CS459/698 Privacy, Cryptography, Network and Data Security

Authentication Protocols

Fall 2024, Tuesday/Thursday 02:30pm-03:50pm

Today’s Lecture – Authentication Protocols

- Symmetric Authentication
  - Needham-Schroeder
  - Kerberos
- Asymmetric Authentication (PKI)
  - DH
  - Certificates
- PAKEs
- Single Sign On
  - SAML
  - OAuth
- DNSSEC

Recall, Definition of Authentication

Authentication = { **Identification** + Verification }

Recall, Types of Authentication Tokens

- Something you know
  - Passwords, pins, etc

- Something you have
  - Mobile phones (SMS), RSA tokens, etc.

- Something you are
  - Fingerprints, retinal scans, etc.

- (Experimental) Something you do
  - Keystroke metrics, typing and speech patterns, etc.
  - May be copied, so they provide less security vs Biometrics.

Today's Focus

- Establishing Keys:
  - Typically, once authenticated, we give access to some service or message
  - Goal will typically be to establish a symmetric key between parties

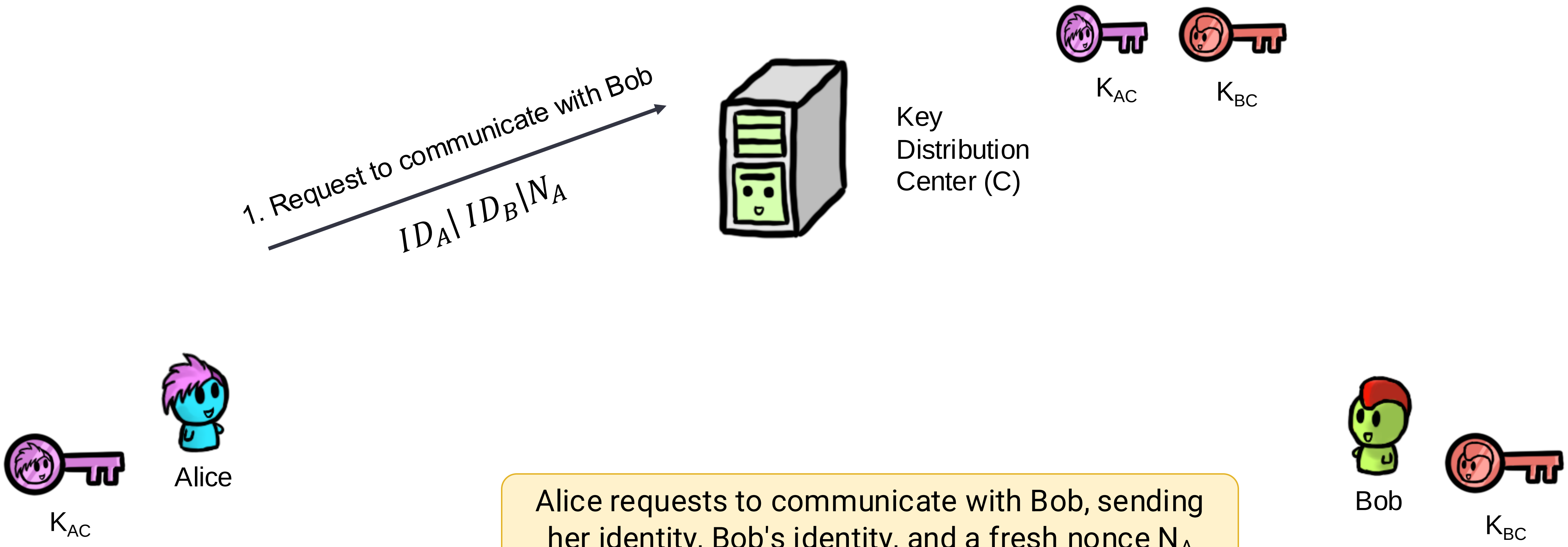
Symmetric Crypto Authentication

Needham-Schroeder

Needham-Schroeder Overview

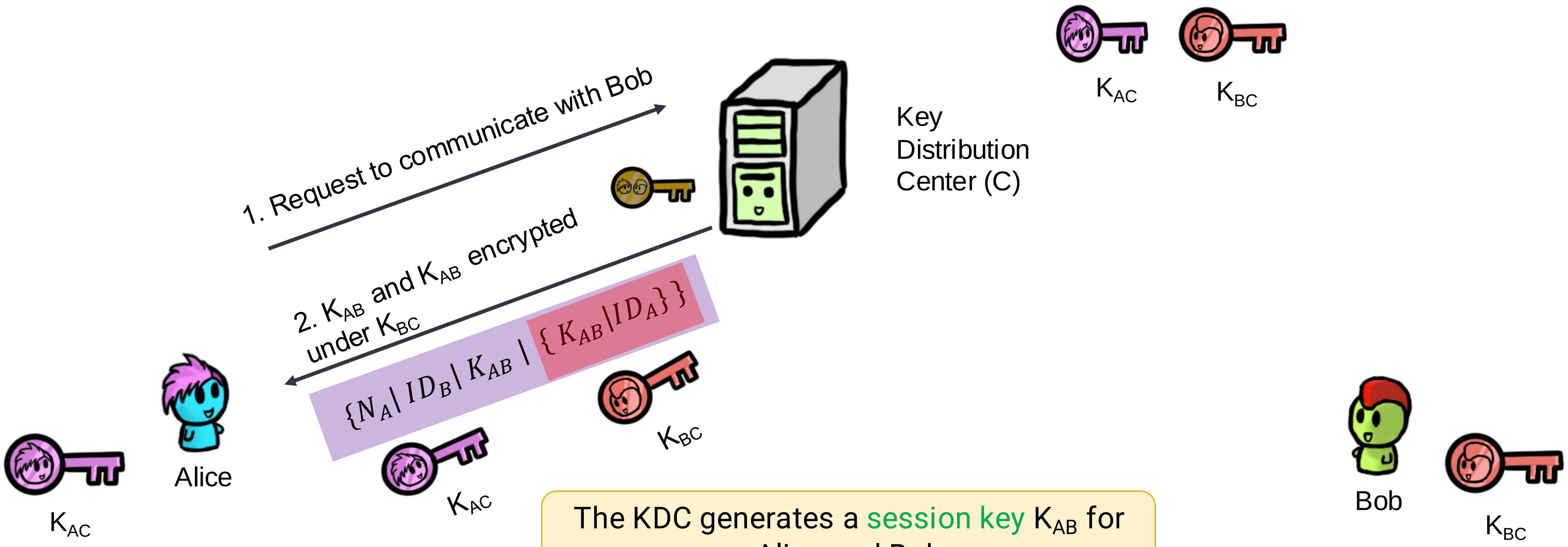
- Alice (A) wants to initiate communication with Bob (B)
- There’s a Trusted Third Party (C) with pre established symmetric keys
- K_{AC} is a symmetric key known only to A and the Key Distribution Center (C)
- K_{BC} is a symmetric key known only to B and the Key Distribution Center (C)
- The server generates K_{AB}, a symmetric key used in the session between A and B
    - Every time Alice wants to talk to Bob, a new symmetric K_{AB} key is provided

Needham-Schroeder Flow



Alice requests to communicate with Bob, sending her identity, Bob's identity, and a fresh nonce N_A

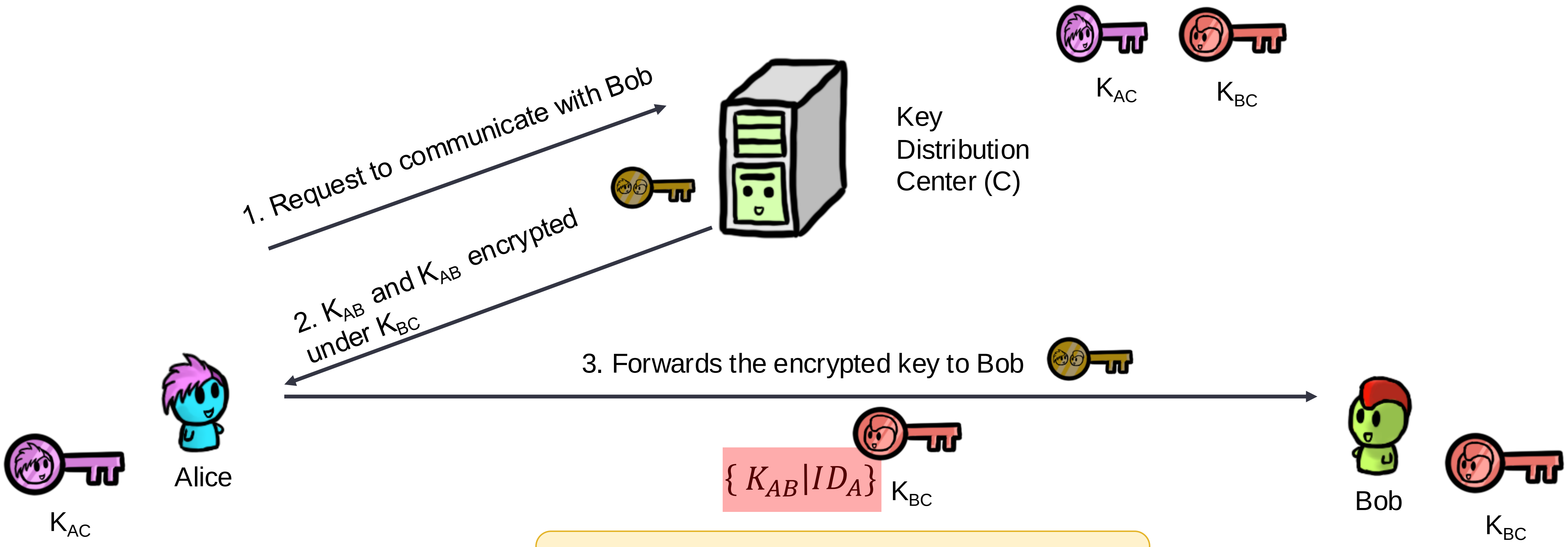
Needham-Schroeder Flow



The KDC generates a session key K_{AB} for Alice and Bob

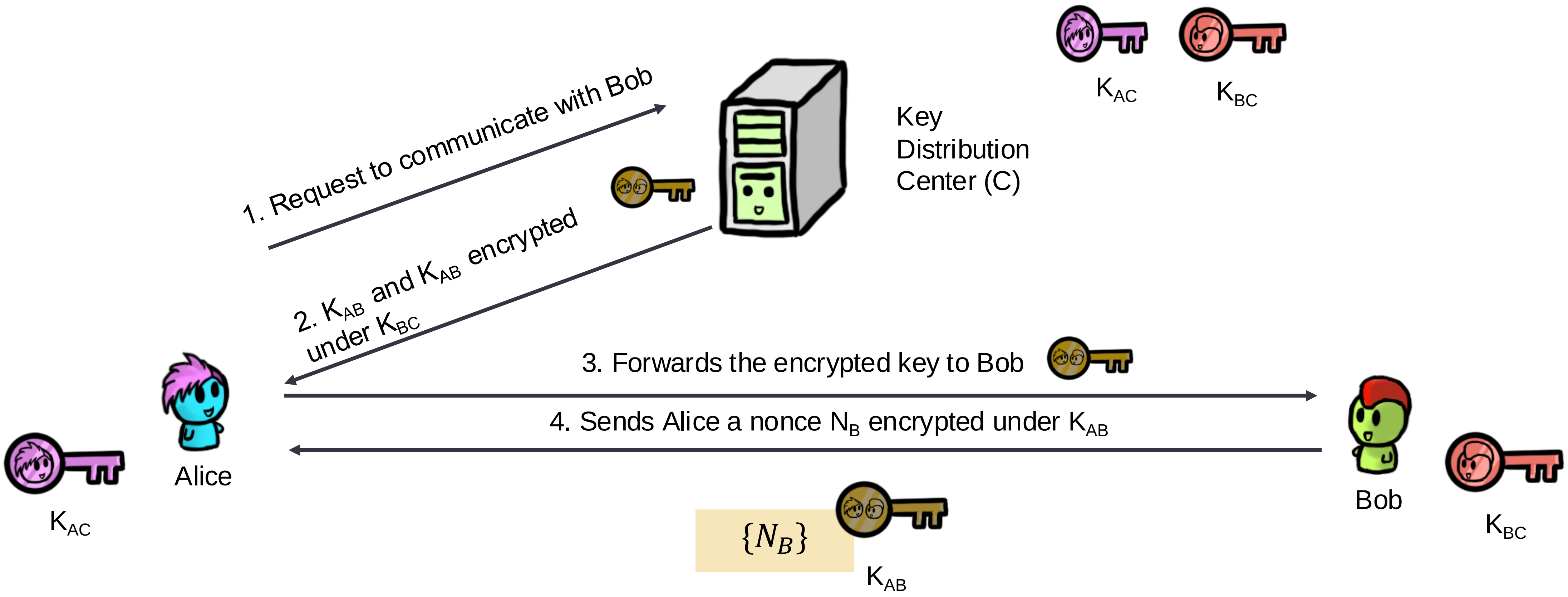
An encrypted ticket containing Alice's identity and the session key, encrypted with Bob's key K_{BC} using K_{AC}

Needham-Schroeder Flow

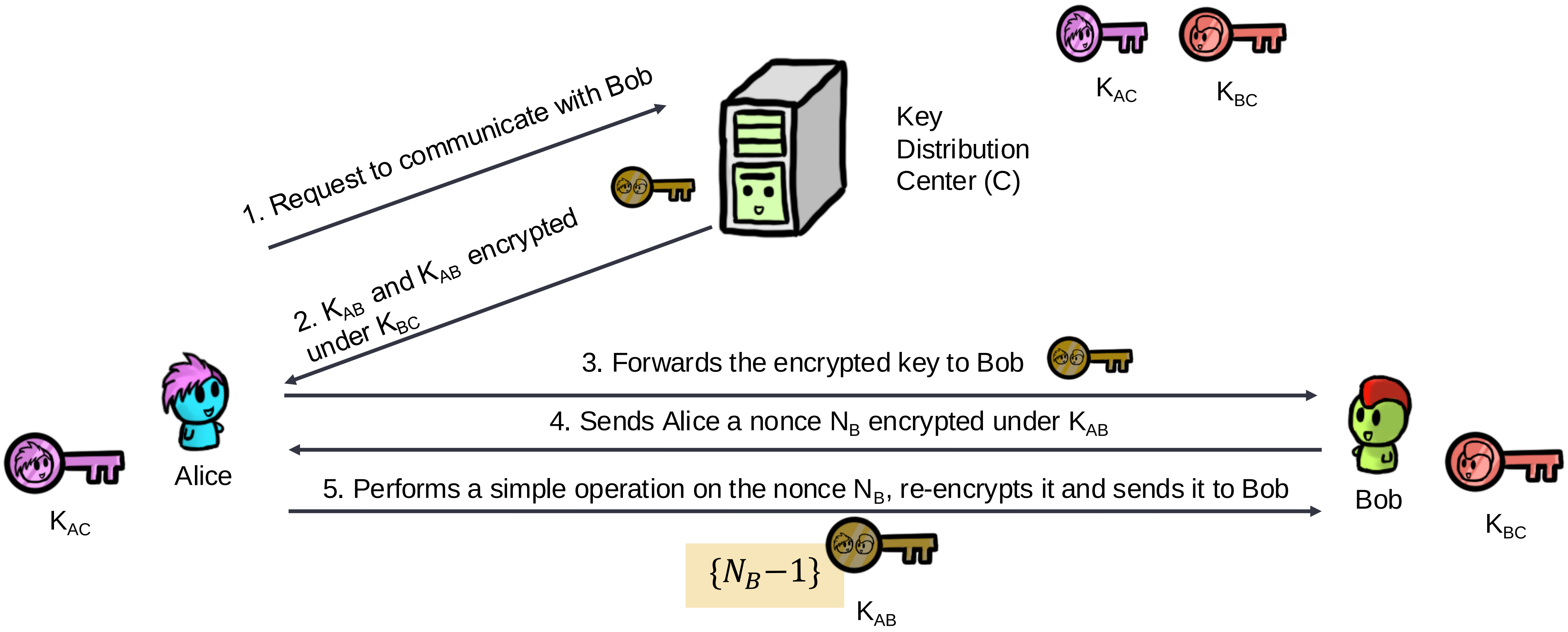


Bob decrypts the session key

Needham-Schroeder Flow



Needham-Schroeder Flow

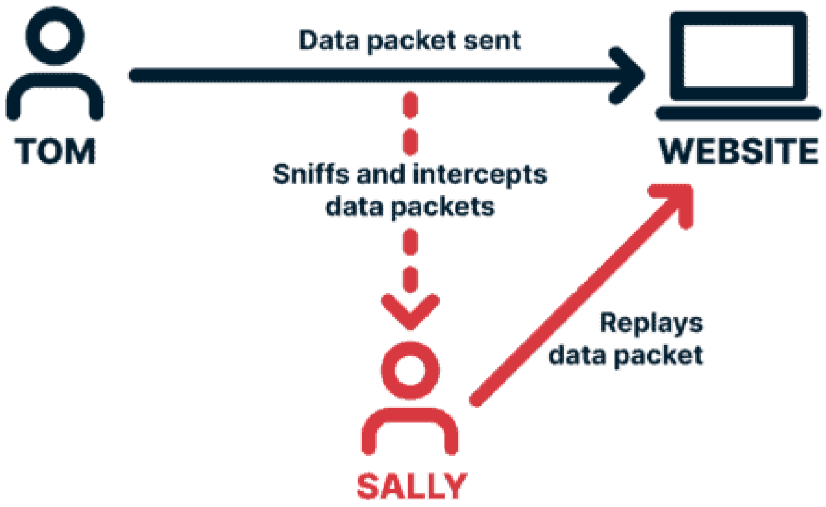


Bob verifies that Alice has the session key

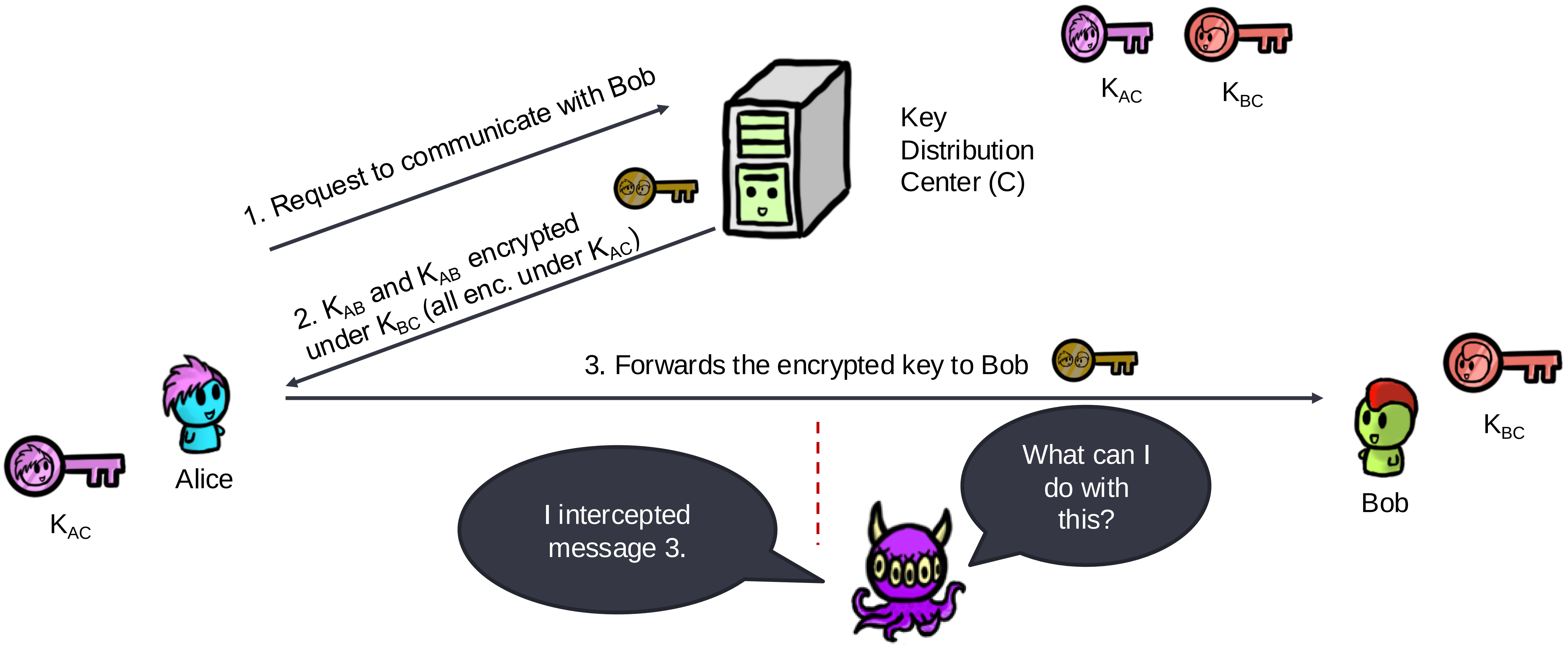
Is Needham-Schroeder Vulnerable to Replay Attacks?

- Replay attack:
  - Mallory intercepts a message meant for some other party
  - They later send this message again pretending to be some other party

- Example
  - Hashed password
  - Car unlocking



Yes, it is ☹



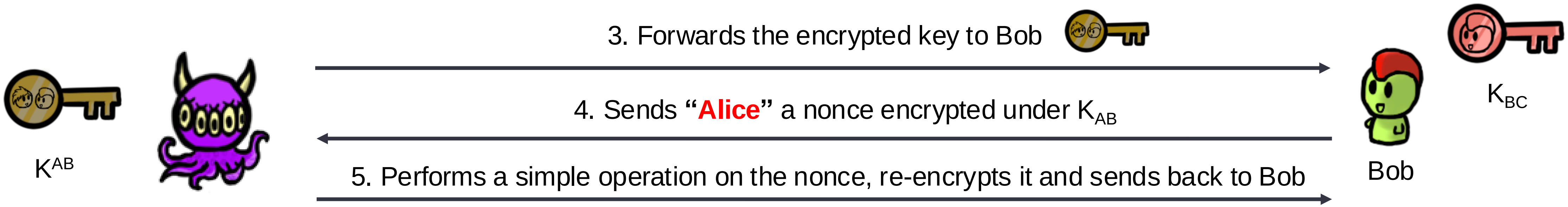
1. Request to communicate with Bob
2. K_{AB} and K_{AB} encrypted under K_{BC} (all enc. under K_{AC})
Key Distribution Center (C)
K_{AC}
K_{BC}
3. Forwards the encrypted key to Bob
K_{AC}
Alice
Bob
K_{BC}
I intercepted message 3.
What can I do with this?

