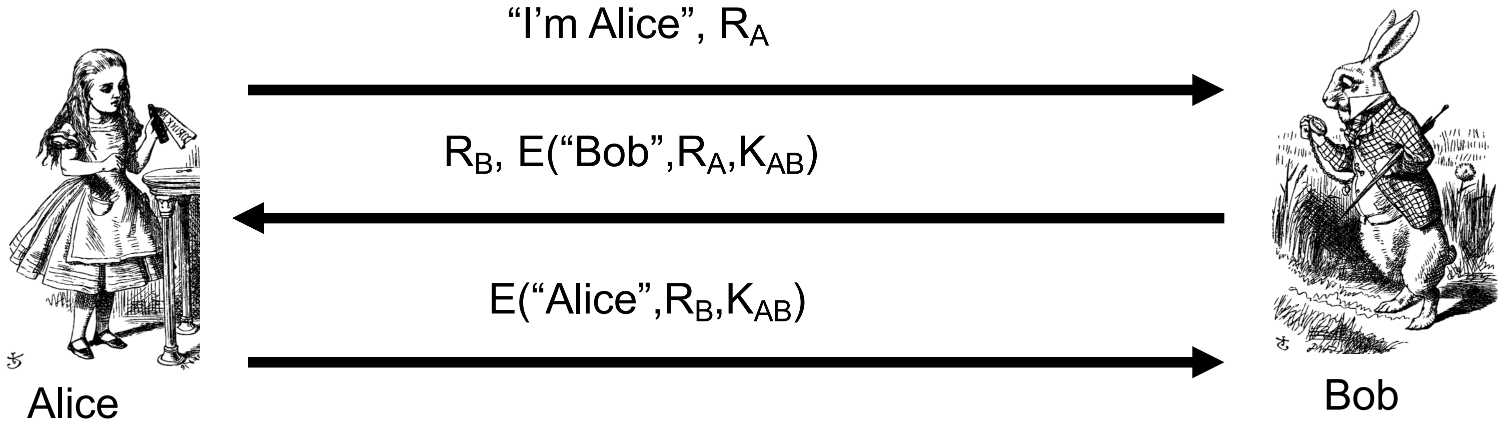




Mutual Authentication

- Our one-way authentication protocol **not** secure for mutual authentication
- Protocols are subtle!
- The "obvious" thing may not be secure
- Also, if assumptions or environment changes, protocol may not work
  - This is a common source of security failure
  - For example, Internet protocols

Symmetric Key Mutual Authentication

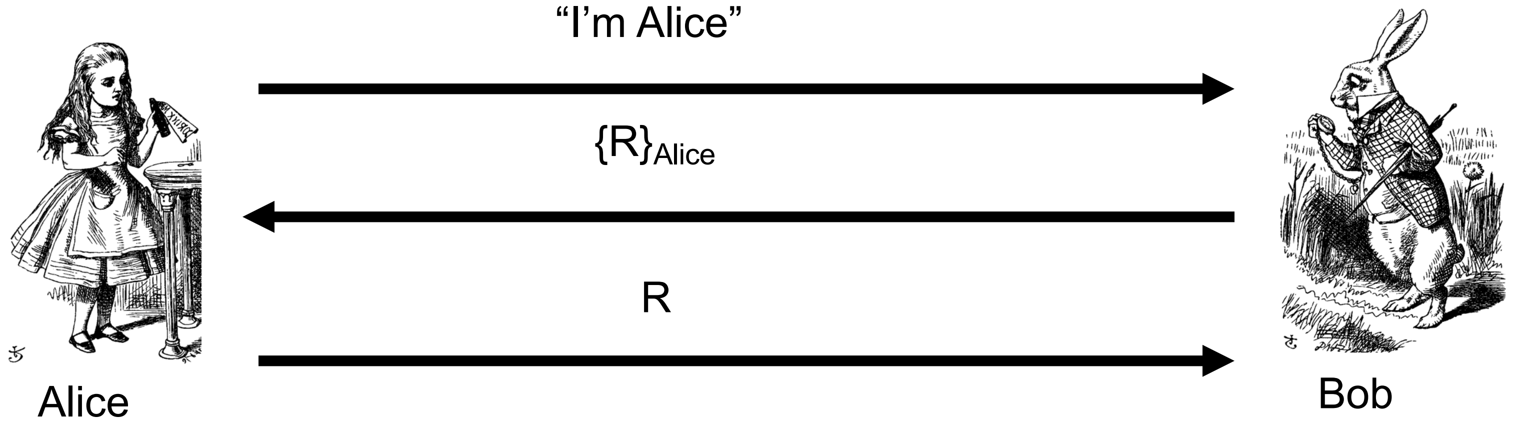


- Do these "insignificant" changes help?
- Yes!

Public Key Notation

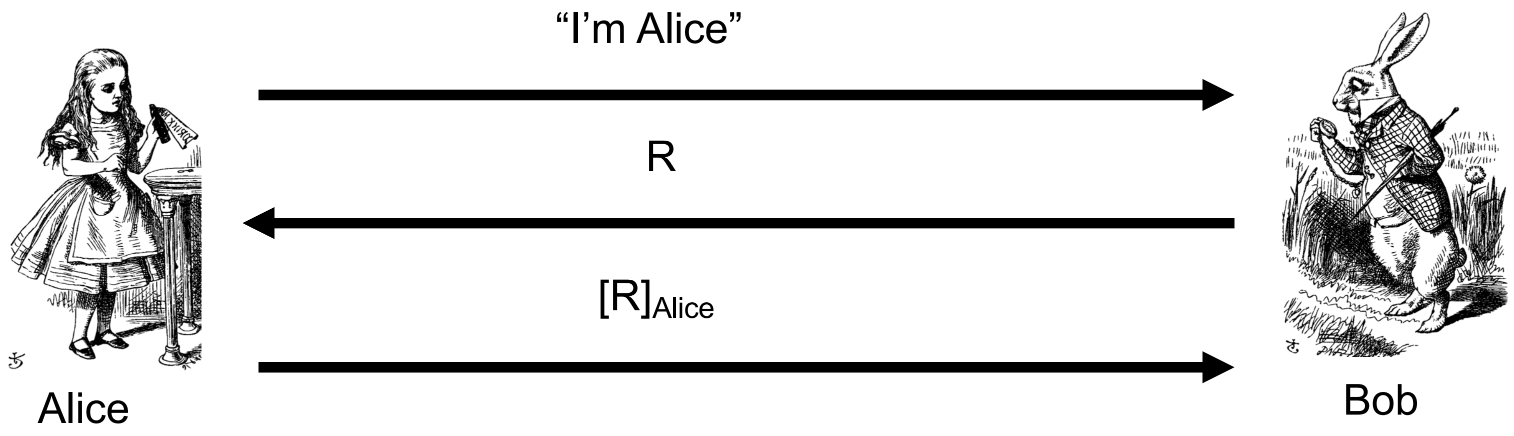
- Encrypt M with Alice's public key: $\{M\}_{Alice}$
- Sign M with Alice's private key: $[M]_{Alice}$
- Then
  - $[\{M\}_{Alice}]_{Alice} = M$
  - $\{[M]_{Alice}\}_{Alice} = M$
- **Anybody** can do **public key** operations
- Only **Alice** can use her **private key** (sign)

Public Key Authentication



- Is this secure?
- Trudy can get Alice to decrypt anything!
  - Must have two key pairs

Public Key Authentication



- Is this secure?
- Trudy can get Alice to sign anything!
  - Must have two key pairs

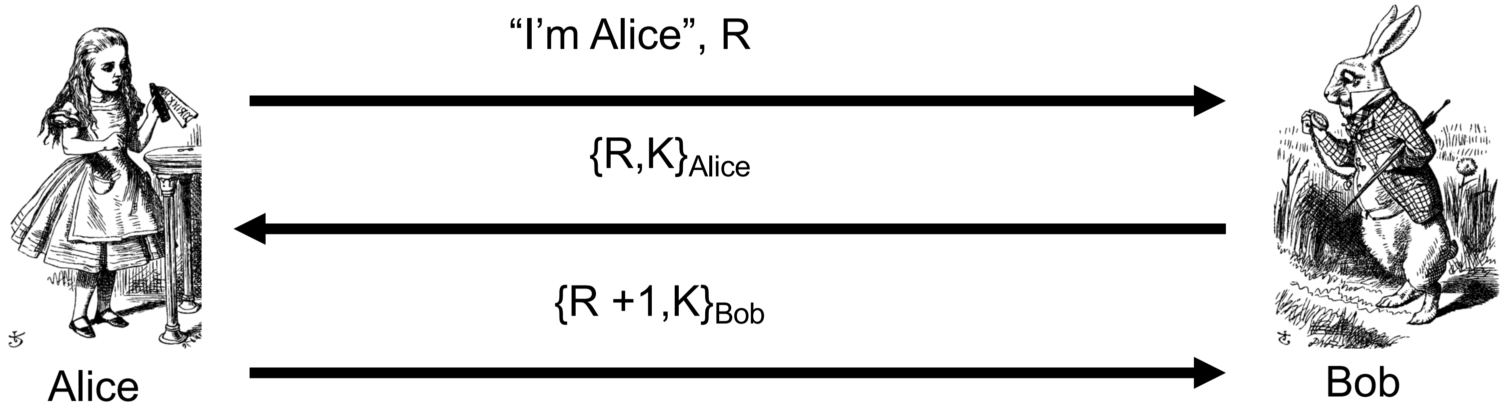
Public Keys

- Never use the same key pair for encryption and signing
- One key pair for encryption/decryption
- A different key pair for signing/verifying signatures

Session Key

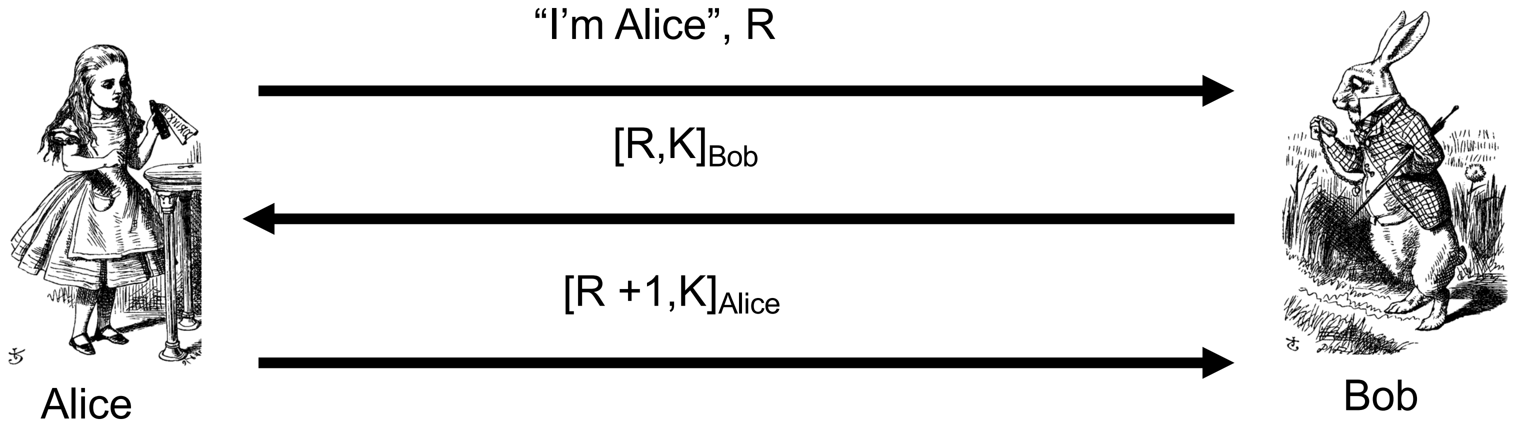
- Usually, a session key is required
  - Symmetric key for a particular session
- Can we authenticate and establish a shared symmetric key?
  - Key can be used for confidentiality
  - Key can be used for integrity
- In some cases, we may also require perfect forward secrecy (PFS)
  - Discussed later…

Authentication & Session Key



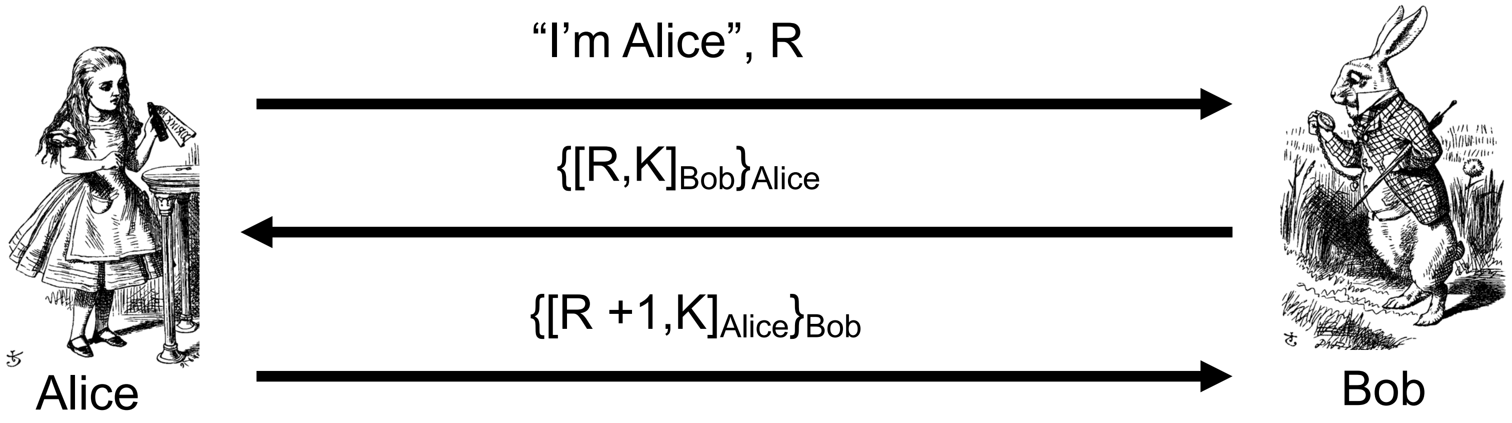
- Is this secure?
- OK for key, but no mutual authentication
- **Note** that K is acting as Bob's nonce

