Public Key Cryptology,
Part I: Intro and Key Exchange

Last updated: 4/6/20

Prof. Amir Herzberg

Public Key Cryptology

- Kerckhoff: cryptosystem (algorithm) is public
- Only the key is secret (unknown to attacker)
- Same key for encryption, decryption
  ⇒ if you can encrypt, you can also decrypt!

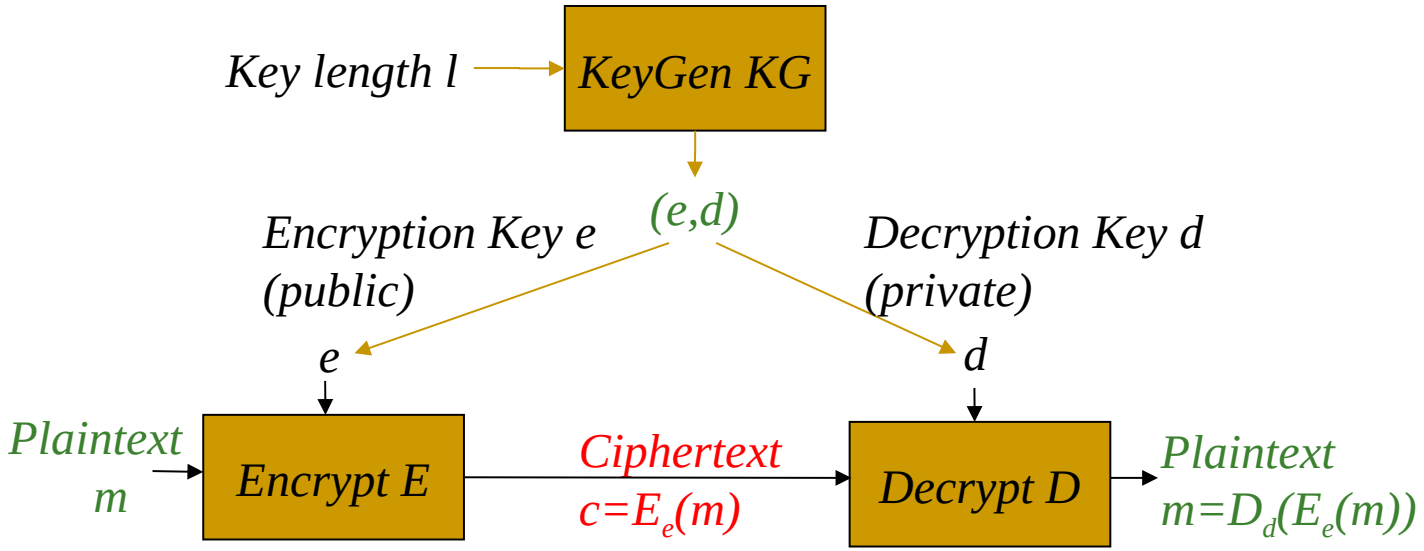
Martin Hellman and Whit Diffie

Martin, help! I want to allow students to send me encrypted email, but not to decrypt emails from other students...

Good idea, Whit! Let me see if I got this right…

Public Key Cryptosystem (PKC)

- Kerckhoff: cryptosystem (algorithm) is public
- [DH76]: can encryption key be public, too??
  - Decryption key will be different (and private)
  - Everybody can send me mail, only I can read it.



Public Key Cryptosystem (PKC)

- Encryption key is public
- Decryption key is private (and different:)
  - Everybody can send me mail, only I can read it

[DH76]: Public Key Cryptology

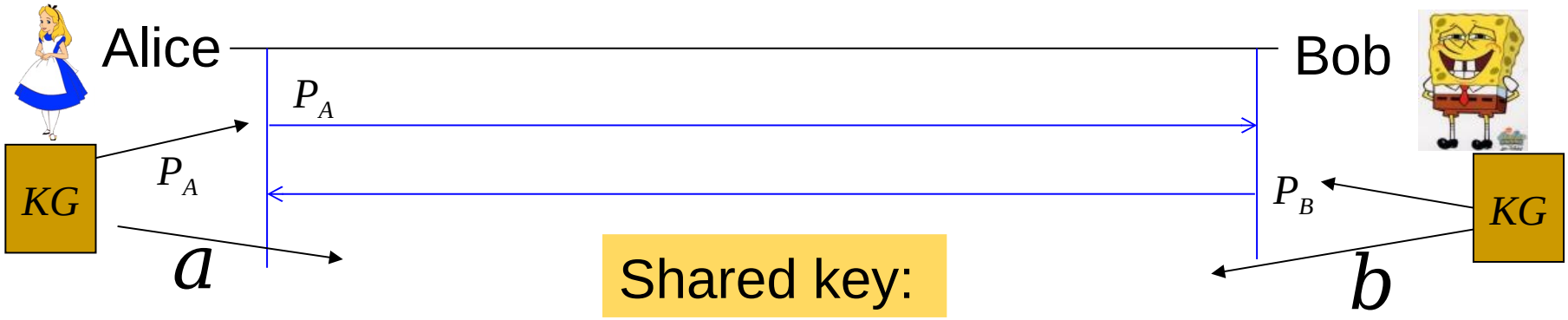
- Public-key Cryptosystem (RSA,…)
  - Public encryption key , private decryption key
- Also: Digital signatures (RSA, DSA,…)
  - Sign with private key s, verify with public key v

Public keys solve more problems…

- Signatures provide **evidences**
  - Everyone can validate, only ‘owner’ can sign
- Establish shared secret keys
  - Use authenticated public keys
    - Signed by trusted certificate authority (CA)
    - Which CA can we trust? What if they fail? …

[DH76]: DH Key-Exchange Protocol

- Public-Key Cryptosystem (RSA,…)
  - Public encryption key , private decryption key
- Also: Digital signatures (RSA, DSA,…)
  - Sign with private key s, verify with public key v
- Key Exchange
  - Use public information from A, B to setup shared secret key. Eavesdropper cannot learn the key.



Public keys solve more problems…

- Signatures provide **evidences**
  - Everyone can validate, only ‘owner’ can sign
- Establish shared secret keys
  - Use authenticated public keys
    - Signed by trusted certificate authority (CA)
  - Or: use DH key exchange
- Stronger resiliency to key exposure
  - Perfect forward+recover secrecy
    - Protect confidentiality from possible key exposures
  - Threshold (and proactive) security
    - Resilient to exposure of out of parties (every period)

Public keys are easier…

- To distribute:
  - From directory (ensure or trust authentication)
  - From incoming message (if authenticated)
  - Less keys to distribute (same public key to all)
- To maintain:
  - Can keep in non-secure storage
  - Validate (e.g. using MAC) before using
  - Less keys: $O(|parties|)$, not $O(|parties|^2)$
- So: why not **always** use public key crypto?