Needham-Schroeder is vulnerable to replay attacks

- 3 weeks later…

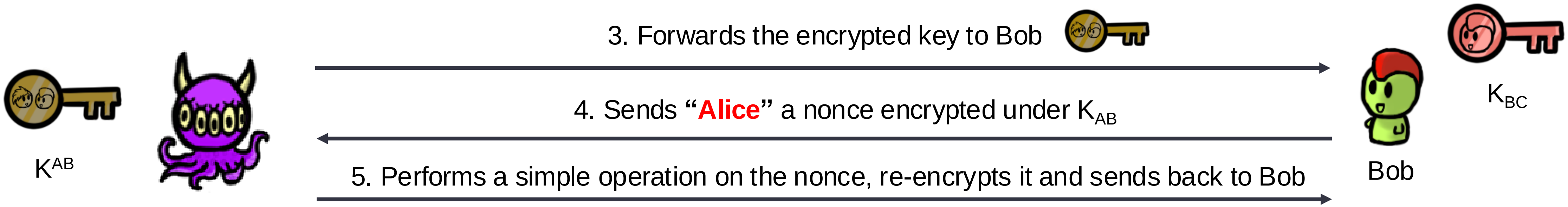
Needham-Schroeder is vulnerable to replay attacks

- 3 weeks later…



Needham-Schroeder is vulnerable to replay attacks

- 3 weeks later…



Bob will believe he is talking to Alice.

Typical Defenses against replays

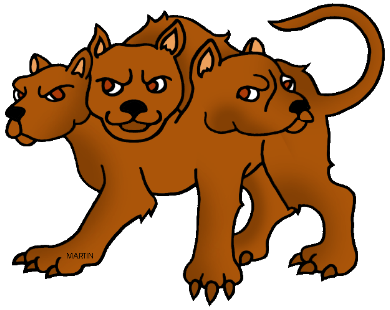
- Need to ensure the data is “fresh”

E.g.

- Using a Nonce
- Timestamps
  - Ensure Synchronization
- Caching Responses

Kerberos

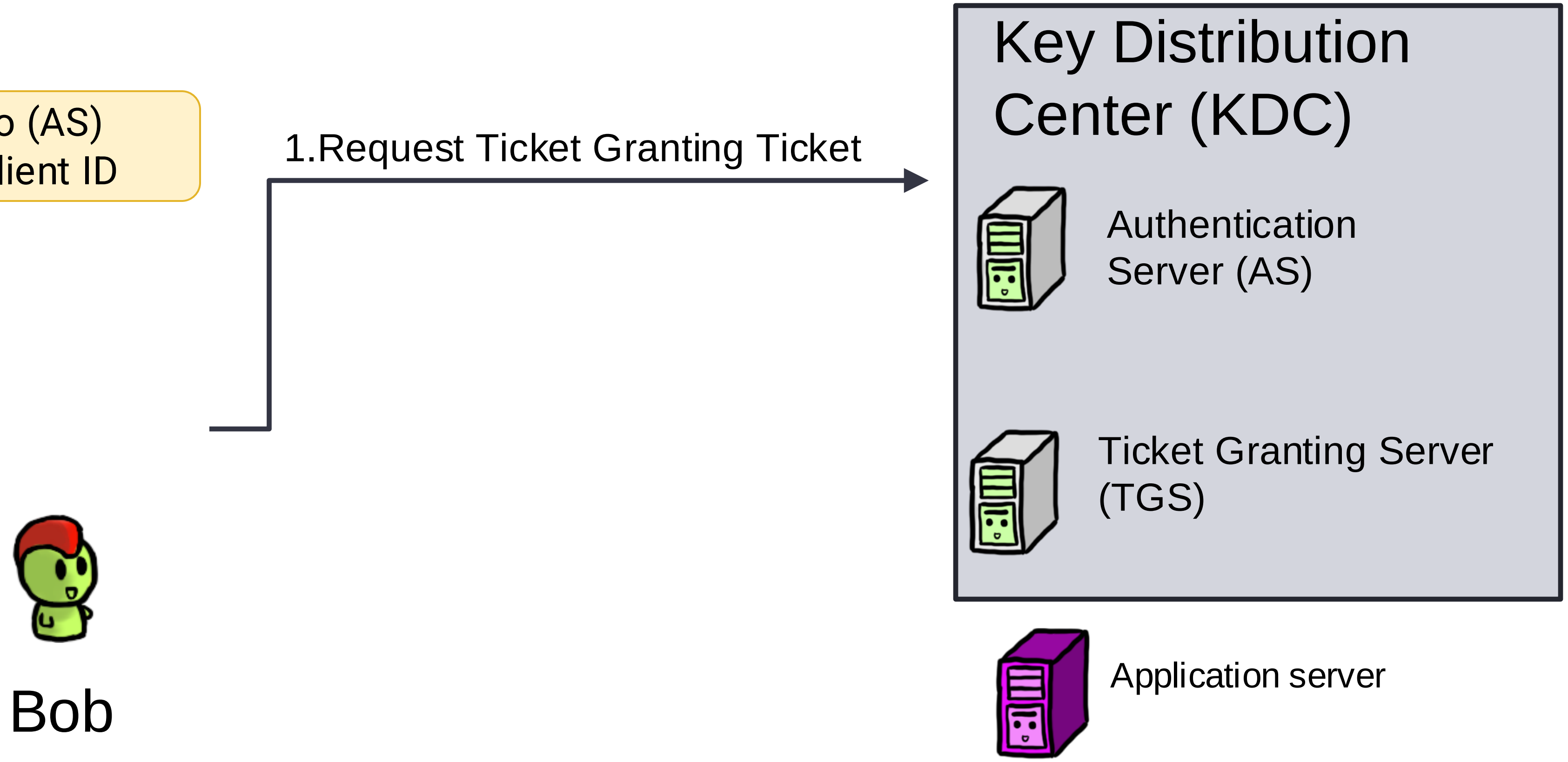
Kerberos



- Based on the Needham-Schroeder protocol
- Fixes the potential for a replay attack vulnerability by adding a timestamp
- Used in Windows Active Directory
  - Developed by MIT, and named after Cerberus Server Client KDC
- Effective Access Control
  - Each client only needs single key.
  - Each server also only needs a single key.
  - Mutual Authentication.

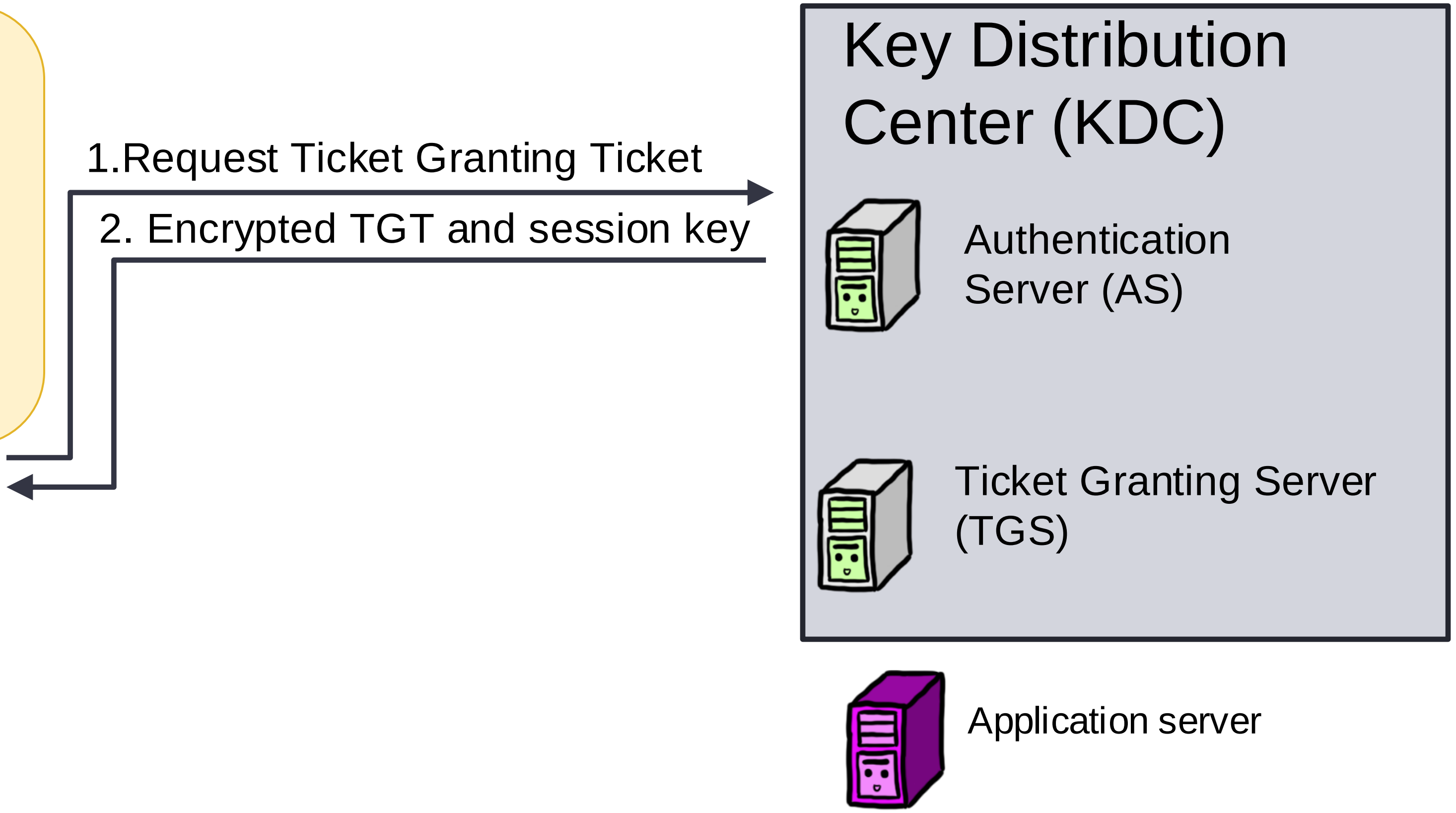
Kerberos Overview

The request to (AS) contains the client ID



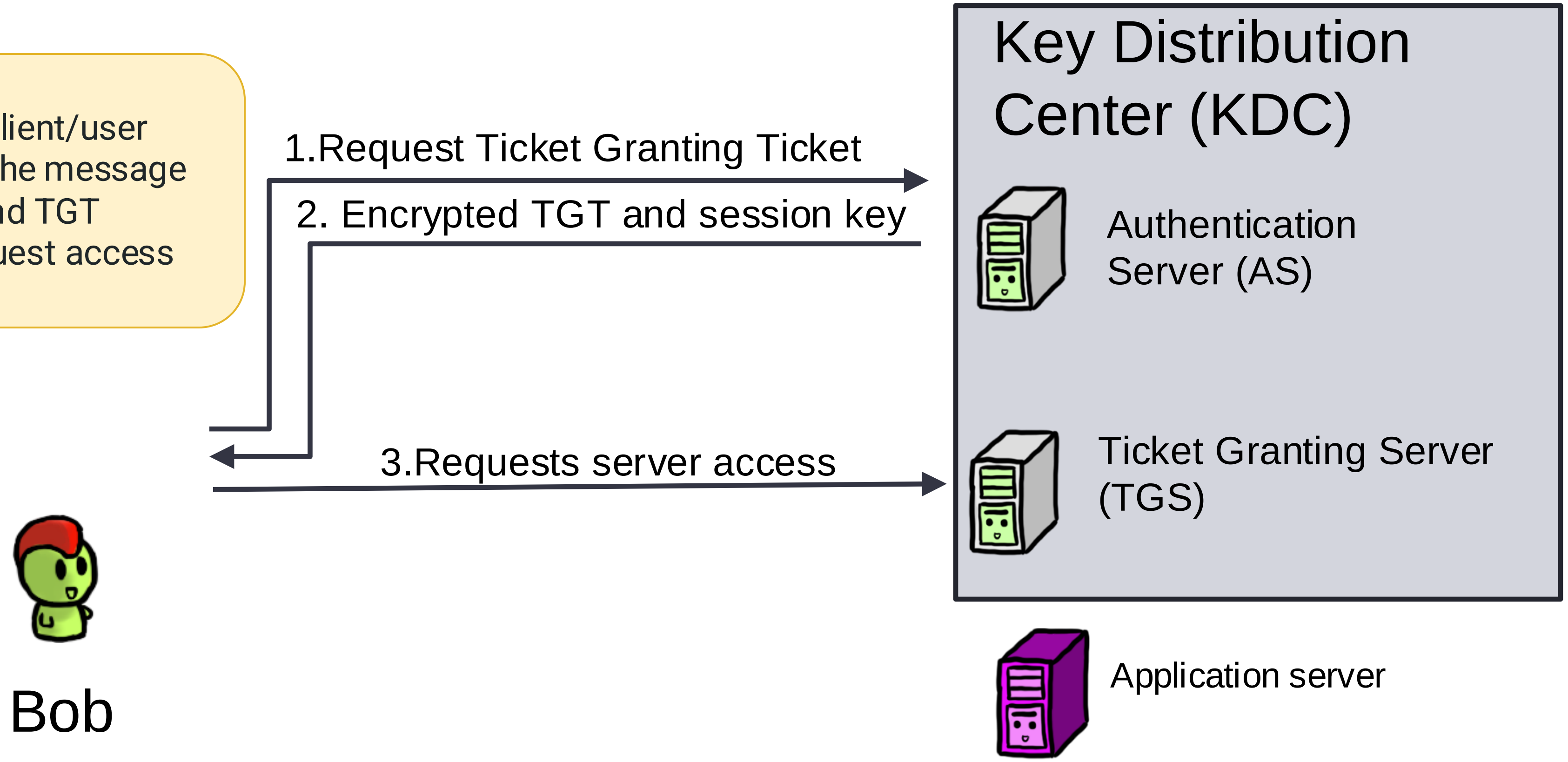
Kerberos Overview

1.KDC verifies ID.
2.(AS) checks for the client and TGT availability, then generates a client/user secret key, employing the user's password hash
3.(AS) computes the TGS secret key → ENC(session_key1)
4.(AS) generates a TGT containing: ID, timestamp, lifetime and SK1



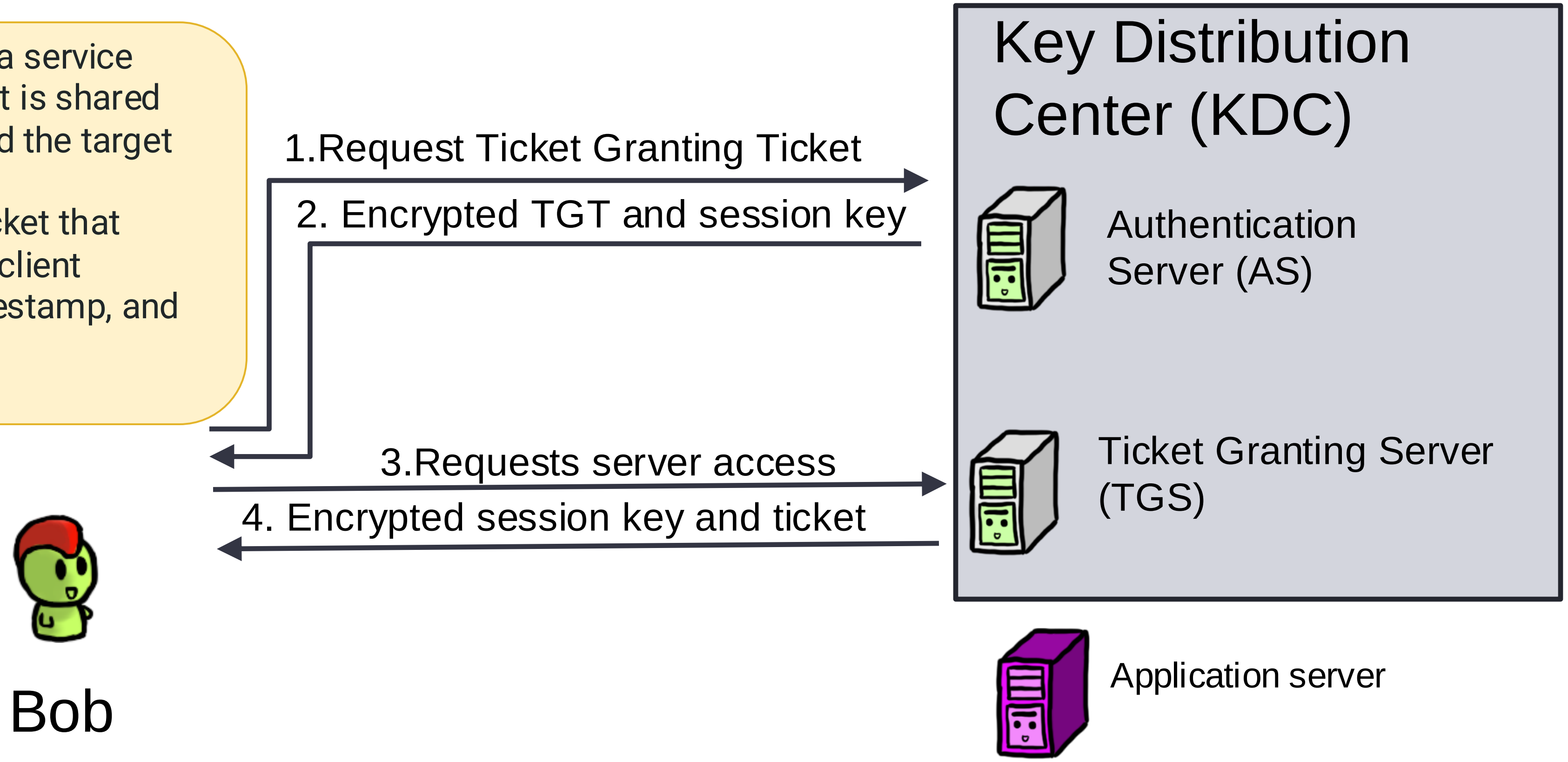
Kerberos Overview

1.The client uses the client/user secret key to decrypt the message and extract the SK1 and TGT
2.Uses the TGT to request access



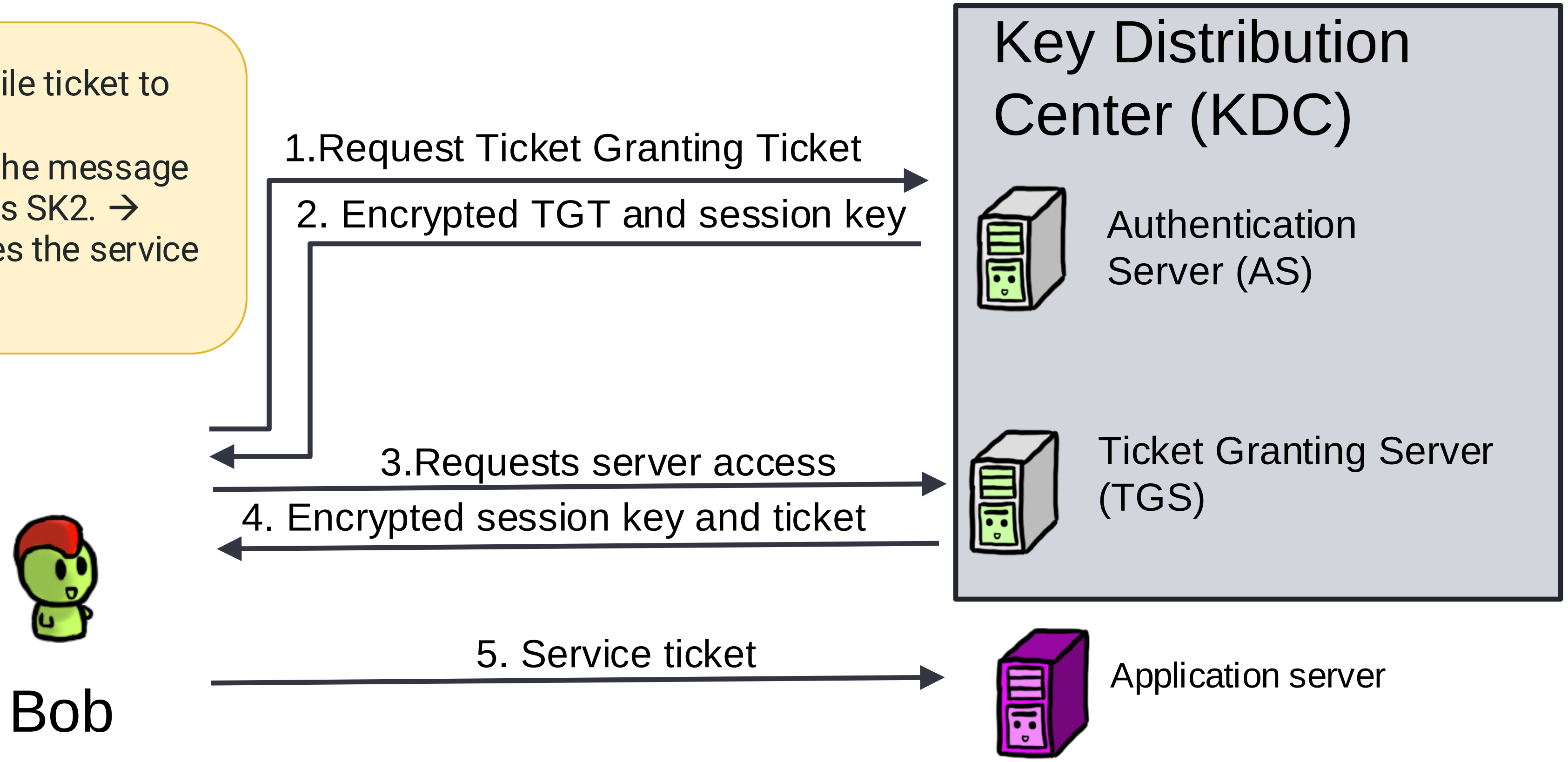
Kerberos Overview

1. The KDC generates a service session key (SK2) that is shared between the client and the target server.
2. Creates a service ticket that includes the client id, client network address, timestamp, and SK2
3. ENC_{SK1}(SK2, ticket)



