# CHAPTER 9
# SIMPLE AUTHENTICATION PROTOCOLS

BRAC UNIVERSITY
Inspiring Excellence

SIMPLE SECURITY PROTOCOL

AUTHENTICATION PROTOCOLS

ZERO KNOWLEDGE PROOFS

THE BEST AUTHENTICATION PROTOCOL?

# PROTOCOLS

- 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 protocols — 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 WEP, GSM and even IPSec
  - Implementation errors can occur
    - Such as IE implementation of SSL
- Not easy to get protocols right…

# IDEAL SECURITY PROTOCOL

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

- Difficult to satisfy all of these!

# SIMPLE SECURITY PROTOCOLS

# SECURE ENTRY TO NSA

1. Insert badge into reader
2. Enter PIN
3. Correct PIN?

   Yes? Enter

   No? Get shot by security guard

# ATM MACHINE PROTOCOL

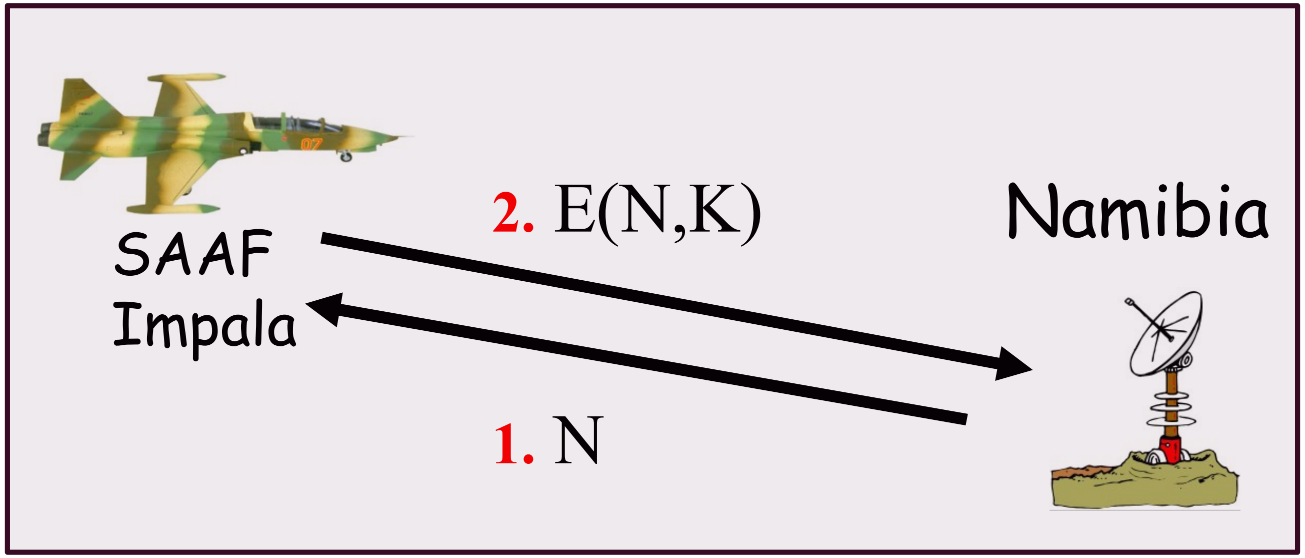
1. Insert ATM card
2. Enter PIN
3. Correct PIN?

    Yes? Conduct your transaction(s)

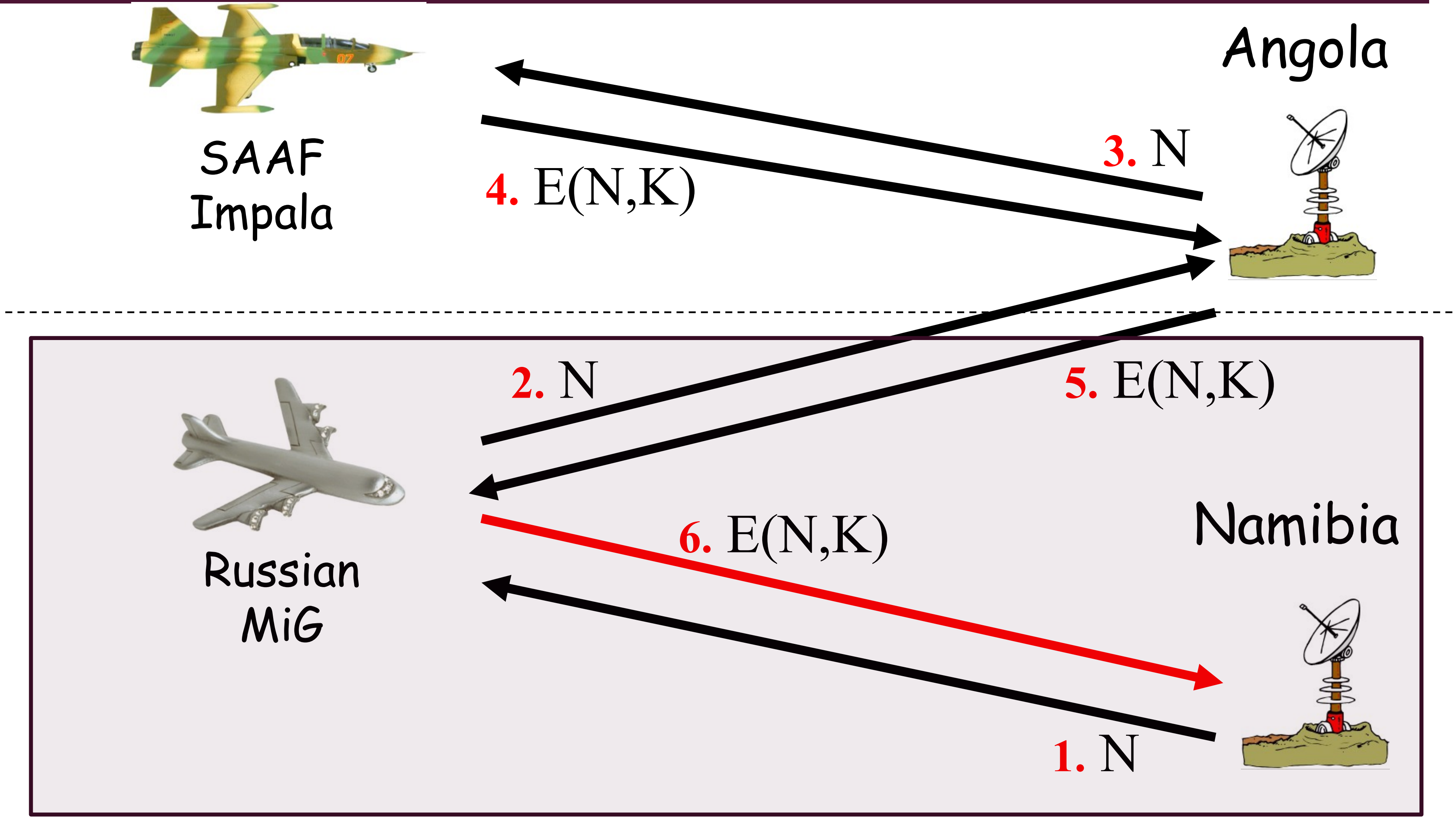
    No? Machine eats card

# IDENTIFY FRIEND OR FOE (IFF)

- Military needs many specialized protocols
- Many cases, it could recognize friends as enemies, or ….