Public key crypto is harder…

- Requires related public, private keys
  - Private key `reverses` public key
  - Public key does not expose private key
- Substantial overhead
  - Successful cryptanalytic shortcuts need long keys (cf. shared key!)
  - Elliptic Curves (EC) may allow shorter key (almost no shortcuts found)
  - Complex computations
  - RSA: very complex (slow) key generation
- Most: based on hard modular math problems

[LV02]	Required key size		
Year	AES	RSA, DH	EC
2010	78	1369	160
2020	86	1881	161
2030	93	2493	176
2040	101	3214	191

Commercial-grade security
Lenstra & Verheul [LV02]

Public key crypto is harder…

Year	Symmetric			Factoring (**RSA**), DiscLog (DH)			EC		
	LV ‘02	NIST 2014	BSI ‘17	LV 2002	NIST 2014	BSI ‘17	LV ‘02	NIST 2014	BSI ‘17
2020	**86**	**112**	**128**	1881	2048	2000	161	224	250
2030	**93**	**112**	**128**	2493	2048	3000	176	224	250
2040	**101**	**128**	**128**	3214	3072	3000	191	256	250
Cr++	**4525 MiB/s AES/CTR 128b**			0.01ms(1024b), 0.03ms(2048b)			**1ms (256b ECIES)**		

Hard Modular Math Problems

- No efficient solution, In spite of extensive efforts
  - But: **verification** of solutions is easy (`one-way' hardness)
    - Discrete log: exponentiation
- Problem 1: Factoring
  - Choose randomly $p,q \in_R LargePrimes$
  - Given pq, it is infeasible to find p,q
  - Verification? Easy, just multiply factors
  - Basis for the RSA cryptosystem and many other tools
- Problem 2: Discrete logarithm in cyclic group G_q
  - Given random number, find its (discrete) logarithm
  - Verification is efficient by exponentiation: $O((\lg n)^3)$
  - Basis for the Diffie-Hellman Key Exchange and many other tools
  - We first discuss key-Exchange problem, then [DH] and disc-log

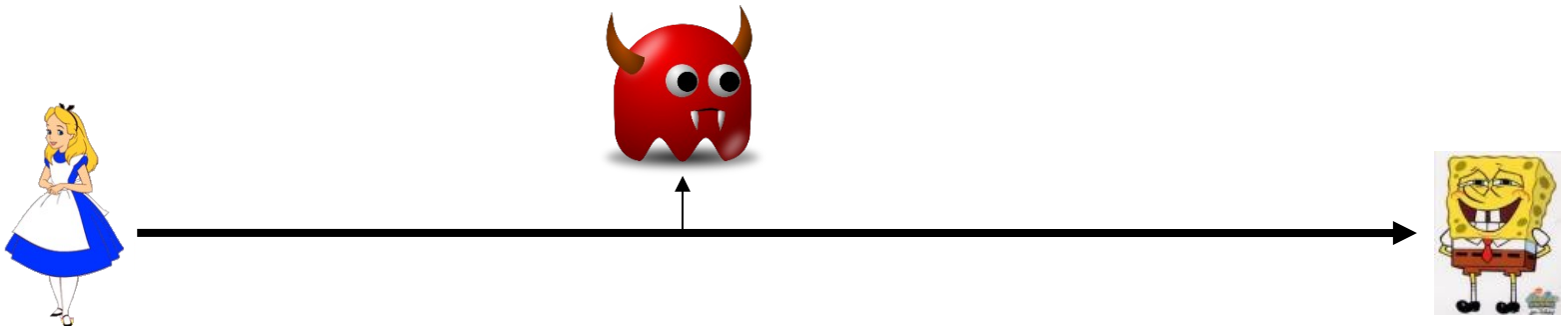
Public Key Cryptology,
Part I: Intro and Key Exchange

- Introduction to Public Key Cryptology
- **The Key Exchange problem**
  - **‘Toy protocols’**
  - The DL assumption
  - The DH protocol and CDH/DDH assumptions
  - Key Derivation Function (KDF)
  - Authenticated DH and PFS
  - Improved resiliency – Ratchet protocols

The Key Exchange Problem

Aka key agreement

- Alice and Bob want to agree on secret (key)
  - Secure against **eavesdropper** adversary
  - Assume no prior shared secrets (key)
    - Otherwise seems trivial
    - Actually, we'll later show it's also useful in this case…
  - Afterwards, may use agreed-on secret as key
- First: Physical Key Exchange