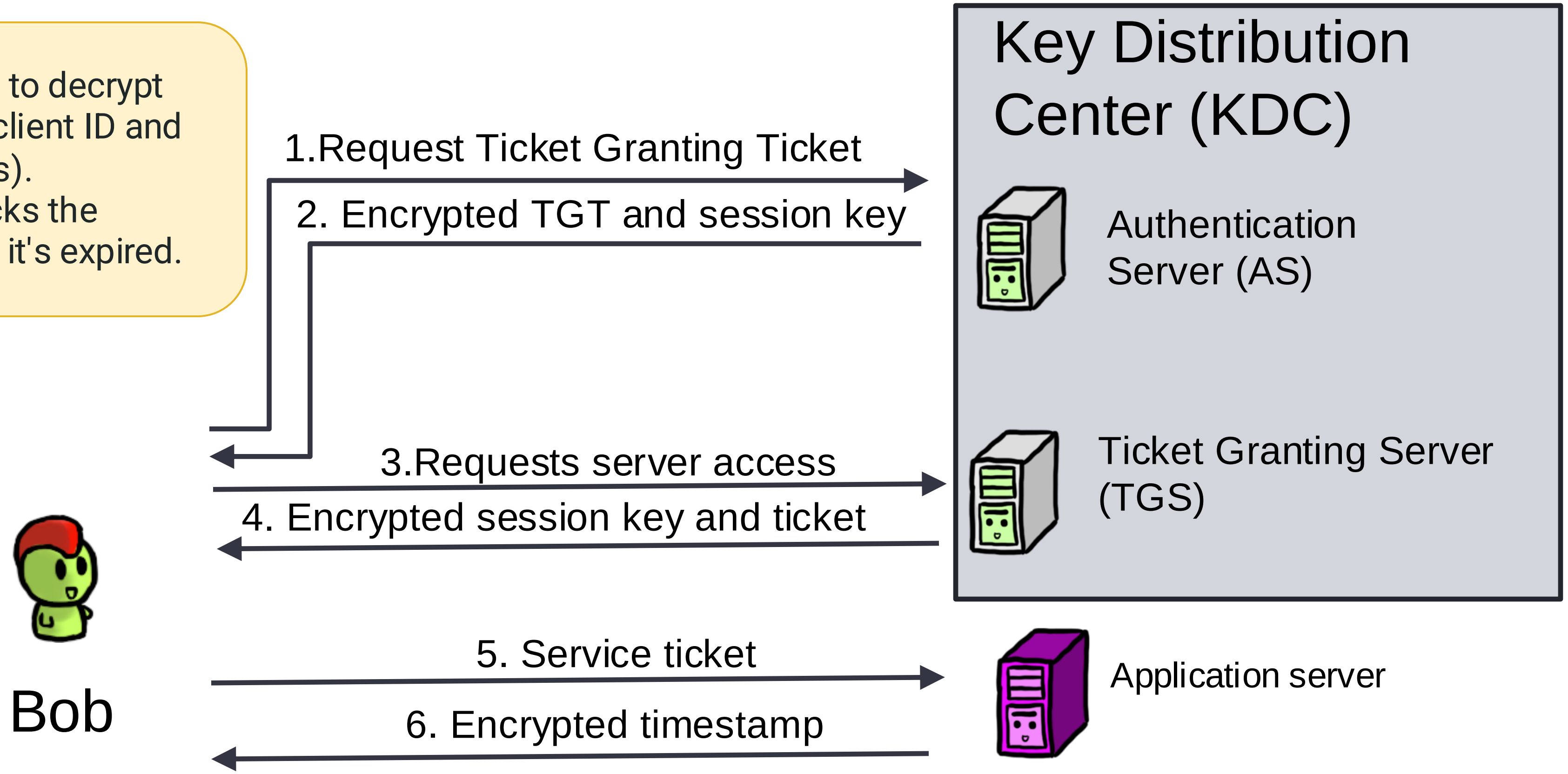
Kerberos Overview

1.The client uses the file ticket to authenticate
2.The client decrypts the message using SK1 and extracts SK2. → This process generates the service ticket.

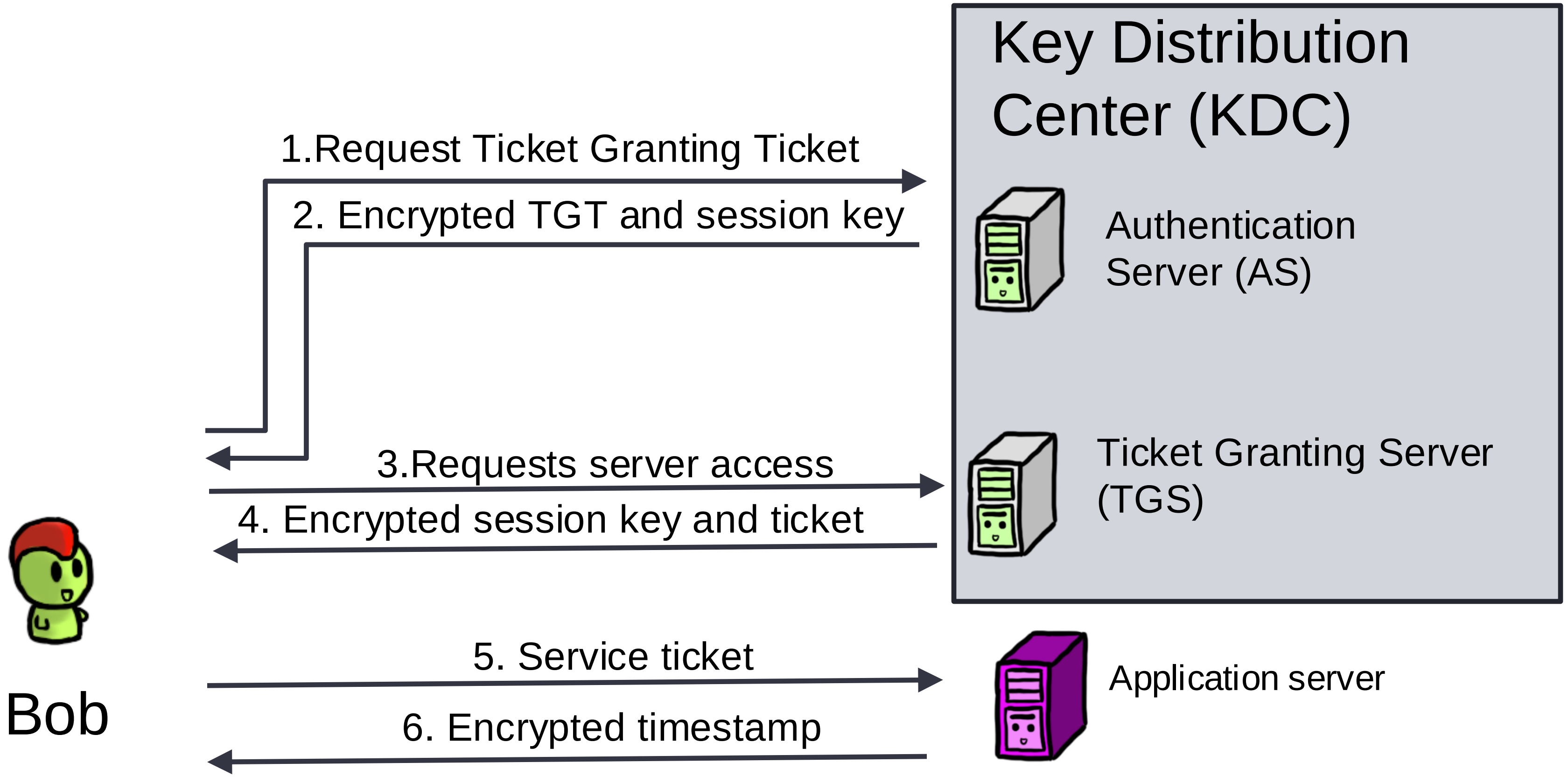


Kerberos Overview

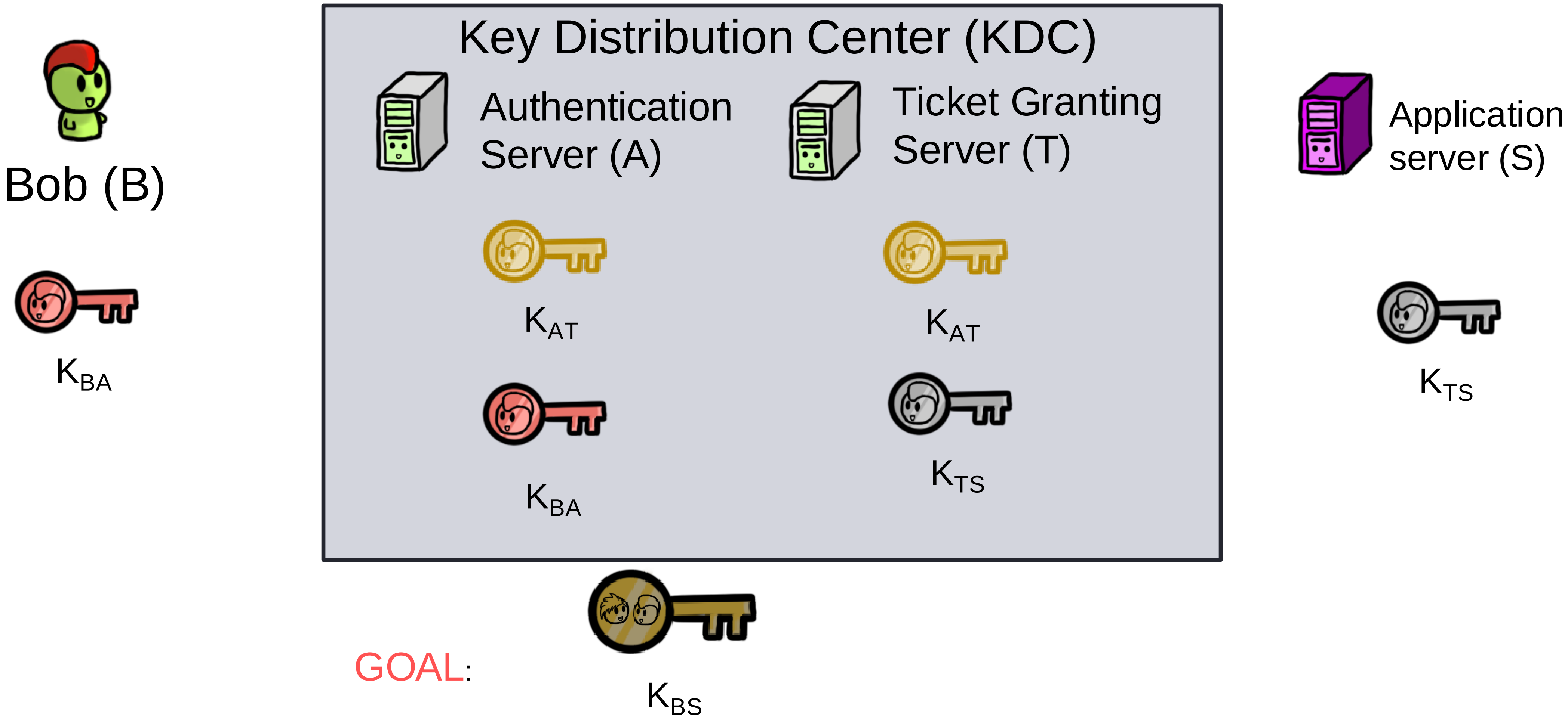
1. The server uses SK2 to decrypt and perform checks (client ID and client network address).
2. The server also checks the service ticket to see if it's expired.



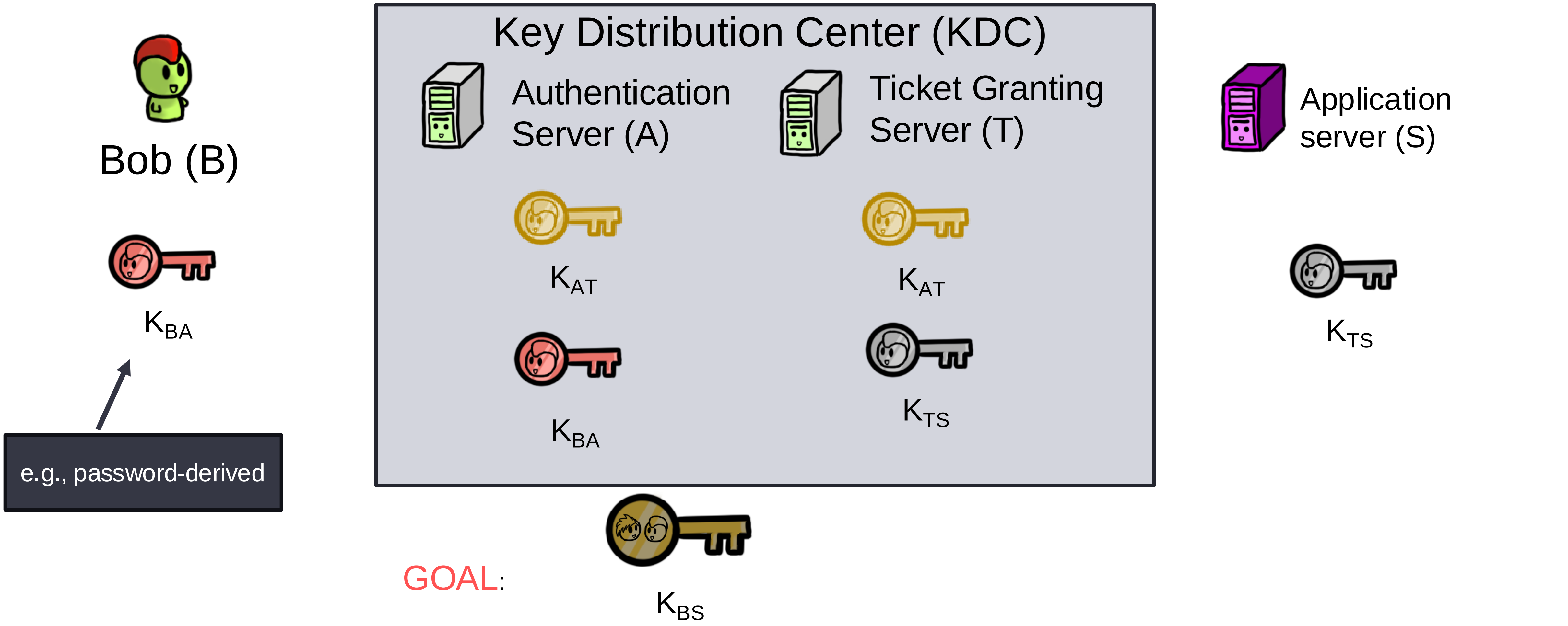
Kerberos Overview



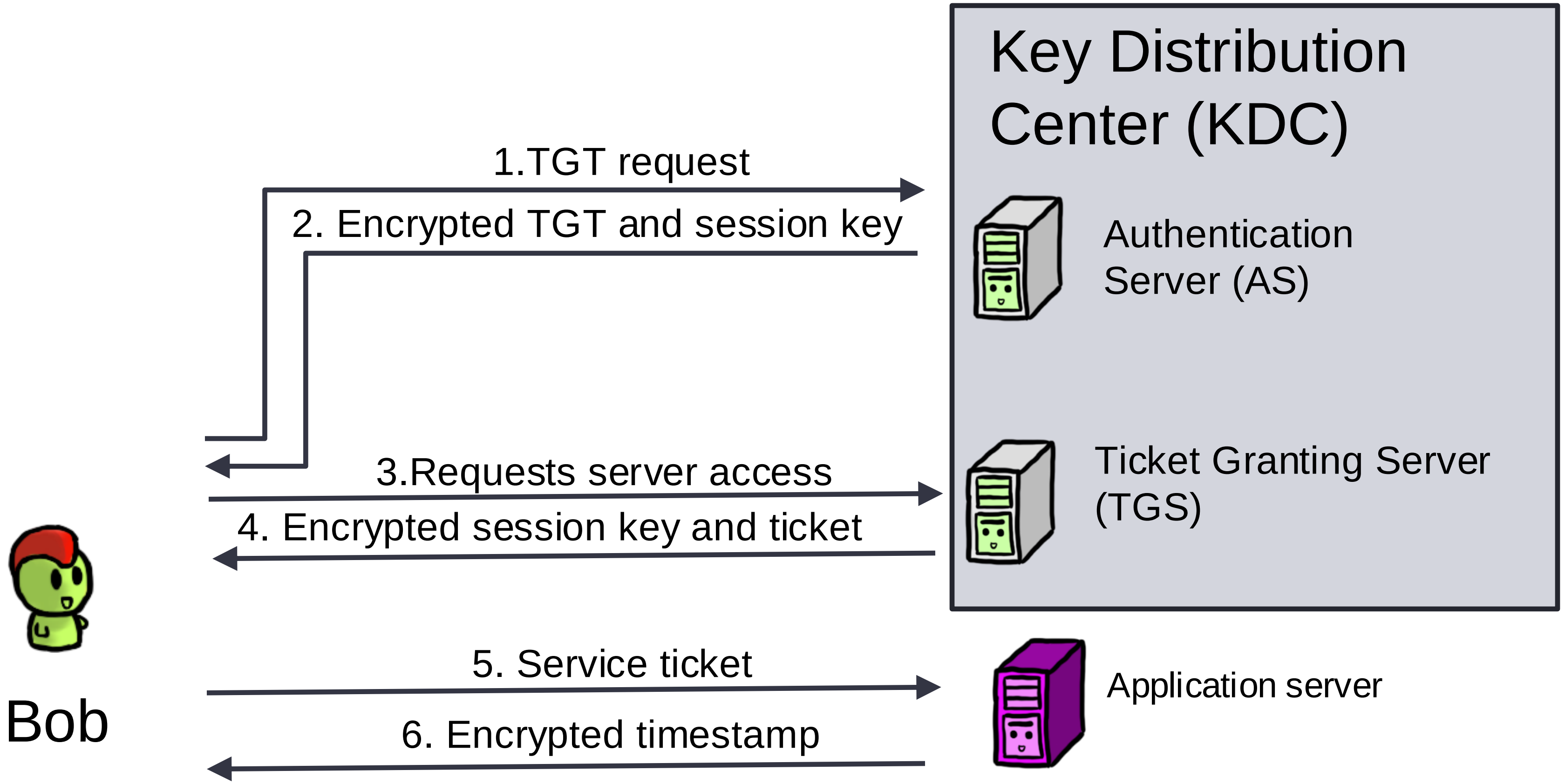
The Keys


Bob (B)
K_{BA}
Key Distribution Center (KDC)
Authentication Server (A)
K_{AT}
K_{BA}
Ticket Granting Server (T)
K_{AT}
K_{TS}
Application server (S)
K_{TS}
GOAL:
K_{BS}

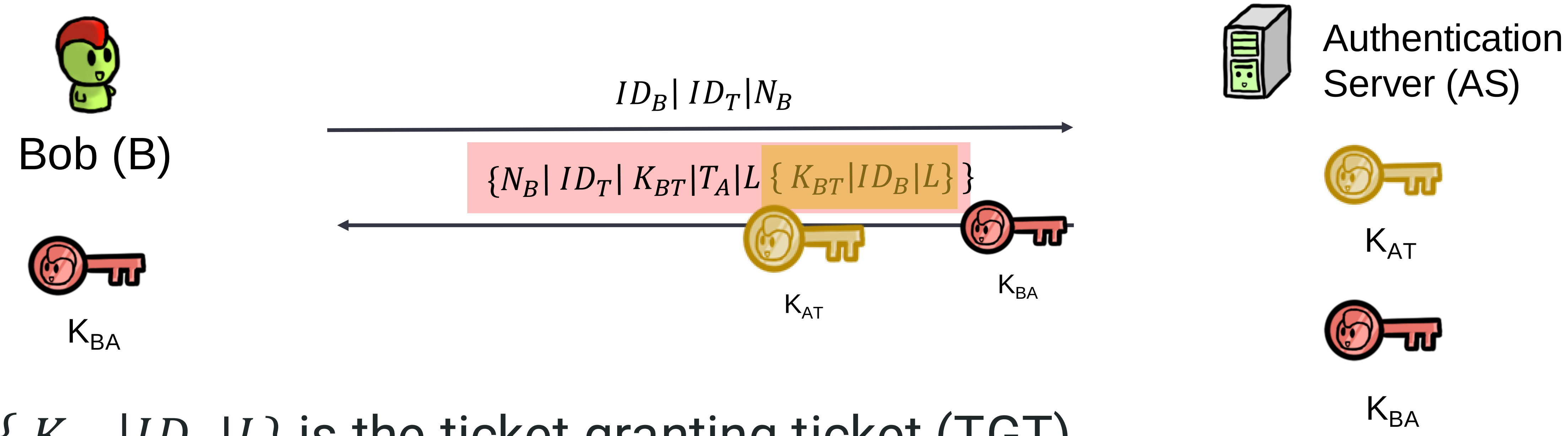
The Keys


Bob (B)
K_{BA}
e.g., password-derived
Key Distribution Center (KDC)
Authentication Server (A)
Ticket Granting Server (T)
K_{AT}
K_{BA}
K_{AT}
K_{TS}
Application server (S)
K_{TS}
GOAL:
K_{BS}