Public Key Authentication and Session Key



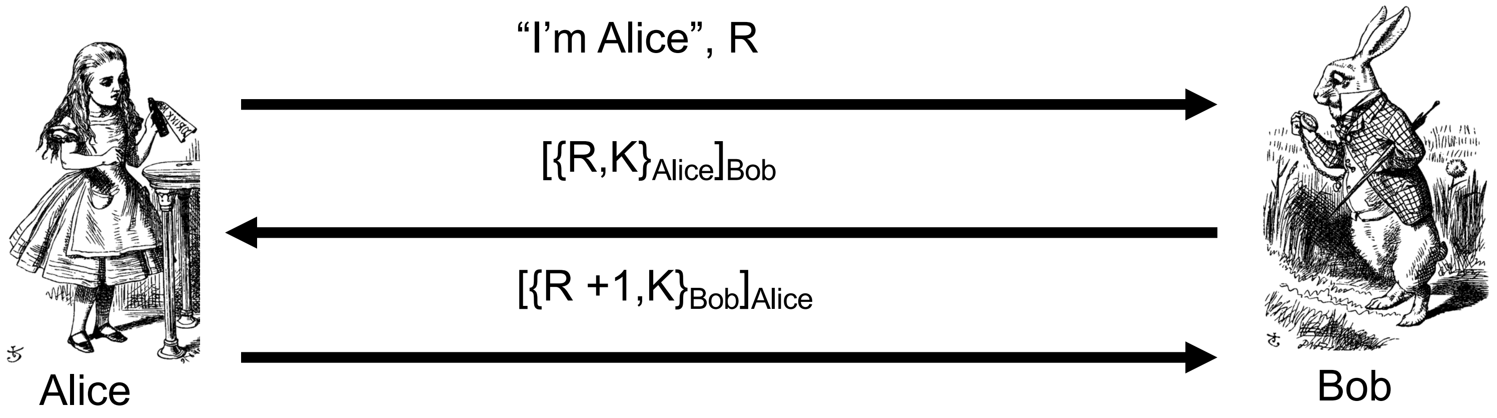
- Is this secure?
- Mutual authentication but key is not secret!

Public Key Authentication and Session Key



- Is this secure?
- Seems to be OK
- Mutual authentication and session key!

Public Key Authentication and Session Key



- Is this secure?
- Seems to be OK
  - Anyone can see $\{R,K\}_{Alice}$ and $\{R+1,K\}_{Bob}$

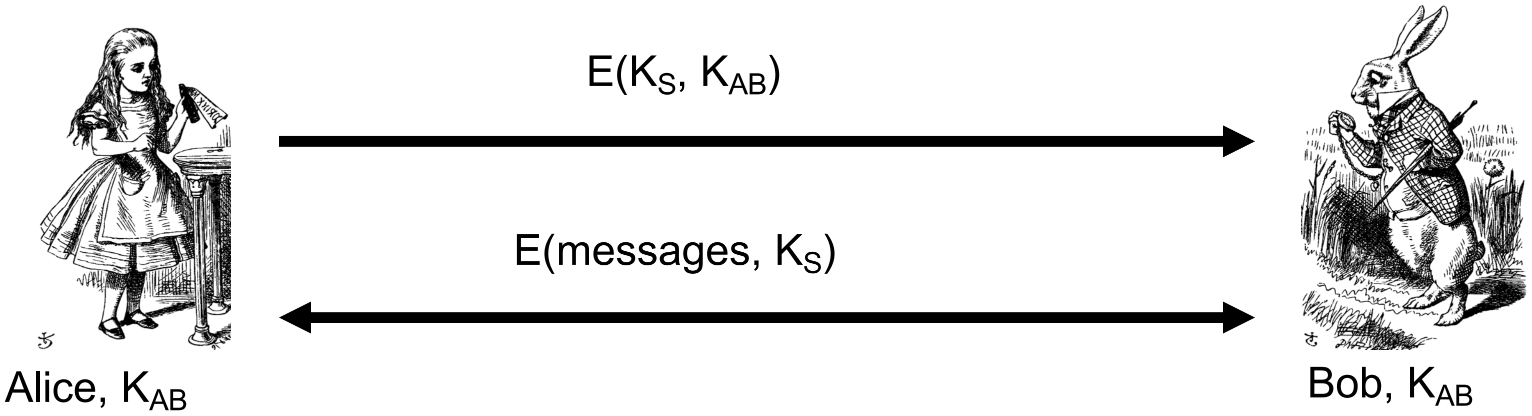
Perfect Forward Secrecy

- The concern…
  - Alice encrypts message with shared key K_{AB} and sends ciphertext to Bob
  - Trudy records ciphertext and later attacks Alice's (or Bob's) computer to find K_{AB}
  - Then Trudy decrypts recorded messages
- **Perfect forward secrecy (PFS):** Trudy cannot later decrypt recorded ciphertext
  - Even if Trudy gets key K_{AB} or other secret(s)
- Is PFS possible?

Perfect Forward Secrecy

- Suppose Alice and Bob share key K_{AB}
- For perfect forward secrecy, Alice and Bob cannot use K_{AB} to encrypt
- Instead they must use a **session key** K_S and forget it after it's used
- Problem: How can Alice and Bob agree on session key K_S and ensure PFS?

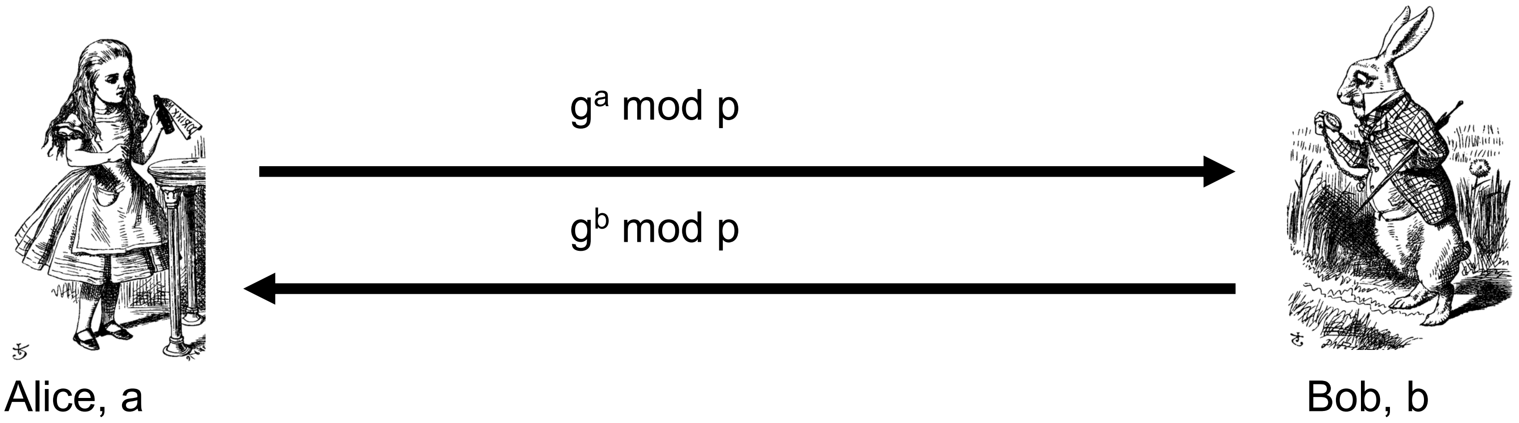
Naïve Session Key Protocol



- Trudy could also record $E(K_S,K_{AB})$
- If Trudy gets K_{AB}, she gets K_S

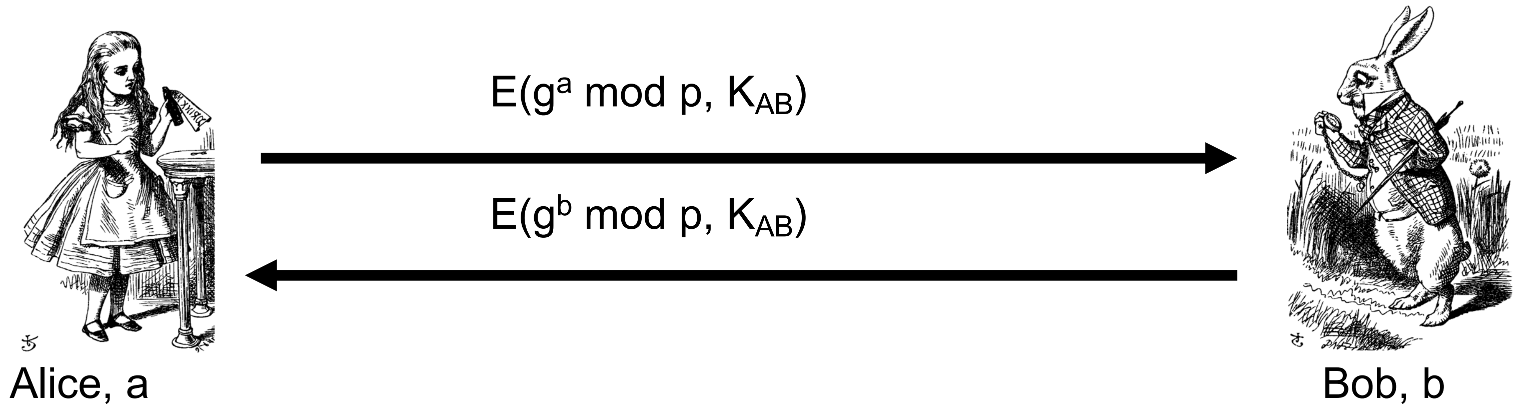
Perfect Forward Secrecy

- Can use **Diffie-Hellman** for PFS
- Recall Diffie-Hellman: public g and p



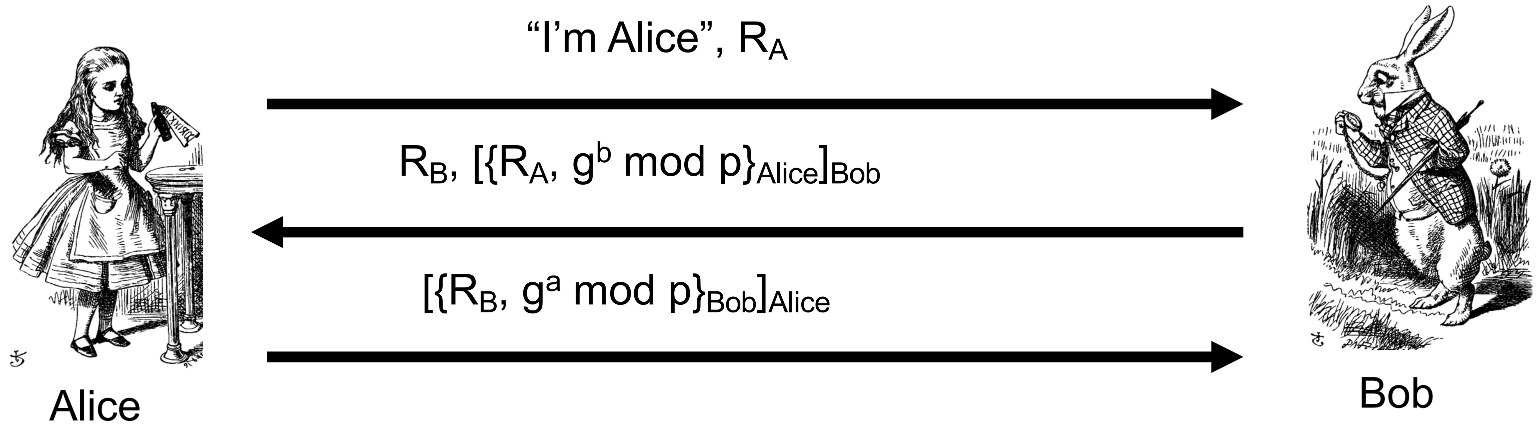
- But Diffie-Hellman is subject to MiM
- How to get PFS and prevent MiM?

Perfect Forward Secrecy



- Session key $K_S = g^{ab} \bmod p$
- Alice forgets a, Bob forgets b
- **Ephemeral Diffie-Hellman**
- Not even Alice and Bob can later recover K_S
- Other ways to do PFS?