Physical Key Exchange

Problem

- Protocol for Alice and Bob:
  - Goal: agree on shared secret key 

  - Alice has:
    - A padlock 

and its key 

    - A box (can be locked) 
  - Bob has only padlock 

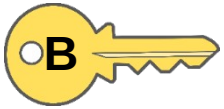
and its key 

  - Attacker is Eavesdropper
    - Can’t open locked box or expose keys

Physical Key Exchange Protocol

Alice

Bob



Put key in box

lock it

Physical Key Exchange Protocol

Alice

Bob

Put key in box

lock it

and send

lock it too

Physical Key Exchange Protocol

Alice

Bob

Put key in box

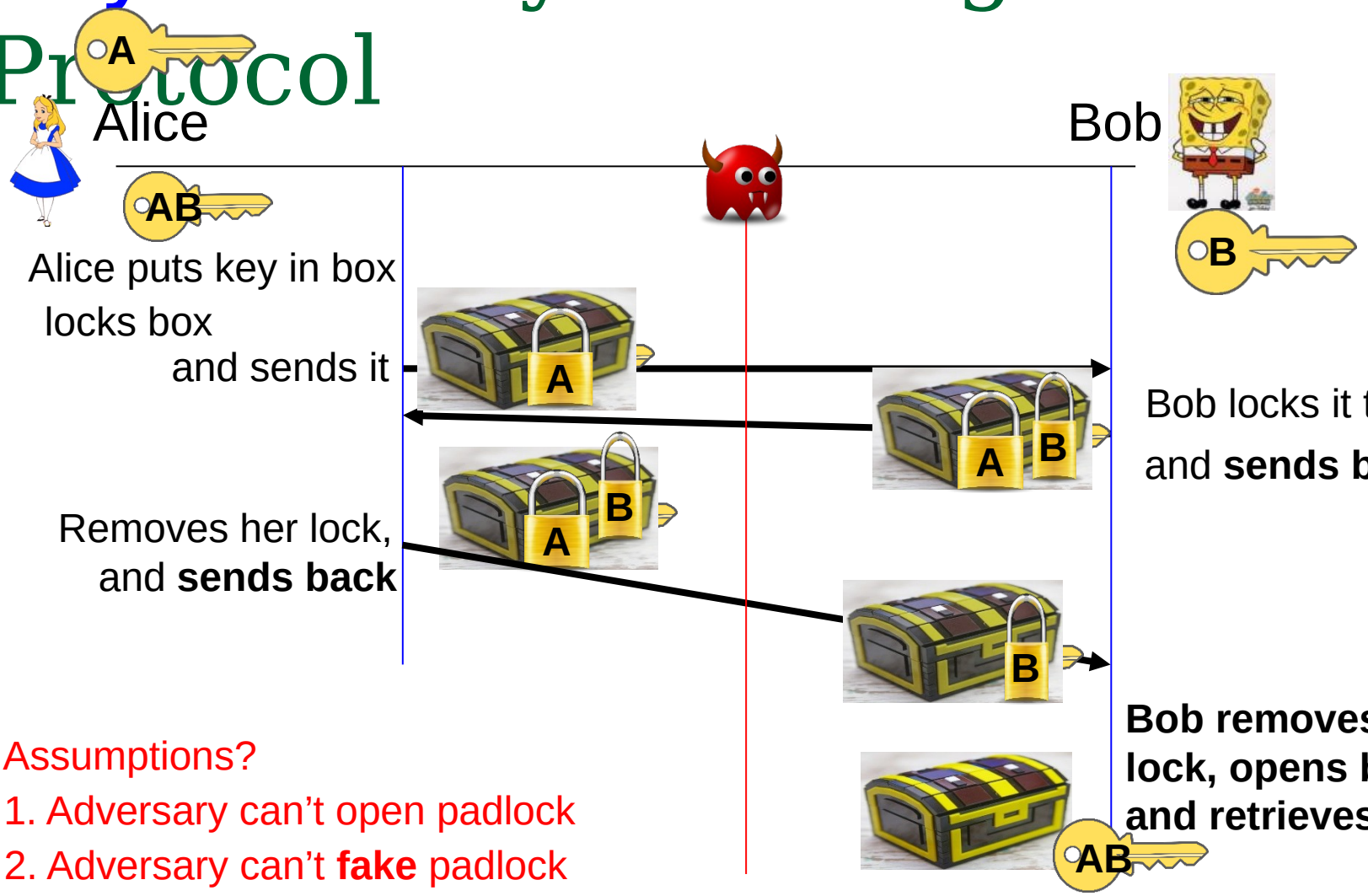
lock it

and send

lock it too,

and **send back**

Physical Key Exchange Protocol



Assumptions?

1. Adversary can't open padlock
2. Adversary can't **fake** padlock

Can we do the same – but only with public info?

XOR (One Time Pad) Key Exchange?

Alice

Eavesdropper

Bob

Alice 'XOR-locks' key $k' = k \oplus k_A$

$k'' = k \oplus k_A \oplus k_B$

Bob also XOR-locks

Alice XORs again to remove her xor-lock $k''' = k'' \oplus k_A$

$k''' = k \oplus k_B$

Bob removes his xor-lock: $k = k''' \oplus k_B$

EXTRA CREDIT: Is this secure?

Can we use XOR (One Time Pad) as lock?

Alice

Bob

Same attack if we multip
(instead of xor)

Alice 'XOR-locks' key

$k' = k \oplus k_A$

$k'' = k \oplus k_A \oplus k_B$

Bob also XOR-locks

Alice XORs again to remove her lock

$k''' = k'' \oplus k_A$

$k''' = k \oplus k_B$

Bob removes his lock:

$k = k''' \oplus k_B$

No! Adversary can find $k = k' \oplus k'' \oplus k''' =$

$= (k \oplus k_B) \oplus (k \oplus k_B \oplus k_A) \oplus (k \oplus k_A)$

Exponentiation Key Exchange Protocol ?

Alice

Bob

Agree on some random

Choose random key and randomizer

g^{kr_a}

Choose randomizer

$g^{kr_a r_b} = \left(g^{kr_a}\right)^{r_b}$

remove

$g^{kr_b} = \left(g^{kr_a r_b}\right)^{(1/r¿¿a)¿}$

Key:

Key:

Is this secure?

No. Computing log over is not hard! So attacker computes…

=_____ ; _____

But **discrete**-log may be hard!

Discrete Exponentiation Key Exchange

Alice

Bob

Is **this** secure???

Not for 'bad' , e.g., for some integer

'**Yes**' [assumption…] for 'safe prime' (for prime)

Public Key Cryptology,
Part I: Intro and Key Exchange

- Introduction to Public Key Cryptology
- The Key Exchange problem
  - ‘Toy protocols’
  - **The DL assumption**
  - The DH protocol and CDH/DDH assumptions
  - Key Derivation Function (KDF)
  - Authenticated DH and PFS
  - Improved resiliency – Ratchet protocols

The Discrete Log Problem

- Computing log is quite efficient – e.g., over the reals
- Consider a cyclic multiplicative group G
  - Cyclic group: exists generator s.t.
  - Discrete log problem: given generator and , find s.t.
  - Verification: exponentiation (efficient algorithm)
  - For prime , the group ={1,…p-1} is cyclic
- Is discrete-log hard?
  - Some ‘weak’ groups, i.e., where disc-log is **not** hard:
    - for prime , where has only ‘small’ prime factors
      - Using the Pohlig-Hellman algorithm
    - Check!! Mistakes/trapdoors found, e.g., in OpenSSL’16
  - Other groups studied, considered Ok (‘hard’)
  - In particular: for **safe prime: for prime**

Discrete Log Assumption [for safe prime group: **for prime**]

Given PPT adversary A, and n-bit safe prime p:

Comments:

1. Similar assumptions for (some) other groups
2. Knowing , it is easy to find a generator
3. Any generator (primitive element) will do