# MIG IN THE MIDDLE



# AUTHENTICATION PROTOCOLS

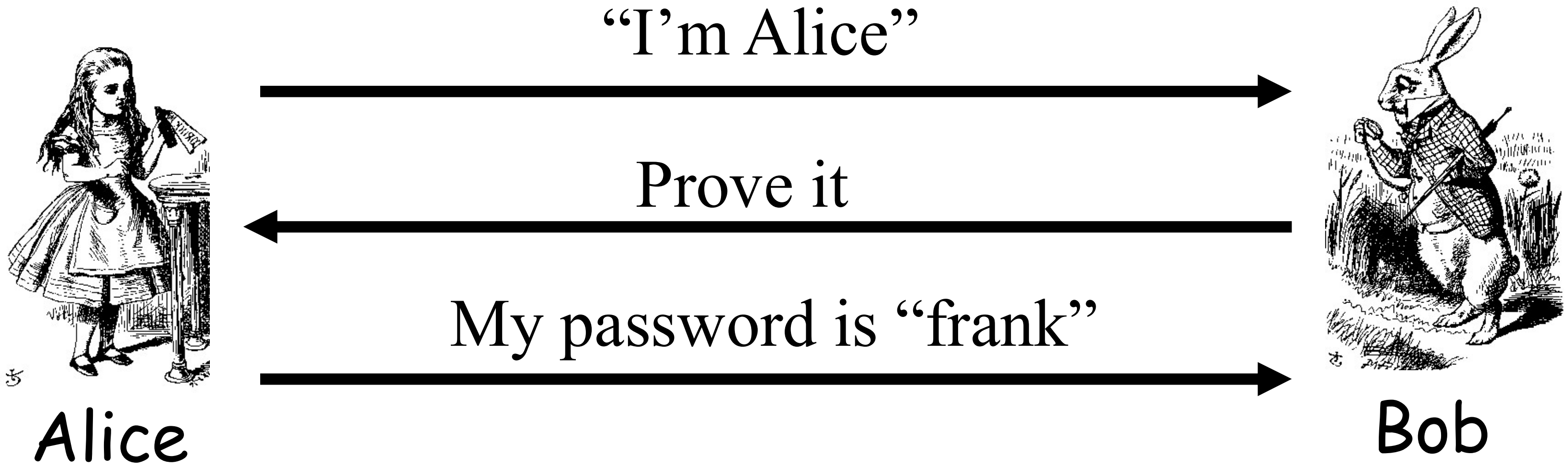
# AUTHENTICATION

- 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

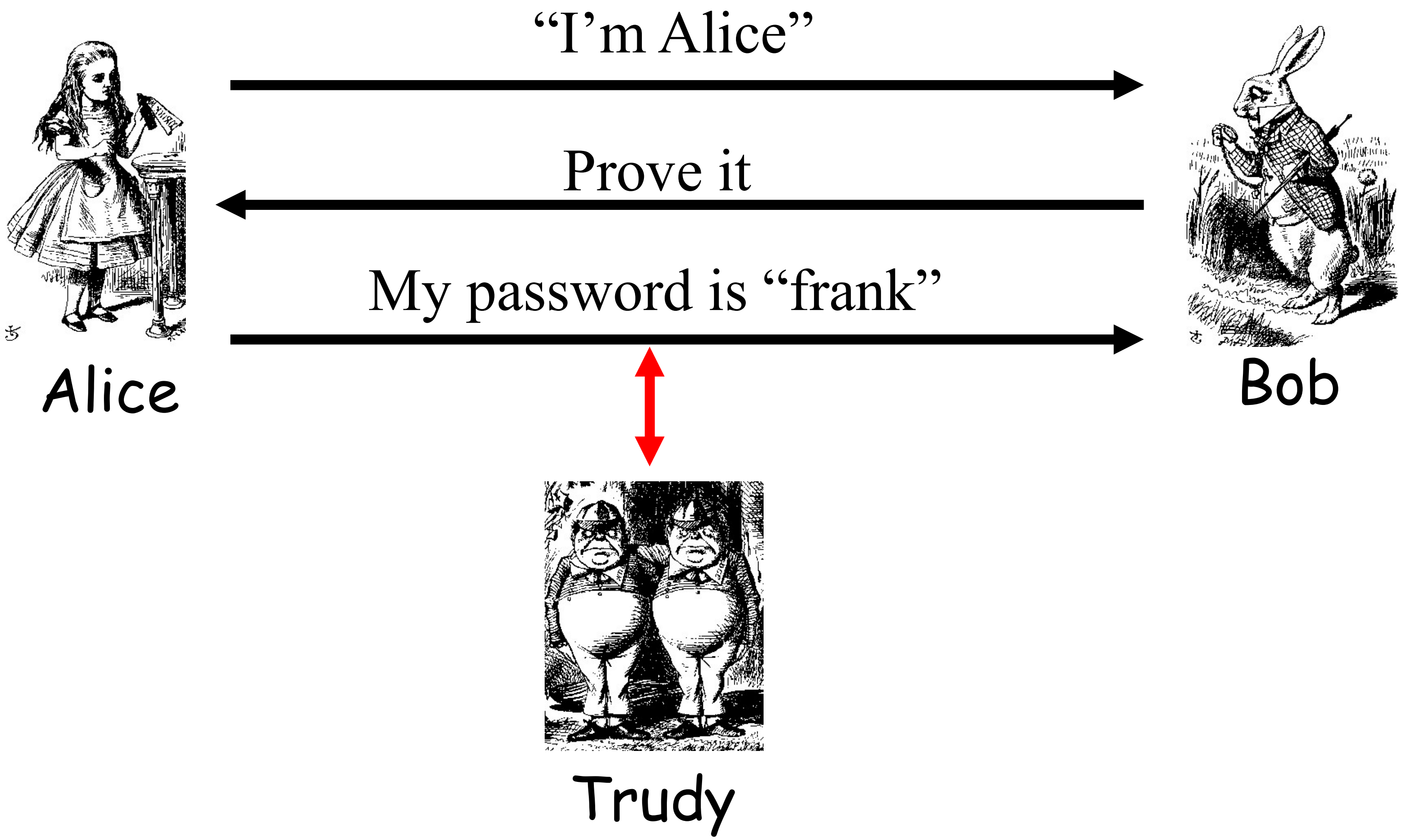
- Authentication on a stand-alone computer is relatively simple
  - "Secure path" is the primary issue
  - Main concern is an attack on authentication software (we discuss software attacks later)
- Authentication over a network is much more complex
  - Attacker can passively observe messages
  - Attacker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

# SIMPLE AUTHENTICATION

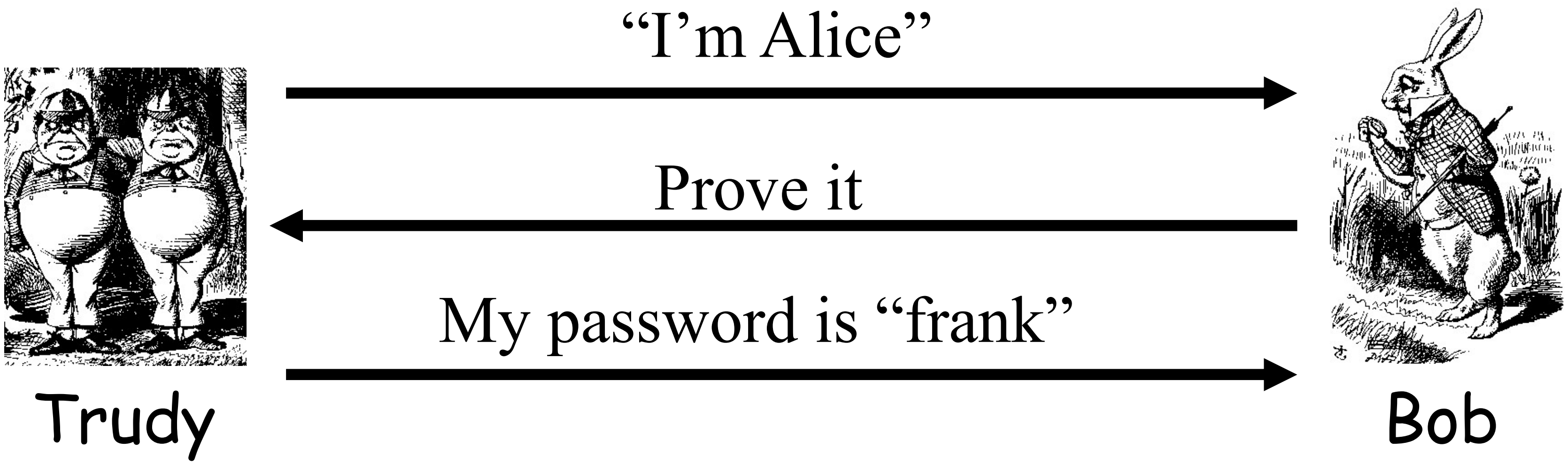



- Simple and may be OK for standalone system
- But insecure for networked system
  - Subject to a replay attack (next 2 slides)
  - Bob must know Alice's password