Mutual Authentication, Session Key and PFS

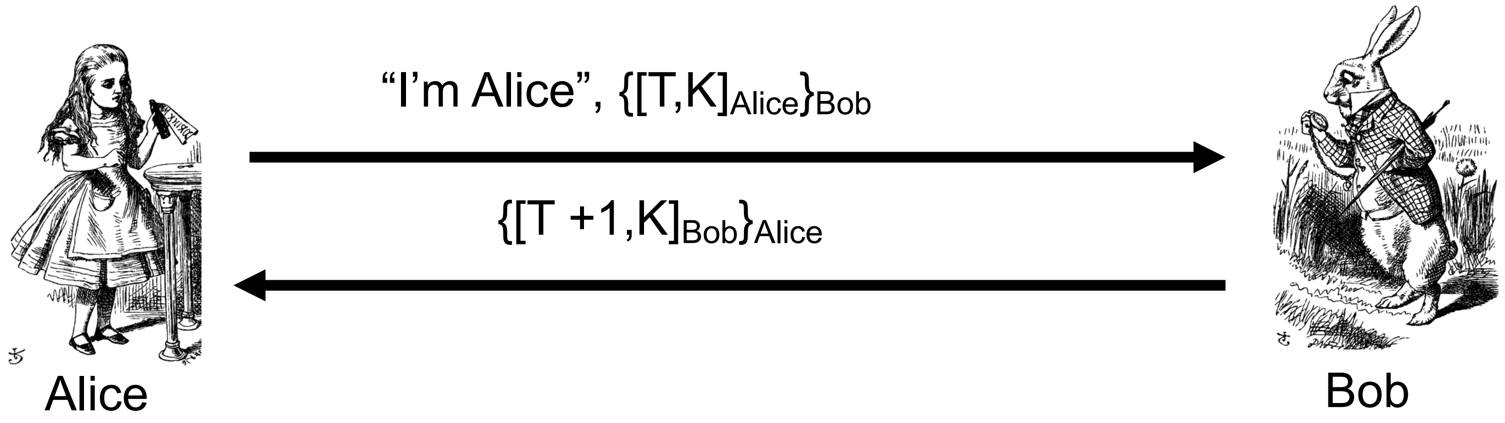

"I'm Alice", R_A
R_B, $[\{R_A, g^b \bmod p\}_{Alice}]_{Bob}$
$[\{R_B, g^a \bmod p\}_{Bob}]_{Alice}$
Alice
Bob

- Session key is $K = g^{ab} \bmod p$
- Alice forgets a and Bob forgets b
- If Trudy later gets Bob's and Alice's secrets, she cannot recover session key K

Timestamps

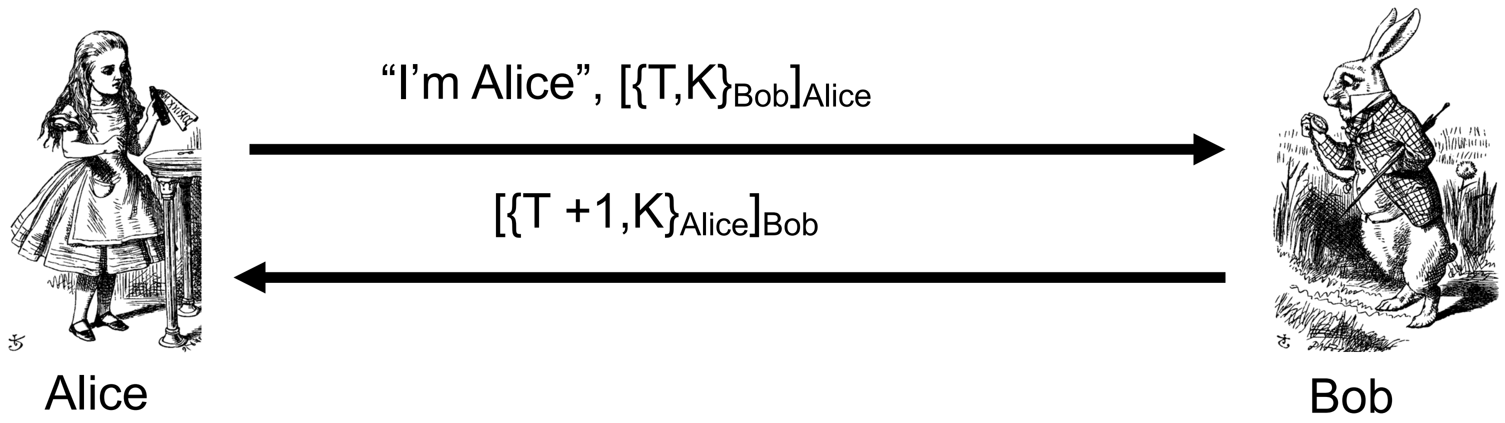
- A timestamp T is the current time
- Timestamps used in many security protocols (Kerberos, for example)
- Timestamps reduce number of messages
  - Like a nonce that both sides know in advance
- But, use of timestamps implies that time is a security-critical parameter
- Clocks never exactly the same, so must allow for **clock skew** — risk of replay
- How much clock skew is enough?

Public Key Authentication with Timestamp T


"I'm Alice", $\{[T,K]_{Alice}\}_{Bob}$
$\{[T+1,K]_{Bob}\}_{Alice}$
Alice
Bob

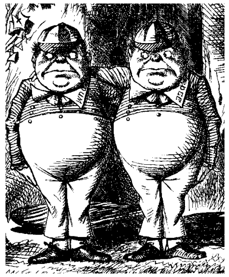
- Is this secure?
- Seems to be OK

Public Key Authentication with Timestamp T


"I'm Alice", $[\{T,K\}_{Bob}]_{Alice}$
$[\{T+1,K\}_{Alice}]_{Bob}$
Alice
Bob

- Is this secure?
- Trudy can use Alice's public key to find $\{T,K\}_{Bob}$ and then…

Public Key Authentication with Timestamp T


Trudy

“I’m Trudy”, $[\{T,K\}_{Bob}]_{Trudy}$ →

← $[\{T+1,K\}_{Trudy}]_{Bob}$

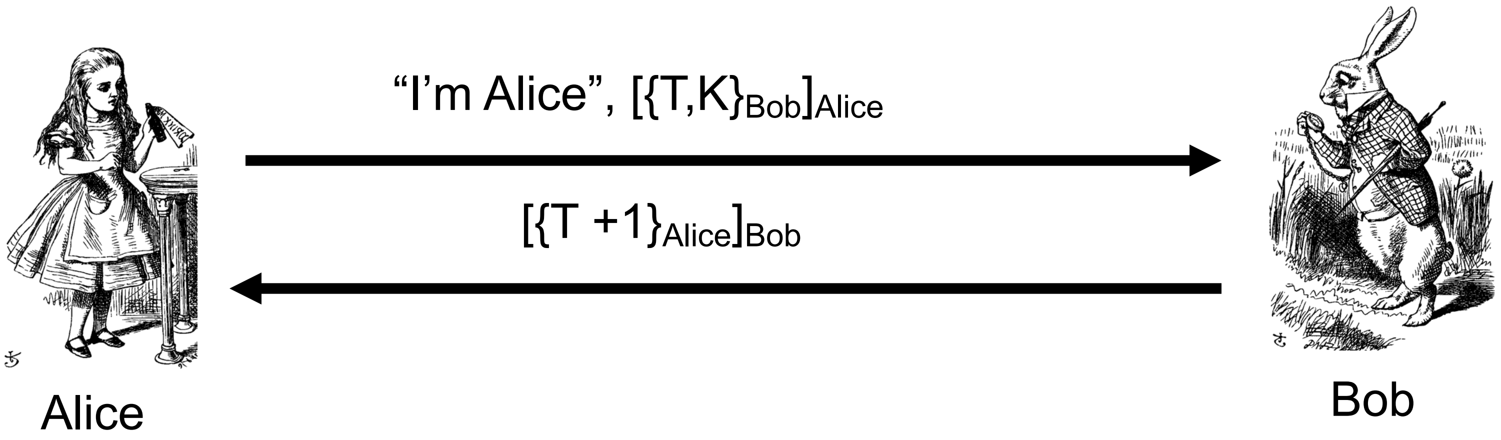
Bob

- Trudy obtains Alice-Bob session key K
- Note: Trudy must act within clock skew

Public Key Authentication

- Sign and encrypt with nonce…
  - **Secure**
- Encrypt and sign with nonce…
  - **Secure**
- Sign and encrypt with timestamp…
  - **Secure**
- Encrypt and sign with timestamp…
  - **Insecure**
- Protocols can be subtle!

Public Key Authentication with Timestamp T


"I'm Alice", $[\{T,K\}_{Bob}]_{Alice}$
$[\{T+1\}_{Alice}]_{Bob}$
Alice
Bob

- Is this "encrypt and sign" secure?
- Yes, seems to be
- Does "sign and encrypt" also work here?

Kerberos

- In Greek mythology, Kerberos is 3-headed dog that guards entrance to Hades
  - "Wouldn't it make more sense to guard the exit?"
- In security, Kerberos is an authentication system based on symmetric key crypto
  - Originated at MIT
  - Based on work by Needham and Schroeder
  - Relies on a **trusted third party (TTP)**

Motivation for Kerberos

- Authentication using public keys
  - N users $\Rightarrow$ N key pairs
- Authentication using symmetric keys
  - N users requires about N^2 keys
- Symmetric key case **does not scale!**
- Kerberos based on symmetric keys but only requires N keys for N users
  - But must rely on TTP
  - Advantage is that no PKI is required

Kerberos KDC

- Kerberos **Key Distribution Center** or **KDC**
  - Acts as a TTP
  - TTP must not be compromised!
  - KDC shares symmetric key K_A with Alice, key K_B with Bob, key K_C with Carol, etc.
  - Master key K_{KDC} known only to KDC
  - KDC enables authentication and session keys
  - Keys for confidentiality and integrity
  - In practice, the crypto algorithm used is DES

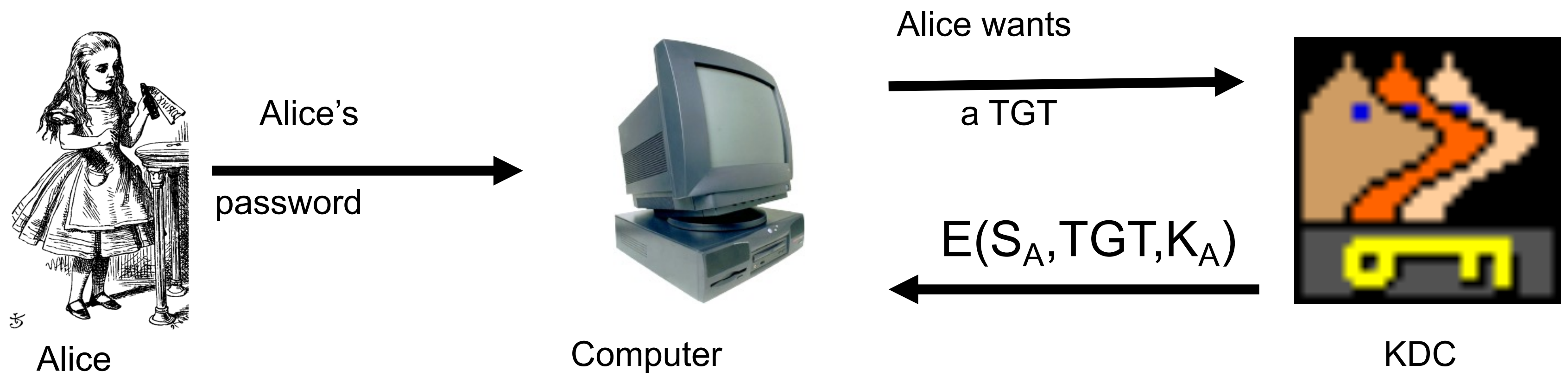
Kerberos Tickets

- KDC issues a **ticket** containing info needed to access a network resource
- KDC also issues **ticket-granting tickets** or **TGTs** that are used to obtain tickets
- Each TGT contains
  - Session key
  - User’s ID
  - Expiration time
- Every TGT is encrypted with K_{KDC}
  - TGT can only be read by the KDC

Kerberized Login

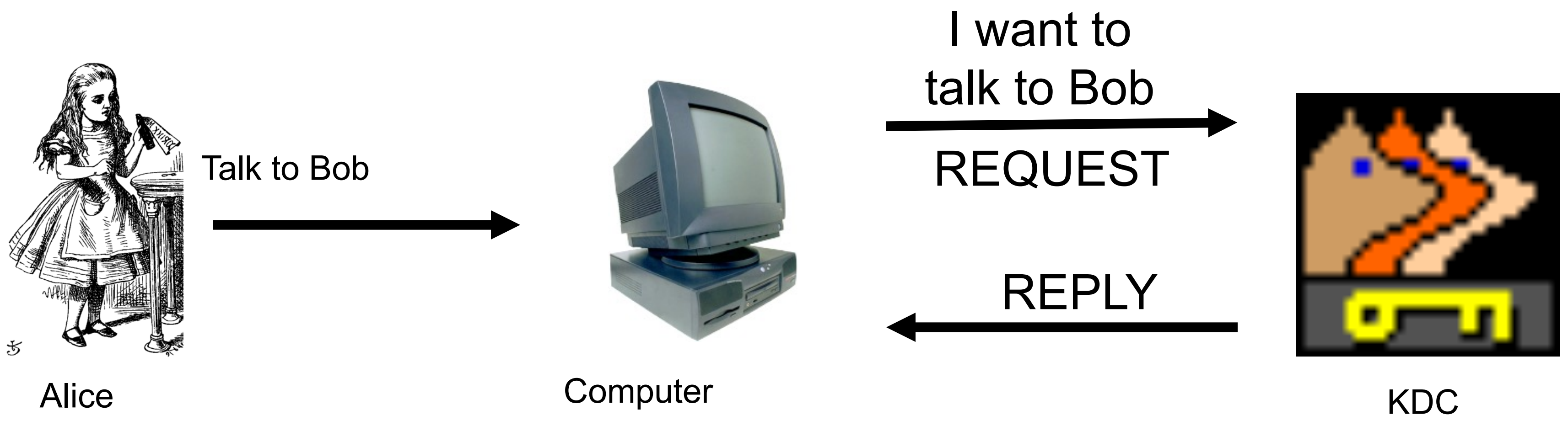
- Alice enters her password
- Alice's workstation
  - Derives K_A from Alice's password
  - Uses K_A to get TGT for Alice from the KDC
- Alice can then use her TGT (credentials) to securely access network resources
- **Plus:** Security is transparent to Alice
- **Minus:** KDC must be secure — it's trusted!