Kerberos Overview

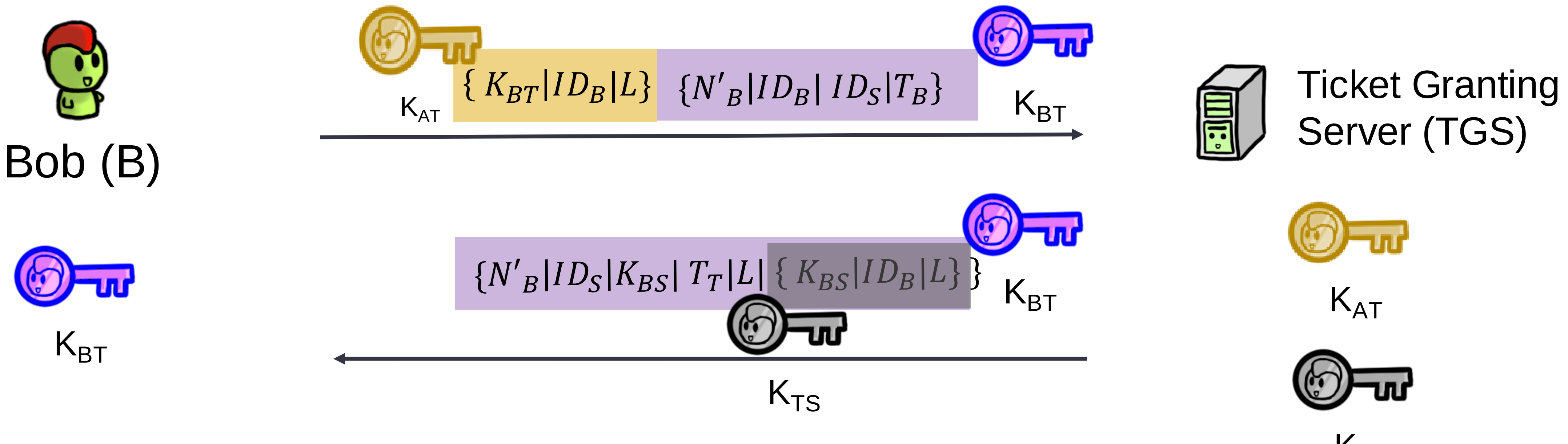


Breaking Down Kerberos – Part 1



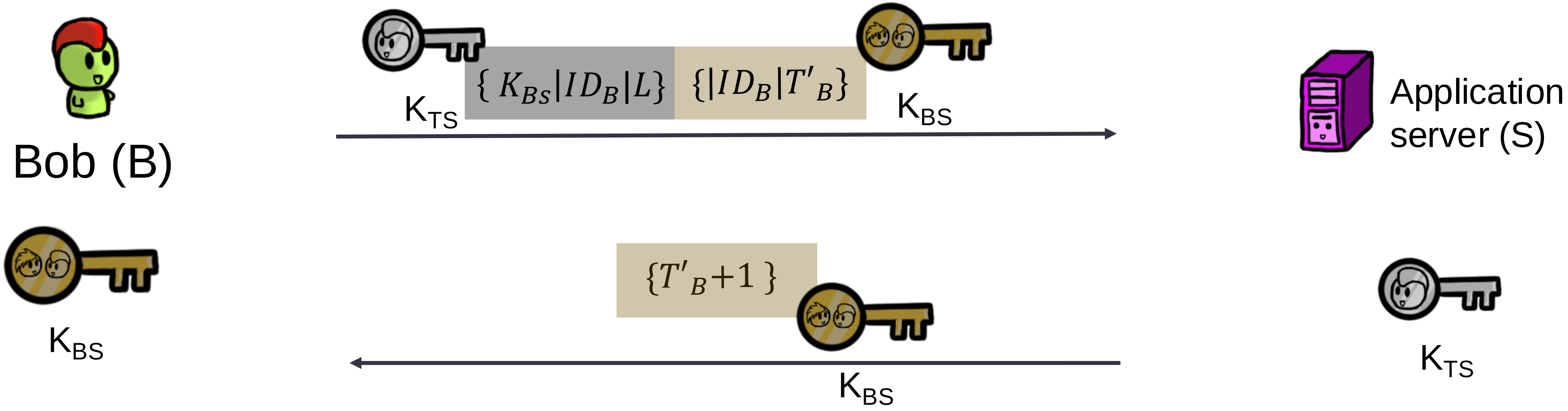
- $\{\ K_{BT}|ID_B|L\}$ is the ticket granting ticket (TGT)
- L is lifetime, T_A is the timestamp at A, N_B is a nonce
- K_{BT} is a session key between Bob and the TGS
- K_{AT} the TGS secret key

Breaking Down Kerberos – Part 2



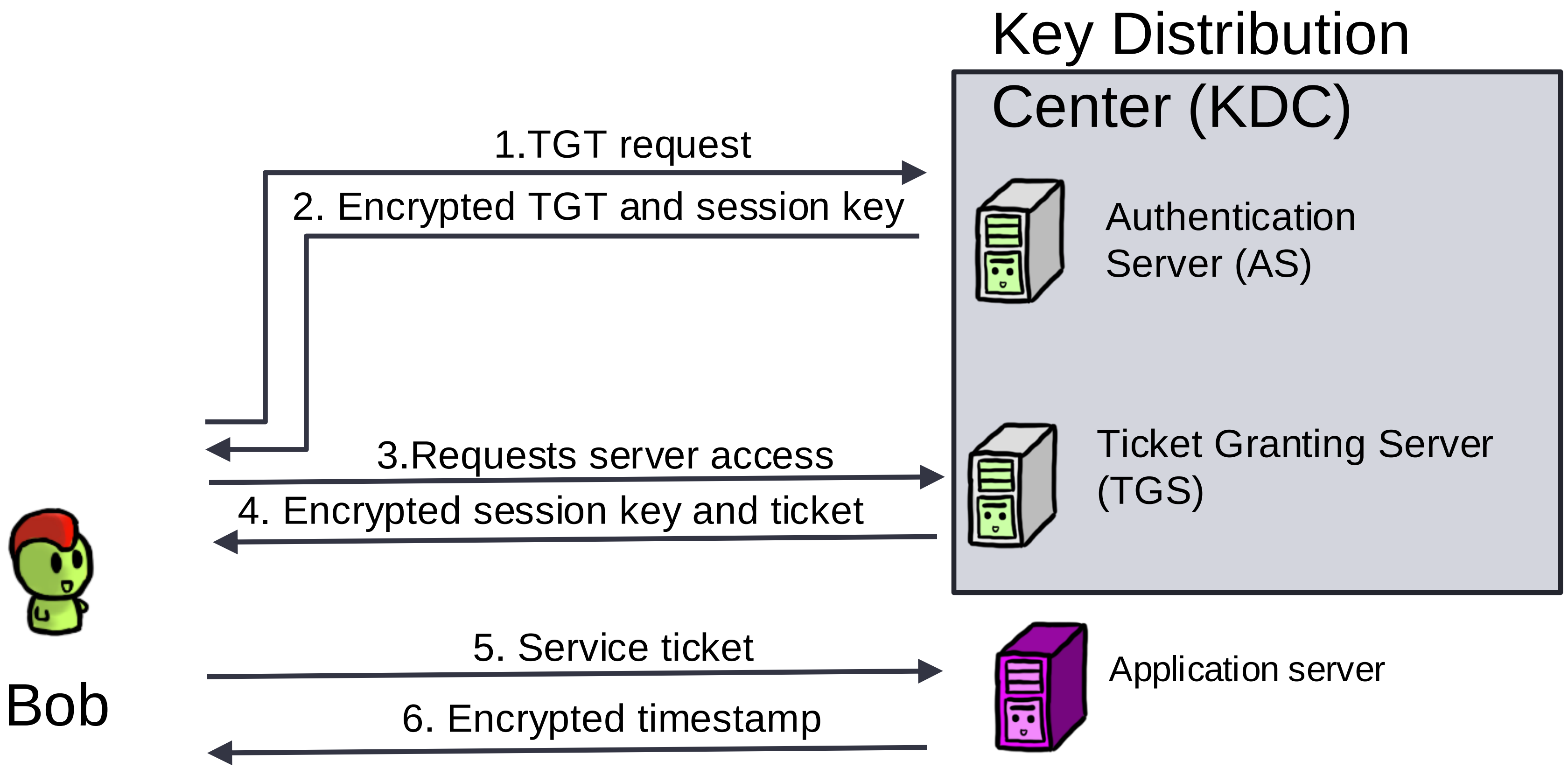
- $\{K_{BT}|ID_B|L\}$ is the ticket granting ticket (TGT)
- $\{K_{BS}|ID_B|L\}$ is the service ticket (ST)
- K_{BT} is a session key between Bob and the TGS

Breaking Down Kerberos – Part 3



- $\{ K_{BS}|ID_B|L\}$ is the service ticket (ST)
- K_{BS} is a session key between Bob and the Server

Kerberos Overview



Reflect, why does Kerberos fix it

- Timestamps in previously insecure messages
- All tickets include a Lifetime (time they expire)

Asymmetric Crypto Authentication

Recap

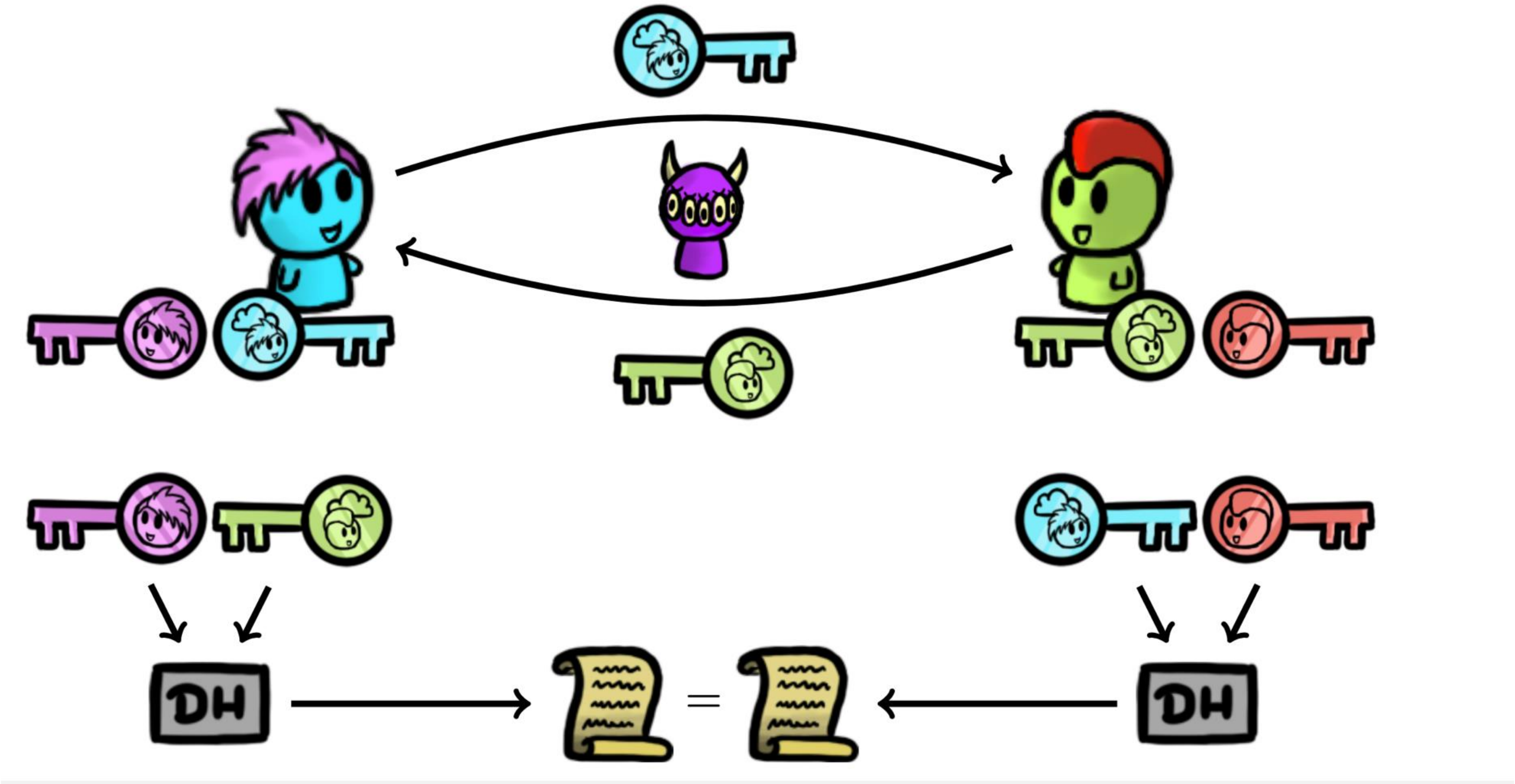
Recall the Diffie-Hellman key exchange

A public-key protocol published in 1976 by Whitfield Diffie and Martin Hellman

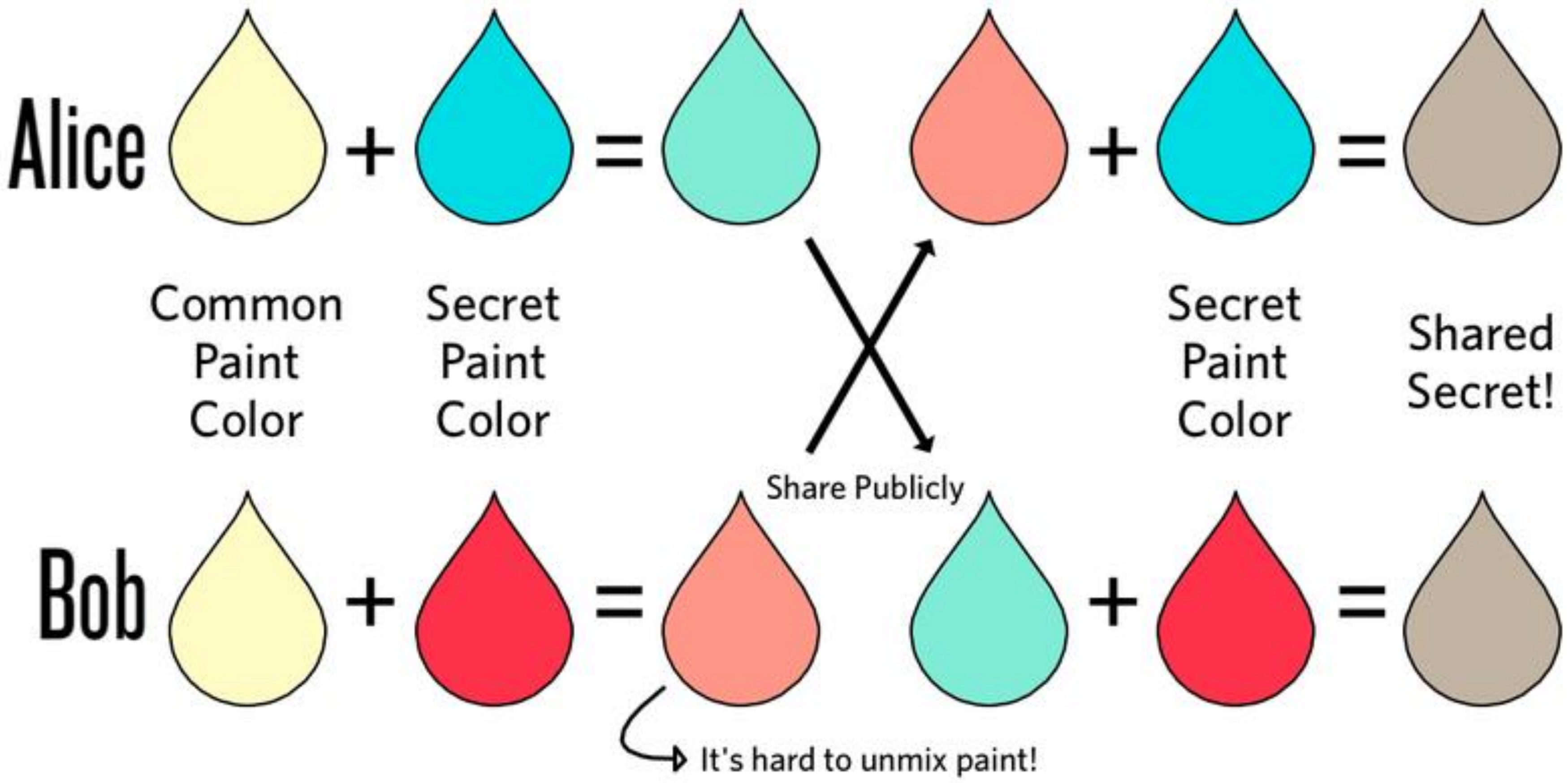
Allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure channel

Key used to encrypt subsequent communications using a symmetric key cipher

Recall the Diffie-Hellman key exchange



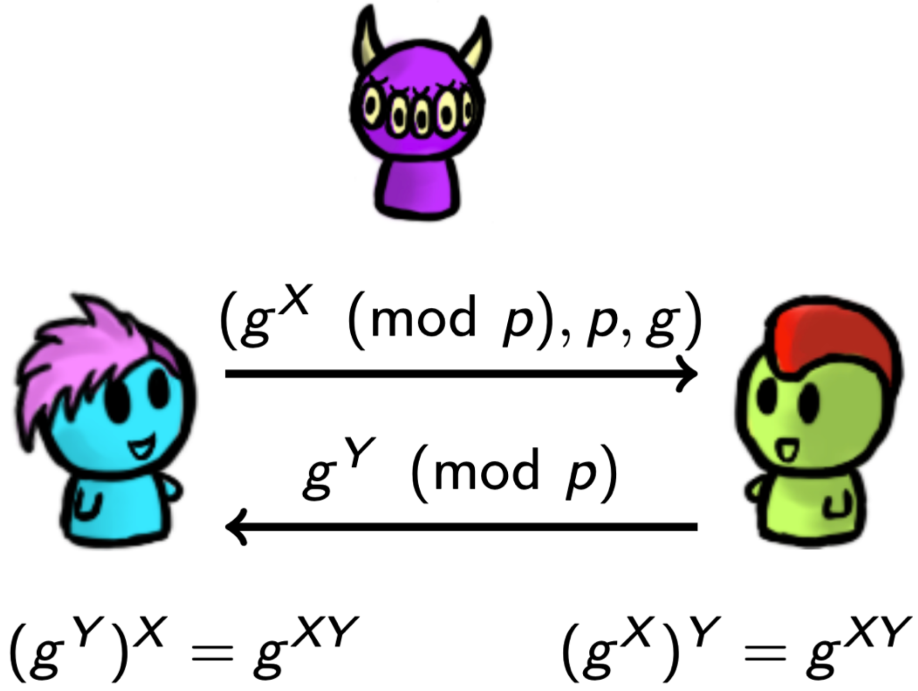
DH as paint!



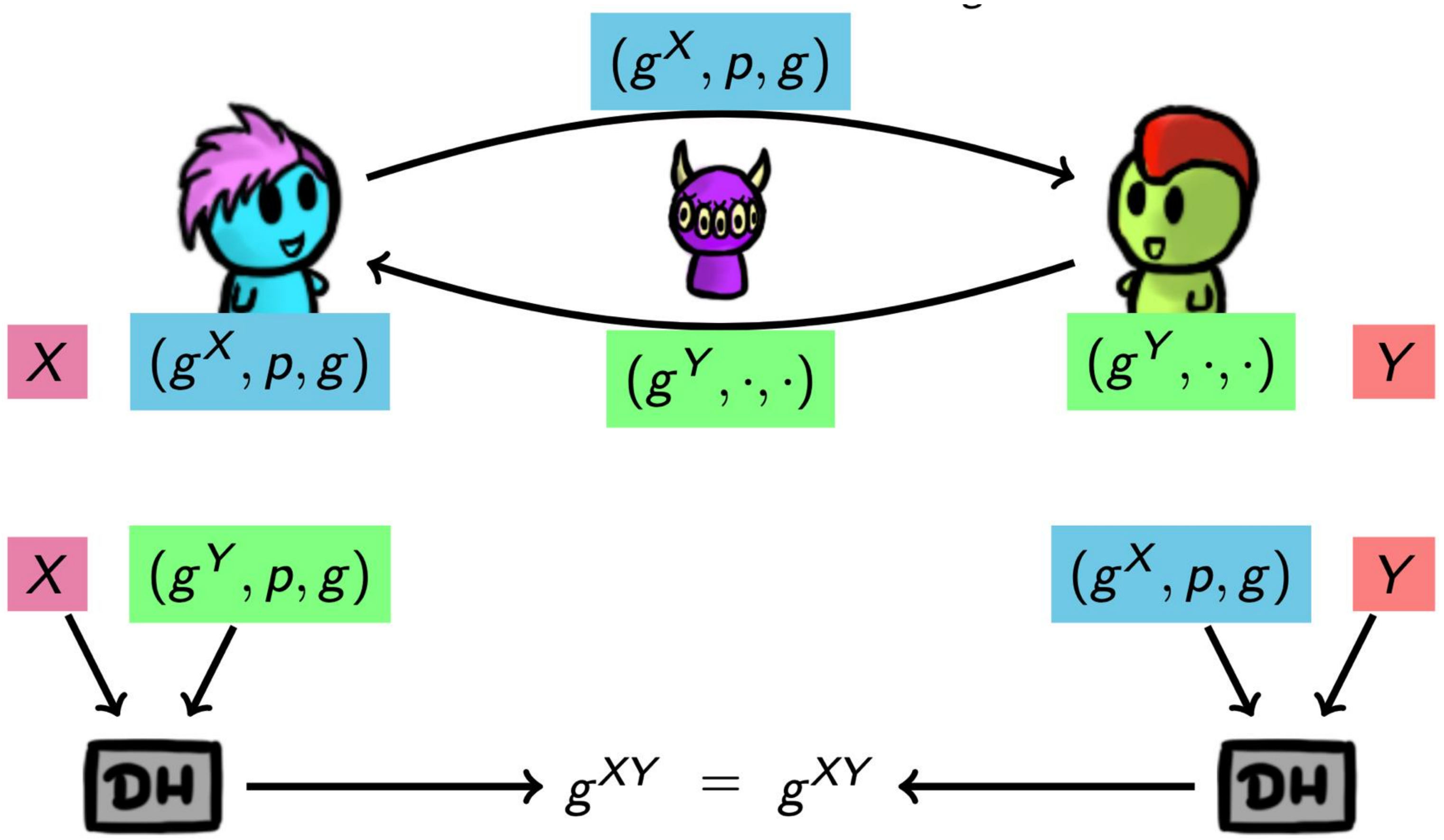
Source: Kelly Robinson

Diffie-Hellman key exchange – The Math

- Alice chooses prime p at random and finds a generator g
- Alice chooses $X \leftarrow_R \{2, 3, \ldots, p-2\}$ and sends $A = g^X \pmod{p}$ to Bob, together with p and g
- Bob chooses $Y \leftarrow_R \{2, 3, \ldots, p-2\}$ and sends $B = g^Y \pmod{p}$ to Alice
- Alice and Bob both compute $s = g^{XY} \pmod{p}$
  - Alice does that by computing $B^X \pmod{p}$
  - Bob does that by computing $A^Y \pmod{p}$
- Now they share a common secret s which can be used to derive a symmetric key