# AUTHENTICATION ATTACK

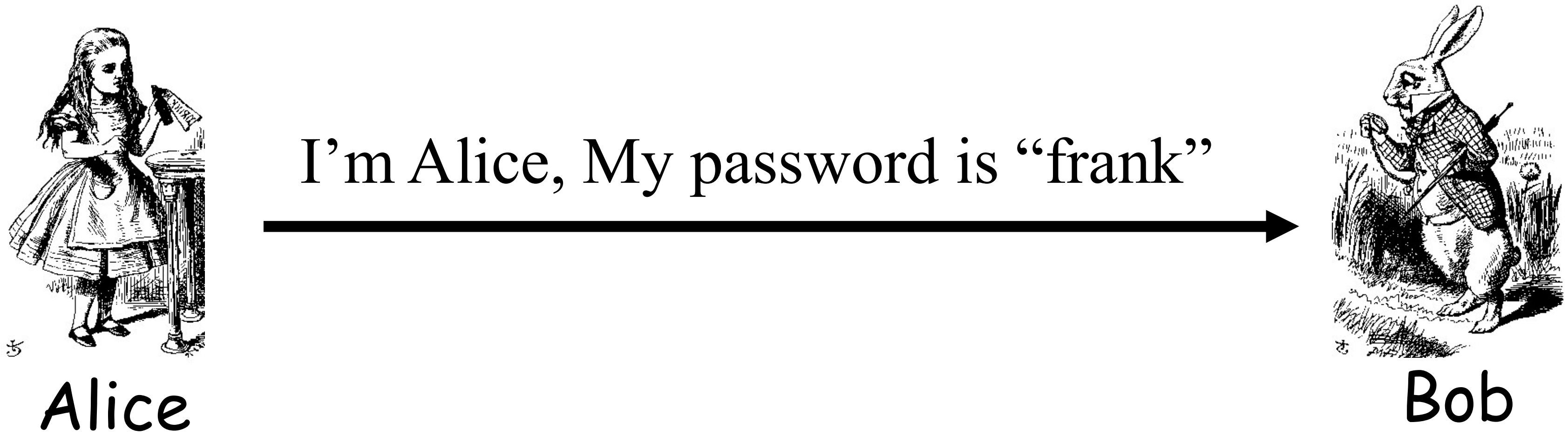


# AUTHENTICATION ATTACK



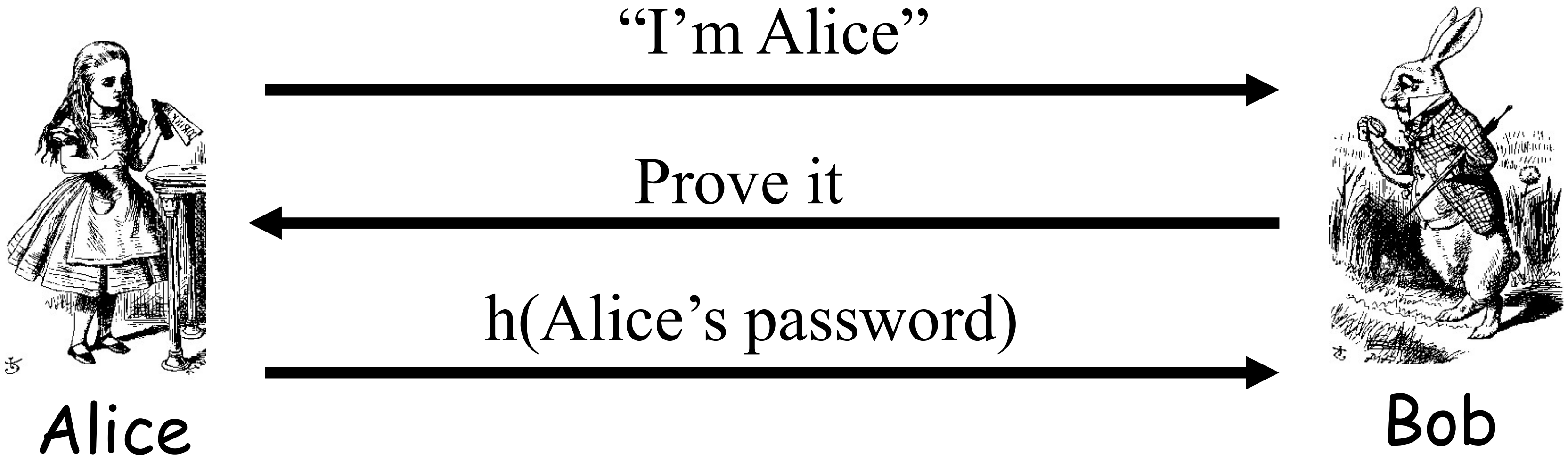

- This is a replay attack
- How can we prevent a replay?

# SIMPLE AUTHENTICATION




- More efficient…
- But same problem as previous version
  - Replay attack

# BETTER AUTHENTICATION




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

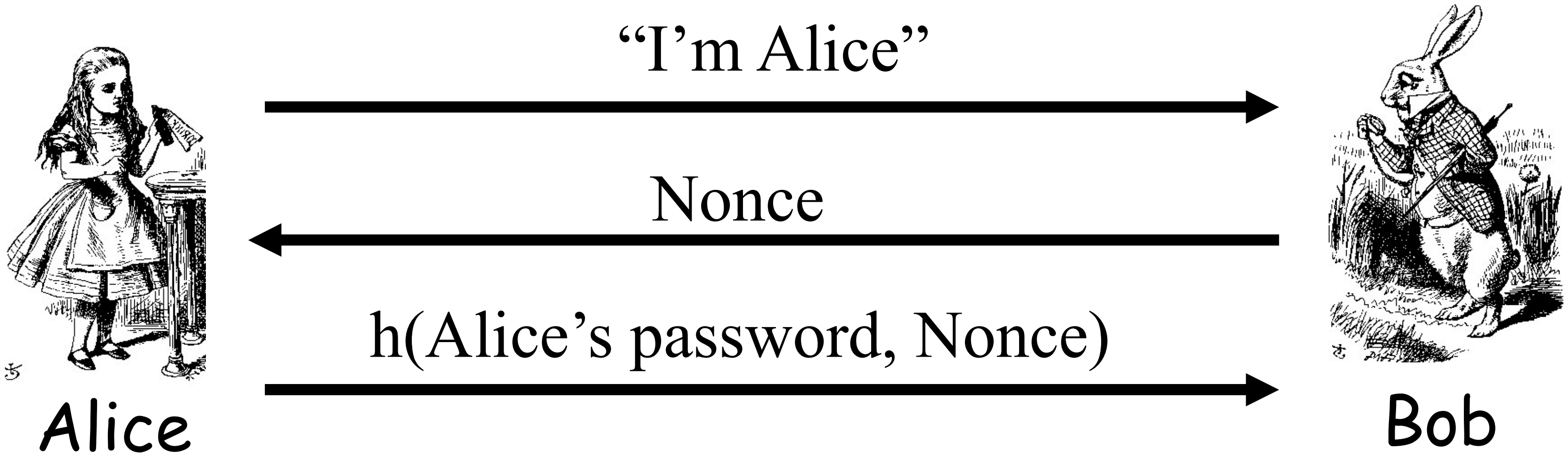
# CHALLENGE-RESPONSE

- To prevent replay, use challenge-response
  - Goal is to ensure freshness"
- Suppose Bob wants to authenticate Alice
  - *Challenge* sent from Bob to Alice
- Challenge is chosen so that
  - Replay is not possible
  - Only Alice can provide the correct *response*
  - Bob can verify the response

# NONCE

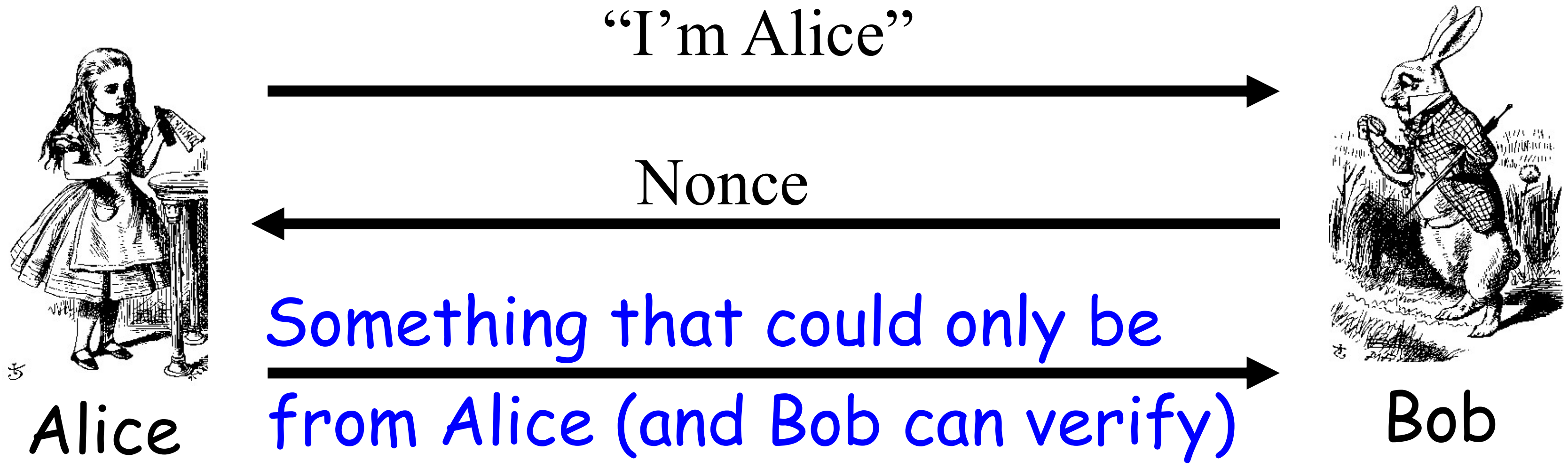
- To ensure freshness, can employ a nonce
  - Nonce == number used once
- What to use for nonces?
  - That is, what is the challenge?
- What should Alice do with the nonce?
  - That is, how to compute the response?
- How can Bob verify the response?
- Should we rely on passwords or keys?

# CHALLENGE-RESPONSE




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

# GENERIC CHALLENGE-RESPONSE




- In practice, how to achieve this?
- Hashed pwd works
- Maybe crypto is better, Why?