Kerberized Login



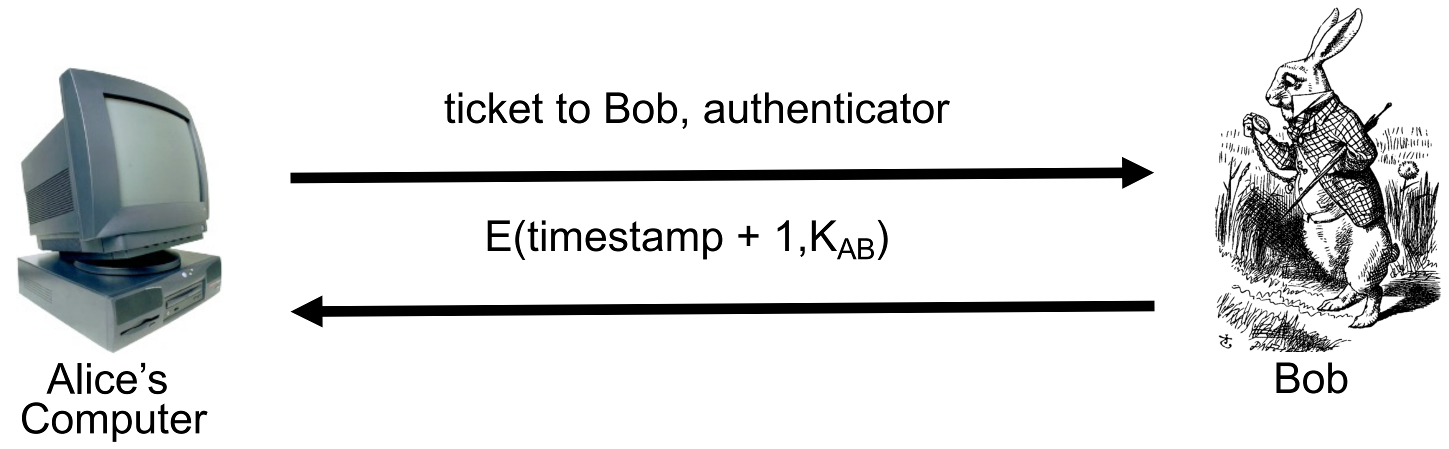
- Key K_A derived from Alice's password
- KDC creates session key S_A
- Workstation decrypts S_A, TGT, forgets K_A
- TGT = E("Alice", S_A, K_{KDC})

Alice Requests Ticket to Bob



- REQUEST = (TGT, authenticator) where
  authenticator = E(timestamp,S_A)
- REPLY = E("Bob", K_{AB}, ticket to Bob, S_A)
- ticket to Bob = E("Alice", K_{AB}, K_B)
- KDC gets S_A from TGT to verify timestamp

Alice Uses Ticket to Bob



- ticket to Bob = E("Alice",K_{AB}, K_B)
- authenticator = E(timestamp, K_{AB})
- Bob decrypts "ticket to Bob" to get K_{AB} which he then uses to verify timestamp

Kerberos

- Session key S_A used for authentication
- Can also be used for confidentiality/integrity
- Timestamps used for mutual authentication
- Recall that timestamps reduce number of messages
  - Acts like a nonce that is known to both sides
  - Note: **time** is a security-critical parameter!

Kerberos Questions

- When Alice logs in, KDC sends E(S_A,TGT,K_A) where TGT = E("Alice",S_A,K_{KDC})

  Q: Why is TGT encrypted with K_A?

  A: Extra work and no added security!
- In Alice's Kerberized login to Bob, why can Alice remain anonymous?
- Why is "ticket to Bob" sent to Alice?
- Where is replay prevention in Kerberos?

Kerberos Alternatives

- Could have Alice’s workstation remember password and use that for authentication
  - Then no KDC required
  - But hard to protect password on workstation
  - Scaling problem
- Could have KDC remember session key instead of putting it in a TGT
  - Then no need for TGTs
  - But **stateless** KDC is big feature of Kerberos

Kerberos Keys

- In Kerberos, K_A = h(Alice's password)
- Could instead generate random K_A and
  - Compute K_h = h(Alice's password)
  - And workstation stores $E(K_A, K_h)$
- Then K_A need not change (on workstation or KDC) when Alice changes her password
- But $E(K_A, K_h)$ subject to password guessing
- This alternative approach is often used in applications (but not in Kerberos)

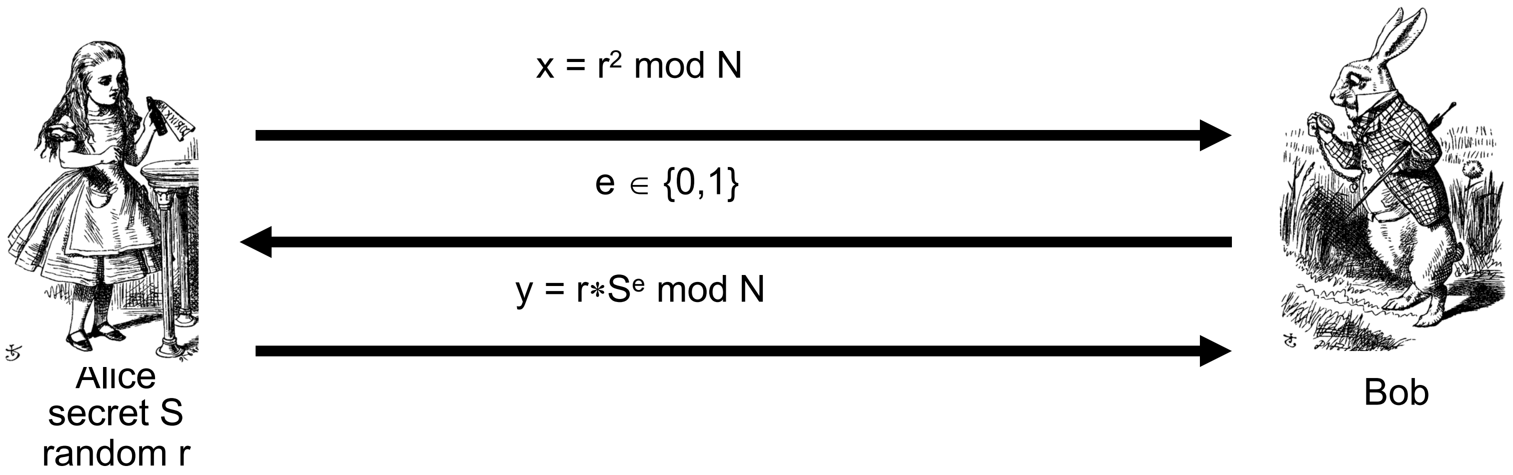
Zero Knowledge Proof (ZKP)

- Alice wants to prove that she knows a secret without revealing **any** info about it
- Bob must verify that Alice knows secret
  - Even though he gains no info about the secret
- Process is probabilistic
  - Bob can verify that Alice knows the secret to an arbitrarily high probability
- An "interactive proof system"

Fiat-Shamir Protocol

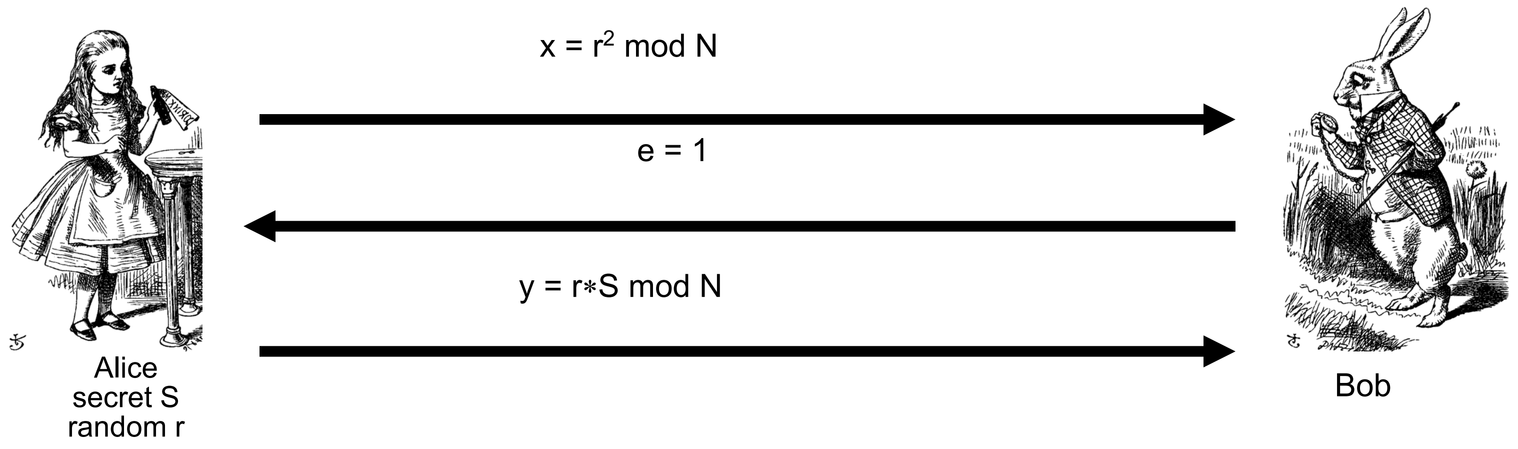
- Finding square roots modulo N is difficult (like factoring)
- Suppose N = pq, where p and q prime
- Alice has a **secret** S
- N and $v = S^2 \bmod N$ are public, S is secret.
- Alice must convince Bob that she knows S without revealing any information about S

Fiat-Shamir



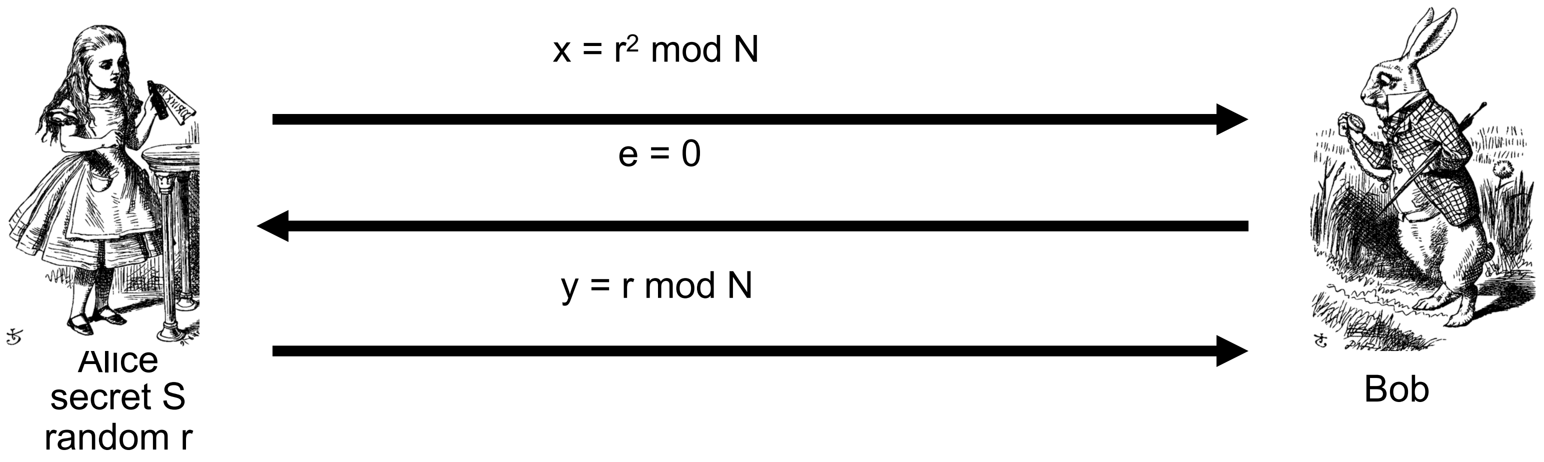
- **Public:** Modulus N and $v = S^2 \bmod N$
- Alice selects random r
- Bob chooses $e \in \{0,1\}$
- Bob verifies that $y^2 = r^2*S^{2e} = r^2*(S^2)^e = x*v^e \bmod N$

Fiat-Shamir: e = 1



- **Public:** Modulus N and $v = S^2 \bmod N$
- Alice selects random r
- Suppose Bob chooses e =1
- Bob must verify that $y^2 = x{*}v \bmod N$
- Alice must know S in this case

Fiat-Shamir: e = 0



- **Public:** Modulus N and $v = S^2 \bmod N$
- Alice selects random r
- Suppose Bob chooses $e = 0$
- Bob must verify that $y^2 = x \bmod N$
- Alice does **not** need to know S in this case!

Fiat-Shamir

- **Public:** modulus N and $v = S^2 \bmod N$
- **Secret:** Alice knows S
- Alice selects random r and **commits** to r by sending $x = r^2 \bmod N$ to Bob
- Bob sends **challenge** $e \in \{0,1\}$ to Alice
- Alice **responds** with $y = r*S^e \bmod N$
- Bob checks that $y^2 = x*v^e \bmod N$
- Does this prove response is from Alice?

Does Fiat-Shamir Work?

- The math works since
  - Public: $v = S^2$
  - Alice to Bob: $x = r^2$ and $y = r*S^e$
  - Bob verifies $y^2 = x*v^e \bmod N$
- Can Trudy convince Bob she is Alice?
  - If Trudy expects $e = 0$, she can send $x = r^2$ in msg 1 and $y = r$ in msg 3 (i.e., follow protocol)
  - If Trudy expects Bob to send $e = 1$, she can send $x = r^2*v^{-1}$ in msg 1 and $y = r$ in msg 3
- If Bob chooses $e \in \{0,1\}$ at random, Trudy can only fool Bob with probability 1/2

Fiat-Shamir Facts

- Trudy can fool Bob with prob 1/2, but…
- …after n iterations, the probability that Trudy can fool Bob is only $1/2^n$
- Bob's $e \in \{0,1\}$ must be unpredictable
- Alice must use new r each iteration or else
  - If e = 0, Alice sends r in message 3
  - If e = 1, Alice sends $r*S$ in message 3
  - Anyone can find S given **both** r and $r*S$

Fiat-Shamir Zero Knowledge?