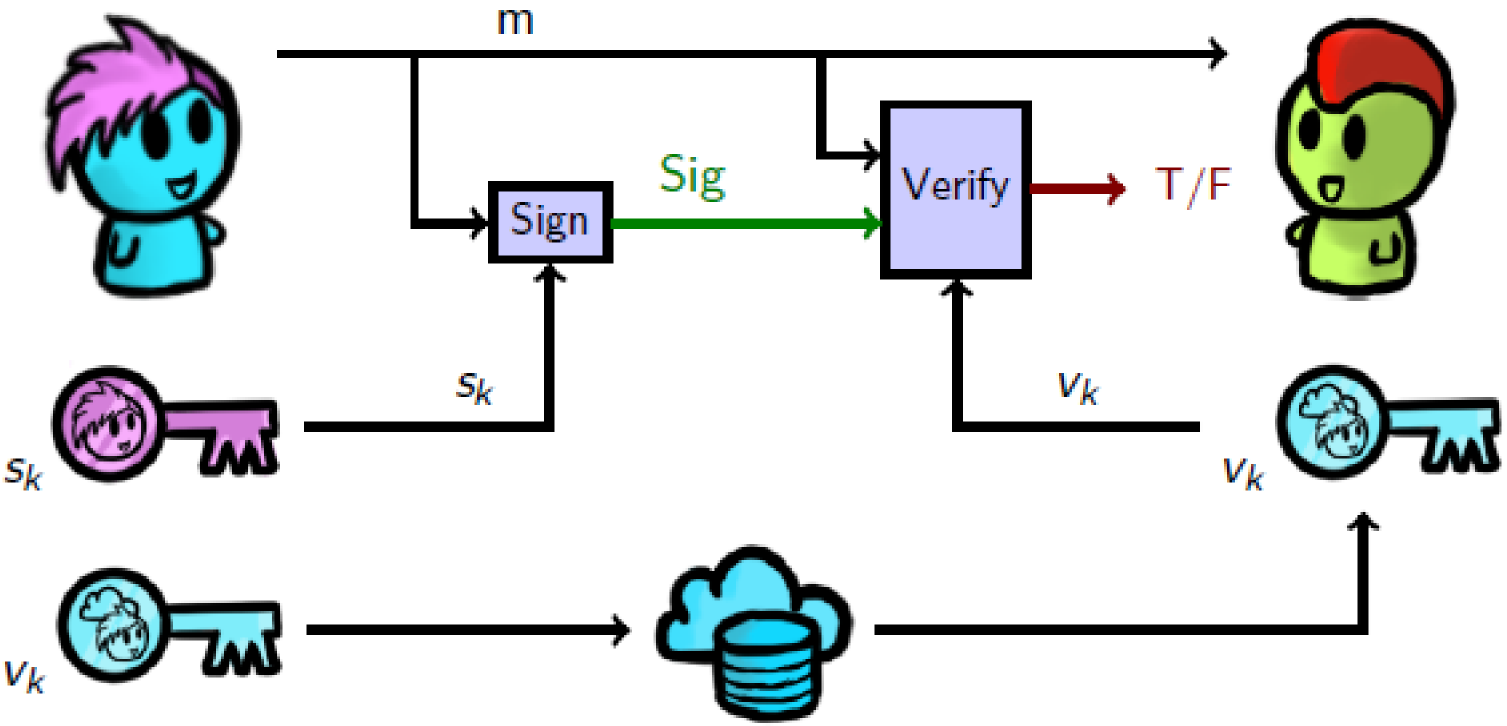
Diffie-Hellman key exchange – Altogether



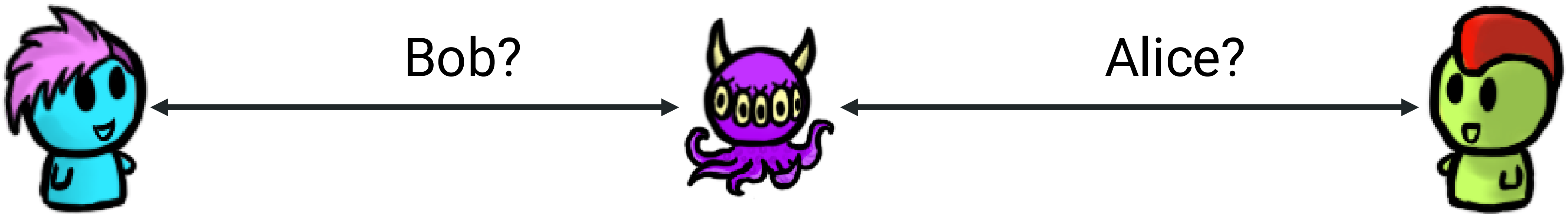
What's the Problem!

- Authentication!
- Need to verify the public keys!

Recall, Digital Signatures

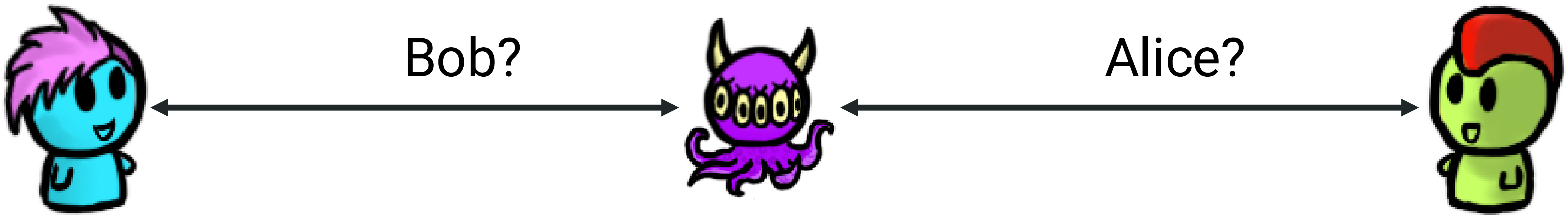


The Key Management Problem



Q: How can Alice and Bob be sure they're talking to each other?

The Key Management Problem



Q: How can Alice and Bob be sure they're talking to each other?

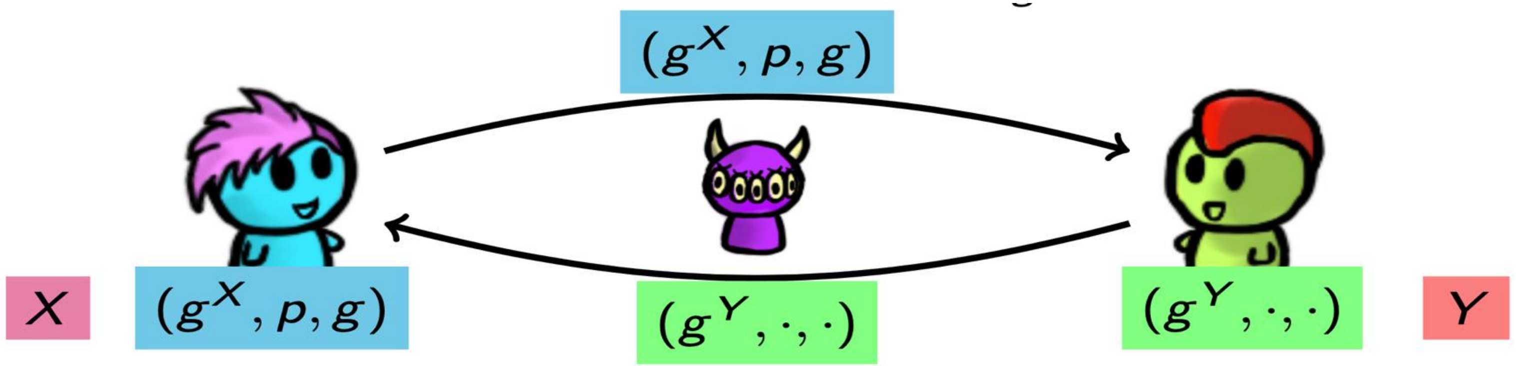
A: By having each other's verification key!

The Key Management Problem

Q: How can Alice and Bob be sure they're talking to each other?

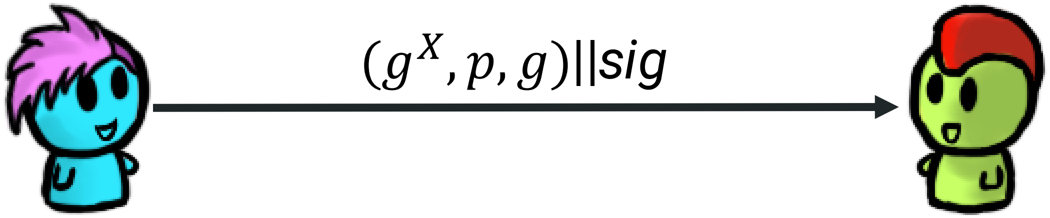
A: By having each other's verification key!

Before



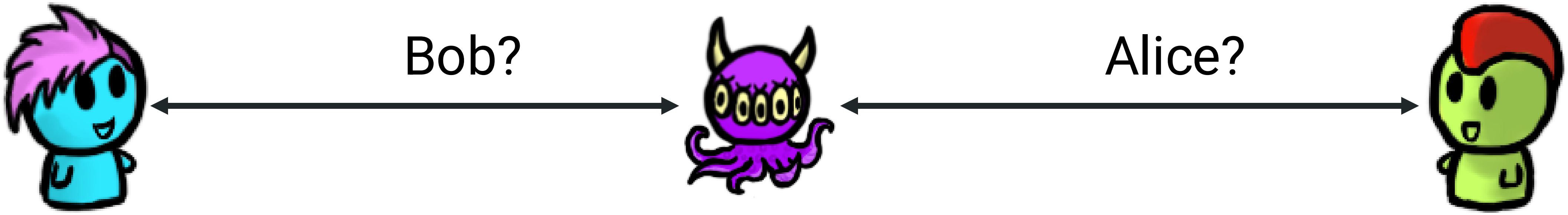
After

sig = $\text{Sign}_{sk}((g^X, p, g))$



$\text{Verify}_{vk}(sig, (g^X, p, g))$?

The Key Management Problem

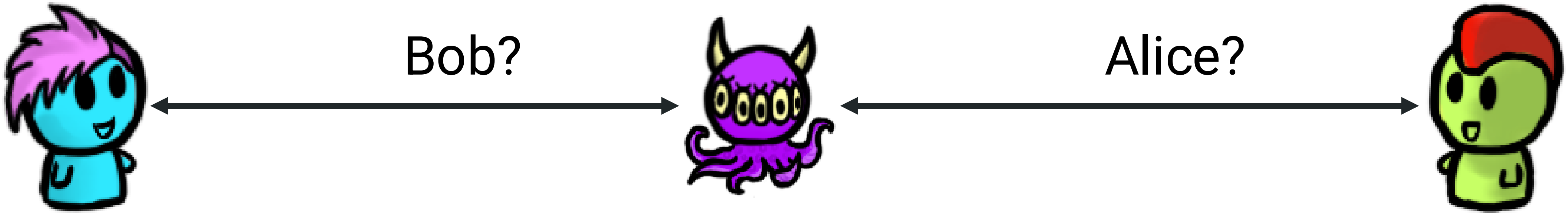


Q: How can Alice and Bob be sure they're talking to each other?

A: By having each other's verification key!

Q: But how do they get the keys…

The Key Management Problem…Solutions?



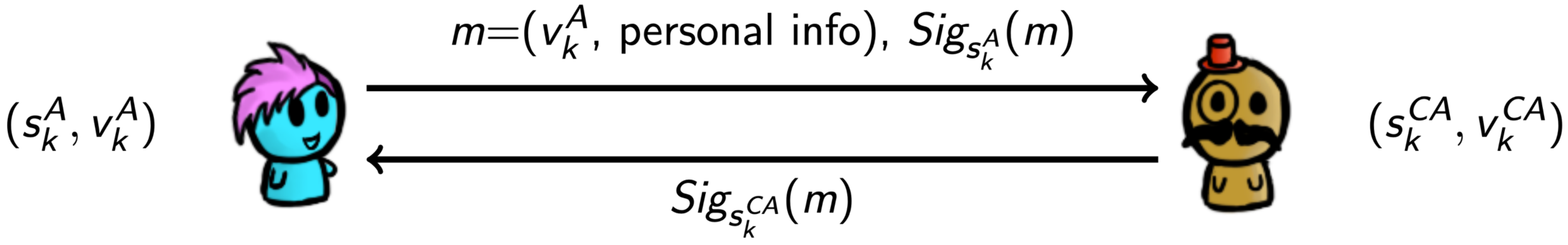
Q: But how do they get the keys…

A: Know it personally (manual keying e.g., SSH)

A: Trust a friend (web of trust e.g, PGP)

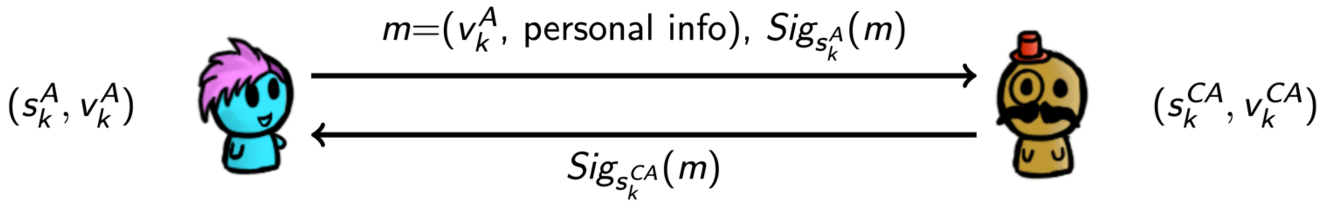
A: Trust some third party to tell them (CAs, e.g., TLS/SSL)

Certificate Authorities (CAs)



A **CA** is a trusted third party who keeps a directory of people's (and organizations') verification keys

Certificate Authorities (CAs)

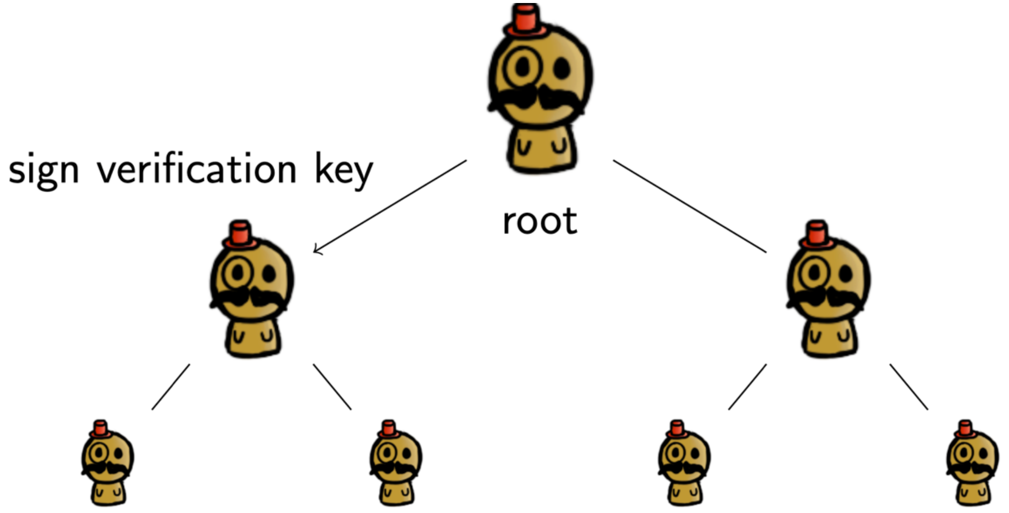


A **CA** is a trusted third party who keeps a directory of people's (and organizations') verification keys

- Alice generates a (s_k^A, v_k^A) key pair, and sends the verification key and personal information, both signed with Alice's signature key, to the CA
- The CA ensures that the personal information and Alice's signature are correct
- The CA generates a certificate consisting of Alice's personal information, as well as her verification key. The entire certificate is signed with the CA's signature key
- Most web traffic now is encrypted. Extended validation certificates (for which CAs charged a lot of money) now not treated differently by browsers.

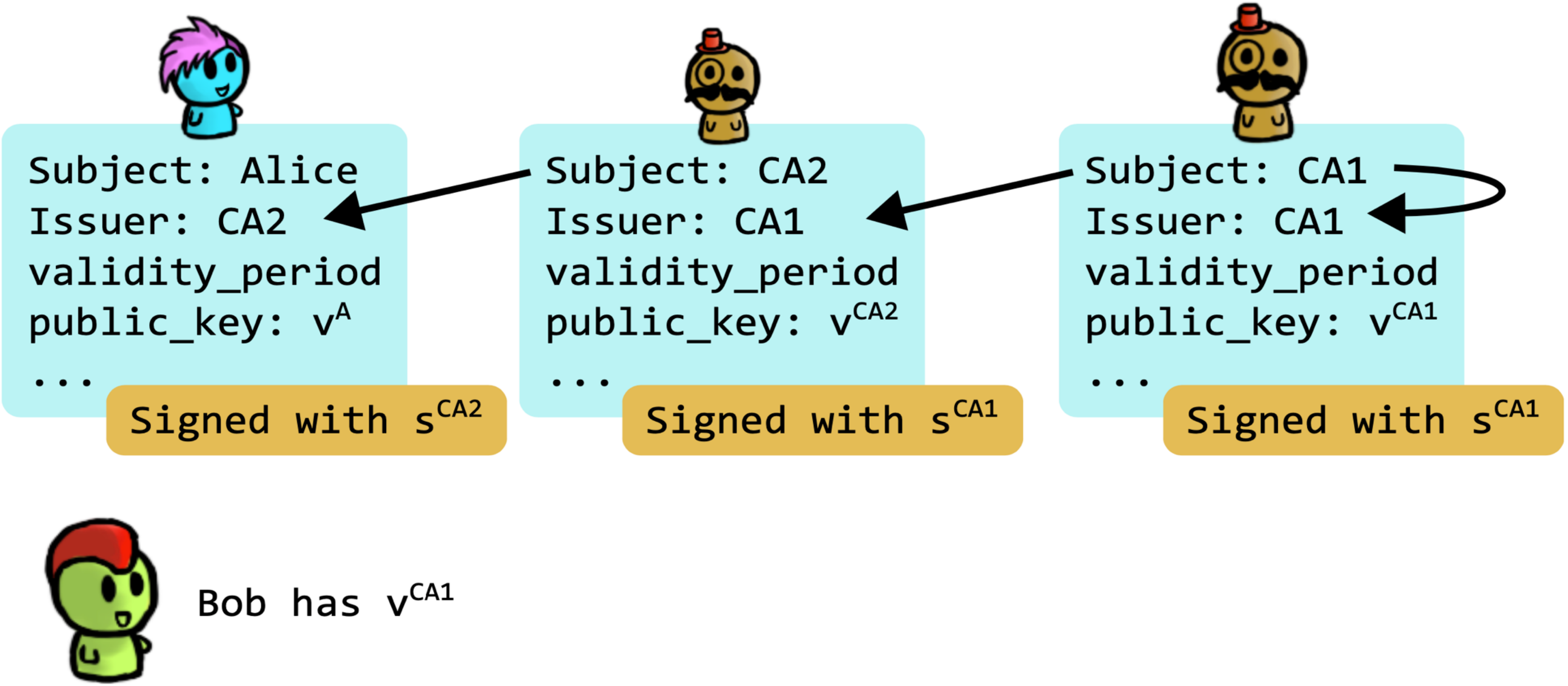
Certificate Authorities

- Everyone is assumed to have a copy of the CA's verification key (s_k^{CA}), so they can verify the signature on the certificate
- There can be multiple levels of certificate authorities; level n CA issues certificates for level n+1 CAs – Public-key infrastructure (PKI)
- Need to have only verification key of root CA to verify the certificate chain



Chain of Certificates

Alice sends Bob the following certificate to prove her identity. Bob can follow the chain of certificates to validate Alice's identity.

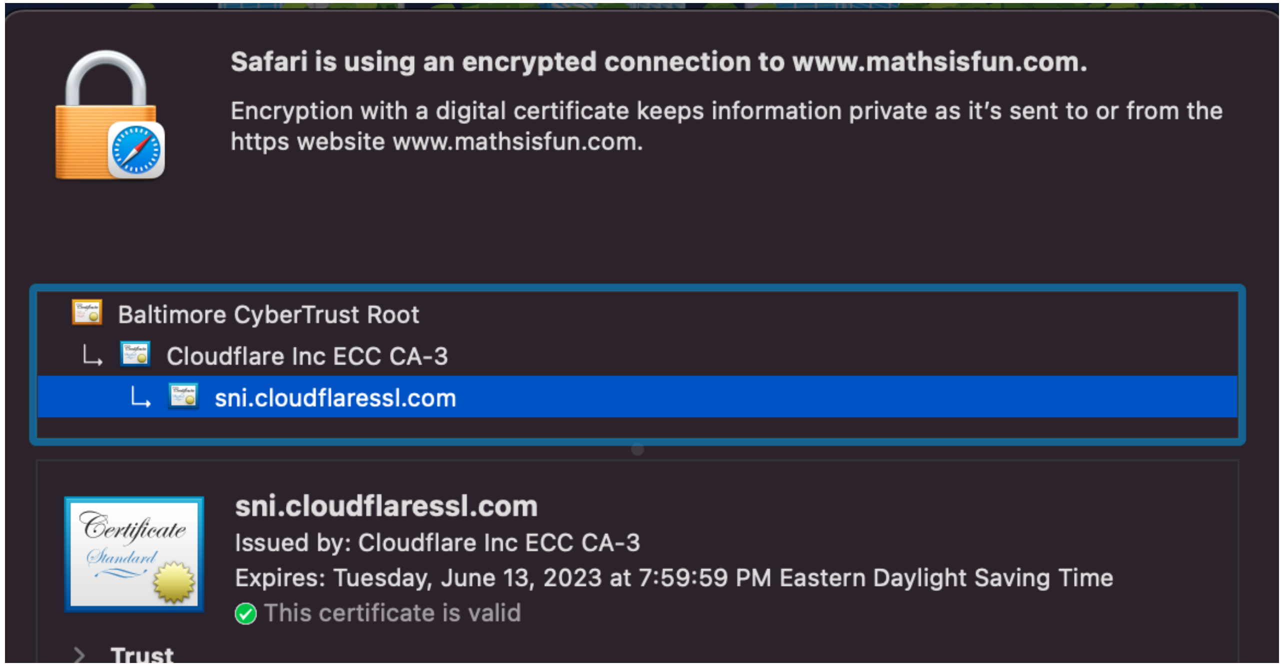


CAs on the web

- Root verification key installed on browser
- https://letsencrypt.org changed the game by offering free certificates
- Other common CAs:

Rank	Issuer	Usage	Market Share
1	IdenTrust	38.5%	43.6%
2	DigiCert Group	13.1%	14.5%
3	Sectigo (Comodo Cybersecurity)	12.1%	13.4%
4	GlobalSign	16.1%	16.7%
5	Let's Encrypt	5.8%	6.4%
6	GoDaddy Group	4.8%	5.3%

Examples


Safari is using an encrypted connection to www.mathsisfun.com.
Encryption with a digital certificate keeps information private as it's sent to or from the https website www.mathsisfun.com.
Baltimore CyberTrust Root
Cloudflare Inc ECC CA-3
sni.cloudflaressl.com
Certificate Standard
sni.cloudflaressl.com
Issued by: Cloudflare Inc ECC CA-3
Expires: Tuesday, June 13, 2023 at 7:59:59 PM Eastern Daylight Saving Time
This certificate is valid
Trust


Details
Subject Name
Country or Region US
State/Province California
Locality San Francisco
Organization Cloudflare, Inc.
Common Name sni.cloudflaressl.com
Issuer Name
Country or Region US
Organization Cloudflare, Inc.
Common Name Cloudflare Inc ECC CA-3
Serial Number 0D 62 A9 13 F8 92 16 F7 74 7D 82 56 83 B4 C1 93
Version 3
Signature Algorithm ECDSA Signature with SHA-256 (1.2.840.10045.4.3.2)
Parameters None
Not Valid Before Sunday, June 12, 2022 at 8:00:00 PM Eastern Daylight Saving Time
Not Valid After Tuesday, June 13, 2023 at 7:59:59 PM Eastern Daylight Saving Time
Public Key Info
Algorithm Elliptic Curve Public Key (1.2.840.10045.2.1)
Parameters Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)
Public Key 65 bytes : 04 74 C2 77 87 04 8D BD E0 C7 C8 8B CF 13 B8 F5 18 40 7E 98 1F C2 F7 9E 4A 66 23 5E C8 C8 93 33 75 CC C2 ED 56 1F AB DA 31 D5 5D 1A AB 39 60 9B 2B E9 91 02 62 8C B2 4D 28 F4 91 07 A8 26 01 44 2D
Key Size 256 bits
Key Usage Encrypt, Verify, Derive
Signature 70 bytes : 30 44 02 20 7A 62 4A 32 ...