# Authentication: Symmetric Key

# SYMMETRIC KEY NOTATION

- Encrypt plaintext **P** with key **K**

  $$C = E(P,K)$$

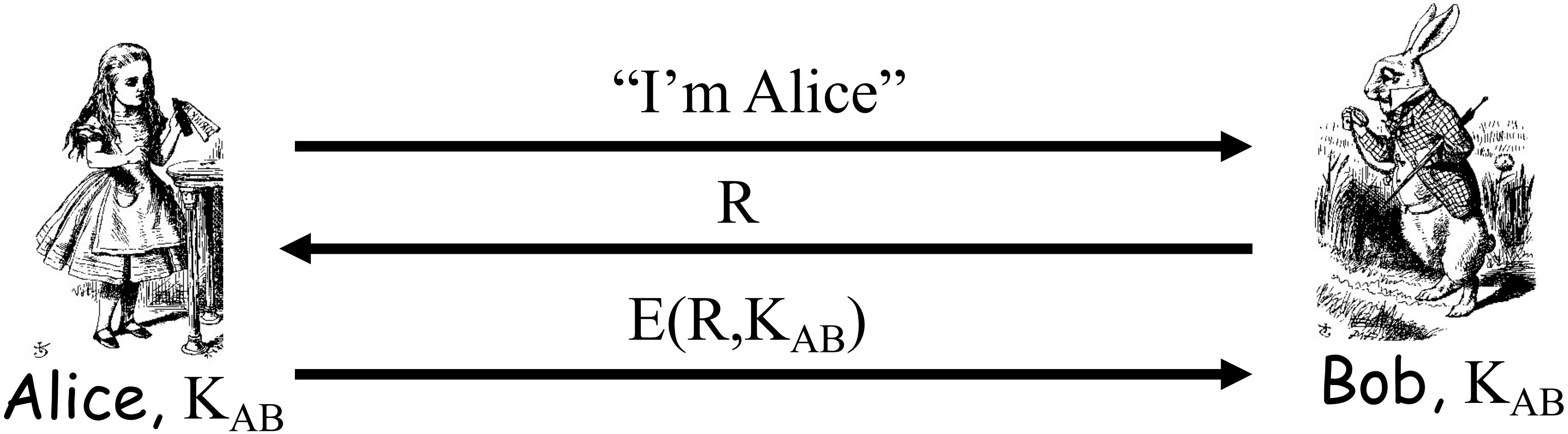
- Decrypt ciphertext **C** with key **K**

  $$P = D(C,K)$$

- Here, we are concerned with attacks on protocols, not attacks on crypto
- So, we assume that crypto algorithm is secure

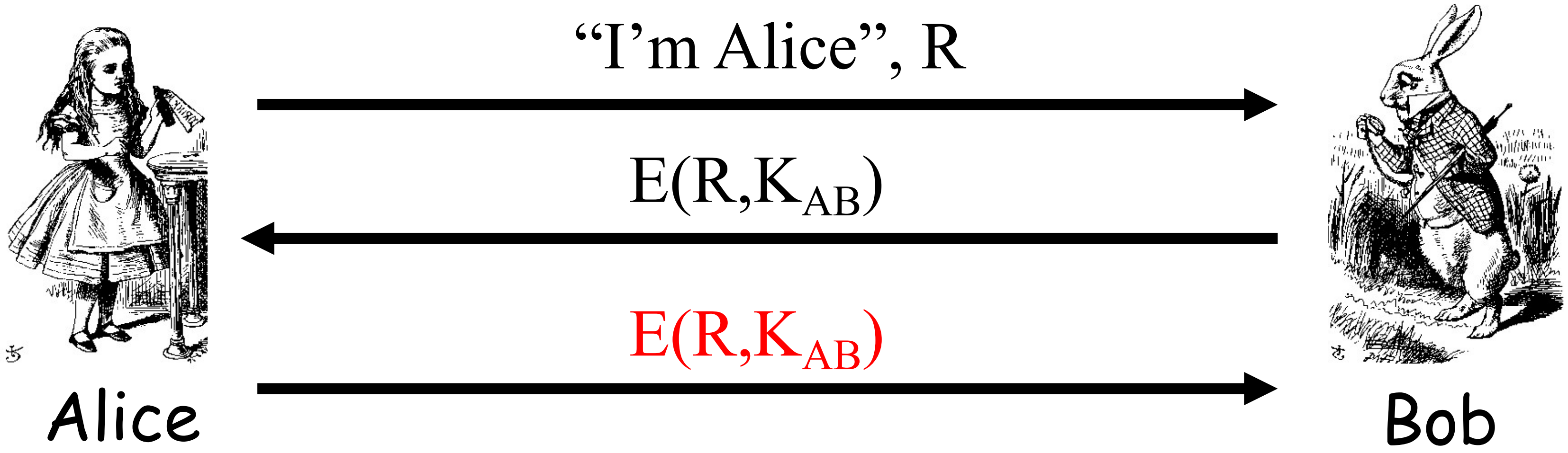
# AUTHENTICATION: SYMMETRIC KEY

- 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 with the following conditions?
  - Must not reveal key
  - Must not allow replay attack
  - Must be verifiable, …




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

# MUTUAL AUTHENTICATION?

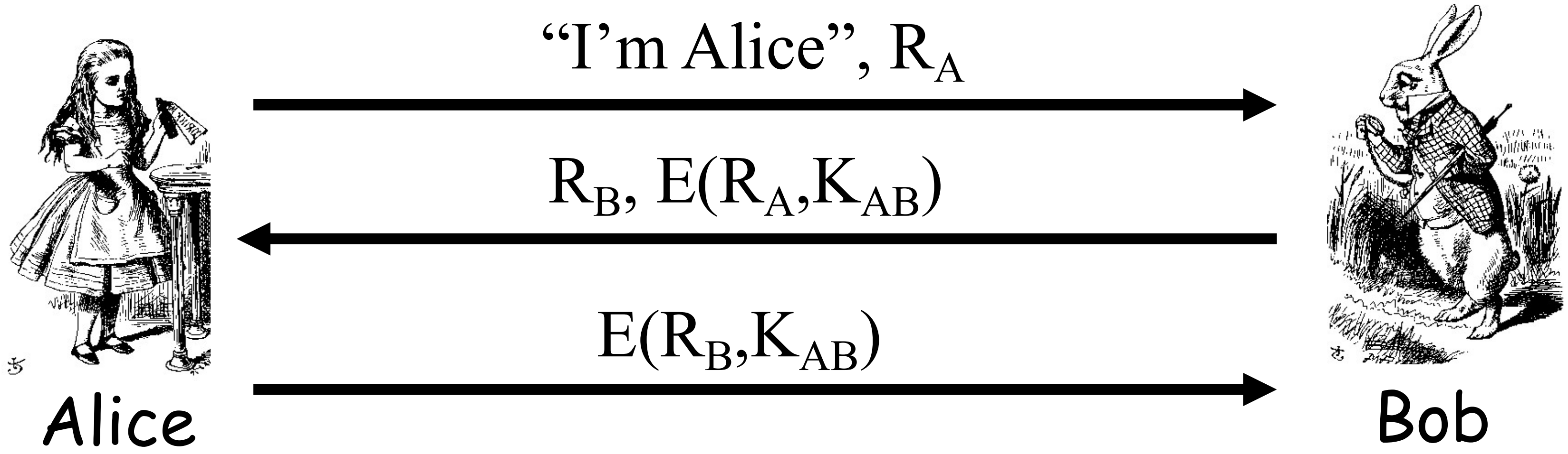



- What's wrong with this picture?
- "Alice" could be Trudy (or anybody else)!

# MUTUAL AUTHENTICATION

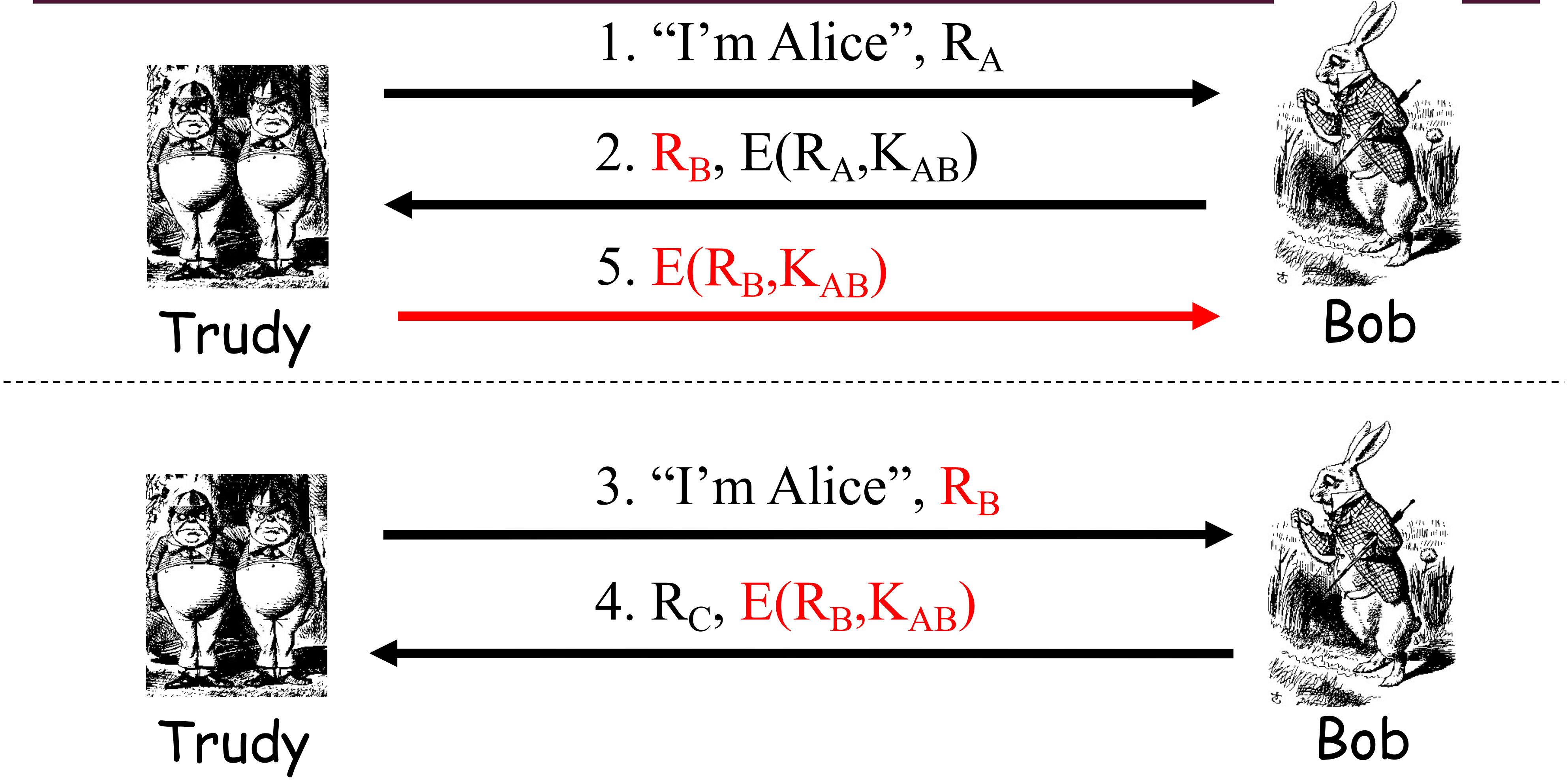
- Since we have a secure one-way authentication protocol…
- The obvious thing to do is to use the protocol twice
  - Once for Bob to authenticate Alice
  - Once for Alice to authenticate Bob
- This has to work…