- Zero knowledge means that Bob learns **nothing** about the secret S
  - **Public:** $v = S^2 \bmod N$
  - Bob sees $r^2 \bmod N$ in message 1
  - Bob sees $r*S \bmod N$ in message 3 (if $e = 1$)
  - If Bob can find r from $r^2 \bmod N$, he gets S
  - But that requires modular square root
  - If Bob can find modular square roots, he can get S from **public** v
- The protocol does not “help” Bob to find S

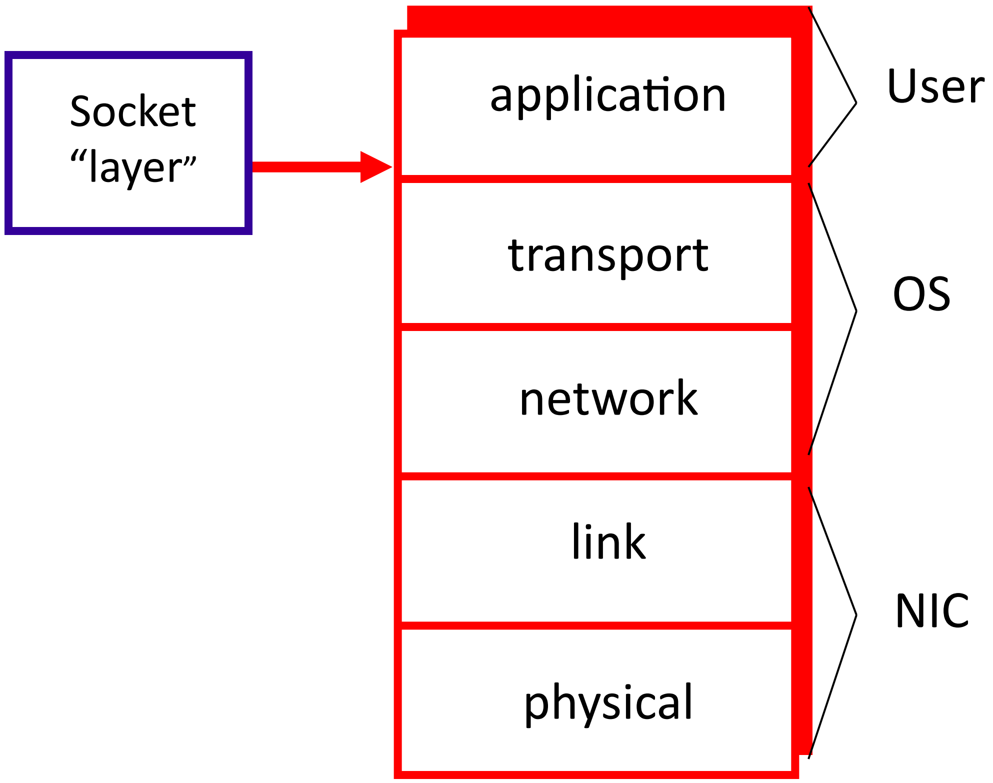
ZKP in the Real World

- Public key certificates identify users
  - No anonymity if certificates transmitted
- ZKP offers a way to authenticate without revealing identities
- ZKP supported in Microsoft’s Next Generation Secure Computing Base (NGSCB)
  - ZKP used to authenticate software “without revealing machine identifying data”

Secure Socket Layer

Socket layer

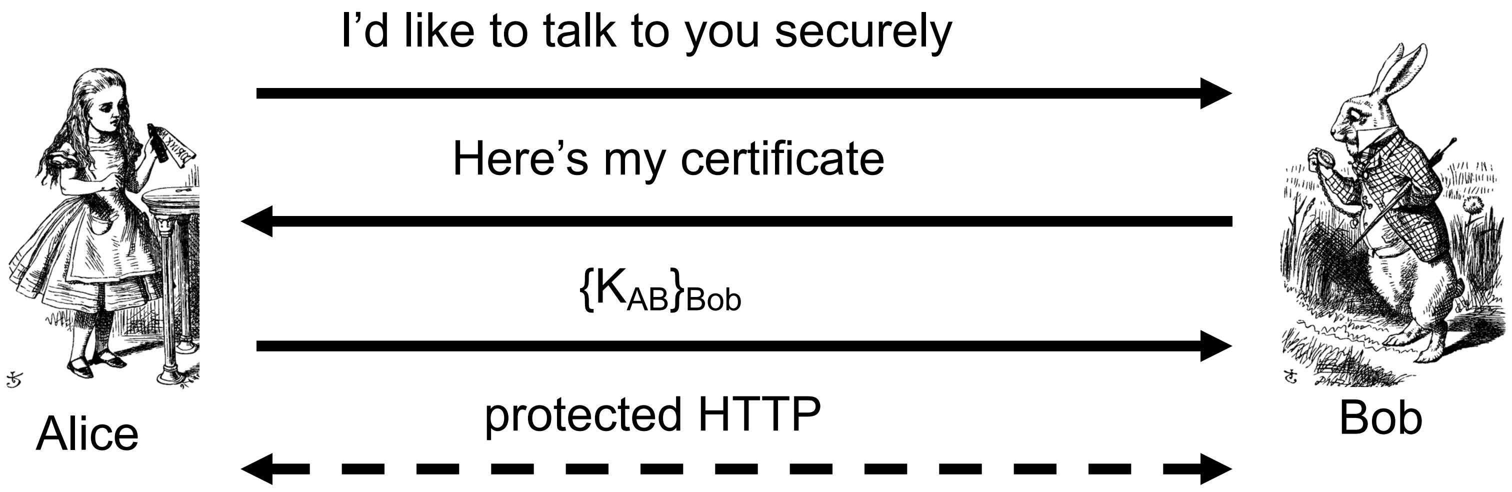
- "Socket layer" lives between application and transport layers
- SSL usually lies between HTTP and TCP



What is SSL?

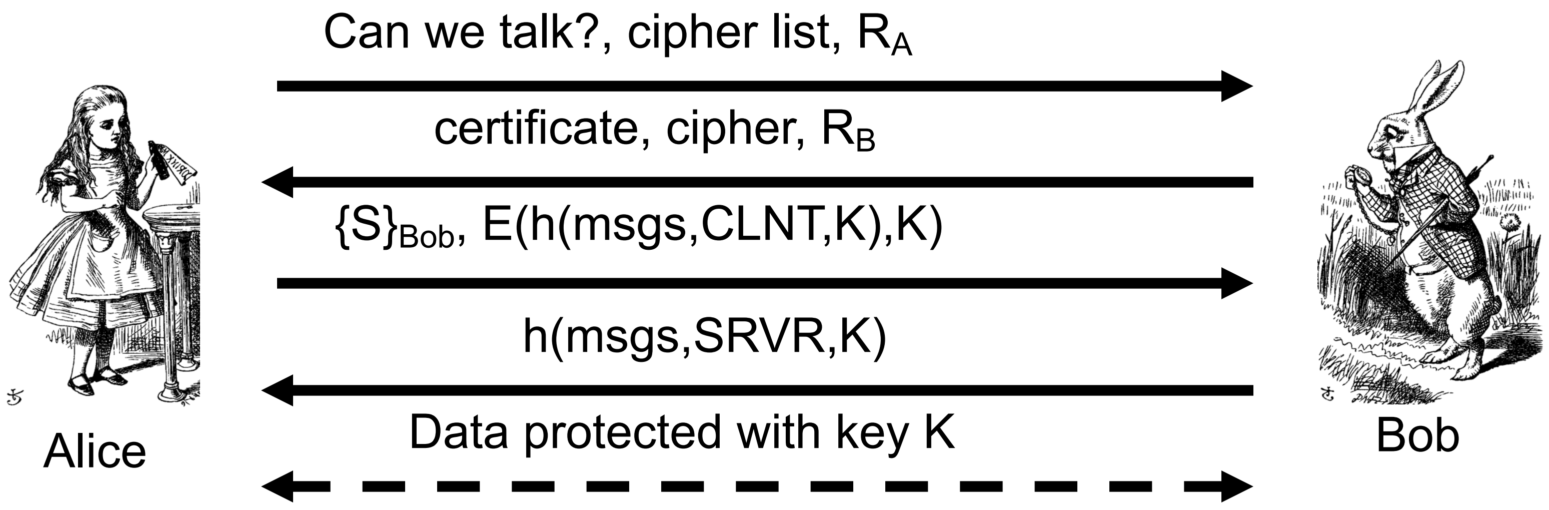
- SSL is the protocol used for most secure transactions over the Internet
- For example, if you want to buy a book at amazon.com…
  - You want to be sure you are dealing with Amazon (**authentication**)
  - Your credit card information must be protected in transit (**confidentiality** and/or **integrity**)
  - 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


Can we talk?, cipher list, R_A
certificate, cipher, R_B
$\{S\}_{Bob}$, E(h(msgs,CLNT,K),K)
h(msgs,SRVR,K)
Data protected with key K
Alice
Bob

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

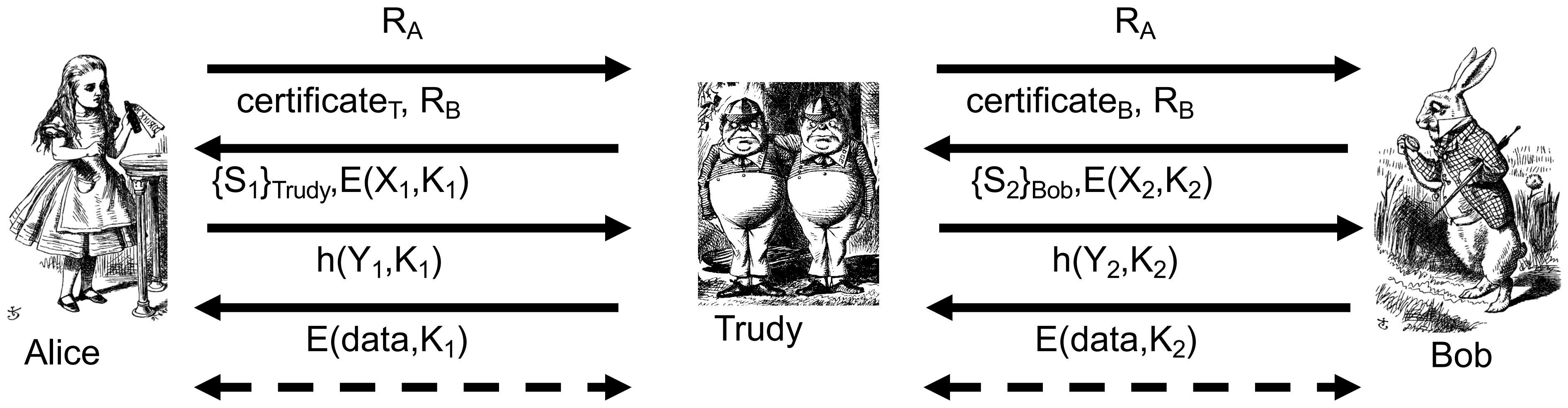
SSL Keys

- 6 "keys" derived from K = 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(msgs,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

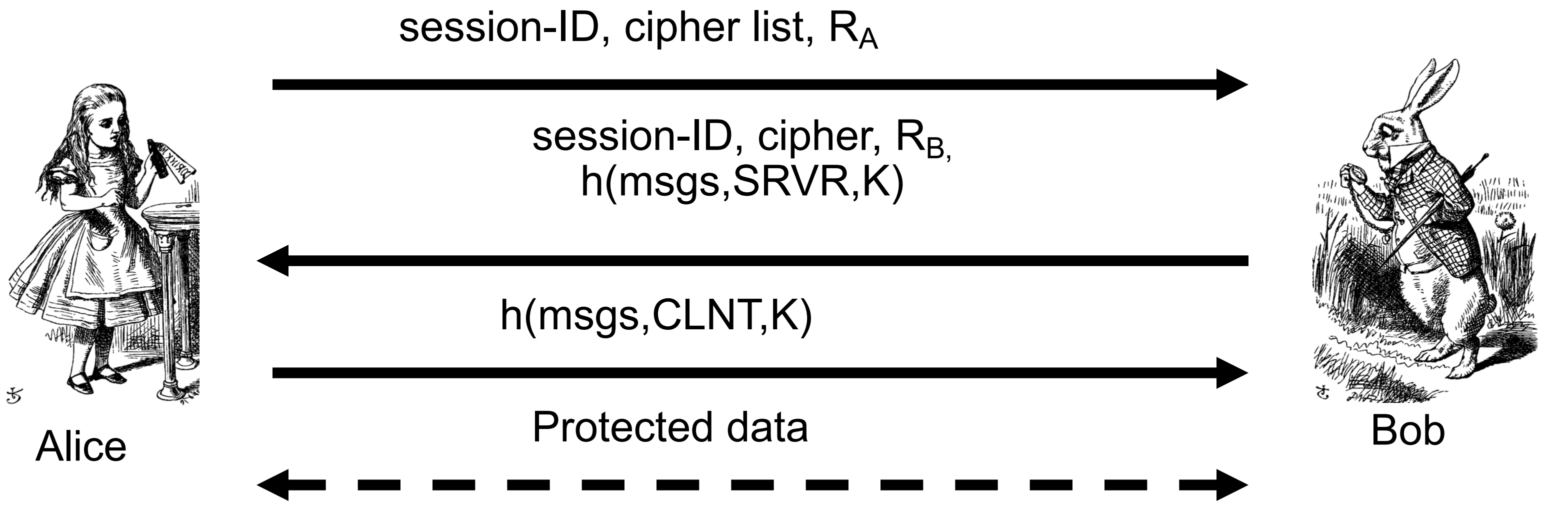


- **Q:** What prevents this MiM attack?
- **A:** Bob's certificate must be signed by a certificate authority (such as Verisign)
- What does Web browser do if sig. not valid?
- What does user do if signature is not valid?