Password-Authenticated Key Exchange

How do we authenticate passwords?

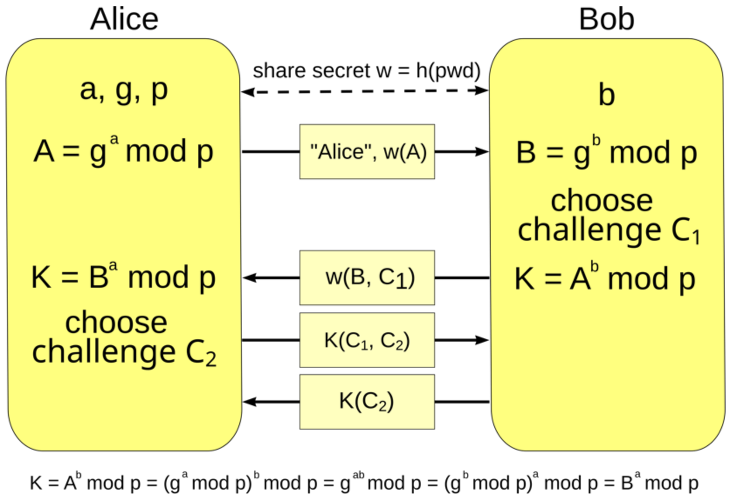
- Typically send password in plain over a secure channel (TLS)

- Server’s store only hash’s (with salt)
    - Will see the password at least briefly

- We are good at crypto, can it help?

Password-authenticated key exchange (PAKE)

- A special form of cryptographic key exchange protocol introduced by Bellovin and Merritt
- Designed to help two parties (Bob and Alice) agree on a shared encryption key using a password
  - Balanced: Both parties have password
  - Augmented: Only client (server does not)
- Problem: Hard to get it right!
  - The password should be **pre-shared** through some secure channel or prior arrangement.



DH-EKE

Goals of PAKEs

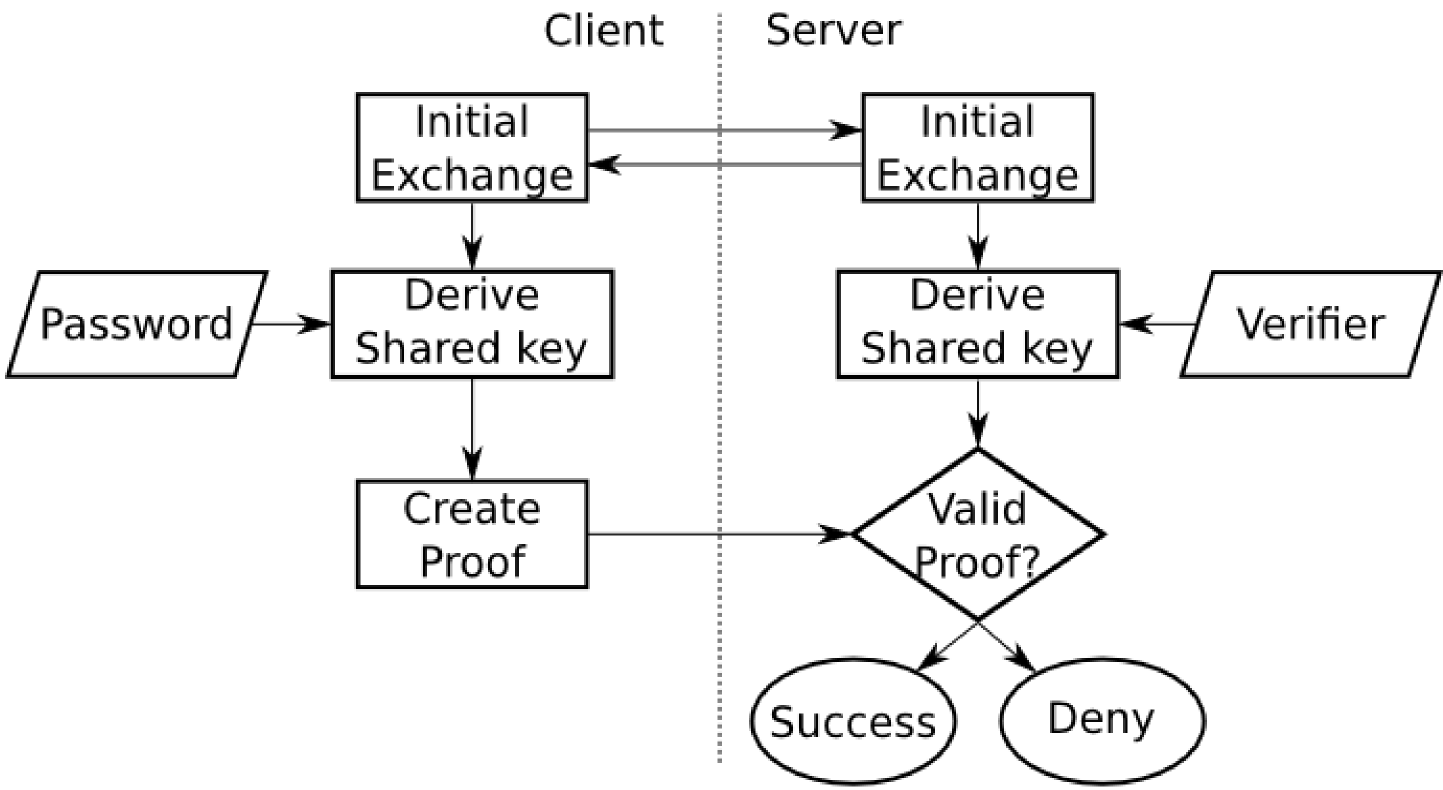
- The secret keys will match if the passwords match, and appear random otherwise.
- Participants do not need to trust third parties (in particular, no Public Key Infrastructure)
- The resulting secret key is not learned by anyone not participating in the protocol - including those who know the password.
- The protocol does not reveal either parties' password to each other (unless the passwords match), or to eavesdroppers.

Attacks on PAKEs

- Off-line dictionary attack
- On-line dictionary attacks
- Replay attacks
- Implementation Issues
- Entropy!?

Example: SRP

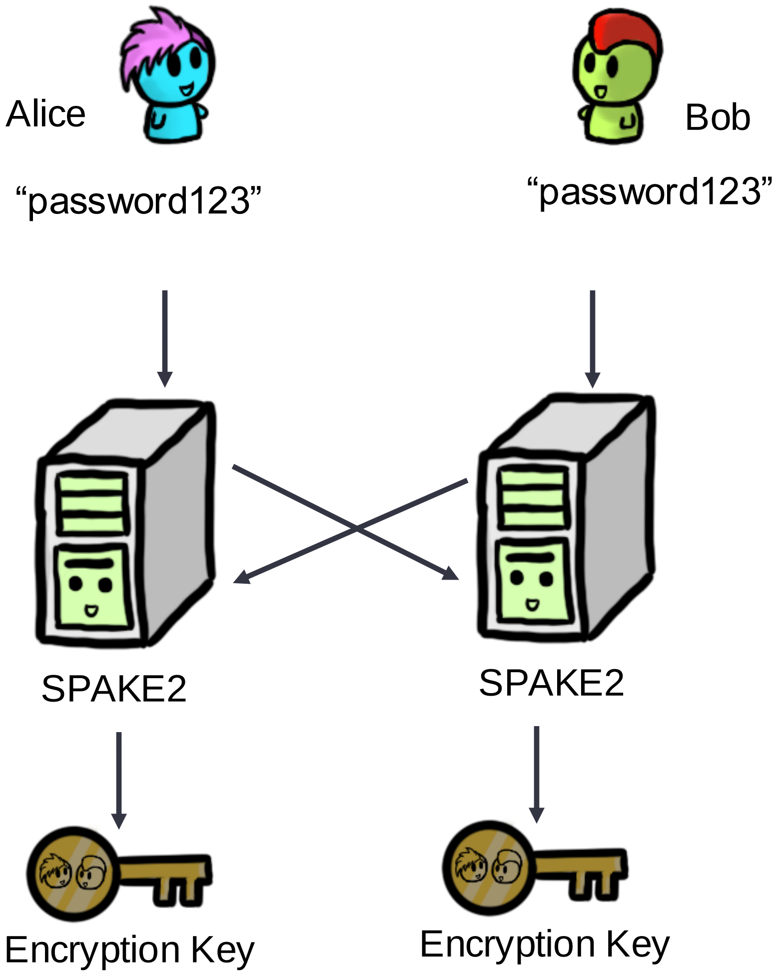
- Early widely deployed PAKEs
  - Apple iCloud!
- Poor security proof
  - On V6a (keeps getting broken)
- Vulnerable to offline dictionary attacks



Example: OPAQUE

- Proposed in 2018
- Has much stronger security proof
- Uses OPRFs to avoid leaking the salt to attacker
- Efficient, works for any hash of passwords on the server
- https://eprint.iacr.org/2018/163.pdf

Example: SPAKE2

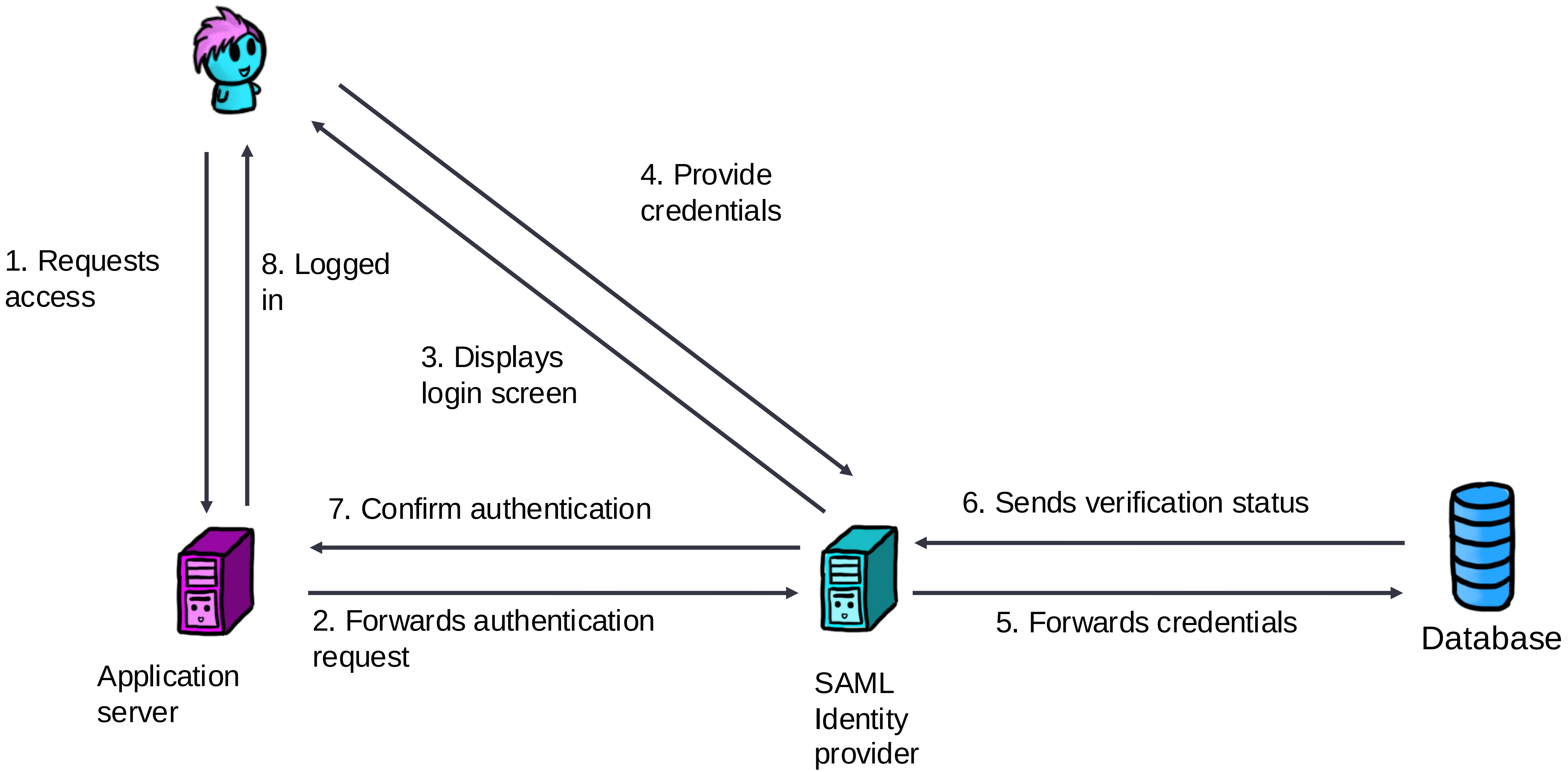


Key Management - Single Sign-On(SSO)

Security Assertion Markup Language (SAML)

- Uses secure tokens (encrypted, digitally signed XML-certificates) instead of credentials
- Allows users to access multiple related independent applications with trusted information with a single log
- Can use whatever authentication protocol you choose
- Primarily a standard for how these communications are formatted

Security Assertion Markup Language (SAML)

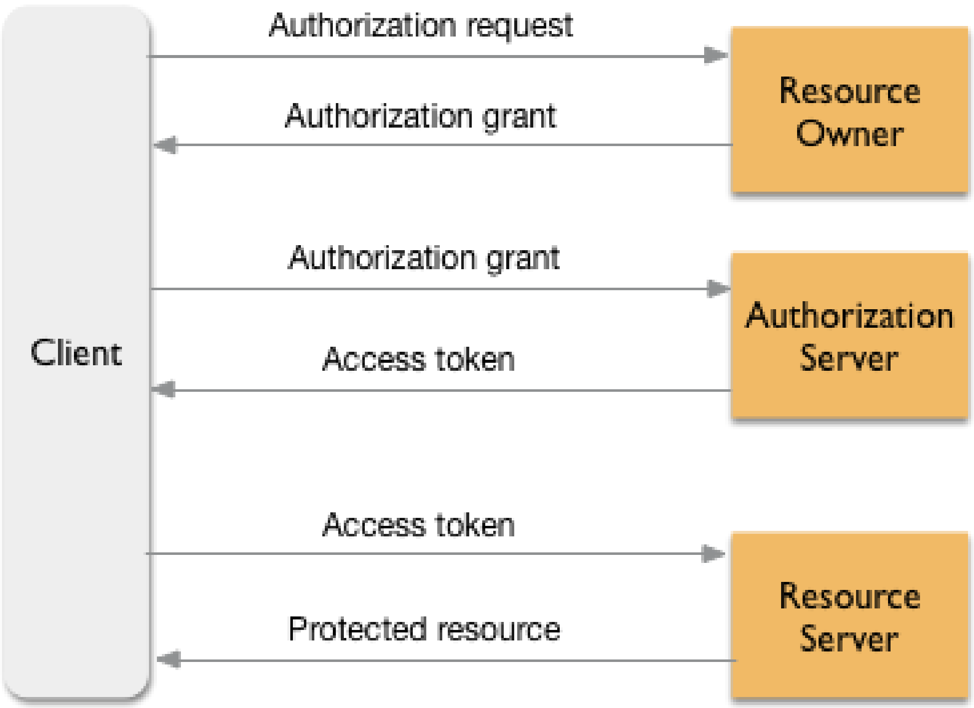


Security Assertion Markup Language (SAML)

- Advantages:
  - Authentication is centralized
  - Loose coupling of directories
  - User errors such as forgotten, weak or leaked password are avoided
  - Improves user experience (single-sign on for multiple applications)
  - XML-based protocol
    - Widely used and known
- Disadvantages
  - Complex to implement
    - Errors
    - Lengthened timelines
  - If down, can remove access from multiple systems

OAuth

- Like SAML it provides a framework and formatting for granting tokens
- Key difference: Authorization not authentication
    - i.e., a set of capabilities not attestation that you are who you say you are
    - Tokens are not tied to you
    - eg. Github, Gitlab, etc



DNSSEC

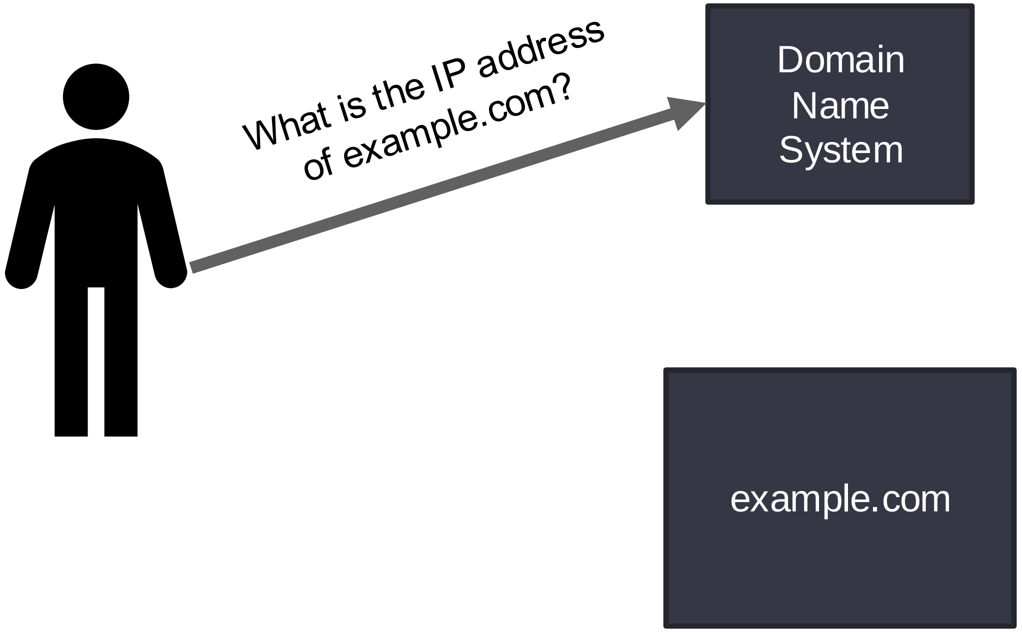
Source: Jason Goertzen and Miti Mazmudar

Recall, what is DNS?