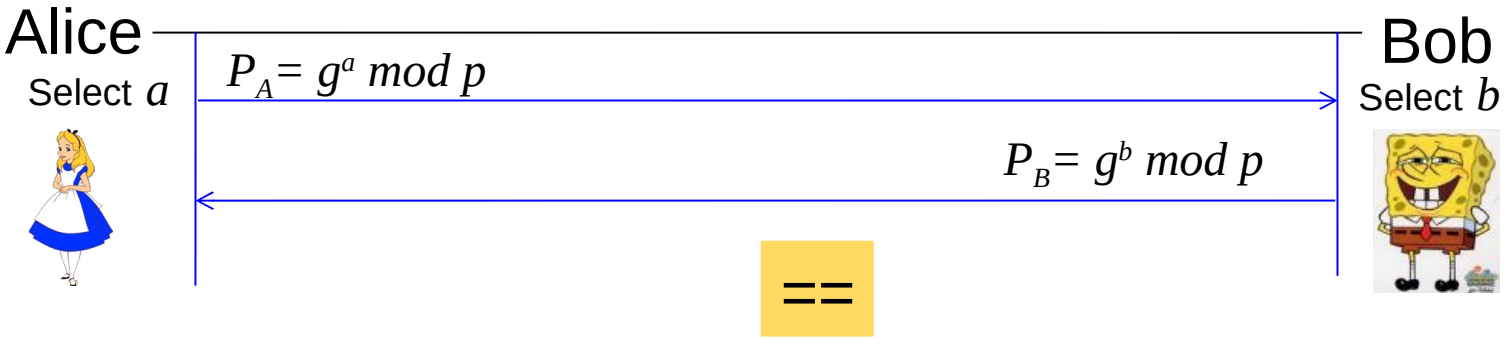
Public Key Cryptology,
Part I: Intro and Key Exchange

- Introduction to Public Key Cryptology
- The Key Exchange problem
  - ‘Toy protocols’
  - The DL assumption
  - **The DH protocol and CDH/DDH assumptions**
  - Key Derivation Function (KDF)
  - Authenticated DH and PFS
  - Improved resiliency – Ratchet protocols

Diffie-Hellman [DH] Key Exchange

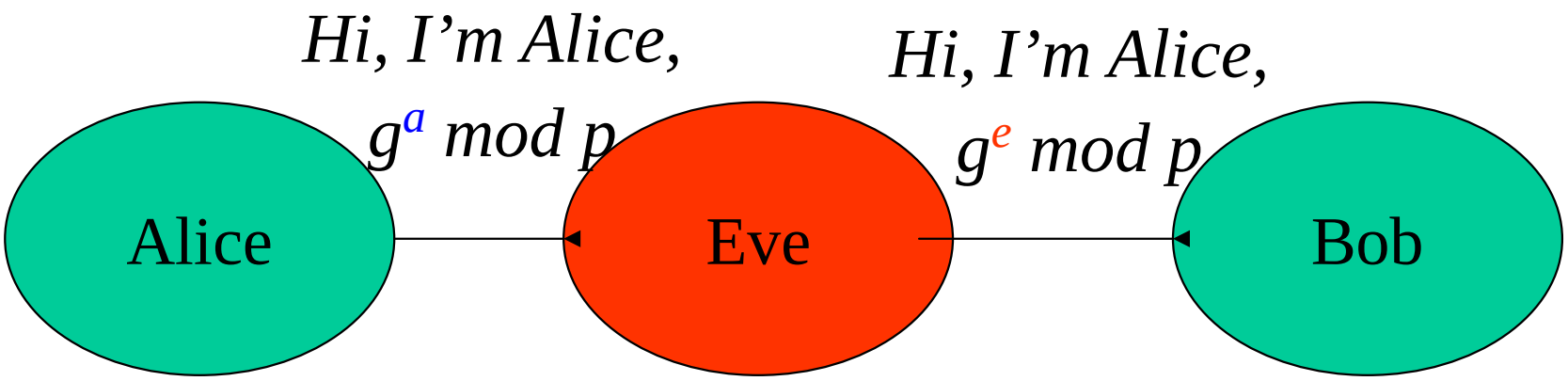
Using cyclic group

- Simplified Discrete Exponentiation Key Exchange
- Agree on a random safe prime *p* and generator *g*
- Alice: secret key a, public key $P_A = g^a \bmod p$
- Bob: secret key b, public key $P_B = g^b \bmod p$
- To set up a shared key :



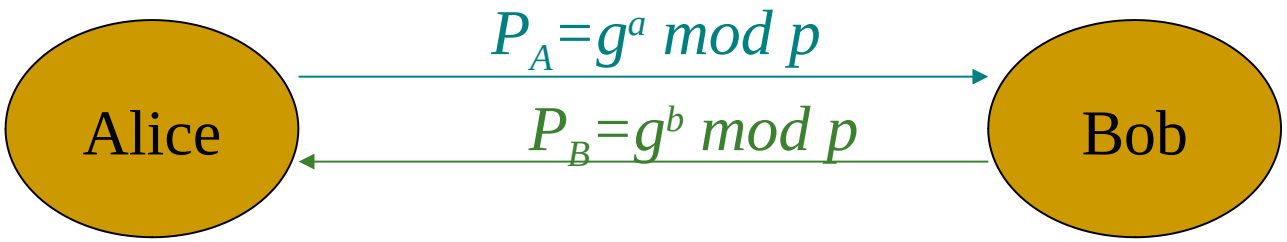
Caution: Authenticate Public Keys!

- Diffie-Hellman key exchange is only secure using the authentic public keys
- If Bob simply receives Alice's public key, [DH] is subject to `Man in the Middle` attack



Security of [DH] Key Exchange

- Assume authenticated communication
- Based on Computational Discrete Log Assumption
- But DH requires stronger assumption than Disc-Log:
  - Maybe from $g^b \bmod p$ and $g^a \bmod p$, Adversary can compute $g^{ab} \bmod p$ (without knowing/learning a,b or ab)?



Computational DH (CDH) Assumption

[for safe prime group]

Given PPT adversary A:

Assume CDH holds. Can we use as key?

Not necessarily; maybe finding some bits of is easy?

Using DH securely?

- Consider (multiplicative group for (safe) prime)
- Can g^a , g^b expose *something* about $g^{ba} \bmod p$?
- Bad news:
  - Finding (at least) one bit about $g^{ba} \bmod p$ is easy!
  - Specifically: if it is quadratic-residue: $x = g^{ba} \bmod p = y^2 \bmod p$
  - Euler showed this holds if $x^{(p-1)/2} = 1 \bmod p$
    - Details: crypto class (and a bit in hidden foil)
- Good news:
  - Many of the bits were shown to be as secure as the whole
  - Also, there are other groups (e.g., Schnorr's), were testing for QR appears a hard problem
- So…how to use DH 'securely'?

Using DH securely?