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

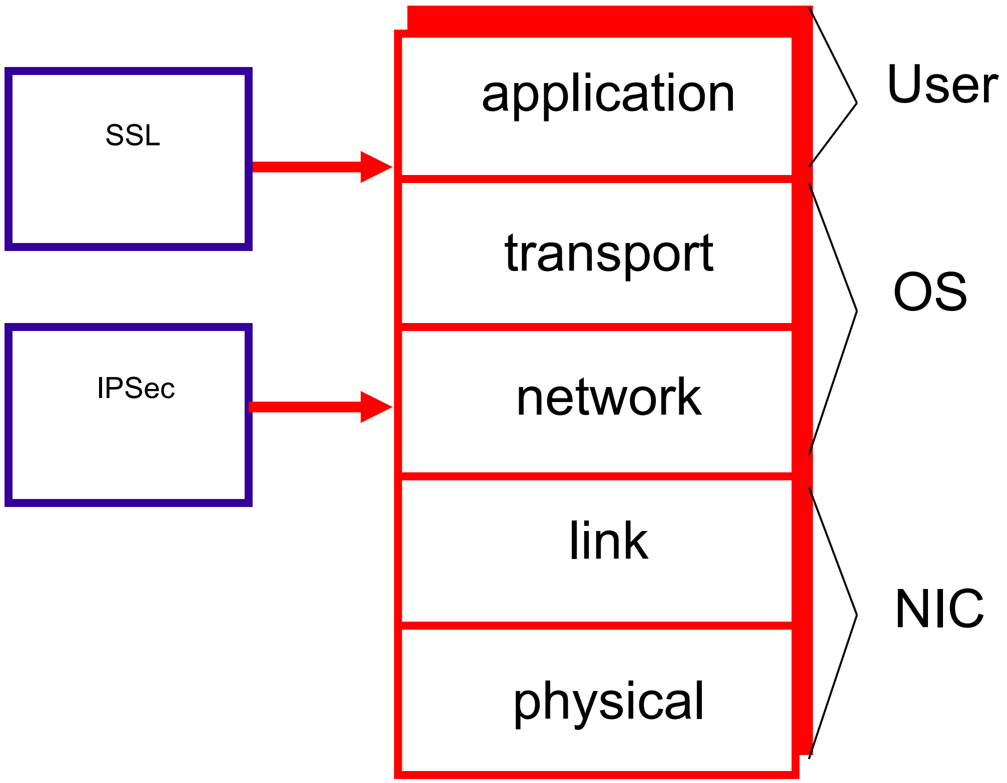
SSL vs IPSec

- 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!**

IPSec

IPSec and SSL

- IPSec lives at the network layer
- IPSec is transparent to applications



IPSec and Complexity

- IPSec is a complex protocol
- Over-engineered
  - Lots of generally useless extra features
- Flawed
  - Some serious security flaws
- Interoperability is serious challenge
  - Defeats the purpose of having a standard!
- Complex
- Did I mention, it’s complex?

IKE and ESP/AH

- Two parts to IPSec
- **IKE:** Internet Key Exchange
  - Mutual authentication
  - Establish shared symmetric key
  - Two "phases" — like SSL session/connection
- **ESP/AH**
  - ESP: Encapsulating Security Payload — for encryption and/or integrity of IP packets
  - AH: Authentication Header — integrity only

IKE

IKE

- IKE has 2 phases
  - Phase 1 — IKE security association (SA)
  - Phase 2 — AH/ESP security association
- Phase 1 is comparable to SSL session
- Phase 2 is comparable to SSL connection
- Not an obvious need for two phases in IKE
- If multiple Phase 2's do not occur, then it is **more** expensive to have two phases!

IKE Phase 1

- Four different "key" options
  - Public key encryption (original version)
  - Public key encryption (improved version)
  - Public key signature
  - Symmetric key
- For each of these, two different "modes"
  - Main mode
  - Aggressive mode
- **There are 8 versions of IKE Phase 1!**
- Evidence that IPSec is over-engineered?

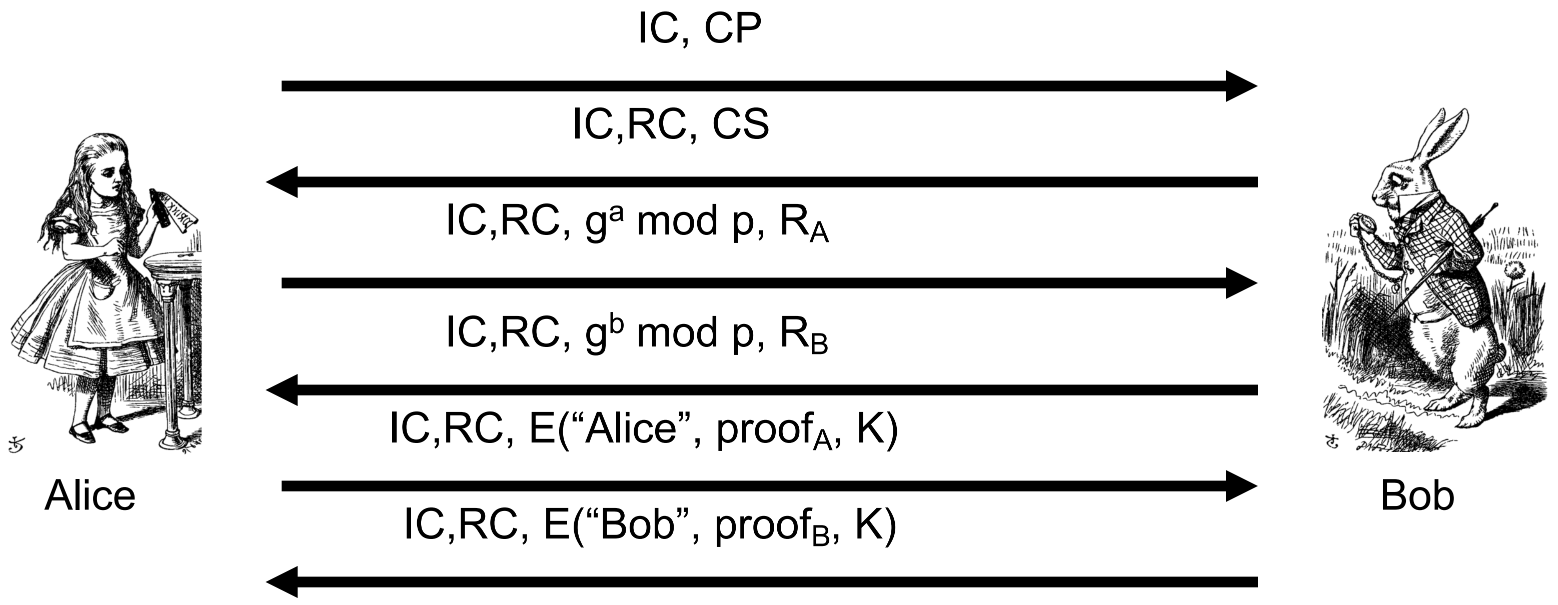
IKE Phase 1

- We'll discuss 6 of 8 phase 1 variants
  - Public key signatures (main and aggressive modes)
  - Symmetric key (main and aggressive modes)
  - Public key encryption (main and aggressive)
- Why public key encryption and public key signatures?
  - Always know your own private key
  - **May not** (initially) know other side's public key

IKE Phase 1

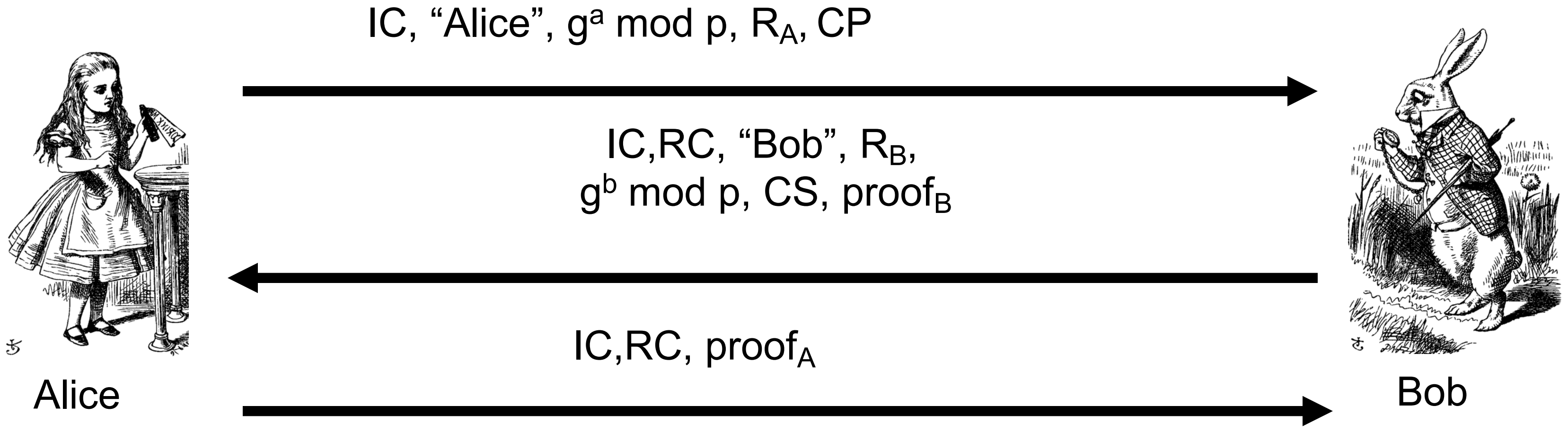
- Uses ephemeral Diffie-Hellman to establish session key
  - Achieves perfect forward secrecy (PFS)
- Let a be Alice's Diffie-Hellman exponent
- Let b be Bob's Diffie-Hellman exponent
- Let g be generator and p prime
- Recall p and g are public

IKE Phase 1: Digital Signature (Main Mode)


IC, CP
IC,RC, CS
IC,RC, g^a mod p, R_A
IC,RC, g^b mod p, R_B
IC,RC, E("Alice", proof_A, K)
IC,RC, E("Bob", proof_B, K)
Alice
Bob

- CP = crypto proposed, CS = crypto selected
- IC = initiator "cookie", RC = responder "cookie"
- K = $h(IC,RC,g^{ab} \bmod p,R_A,R_B)$
- SKEYID = $h(R_A, R_B, g^{ab} \bmod p)$
- proof_A = $[h(\text{SKEYID},g^a,g^b,IC,RC,CP,\text{"Alice"})]_{\text{Alice}}$

IKE Phase 1: Public Key Signature (Aggressive Mode)

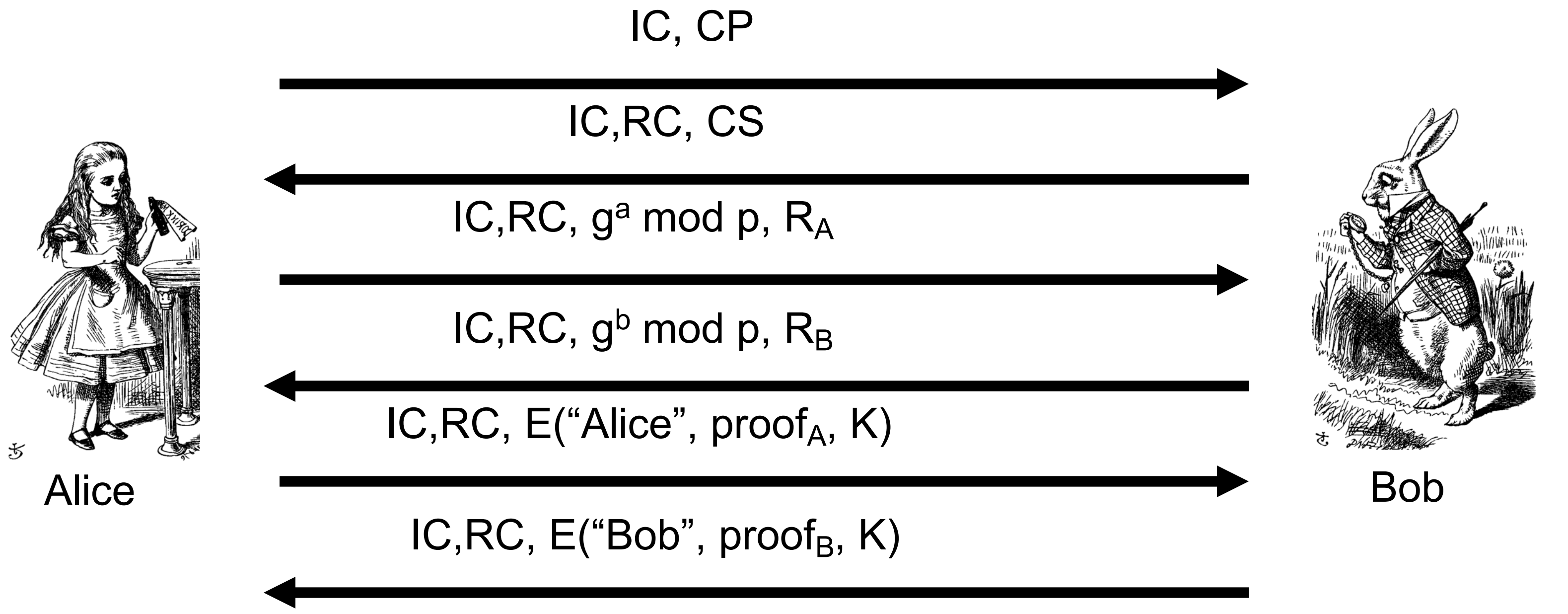

IC, “Alice”, g^a mod p, R_A, CP
IC,RC, “Bob”, R_B, g^b mod p, CS, proof_B
IC,RC, proof_A
Alice
Bob