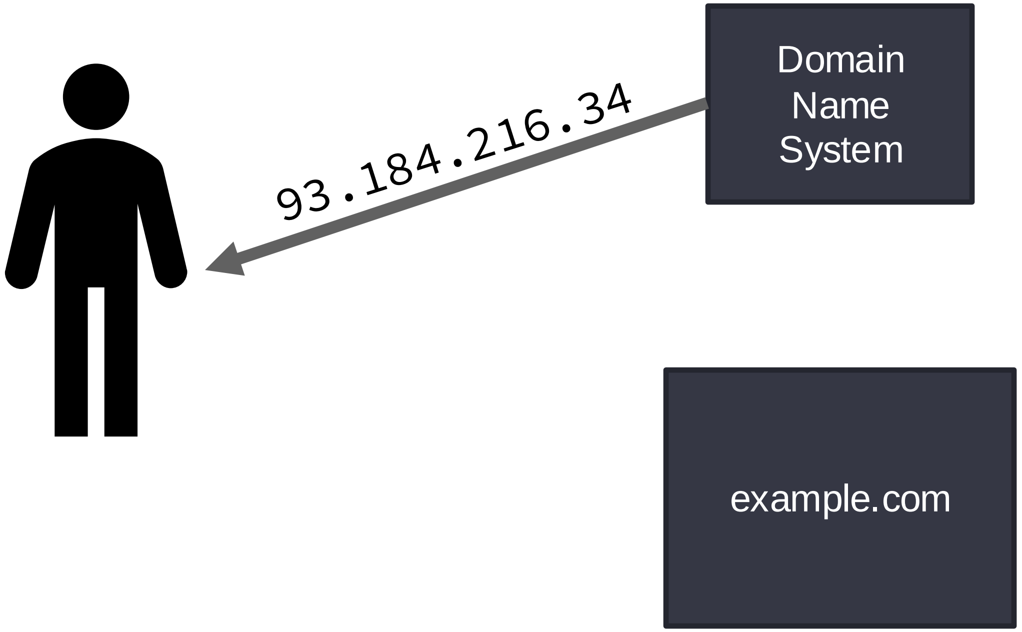
- The internet uses IP addresses to determine where to send messages
- IP addresses are difficult for people to remember!
- The Domain Name System is responsible to translating something easy for a human to remember into IP addresses

```
example.com -> 93.184.216.34
```

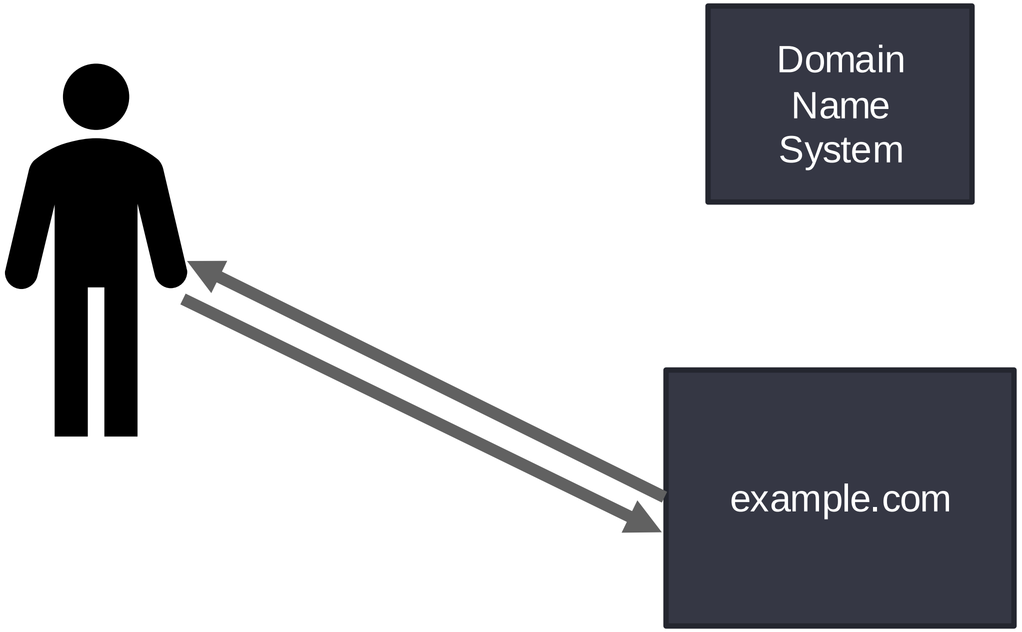
WHAT IS DNS?


What is the IP address
of example.com?
Domain
Name
System
example.com

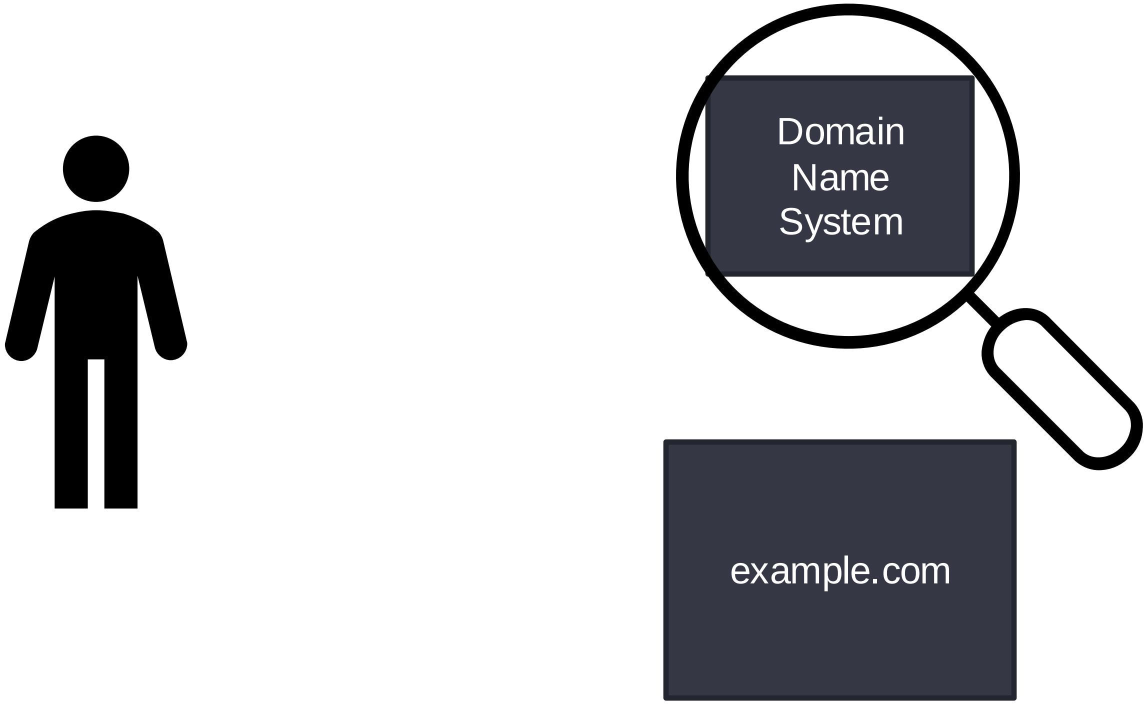
WHAT IS DNS?



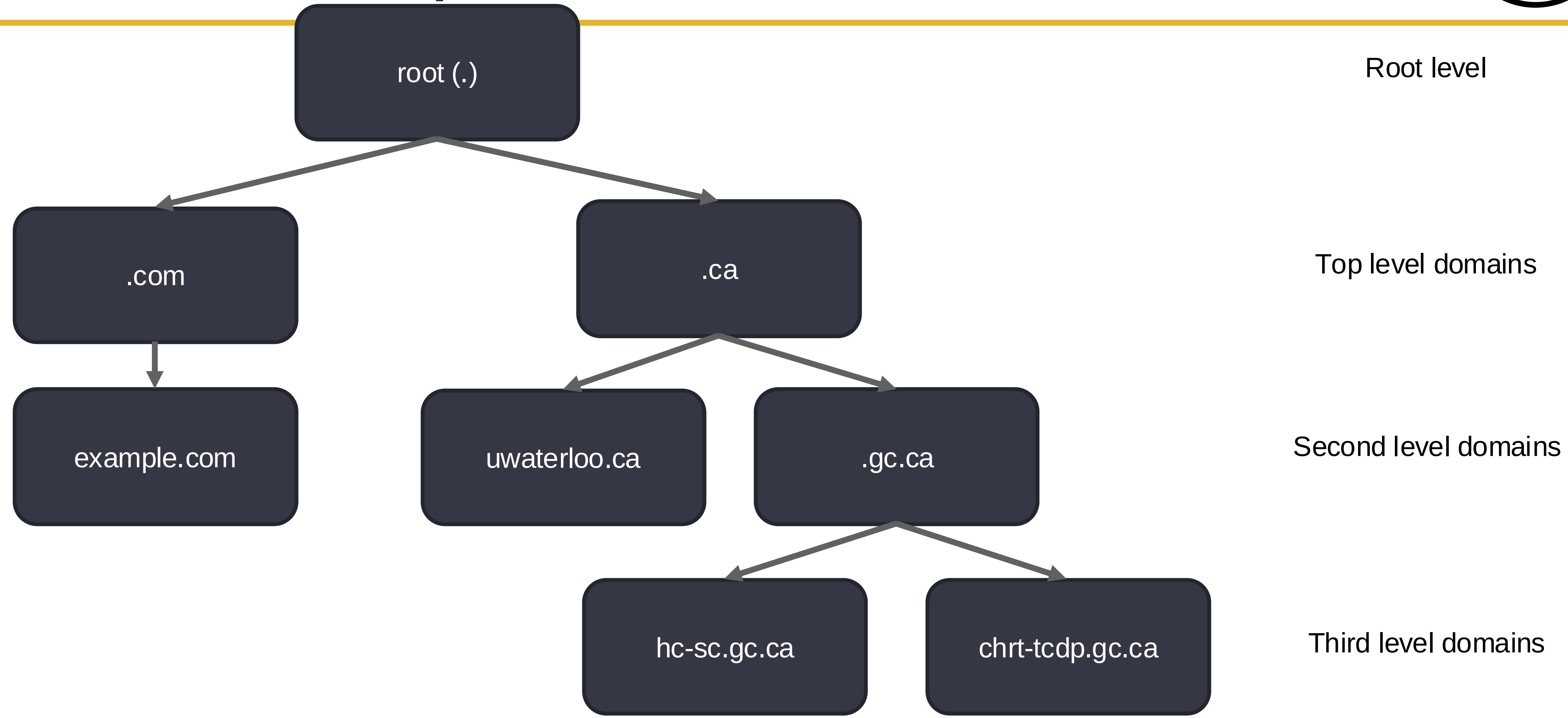
WHAT IS DNS?



WHAT IS DNS?



DNS is broken up into zones



DNS request - *dig* command

```
; <<>> DiG 9.16.15 <<>> crysp.uwaterloo.ca
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 34154
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1280
;; QUESTION SECTION:
;crysp.uwaterloo.ca.            IN      A

;; ANSWER SECTION:
crysp.uwaterloo.ca.     4552    IN      A       129.97.167.73

;; Query time: 0 msec
;; SERVER: 192.168.0.1#53(192.168.0.1)
;; WHEN: Wed May 19 15:10:46 EDT 2021
;; MSG SIZE  rcvd: 63
```

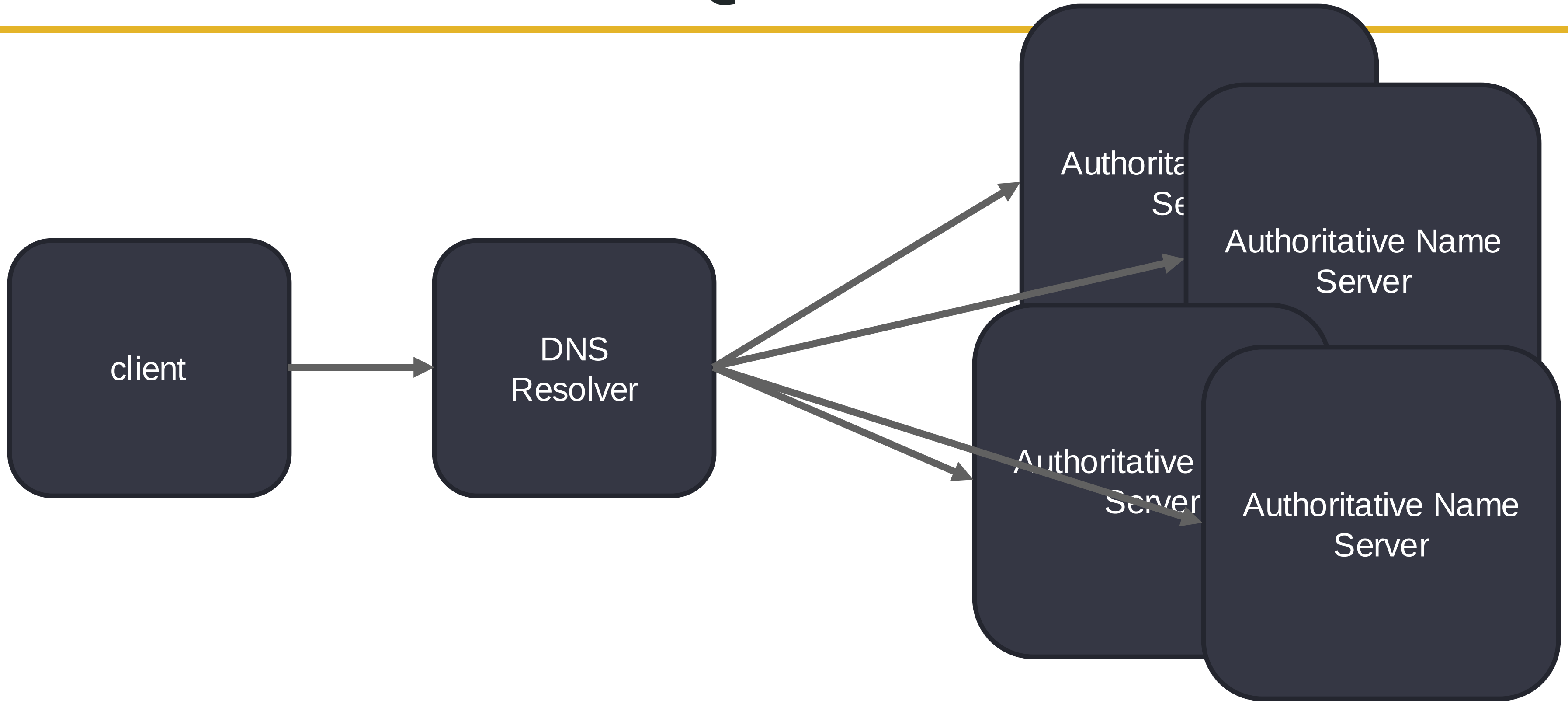
`dig crysp.uwaterloo.ca`

ZONES CONTAIN RESOURCE RECORDS

- You can look up more than just IPs

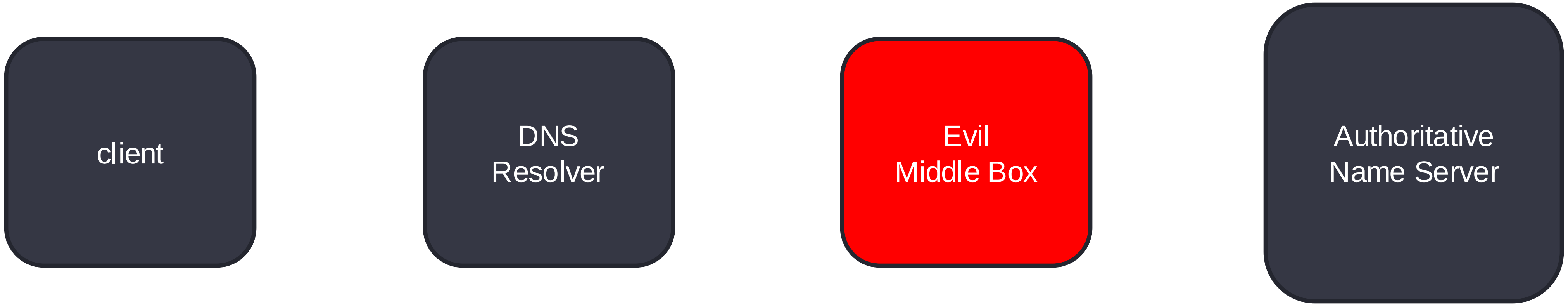
example.com. 57094 IN AAAA	2606:2800:220:1:248:1893:25c8:1946
example.com. 57047 IN A	93.184.216.34
example.com. 57094 IN NS	b.iana-servers.net.
example.com. 57094 IN NS	a.iana-servers.net.

