Catch Me if You Can: An Account Based End-to-end Encryption for 1/1 Snaps

Subhash Sankuratripati
Moti Yung,
Anirudh Garg,
Wentao Huang

Team that built this

Anirudh Garg

Eddie Xue

Mike Duong

Subhash Sankuratripati

Moti Yung

Janelle Tiulentino

Wentao Huang

Daniel Hwang

Charles Huang

In order of when they started working on the project

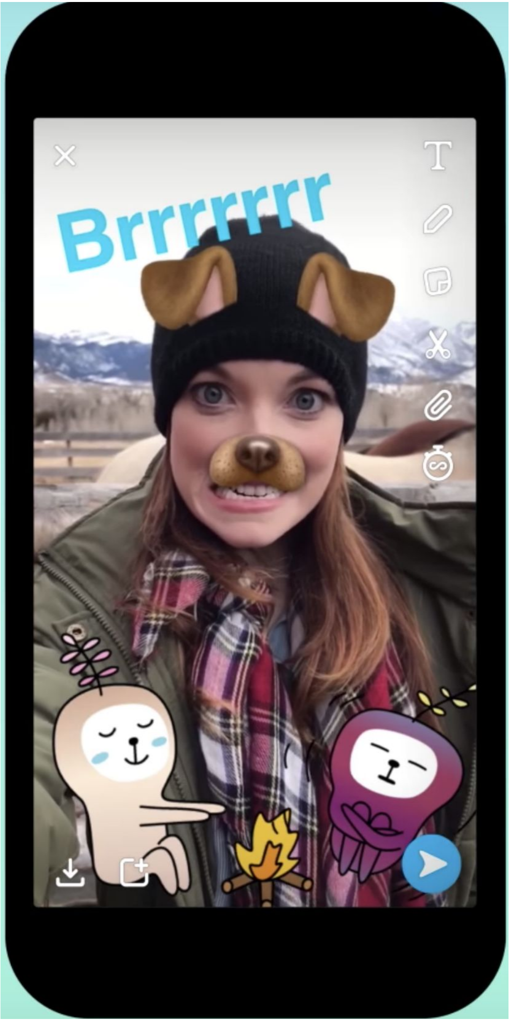
What is a Snap?

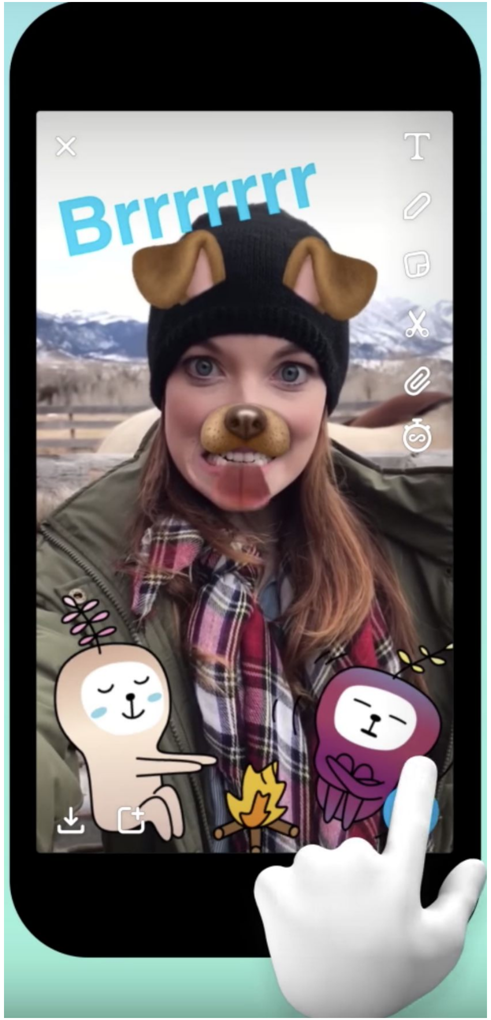
What is a Snap?

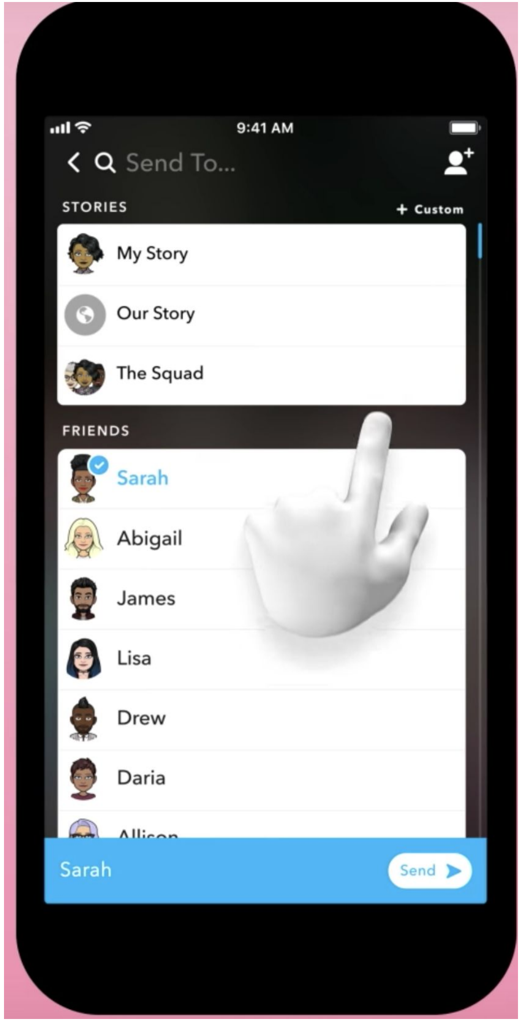
- A multimedia message that is shared between users of the Snapchat App