- Adversary may compute *some* bits over $g^{ba} \bmod p$
- So…how to use DH ‘securely’? Two options!
- Option 1: Use DH but with a `stronger’ group (not mod safe-prime)
  - The (stronger) **Decisional DH (DDH) Assumption:** adversary can’t distinguish between and , for random a, b, c.
- Option 2: use DH with safe prime p… but use a **key derivation function (KDF)** to derive a secure shared key
- Applied crypto mostly uses KDF… and we do too

Public Key Cryptology,
Part I: Intro and Key Exchange

- Introduction to Public Key Cryptology
- The Key Exchange problem
  - ‘Toy protocols’
  - The DL assumption
  - The DH protocol and CDH/DDH assumptions
  - **DH with Key Derivation Function (KDF)**
  - Authenticated DH for key-resiliency and PFS
  - Improved resiliency – Ratchet protocols

Using DH 'securely': CDH + KDF

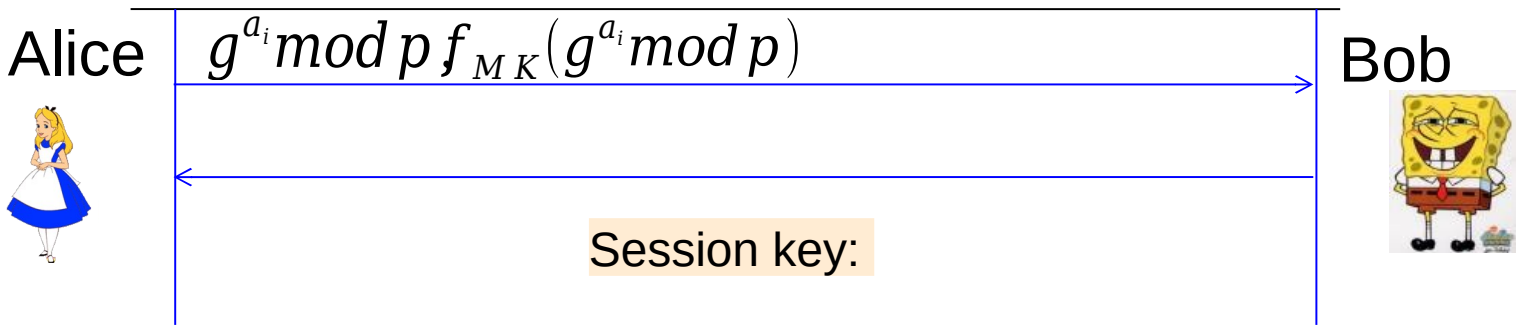
- With CDH, adversary may be able to compute some *partial* information about $g^{ba} \bmod p$ …
  - But 'most bits are random'
- Solution: **Key Derivation Function (KDF)**
  - Two variants: random-keyed and unkeyed (deterministic)
- Randomized - KDF: where KDF is a key derivation function and is public random ('salt')
- Deterministic - crypto-hash: where h is randomness-extracting crypto-hash
  - No need in salt, but **not** provably-secure
- Question: isn't (every) PRF a KDF? [not that easy]
- Note: definition of KDF isn't trivial

Authenticated DH

- Recall: DH not secure against MitM attacker
  - We assumed authenticated channel [shared key?]
  - If we have shared key, why not just use it??
- Use DH for **resiliency to key exposure**
  - Do authenticated DH periodically
  - Use derived key for confidentiality, authentication
    - Some protocols use key to authenticate next exchange
  - Perfect Forward Secrecy (PFS):
    - Confidentiality of session is resilient to exposure of all keys, except -th session key, after session ended

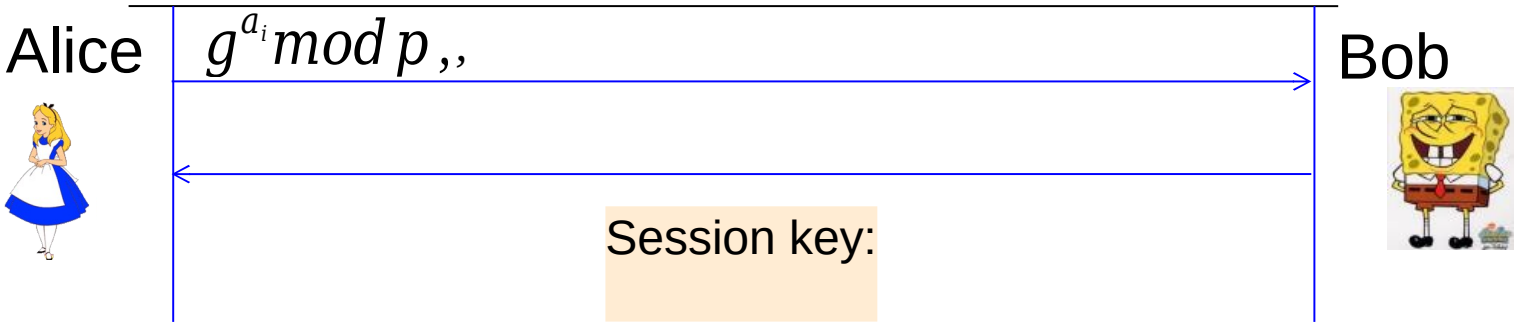
Authenticated DH: using KDF/PRF [TLS]

- Assume which is <u>both</u> a PRF and a KDF
- is secret + is PRF authentication
  - And, as long as MK is secret, session keys are secure – even if disc-log would be easy (quantum computers or math break-thru)
- Assuming CDH: secure if flows are authentic
  - Even if MK is exposed, since: (1) MK is random, (2) is KDF, and (3) most bits of are secret.
  - Authentication: eavesdropping adv. OR secret MK OR exposure only after key exchange complete (PFS).



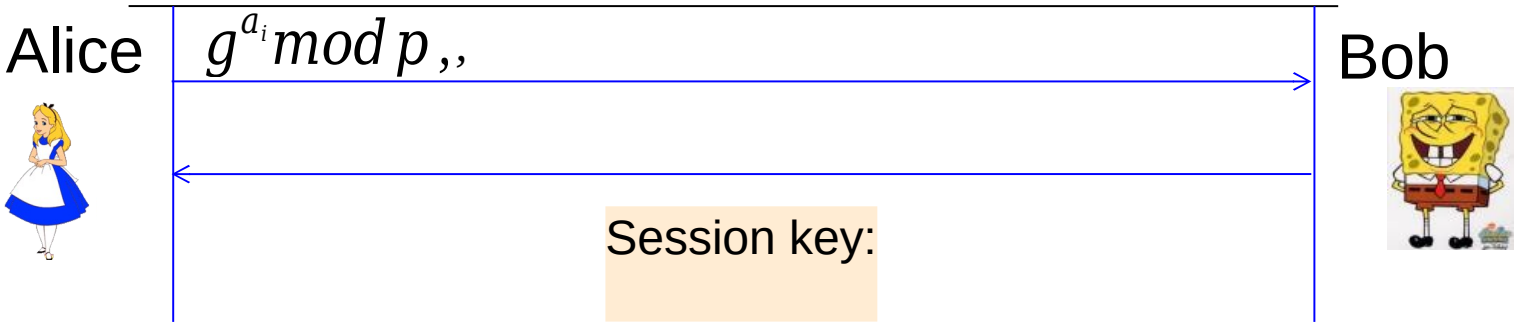
Exercise: what about this variant?