- Main difference from main mode
  - Not trying to protect identities
  - Cannot negotiate g or p

Main vs Aggressive Modes

- Main mode **MUST** be implemented
- Aggressive mode **SHOULD** be implemented
  - In other words, if aggressive mode is not implemented, “you should feel guilty about it”
- Might create interoperability issues
- For public key signature authentication
  - Passive attacker knows identities of Alice and Bob in aggressive mode
  - Active attacker can determine Alice’s and Bob’s identity in main mode

IKE Phase 1: Symmetric Key (Main Mode)

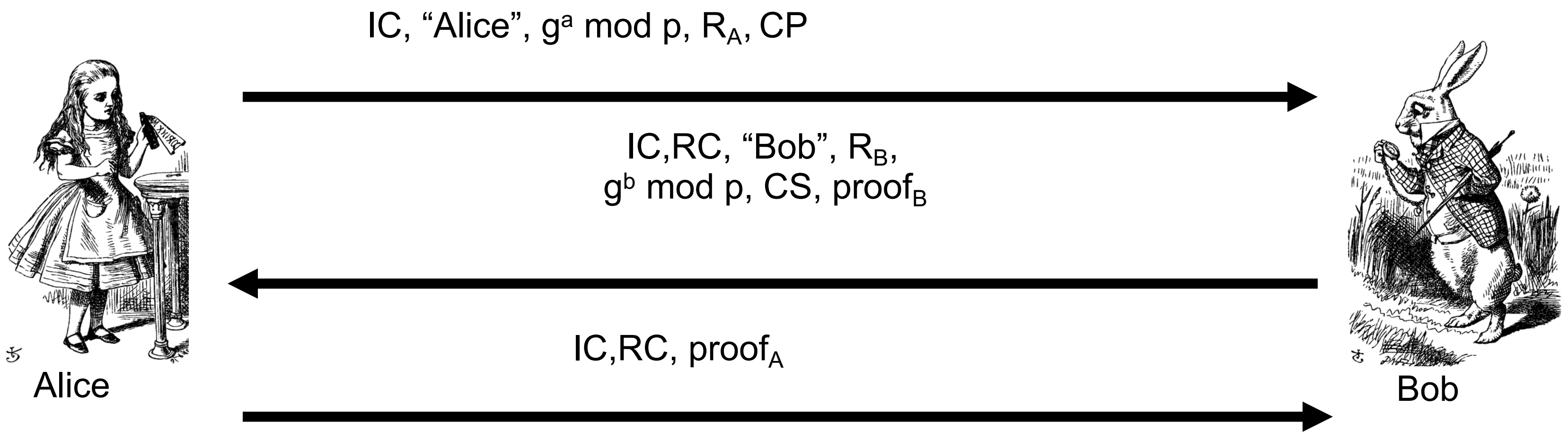

IC, CP
IC,RC, CS
IC,RC, g^a mod p, R_A
IC,RC, g^b mod p, R_B
IC,RC, E("Alice", proof_A, K)
IC,RC, E("Bob", proof_B, K)
Alice
Bob

- Same as signature mode except
  - K_{AB} = symmetric key shared in advance
  - K = h(IC,RC,g^{ab} mod p,R_A,R_B,K_{AB})
  - SKEYID = h(K, g^{ab} mod p)
  - proof_A = h(SKEYID,g^a,g^b,IC,RC,CP,"Alice")

Problems with Symmetric Key (Main Mode)

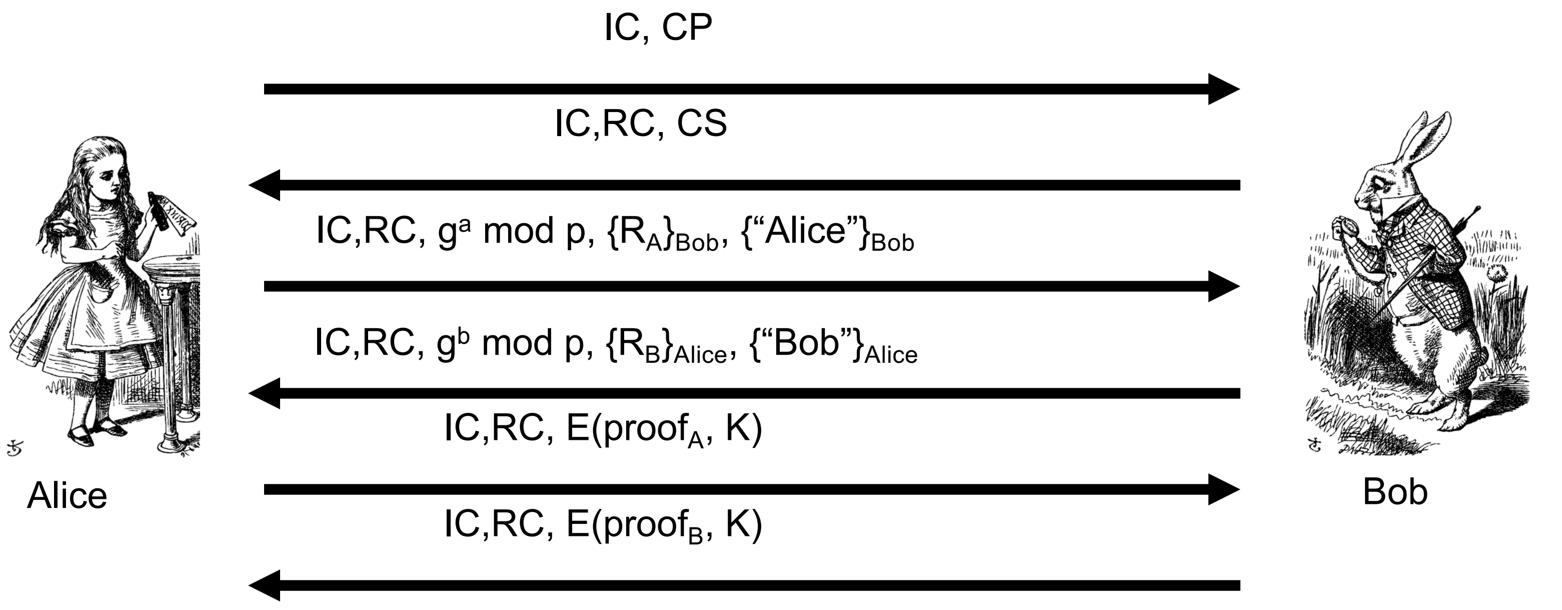
- Catch-22
  - Alice sends her ID in message 5
  - Alice’s ID encrypted with K
  - To find K Bob must know K_{AB}
  - To get K_{AB} Bob must know he’s talking to Alice!
- Result: **Alice’s ID must be IP address!**
- Useless mode for the “road warrior”
- Why go to all of the trouble of trying to hide identities in 6 message protocol?

IKE Phase 1: SymmetricKey (Aggressive Mode)


IC, “Alice”, g^a mod p, R_A, CP
IC,RC, “Bob”, R_B, g^b mod p, CS, proof_B
IC,RC, proof_A
Alice
Bob

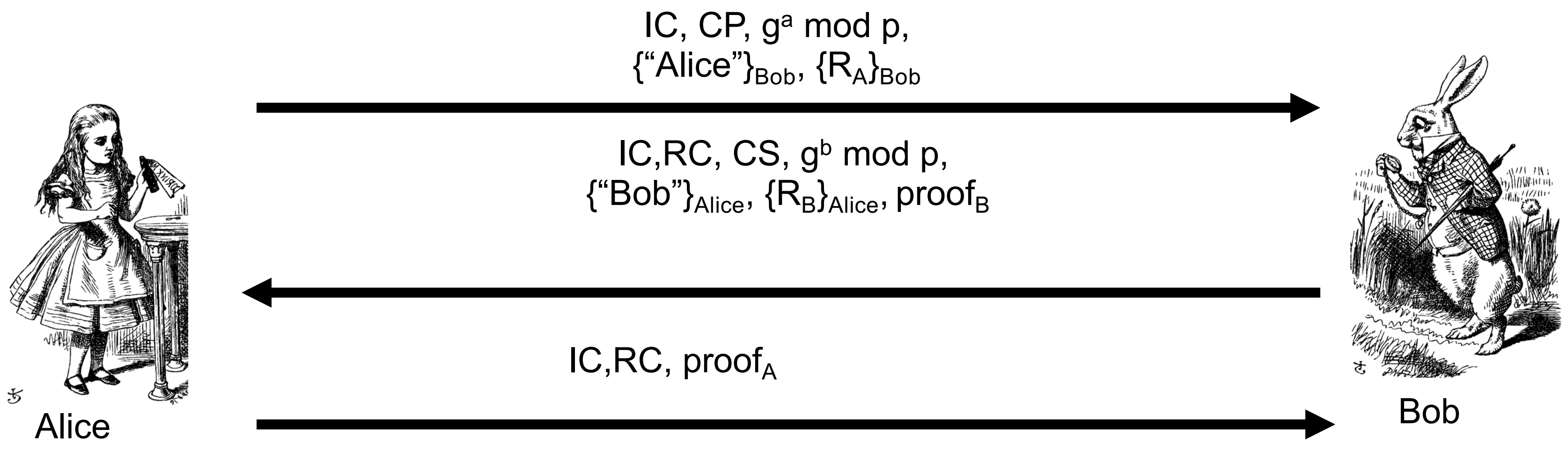
- Same format as digital signature aggressive mode
- Not trying to hide identities…
- As a result, does **not** have problems of main mode
- But does not (pretend to) hide identities

IKE Phase 1: Public Key Encryption (Main Mode)


IC, CP
IC,RC, CS
IC,RC, g^a mod p, $\{R_A\}_{Bob}$, $\{$“Alice”$\}_{Bob}$
IC,RC, g^b mod p, $\{R_B\}_{Alice}$, $\{$“Bob”$\}_{Alice}$
IC,RC, E(proof$_A$, K)
IC,RC, E(proof$_B$, K)
Alice
Bob

- CP = crypto proposed, CS = crypto selected
- IC = initiator “cookie”, RC = responder “cookie”
- K = h(IC,RC,g^{ab} mod p,R_A,R_B)
- SKEYID = h(R_A, R_B, g^{ab} mod p)
- proof$_A$ = h(SKEYID,g^a,g^b,IC,RC,CP,“Alice”)

IKE Phase 1: Public Key Encryption (Aggressive Mode)


IC, CP, $g^a \bmod p$,
{“Alice”}$_{Bob}$, $\{R_A\}_{Bob}$
IC,RC, CS, $g^b \bmod p$,
{“Bob”}$_{Alice}$, $\{R_B\}_{Alice}$, proof_B
IC,RC, proof_A
Alice
Bob

- K, proof_A, proof_B computed as in main mode
- Note that identities are hidden
  - The only aggressive mode to hide identities
  - Then why have main mode?

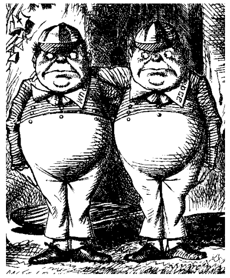
Public Key Encryption Issue?

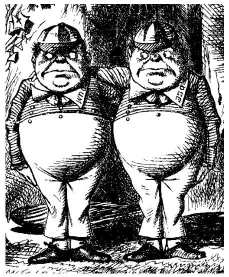
- Public key encryption, aggressive mode
- Suppose Trudy generates
  - Exponents a and b
  - Nonces R_A and R_B
- Trudy can compute “valid” keys and proofs: g^{ab} mod p, K, SKEYID, proof_A and proof_B
- Also true of main mode

Public Key Encryption Issue?

IC, CP, $g^a \bmod p$,
$\{\text{“Alice”}\}_{\text{Bob}}$, $\{R_A\}_{\text{Bob}}$

$\longrightarrow$


Trudy
as Alice


Trudy
as Bob

IC,RC, CS, $g^b \bmod p$,
$\{\text{“Bob”}\}_{\text{Alice}}$, $\{R_B\}_{\text{Alice}}$, proof_B

$\longleftarrow$

IC,RC, proof_A

$\longrightarrow$

- Trudy can create exchange that appears to be between Alice and Bob
- Appears valid to any observer, including Alice and Bob!

Plausible Deniability

- Trudy can create “conversation” that appears to be between Alice and Bob
- Appears valid, even to Alice and Bob!
- A security failure?
- In this mode of IPSec, it is a feature
  - **Plausible deniability:** Alice and Bob can deny that any conversation took place!
- In some cases it might be a security failure
  - If Alice makes a purchase from Bob, she could later repudiate it (unless she had signed)

IKE Phase 1 Cookies

- Cookies (or “anti-clogging tokens”) supposed to make denial of service more difficult
- No relation to Web cookies
- To reduce DoS, Bob wants to remain stateless as long as possible
- But Bob must remember CP from message 1 (required for proof of identity in message 6)
- Bob must keep state from 1st message on!
- These cookies offer little DoS protection!