# MUTUAL AUTHENTICATION




- This provides mutual authentication
- Is it secure? See the next slide…

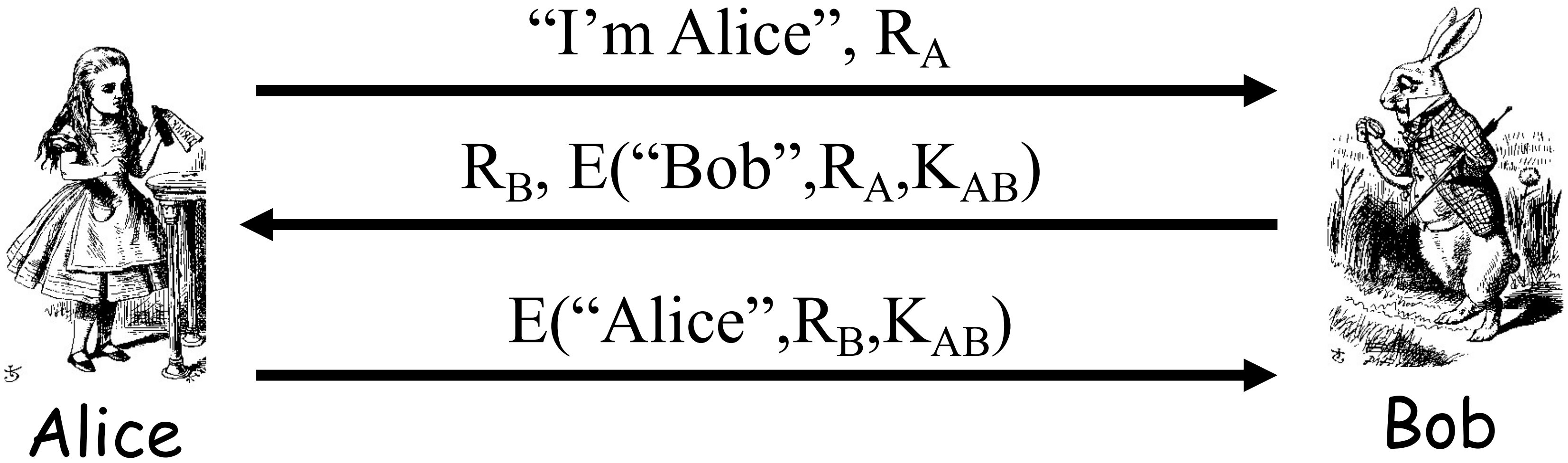
# MUTUAL AUTHENTICATION ATTACK



# 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

# SYM KEY MUTUAL AUTHENTICATION



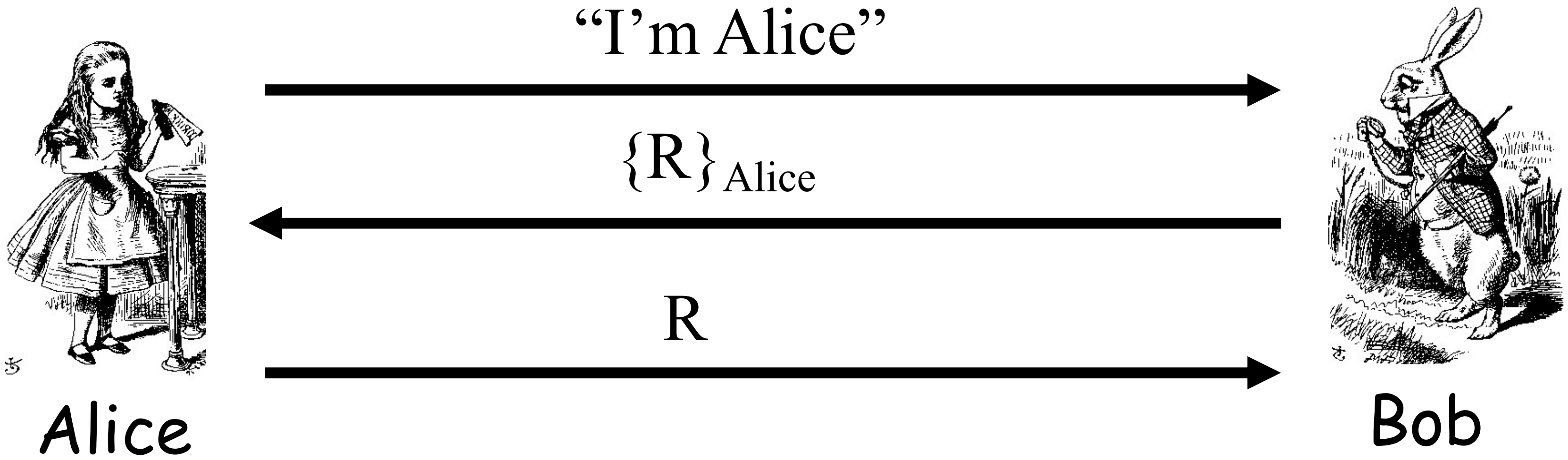

- Do these "insignificant" changes help?
- Yes!

# Public Key Authentication

# PUBLIC KEY NOTATION

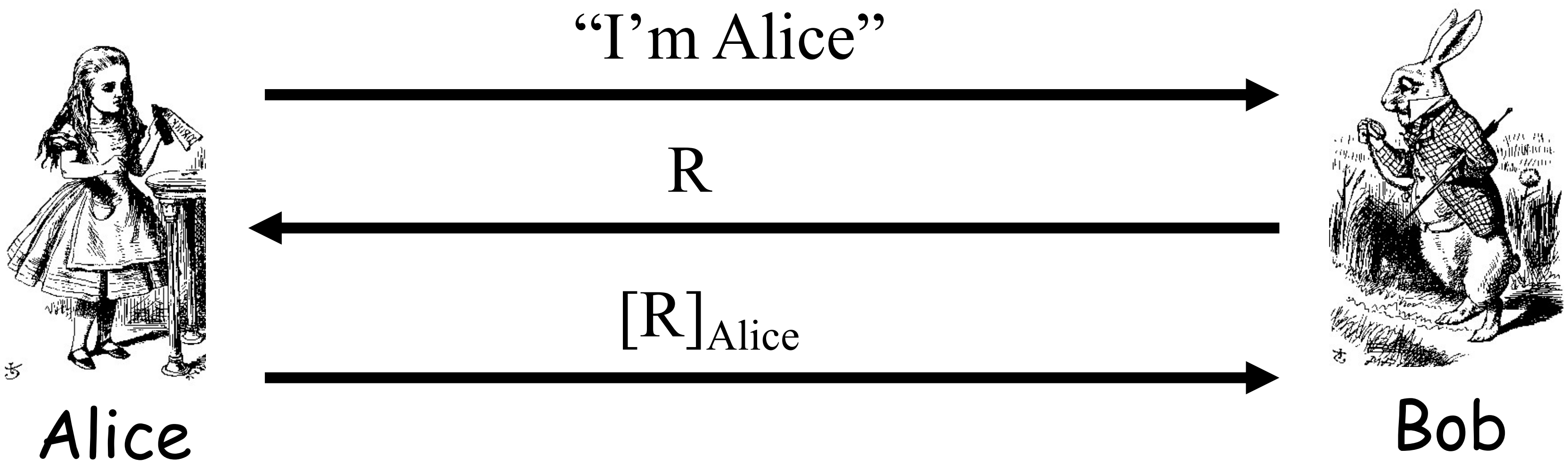
- Encrypt **M** with Alice's public key: $\{M\}_{Alice}$
- Sign **M** with Alice's private key: $[M]_{Alice}$
- Then
  - $[\{M\}_{Alice}]_{Alice} = M$
  - $\{[M]_{Alice}\}_{Alice} = M$
- Anybody can do public key operations
- Only Alice can use her private key (sign)