- Use two functions, :



Exercise: what about this variant?

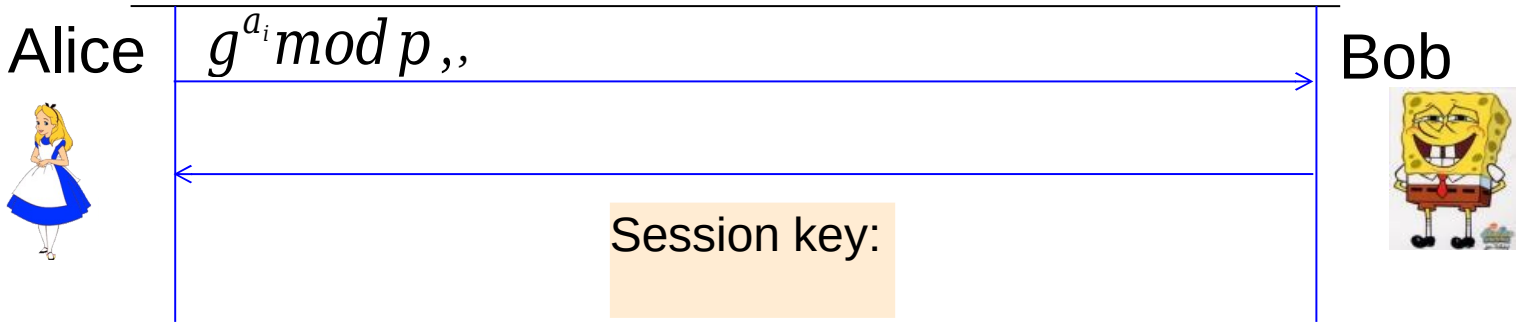
- Use two functions, :



- Not secure !!
- Why?
- How to fix?

Exercise: what about this variant?

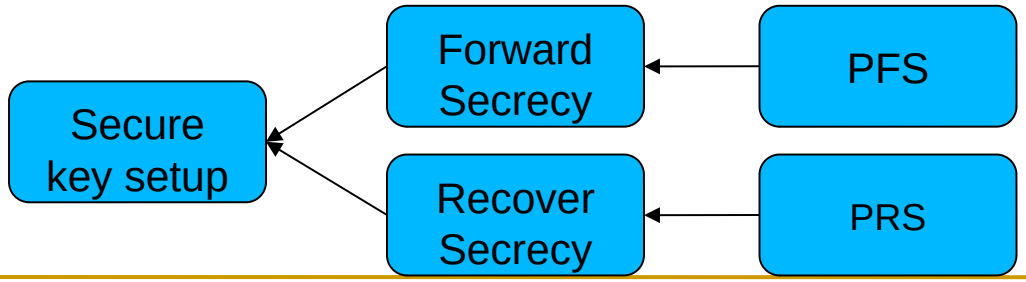
- Two master keys and functions, :



- Secure
- Authentication is secure if either is a MAC
- Key is pseudorandom if either:
  - are secret, random and is PRF, or
  - are random and / is KDF and DH is hard

Resiliency Notions: Shared + P

Notion	Session is secure, when:	Crypto
Secure key-setup	session key is exposed Not resilient to exposure of master key!	Shared key
Forward secrecy	Expose all keys kept <u>after</u> session ended	Shared key
Perfect Forward Secrecy (PFS)	Expose all keys <u>before and after</u> session , and (only) eavesdropping during session	Public key
Recover Secrecy	Expose all , except in sessions (for some) **no eavesdropping during session**	Shared key
Perfect Recover Secrecy (PRS)	Expose all keys, except in sessions (for some) and (only) eavesdropping during session '.	Public key



Auth-DH's Exposure-Resiliency

Notion	Session is secure, when:	Auth-DH
Secure key-setup	session key is exposed Not resilient to exposure of master key!	
Forward secrecy	Expose all keys kept <u>after</u> session ended	
Perfect Forward Secrecy (PFS)	Expose all keys <u>before and after</u> session , and (only) eavesdropping during session	
Recover Secrecy	Expose all , except in sessions (for some) **no eavesdropping during session**	No! why?
Perfect Recover Secrecy (PRS)	Expose all keys, except in sessions (for some) and (only) eavesdropping during session '.	

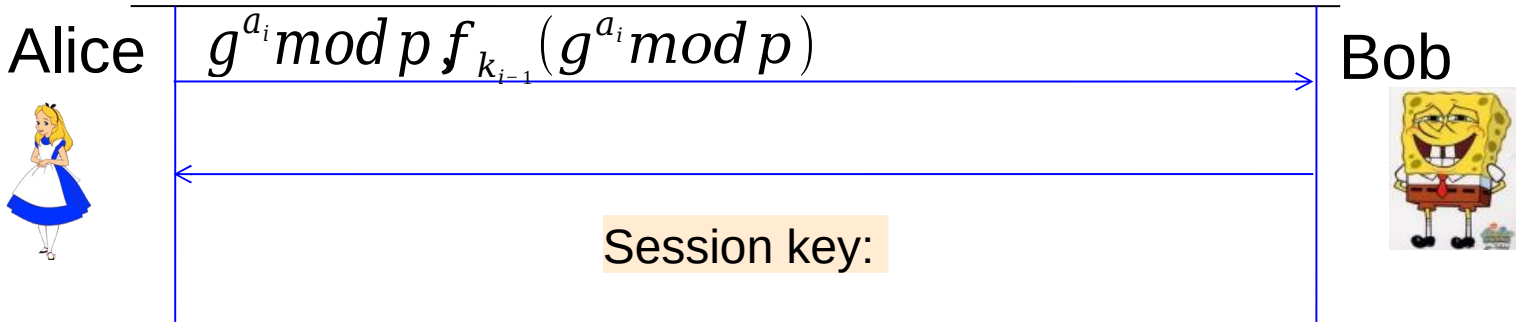
Exposing master key makes <u>all future session vulnerable</u> (to MitM)

Public Key Cryptology,
Part I: Intro and Key Exchange

- Introduction to Public Key Cryptology
- The Key Exchange problem
  - ‘Toy protocols’
  - The DL assumption
  - The DH protocol and CDH/DDH assumptions
  - Key Derivation Function (KDF)
  - Auth.-DH for key-resiliency: PFS
  - **Improved resiliency – Ratchet protocols**