CLIENTS RARELY QUERY DIRECTLY



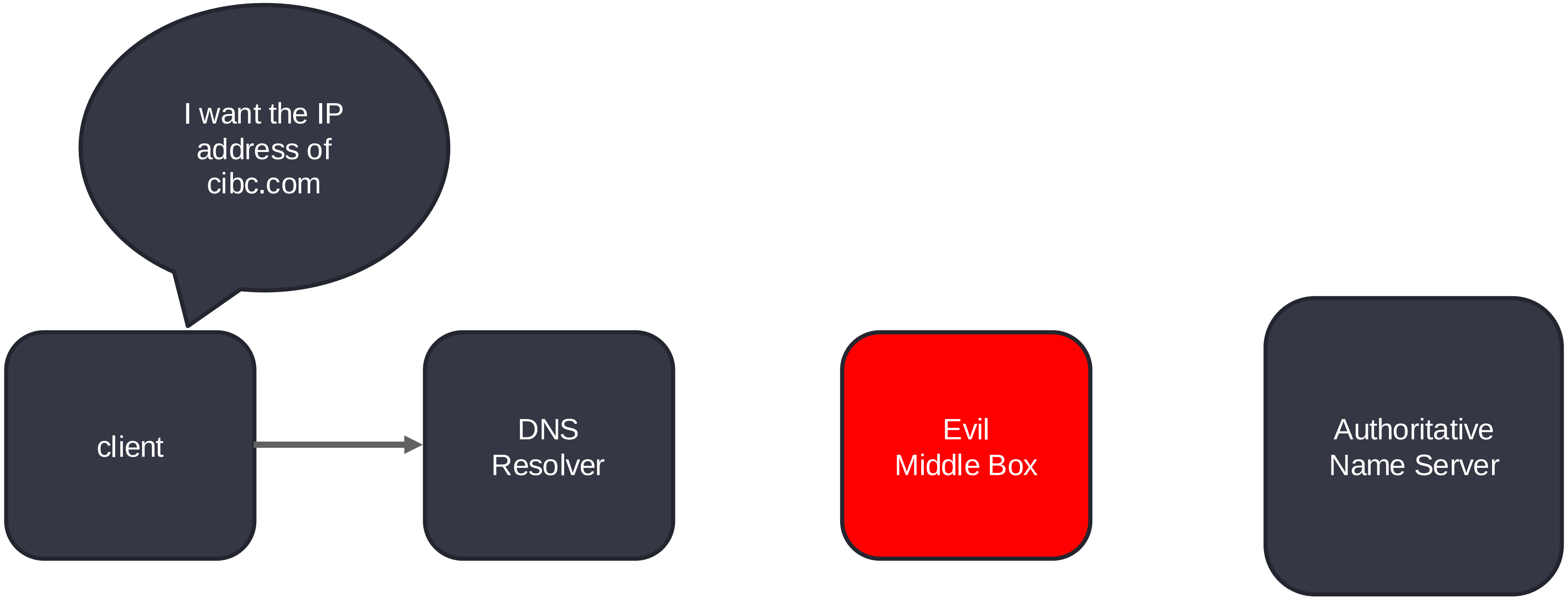
PROBLEM WITH DNS

- Designed with no integrity projection



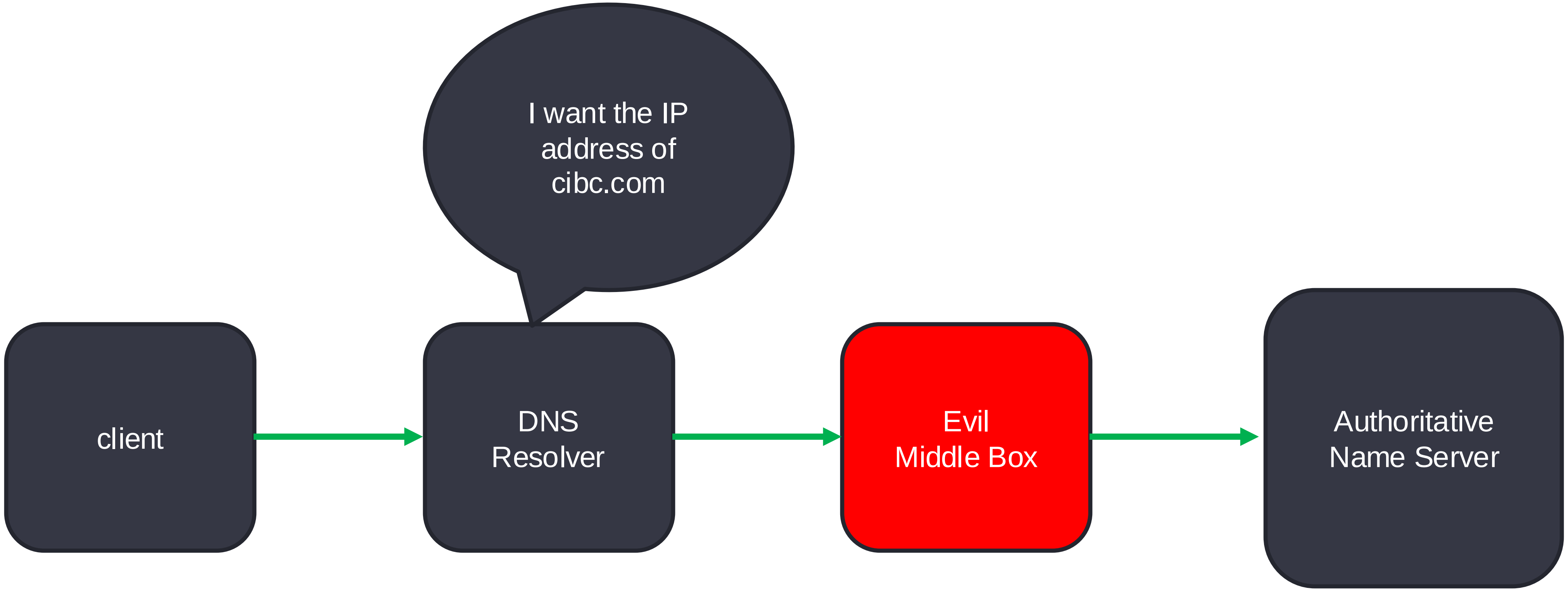
PROBLEM WITH DNS

- Designed with no integrity projection

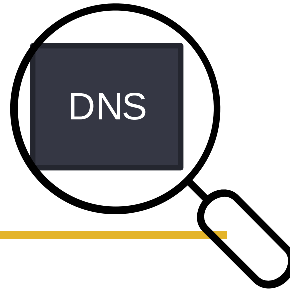


PROBLEM WITH DNS

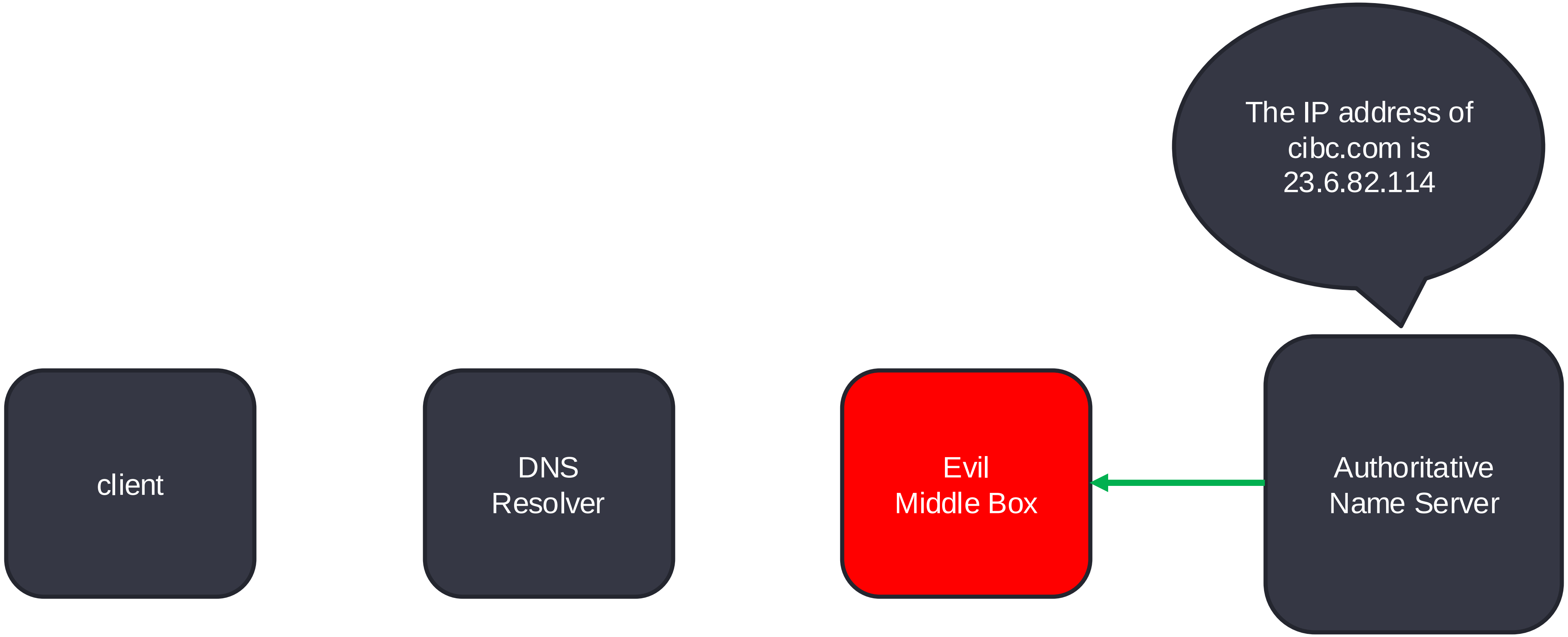
- Designed with no integrity projection



PROBLEM WITH DNS

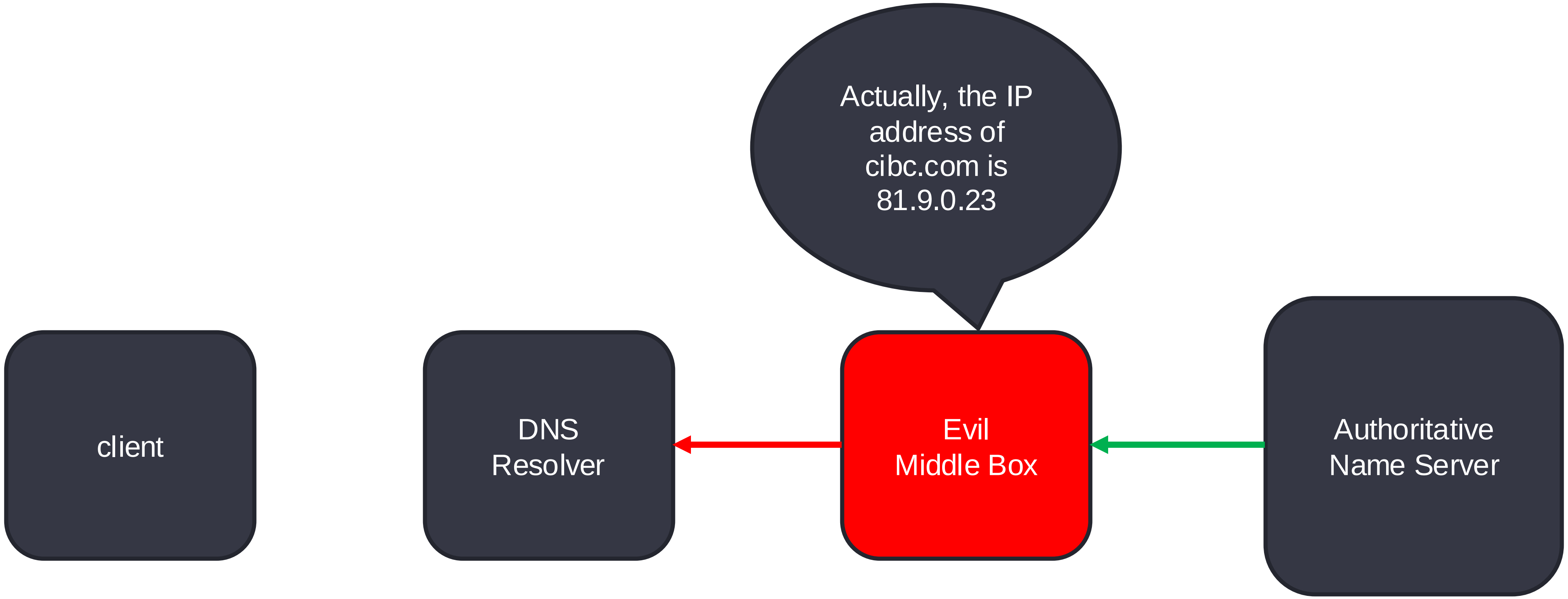


- Designed with no integrity projection



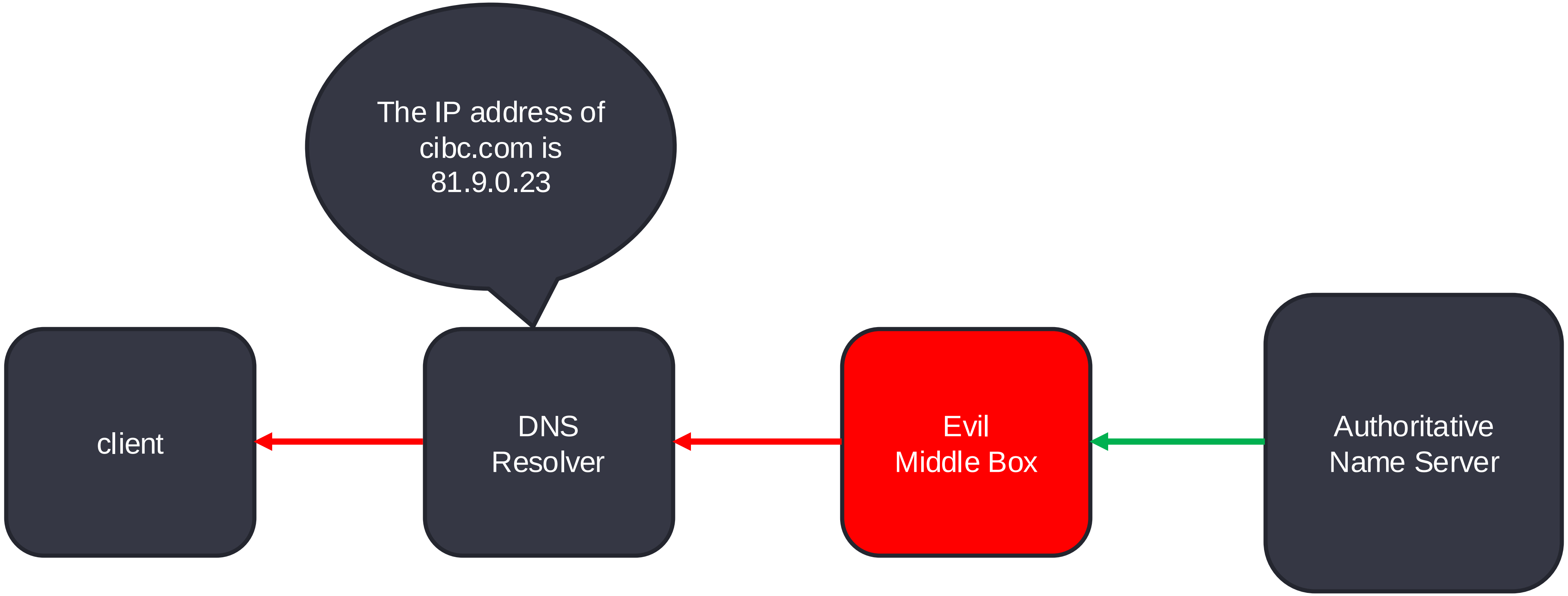
PROBLEM WITH DNS

- Designed with no integrity projection

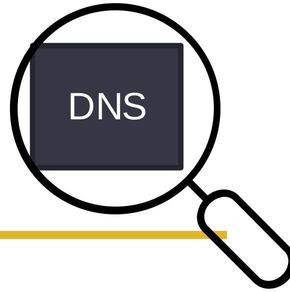


PROBLEM WITH DNS

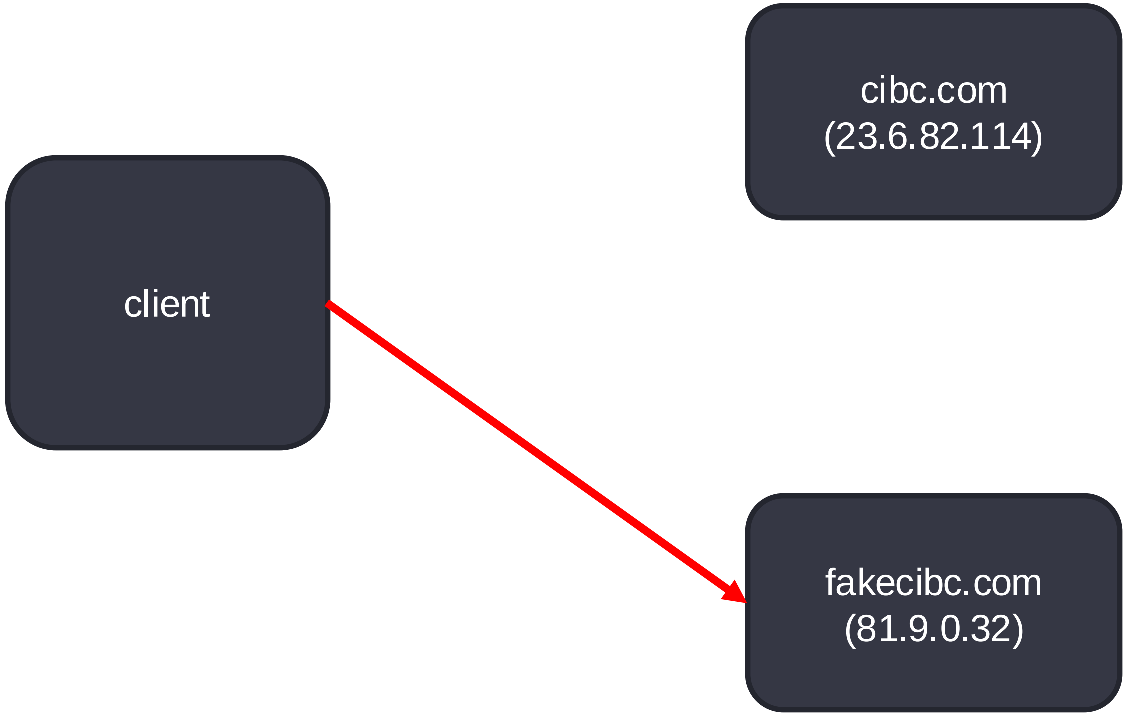
- Designed with no integrity projection



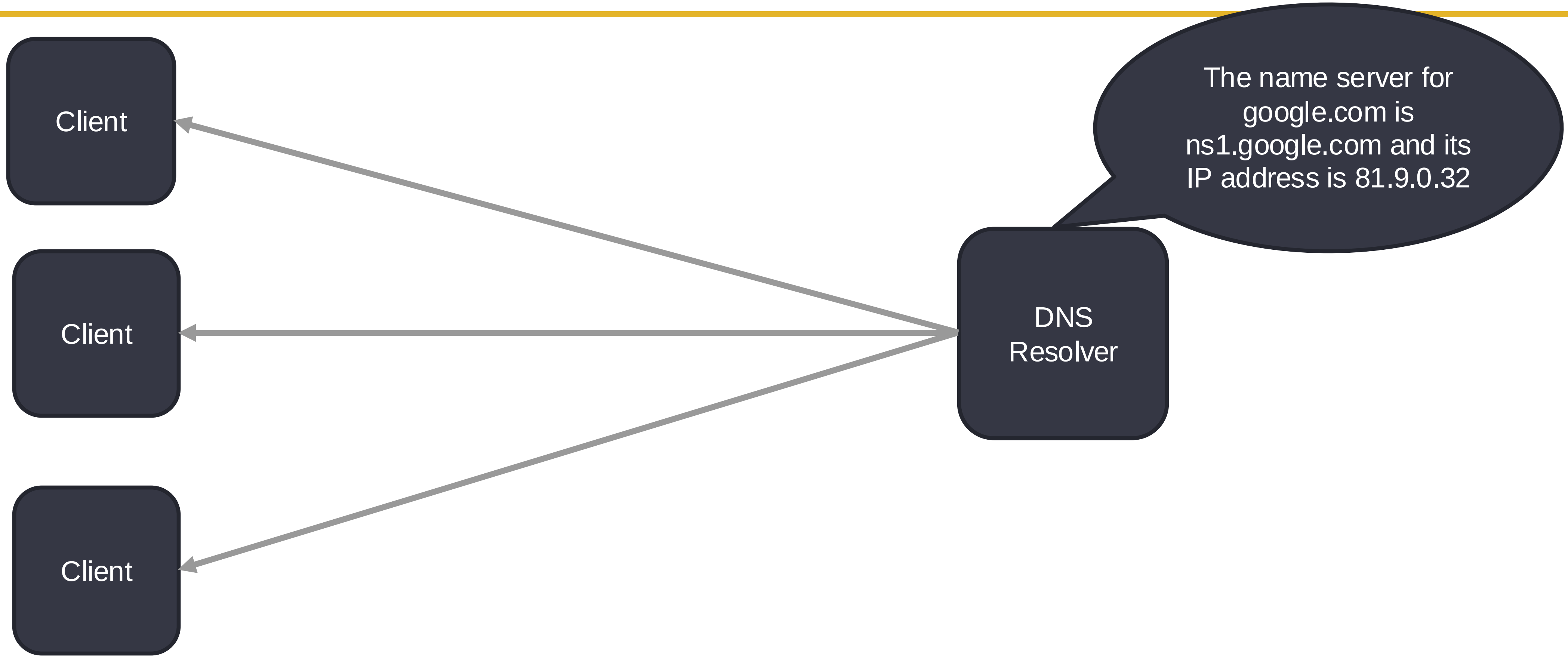
PROBLEM WITH DNS



- Designed with no integrity projection



PROBLEM WITH DNS



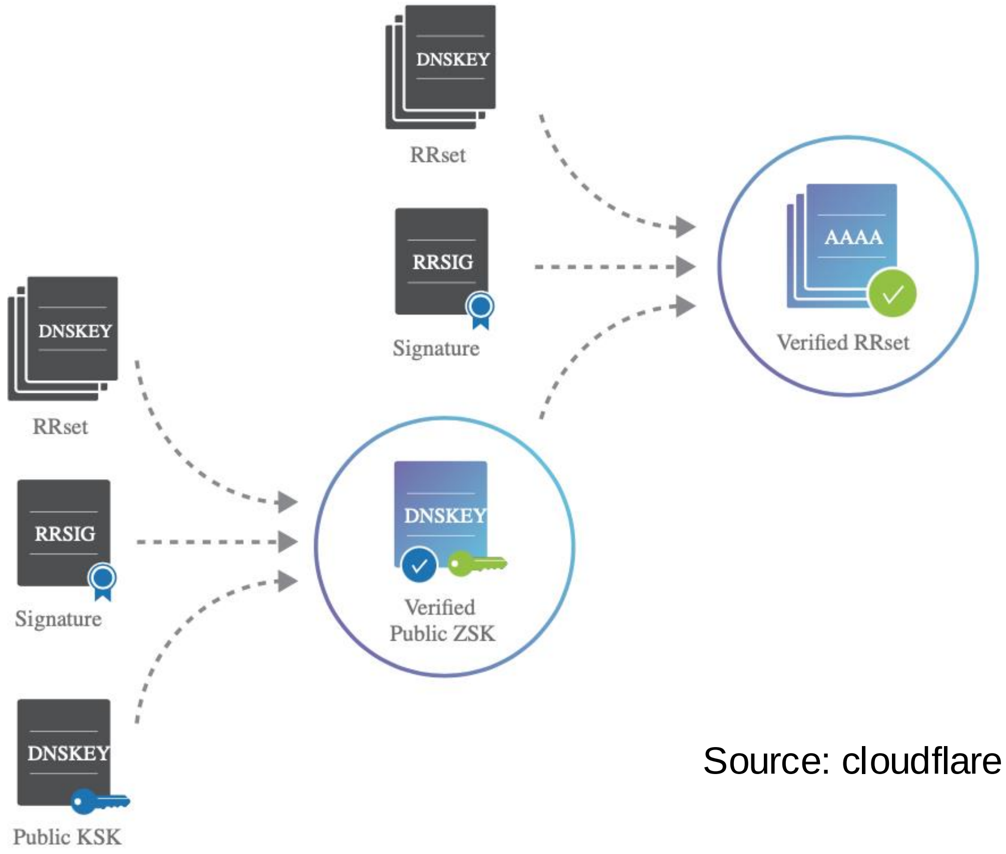
Securing DNS

Use **digital signatures** to make sure a correct and unmodified message is received from the correct entity!

- New records added to DNSSEC signed zone
- Record sets (RRSets) are signed, instead of individual records
- Have two keys:
  - **Key Signing Key (KSK):** kept in trusted hardware, hard to change
  - **Zone Signing Key (ZSK):** changed more often, smaller, used for records

Verification Procedure

- Assume you trust the public **KSK** held by a <u>"trust anchor"</u>
- Use it to verify the RRset containing a given **ZSK**
- Then use **ZSK** to verify the records



Source: cloudflare blog

How do we maintain key integrity?

Construct a chain of trust!

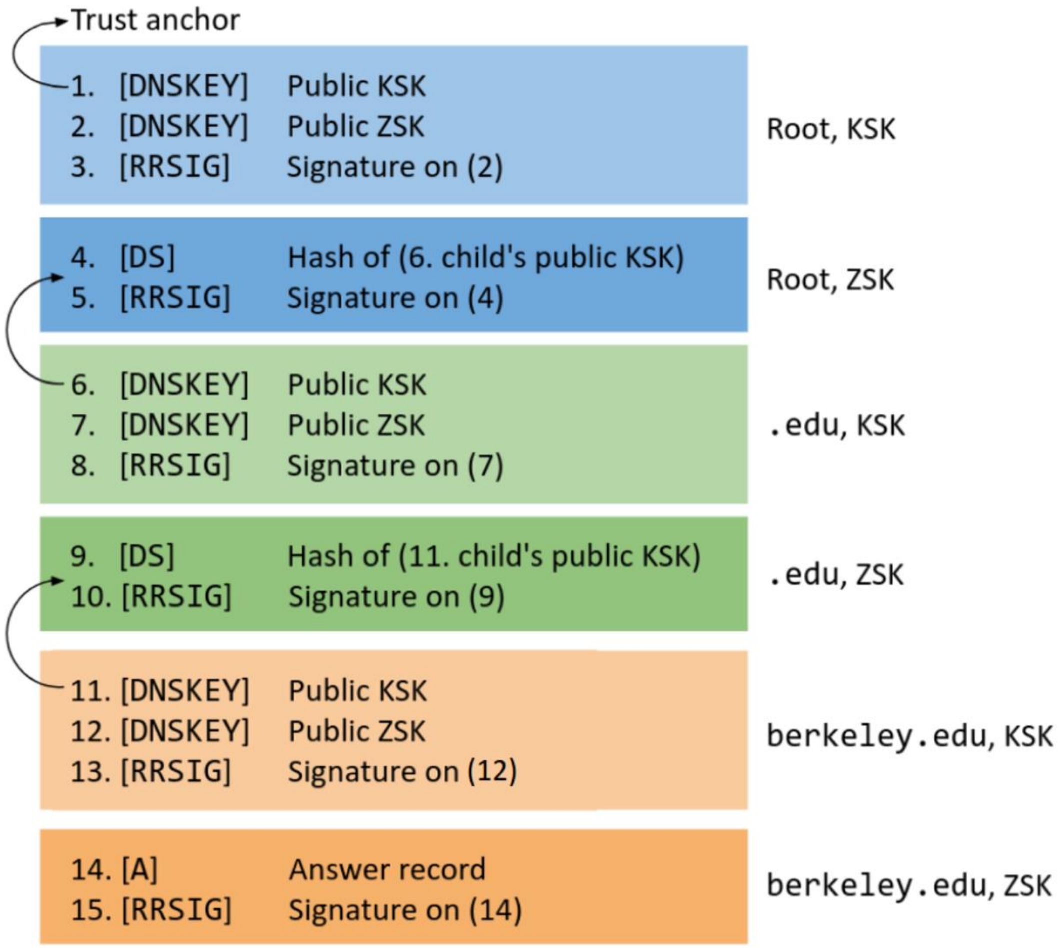
- The root verification **KSK** must be manually configurated on the machine making the request
- When the root **ZSK** is queried use the trust anchor to verify key and its signature (https://www.cloudflare.com/dns/dnssec/root-signing-ceremony/)

- Each zone's parent zone contains a "Delegate signer" (DS) record which is used to verify zone's **KSK**
    - Hash of **KSK**

The verification process

- **Light blue:** Because of our trust anchor, we trust the KSK of the root (1). The root's KSK signs its ZSK, so now we trust the root's ZSK (2-3).
- **Dark blue:** We trust the root's ZSK. The root's ZSK signs .edu's KSK (4-5), so now we trust .edu's KSK.
- **Light green:** We trust the .edu's KSK (6). .edu's KSK signs .edu's ZSK, so now we trust .edu's ZSK (7-8).
- **Dark green:** We trust .edu's ZSK. .edu's ZSK signs berkeley.edu's KSK (9-10), so now we trust berkeley.edu's KSK.
- **Light orange:** We trust the berkeley.edu's KSK (11). berkeley.edu's KSK signs berkeley.edu's ZSK, so now we trust berkeley.edu's ZSK (12-13).
- **Dark orange:** We trust berkeley.edu's ZSK. berkeley.edu's ZSK signs the final answer record (14-15), so now we trust the final answer.

https://textbook.cs161.org/network/dnssec.html



Next Class, Security through the layers!