# PUBLIC KEY AUTHENTICATION




- Is this secure?
- Trudy can get Alice to decrypt anything!
  - Should not use the key for encryption
  - Must have two key pairs

# PUBLIC KEY AUTHENTICATION




- Is this secure?
- Trudy can get Alice to sign anything!
  - Should not use the key for sign
  - Must have two key pairs

# PUBLIC KEYS

- Generally, a bad idea to use the same key pair for encryption and signing
- Instead, should have…
  - …one key pair for encryption/decryption
  - … and a different key pair for signing/verifying signatures

# SESSION KEY

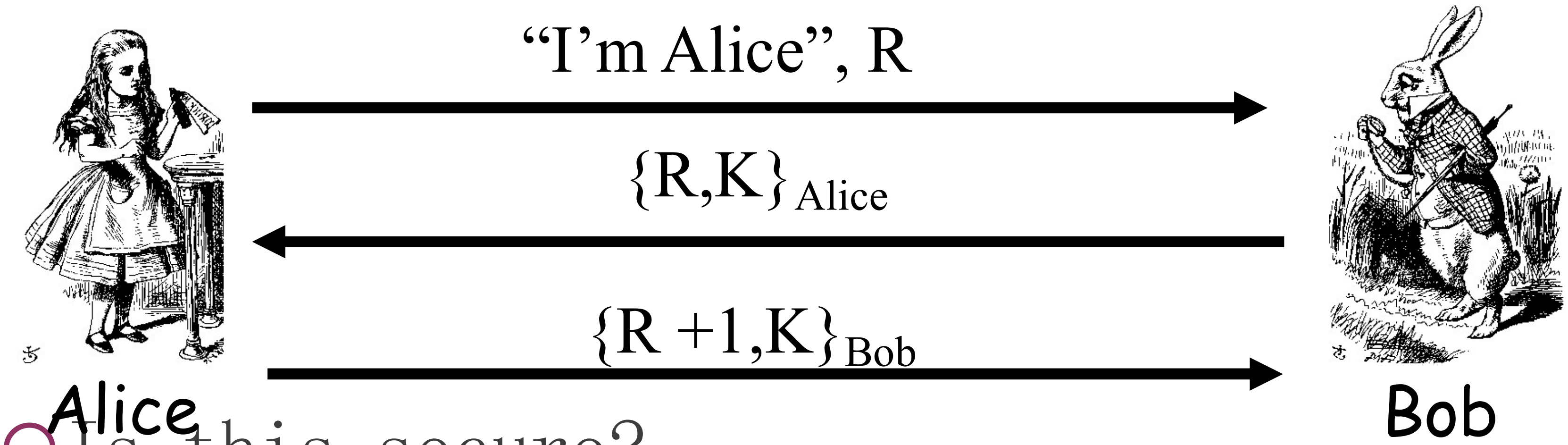
- Session key: temporary key, used for a short time period
- Usually, a session key is required
  - i.e. a symmetric key for a particular session
  - used for confidentiality and/or integrity
  - Limit damage if one session key compromised

# SESSION KEY

- How to authenticate and establish a session key (i.e. shared symmetric key)?
  - When authentication completed, want Alice and Bob to share a session key
  - Trudy cannot break the authentication…
  - …and Trudy cannot determine the session key

# PUB KEY AUTHEN AND SESS KEY

- Using Encryptions of Alice and Bob




- Is this secure?
  - Alice is authenticated and session key is secure
  - Alice's "nonce", R, useless to authenticate Bob
  - The key K is acting as Bob's nonce to Alice
- No mutual authentication

- next

# PUB KEY AUTHEN AND SESS KEY

- Using Signs of Alice and Bob

Alice

"I'm Alice", R →

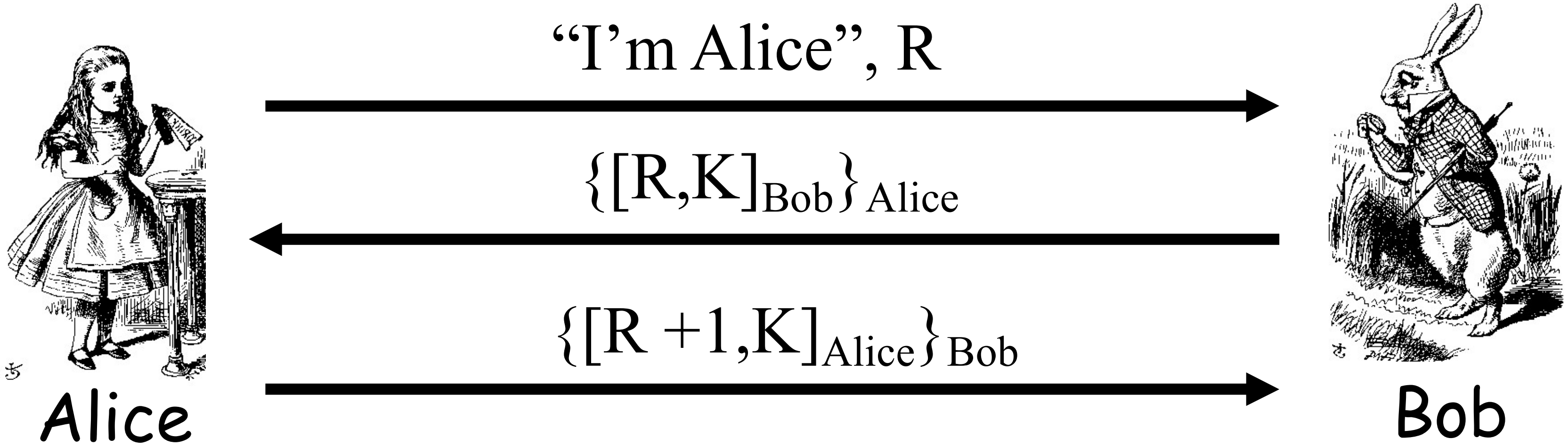
← $[R,K]_{Bob}$

$[R+1,K]_{Alice}$ →

Bob

- Is this secure?
  - Mutual authentication (good), but…
  - … session key is not secret (very bad)

- First Sign and encrypt




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

- First encrypt and Sign

Alice

“I’m Alice”, R $\rightarrow$

$\leftarrow$ $[\{R,K\}_{Alice}]_{Bob}$

$[\{R+1,K\}_{Bob}]_{Alice}$ $\rightarrow$

Bob

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

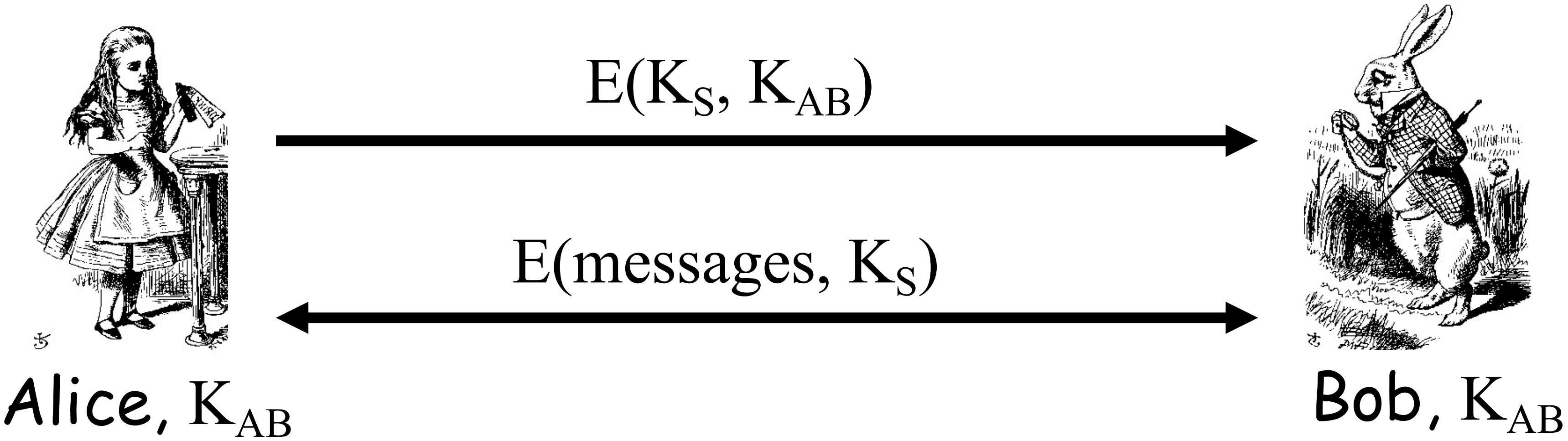
# PERFECT FORWARD SECRECY

- 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 ensures PFS?