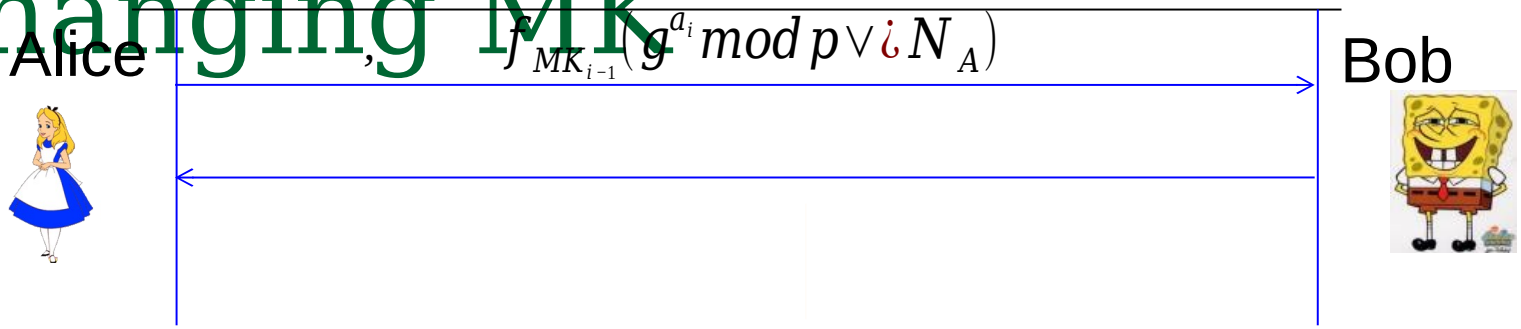
Perfect Recover Secrecy: Sync Ratchet

- Extend Auth-DH to perfect recover secrecy (PRS)
  - Idea: avoid fixed master key; use `ratchet' of keys



- PRS: keys of sessions before , exposed only after session i' ended
- During session , previous key was secret session was authenticated is secret
- Similarly: all following keys are secret

Exercise: Auth-DH with changing MK

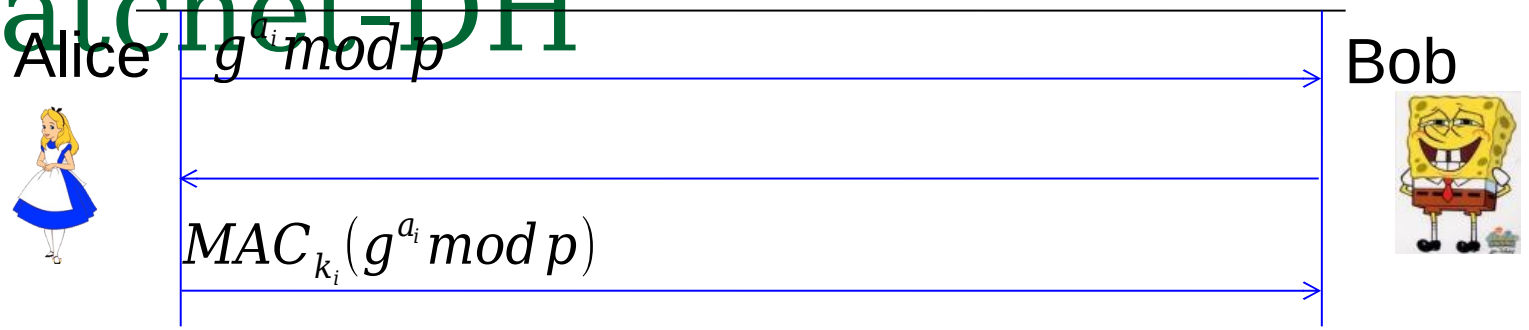


- Assume: protocol is run daily (from day 1)
- is random secret initial key, shared btw Alice and Bob
- Attacker eavesdrops on communication (all days)
- Attacker can spoof messages, be MitM on days 3, 6, 9…
- On day 5, attacker is given key
- For given day, messages of which days are exposed to attacker?
- Compare to authenticated DH and to Sync-Ratchet DH

Ratchet-DH's Exposure-Resiliency

Notion	Session is secure, when:	Ratchet-DH
Secure key-setup	session key is exposed Not resilient to exposure of master key!	
Forward secrecy	Expose all keys kept after session ended	
Perfect Forward Secrecy (PFS)	Expose all keys before and after session , and (only) eavesdropping during session	
Recover Secrecy	Expose all , except in sessions (for some) no eavesdropping/MitM during session	
Perfect Recover Secrecy (PRS)	Expose all keys, except in sessions (for some) and (only) eavesdropping during session '.	

Exercise: Variant on Sync-Ratchet-DH

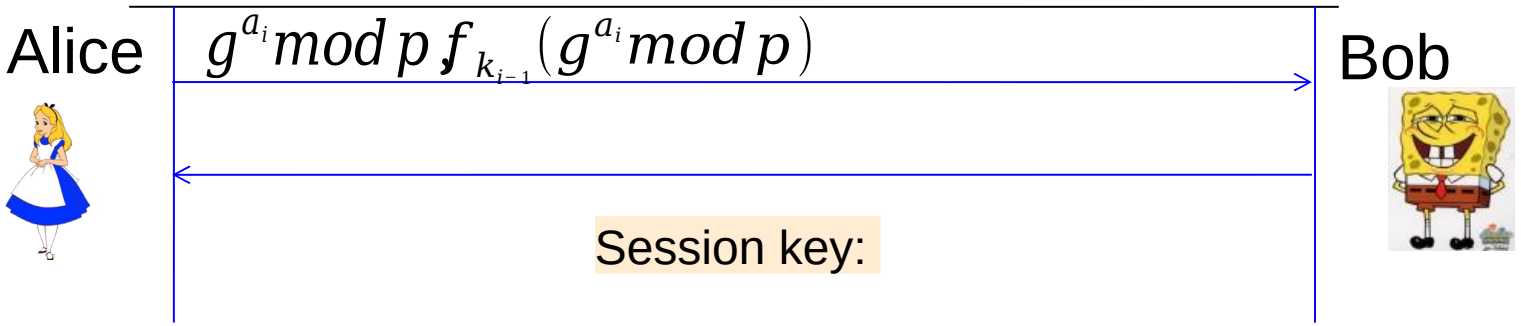


Session key:

- Secure? Present argument, and extend to also send confidential request in third flow
- Insecure? Present attack (sequence diagram)