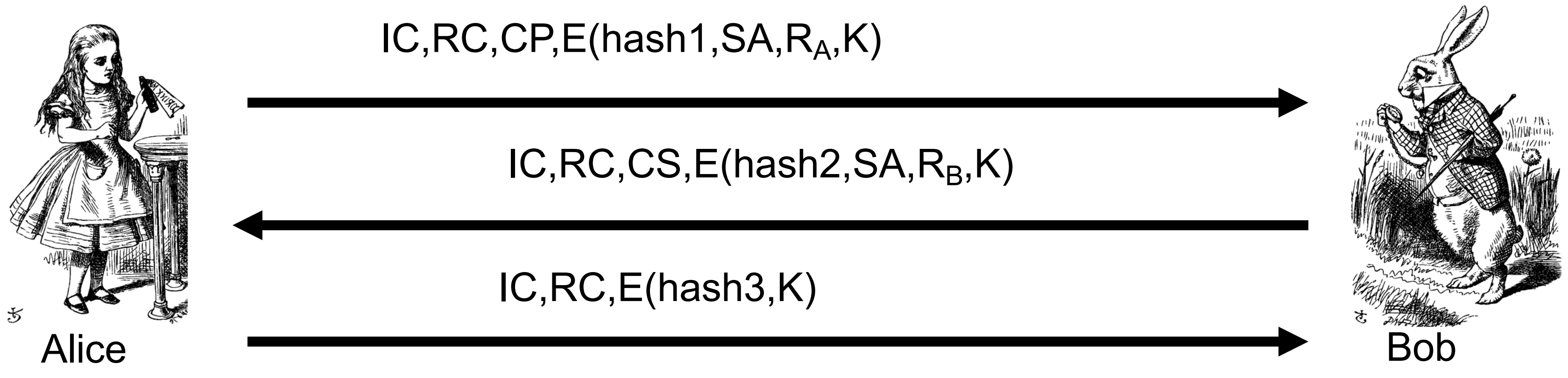
IKE Phase 1 Summary

- Result of IKE phase 1 is
  - Mutual authentication
  - Shared symmetric key
  - IKE **Security Association** **(SA)**
- But phase 1 is expensive (in public key and/or main mode cases)
- Developers of IKE thought it would be used for lots of things — not just IPSec
- Partly explains over-engineering…

IKE Phase 2

- Phase 1 establishes IKE SA
- Phase 2 establishes IPSec SA
- Comparison to SSL
  - SSL session is comparable to IKE Phase 1
  - SSL connections are like IKE Phase 2
- IKE **could** be used for lots of things
- But in practice, it's **not!**

IKE Phase 2



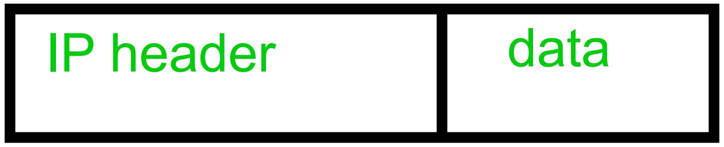
- Key K, IC, RC and SA known from Phase 1
- Proposal CP includes ESP and/or AH
- Hashes 1,2,3 depend on SKEYID, SA, R_A and R_B
- Keys derived from KEYMAT = h(SKEYID,R_A,R_B,junk)
- Recall SKEYID depends on phase 1 key method
- Optional PFS (ephemeral Diffie-Hellman exchange)

IPSec

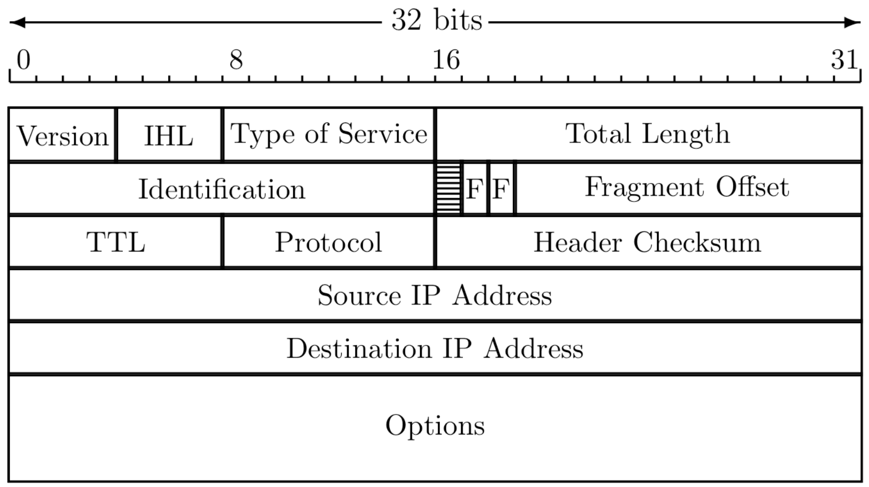
- After IKE Phase 1, we have an IKE SA
- After IKE Phase 2, we have an IPSec SA
- Both sides have a shared symmetric key
- Now what?
  - We want to protect IP datagrams
- But what is an IP datagram?
  - From the perspective of IPSec…

IP Review

- IP datagram is of the form

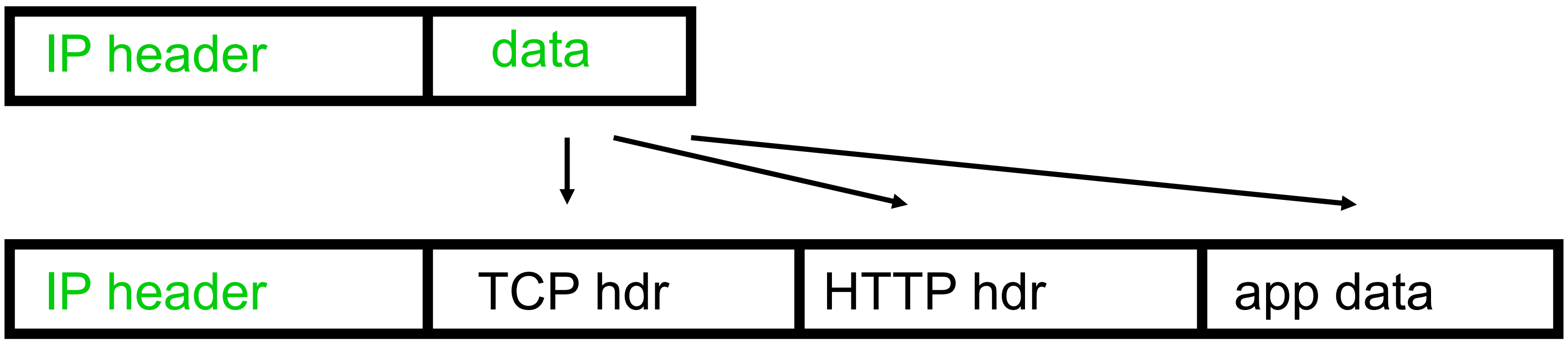


- Where IP header is



IP and TCP

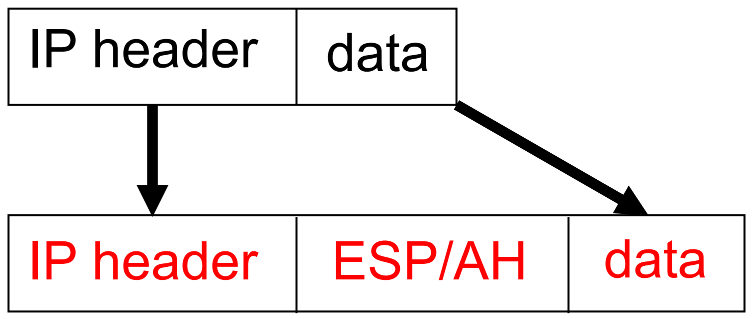
- Consider HTTP traffic (over TCP)
- IP encapsulates TCP
- TCP encapsulates HTTP



- IP data includes TCP header, etc.

IPSec Transport Mode

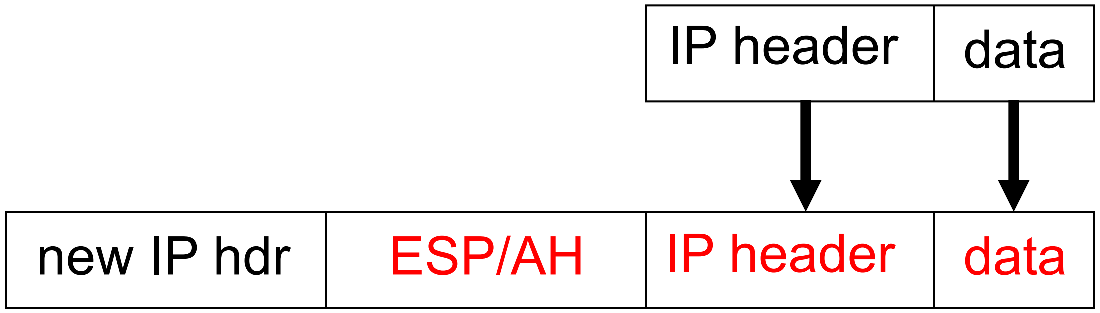
- IPSec **Transport Mode**



- Transport mode designed for host-to-host
- Transport mode is efficient
  - Adds minimal amount of extra header
- The original header remains
  - Passive attacker can see who is talking

IPSec Tunnel Mode

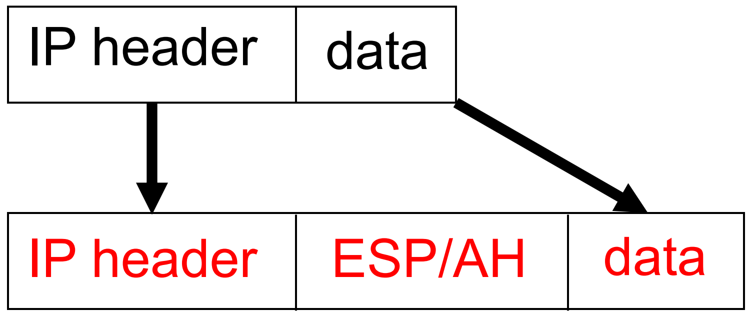
- IPSec Tunnel Mode



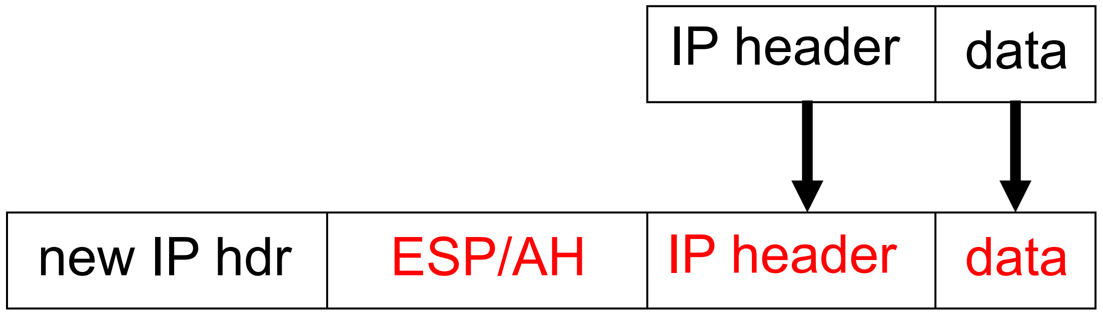
- Tunnel mode for firewall to firewall traffic
- Original IP packet encapsulated in IPSec
- Original IP header not visible to attacker
  - New header from firewall to firewall
  - Attacker does not know which hosts are talking

Comparison of IPSec Modes

- Transport Mode



- Tunnel Mode



- Transport Mode
  - Host-to-host
- Tunnel Mode
  - Firewall-to-firewall
- Transport mode not necessary
- Transport mode is more efficient

IPSec Security

- What kind of protection?
  - Confidentiality?
  - Integrity?
  - Both?
- What to protect?
  - Data?
  - Header?
  - Both?
- ESP/AH do some combinations of these

AH vs ESP

- AH
  - Authentication Header
  - **Integrity only** (no confidentiality)
  - Integrity-protect everything beyond IP header and some fields of header (why not all fields?)
- ESP
  - Encapsulating Security Payload
  - **Integrity and confidentiality**
  - Protects everything beyond IP header
  - Integrity only by using NULL encryption

ESP’s NULL Encryption

- According to RFC 2410
  - NULL encryption “is a block cipher the origins of which appear to be lost in antiquity”
  - “Despite rumors”, there is no evidence that NSA “suppressed publication of this algorithm”
  - Evidence suggests it was developed in Roman times as exportable version of Caesar’s cipher
  - Can make use of keys of varying length
  - No IV is required
  - Null(P,K) = P for any P and any key K
- Security people have a strange sense of humor!

Why Does AH Exist? (1)

- Cannot encrypt IP header
  - Routers must look at the IP header
  - IP addresses, TTL, etc.
  - IP header exists to route packets!
- AH protects **immutable fields** in IP header
  - Cannot integrity protect all header fields
  - TTL, for example, must change
- ESP does not protect IP header at all

Why Does AH Exist? (2)

- ESP encrypts everything beyond the IP header (if non-null encryption)
- If ESP encrypted, firewall cannot look at TCP header (e.g., port numbers)
- Why not use ESP with null encryption?
  - Firewall sees ESP header, but does not know whether null encryption is used
  - End systems know, but **not** firewalls
- Aside 1: Do firewalls reduce security?
- Aside 2: Is IPSec compatible with NAT?

Why Does AH Exist? (3)

- The real reason why AH exists
  - At one IETF meeting “someone from Microsoft gave an impassioned speech about how AH was useless…”
  - “…everyone in the room looked around and said `Hmm. He’s right, and we hate AH also, but if it annoys Microsoft let’s leave it in since we hate Microsoft more than we hate AH.”

Best Authentication Protocol?

- What is best depends on many factors…
- The sensitivity of the application
- The delay that is tolerable
- The cost (computation) that is tolerable
- What crypto is supported
  - Public key, symmetric key, hash functions
- Is mutual authentication required?
- Is a session key required?
- Is PFS a concern?
- Is anonymity a concern?, etc.

End