# NAÏVE SESSION KEY PROTOCOL




- Trudy could also record $E(K_S,K_{AB})$
- If Trudy later gets $K_{AB}$, she can get $K_S$
  - Then Trudy can decrypt recorded messages

# PERFECT FORWARD SECRECY

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

$g^a \bmod p$ →

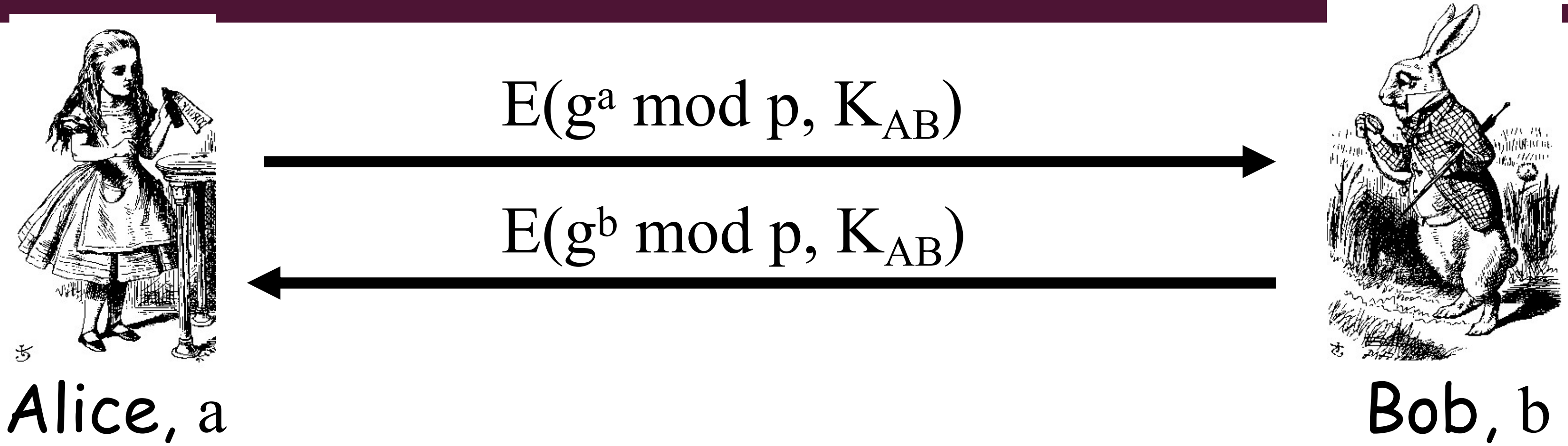
← $g^b \bmod p$

Alice, a

Bob, b

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

# PERFECT FORWARD SECRECY




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

# MUTUAL AUTHEN, SESS KEY & PFS

Alice

$$\xrightarrow{\text{“I'm Alice”}, R_A}$$

$$\xleftarrow{R_B, [\{R_A, g^b \bmod p\}_{Alice}]_{Bob}}$$

$$\xrightarrow{[\{R_B, g^a \bmod p\}_{Bob}]_{Alice}}$$

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 secret s, 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?

# PUB KEY AUTHEN WITH TIMESTAMP T

Alice

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

$\longleftarrow$ $\{[T+1,K]_{Bob}\}_{Alice}$

Bob

- Secure mutual authentication?
- Session key?
- Seems to be OK

# PUB KEY AUTHEN WITH TIMESTAMP T

Alice

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

← $[\{T+1,K\}_{Alice}]_{Bob}$

Bob

- Secure authentication and session key?
- Trudy can use Alice’s public key to find $\{T,K\}_{Bob}$ and then…

# PUB KEY AUTHEN WITH TIMESTAMP T


Trudy

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

$\longleftarrow$ $[\{T+1,K\}_{Trudy}]_{Bob}$


Bob

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

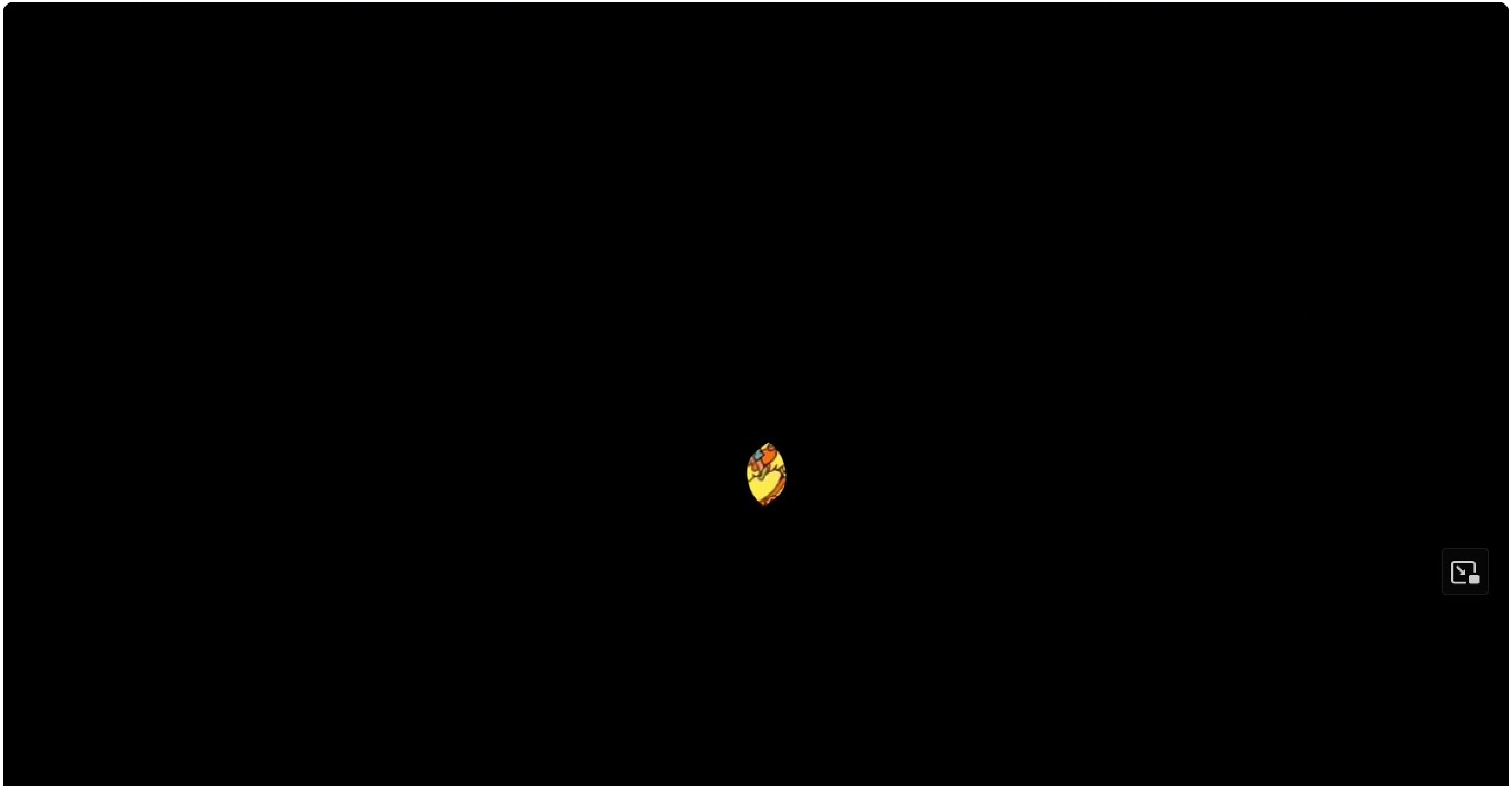
# PUBLIC KEY AUTHENTICATION

- 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!

# ZERO KNOWLEDGE PROOF (ZKP)

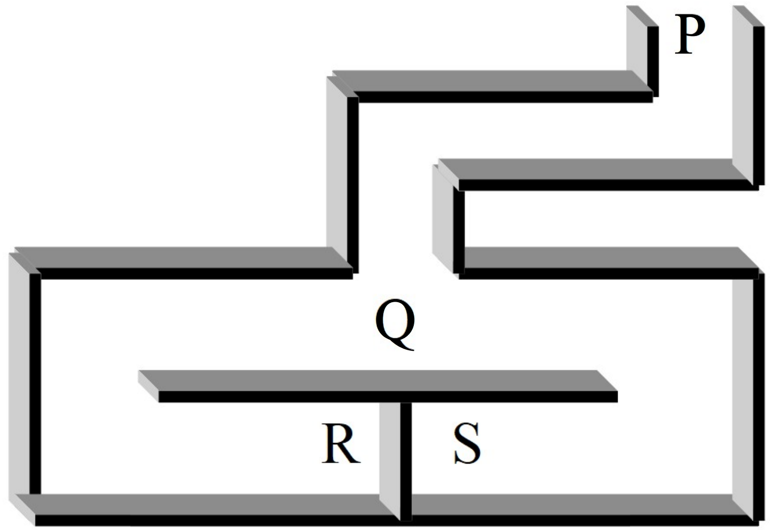
# ZERO KNOWLEDGE PROOF (ZKP)

- 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"