Sync-Ratchet DH is Synchronous

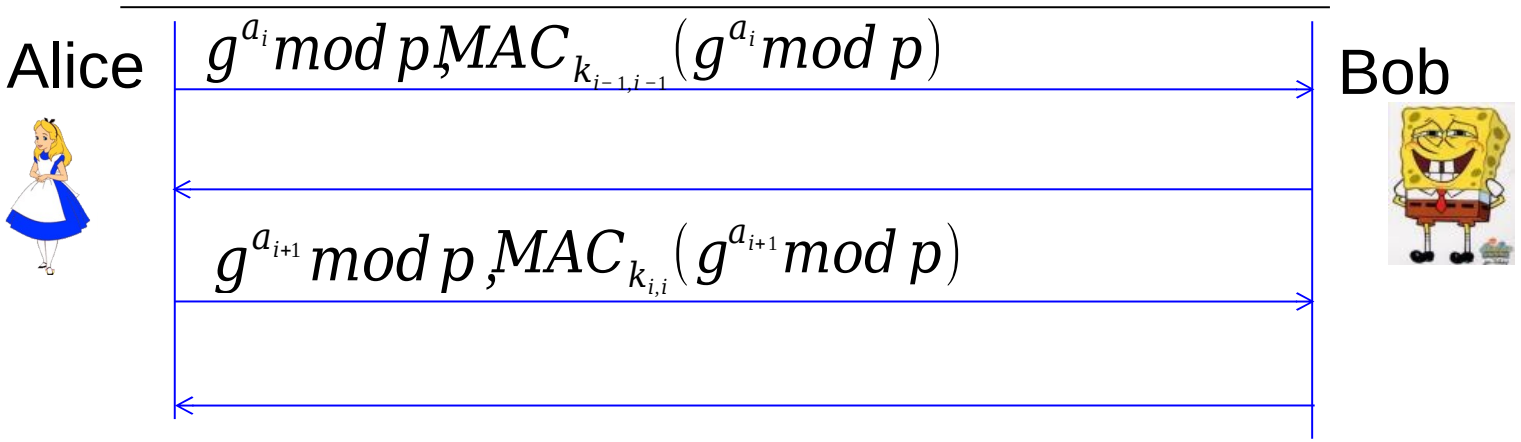
- Recall Sync-Ratchet DH
  - Avoid fixed master key; use `ratchet' of keys



- Drawback: Synchronous
  - Wait for response to refresh key
  - Could be long wait, e.g., if Bob is offline (think messaging app)
- Can we do an asynchronous variant?

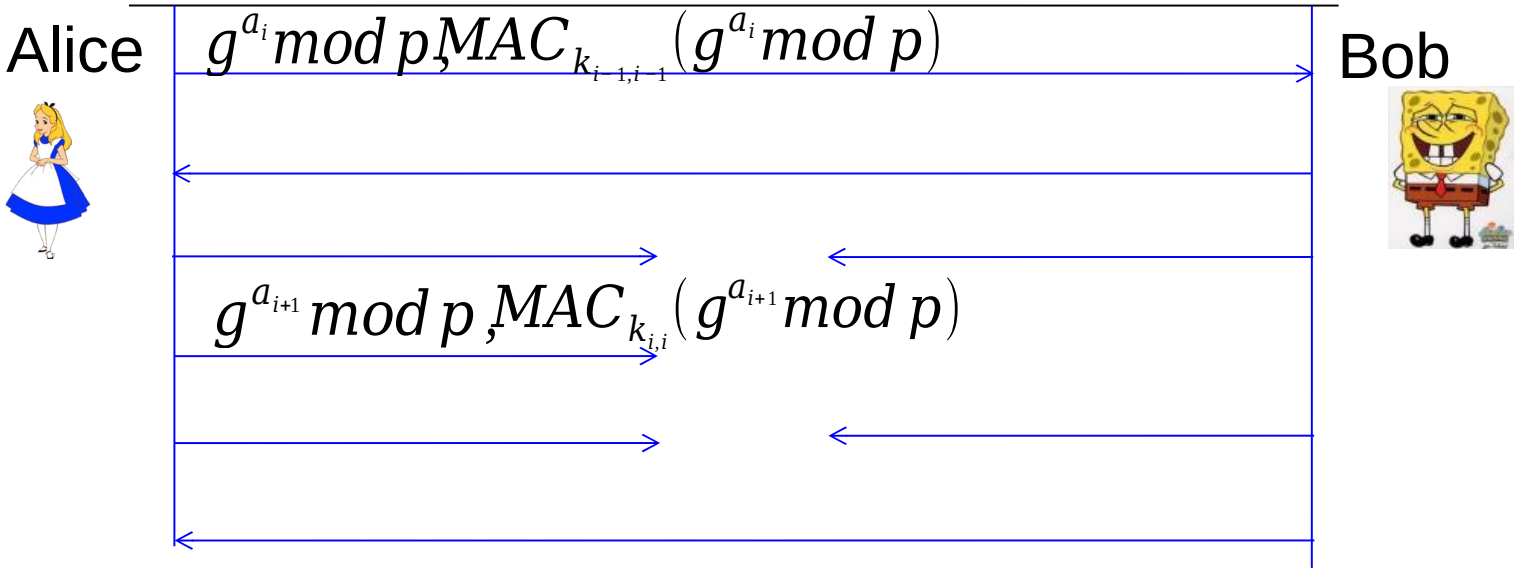
Async DH Ratchet

- Can Alice change key with Bob offline?
  - Not with ‘regular’ DH : keys are synchronized
  - But a small twist allows this: use previous !
  - Async DH Ratchet keys:



Async DH Ratchet: authenticating msgs

- Async DH Ratchet key:
- Always use most recent public key from peer
- With most recent public key sent to peer



‘Double Ratchet’: Further Resiliency!

- Idea: more frequent key changes more resiliency
  - sender key exposes only messages sent since last change
- How? Combine DH and PRF/PRG ratchets!
  - Use the key from DH-ratchet to initialize the PRG/PRF ratchet:
    ,
  - Periodically or even every message, use new PRG ratchet key: (and then erase old key)
- Derive per-goal shared keys [principle of key separation]:
  - For authentication:
  - For encryption:

Used in WhatsApp, Signal, Viber, Telegram…
to **derive, refresh end-to-end keys**

But: usually not used securely… See [HL16] and usability lecture

Summary: PKC part I: Intro and DH

- Public Key Cryptology:
  - Powerful, useful functionalities:
    - Everyone encrypts with public key, only I can decrypt
    - Everyone can verify digital signatures
    - Establish shared secret key using authenticated channel – allows perfect forward and recover security
  - But: considerable computational costs
- Next: Public Key Cryptosystem (Encryption)
  - We'll begin by turning DH into PKC – easily!