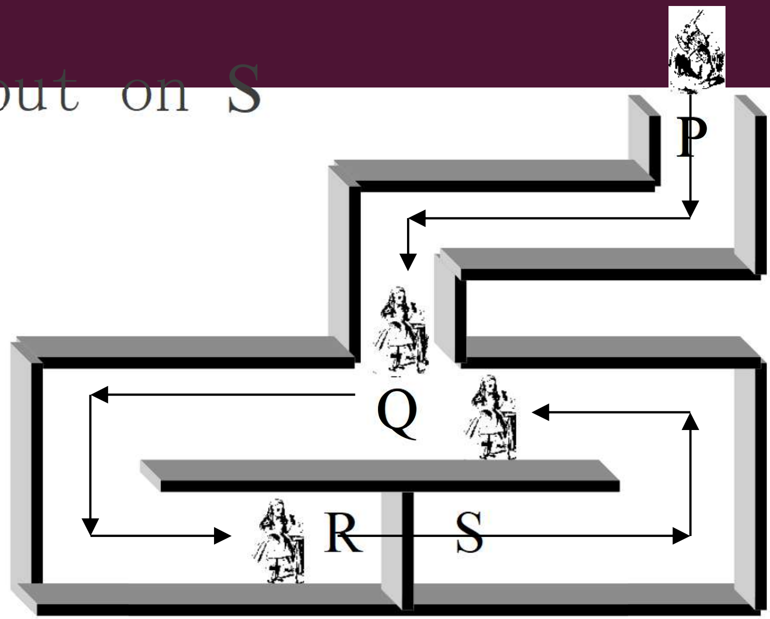
# BOB'S CAVE

- Alice claims to know secret phrase to open path between R and S ("open sasparilla")
- Can she convince Bob that she knows the secret without revealing phrase?



# BOB'S CAVE

- Bob: "Alice come out on S side"
- Alice (quietly): "Open sasparilla"
- Apse Alice does not know secret




- Without knowing secret, Alice could come out from the correct side with probability ½
- If Bob repeats this $n$ times, then Alice can only fool Bob with probability $1/2^n$

# FIAT-SHAMIR PROTOCOL

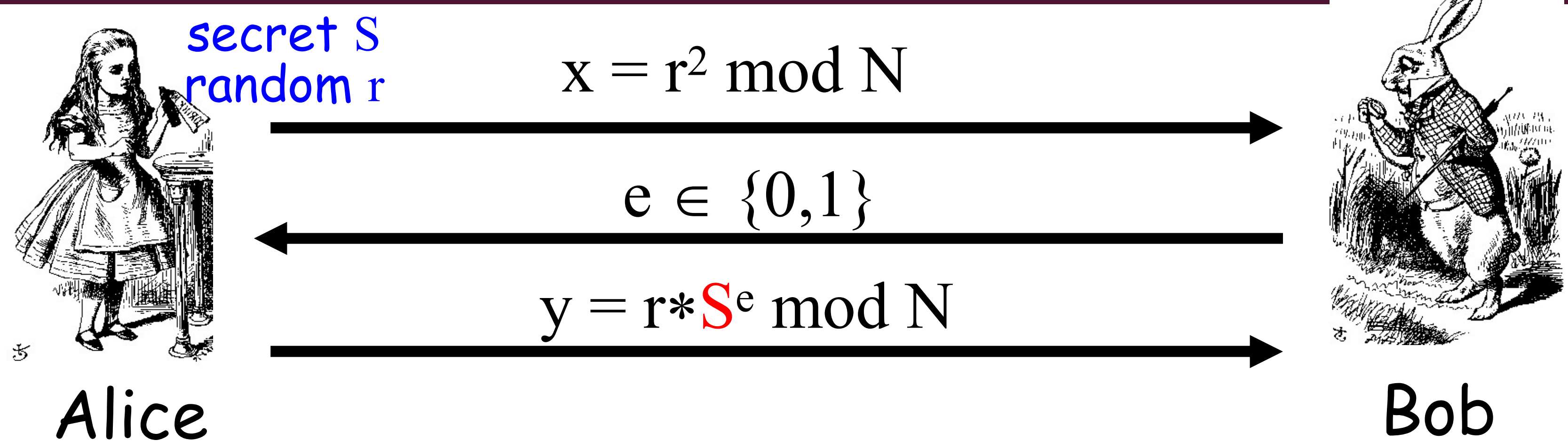
- Cave-based protocols are inconvenient
  - Can we achieve same effect without a cave?
- It is known that finding square roots modulo N is difficult (like factoring)
- Suppose $N = pq$, where p and q prime
- Alice has a secret S
  - N and $v = S^2 \bmod N$ are public, S is secret
- Alice must convince Bob that she knows S without revealing any information about S

# FIAT-SHAMIR PROTOCOL

- N and $v = S^2 \bmod N$ are public, S is secret
- Example
  - P = 7, q = 5, N = 35
  - S=10, $S^2$ =100
  - 100 mod 35 = 30 mod 35
  - 35 and 30: public, 10: secret

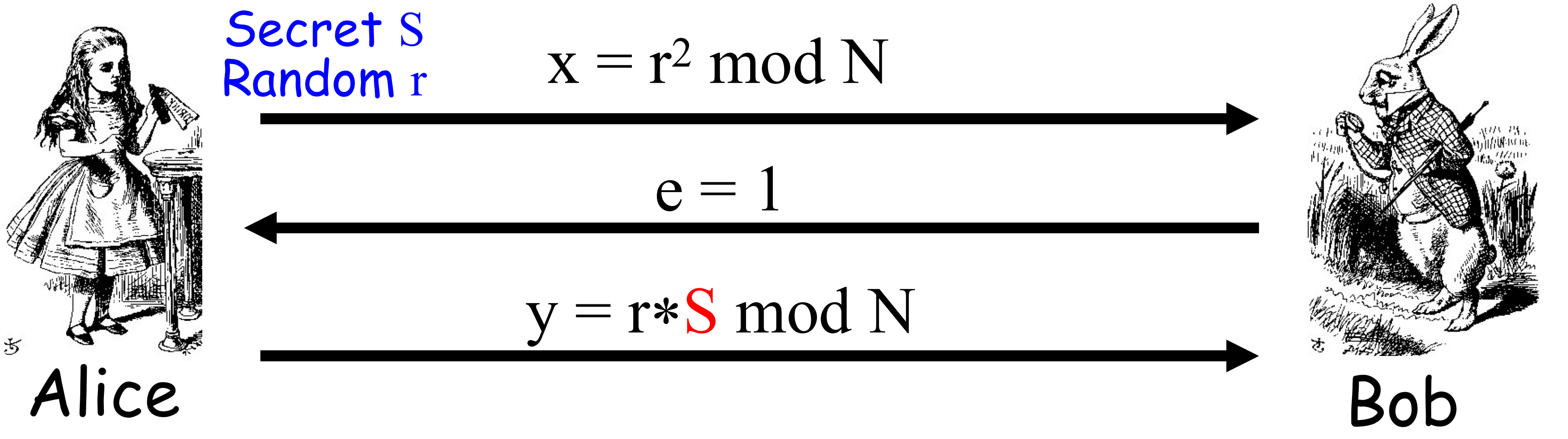
$$\sqrt{30 \bmod 35} = ???$$

# FIAT-SHAMIR




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

# FIAT-SHAMIR: E = 1




- **Public:** Modulus N and $v = S^2 \bmod N$
- Alice selects random **r**, Bob chooses $e = 1$
- If $y^2 = x \cdot v \bmod N$ then Bob accepts it
  - I.e., "Alice" passes this iteration of the protocol
- Note that Alice must know S in this case

# FIAT-SHAMIR: E = 0

Alice

Secret S
Random r

$x = r^2 \bmod N$ →

← $e = 0$

$y = r \bmod N$ →

Bob

Random e

- **Public:** Modulus N and $v = S^2 \bmod N$
- Alice selects random r, 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?

- If everyone follows protocol, math works:
  - Public: $v = S^2 \bmod N$
  - Alice to Bob: $x = r^2 \bmod N$ and $y = r \cdot S^e \bmod N$
  - Bob verifies: $y^2 = x \cdot v^e \bmod N$
- 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 $e = 1$, 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 trick Bob with probability 1/2

# FIAT-SHAMIR FACTS

- Trudy can trick Bob with prob 1/2, but…
  - …after **n** iterations, the probability that Trudy can convince Bob that she is Alice is only $1/2^n$
  - Just like Bob's cave!
- Bob's **e** ∈ {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 nobody learns **anything** about the secret $S$
  - Public: $v = S^2 \bmod N$
  - Trudy sees $r^2 \bmod N$ in message 1
  - Trudy sees $r*S \bmod N$ in message 3 (if $e = 1$)
- If Trudy can find $r$ from $r^2 \bmod N$, gets $S$
  - But that requires modular square root
  - If Trudy could find modular square roots, she can get $S$ from public $v$
- The protocol does not "help" to find $S$

# ZKP IN THE REAL WORLD

- Public keys identify users
  - No anonymity if public keys 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"
  - ZKP not just fun and games for mathematicians!

# BEST AUTHENTICATION PROTOCOL?

- It depends on…
  - The sensitivity of the application/data
  - The delay that is tolerable
  - The cost (computation) that is tolerable
  - What crypto is supported
    - Public key, symmetric key, hash functions
  - Whether mutual authentication is required
  - Whether PFS, anonymity etc. area concern
- …and possibly other factors