- The app is used by 186 million users on a daily basis (Q3 2018)

- Billions of Snaps are exchanged everyday


Brrrrrrr


Brrrrrrr


9:41 AM
Send To...
STORIES
+ Custom
My Story
Our Story
The Squad
FRIENDS
Sarah
Abigail
James
Lisa
Drew
Daria
Allison
Sarah
Send

Snaps have inherent privacy protections

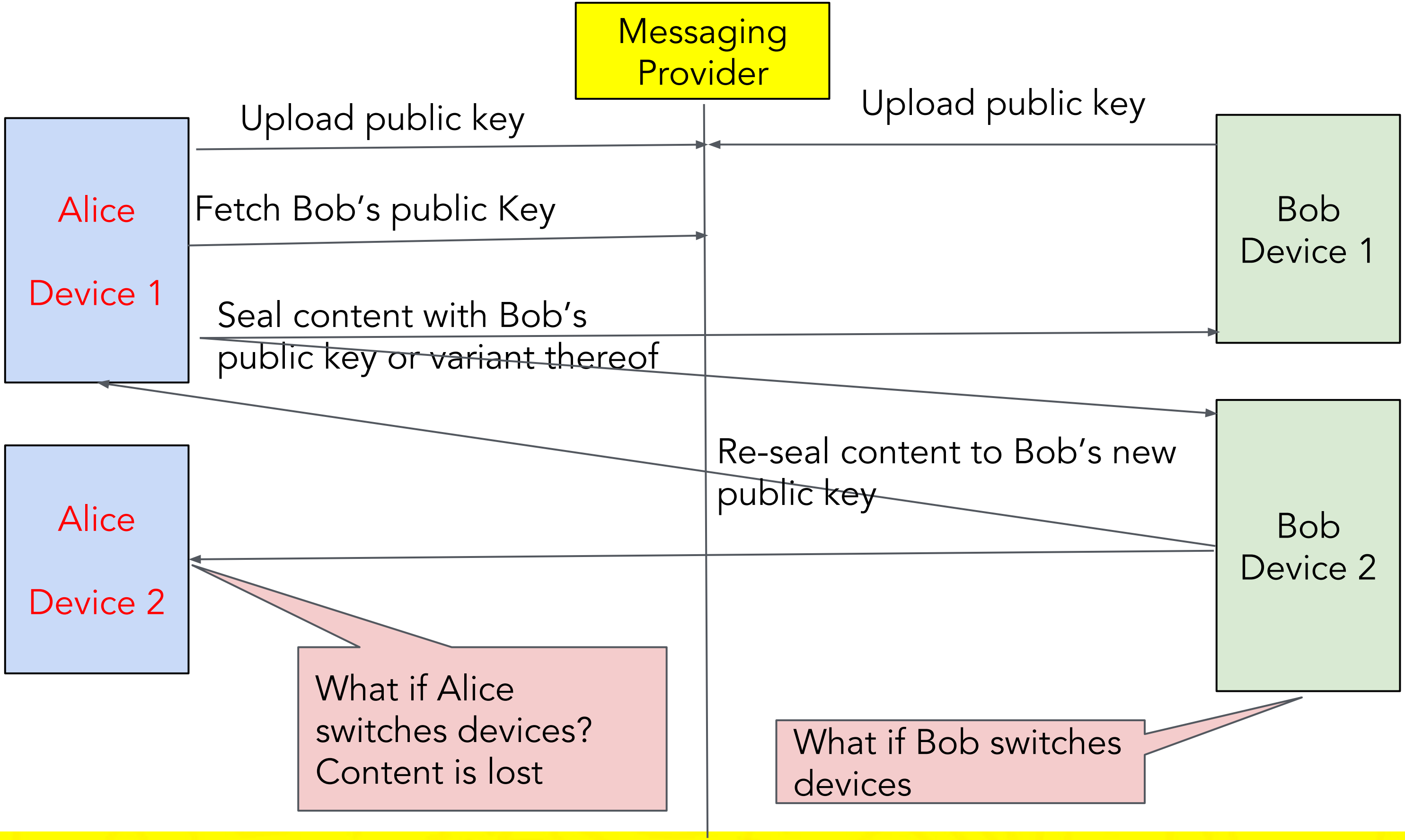
- They are ephemeral
  - Deleted right after viewing
  - Deleted in 30 days if not viewed

Why end-to-end encryption (E2EE)?

Why?

- Defense in depth
- Increased assurances around privacy to our users

Well, E2EE is a solved problem


Messaging Provider
Upload public key
Upload public key
Alice Device 1
Fetch Bob's public Key
Seal content with Bob's public key or variant thereof
Bob Device 1
Re-seal content to Bob's new public key
Alice Device 2
Bob Device 2
What if Alice switches devices? Content is lost
What if Bob switches devices

Key requirements

- Fast key distribution
- A fast mechanism to retry
- Retry delays increase probability of sender device churn and hence content loss

Industry Status quo

- iMessage, WhatsApp and Signal have deployed, an, on by default, E2EE system at scale for 4+ years!

But, they differ from Snapchat in that:

- None of them have an ‘easy’ logout mechanism
- Couple logout with single session restrictions
- Their authentication model relies on device identity (phone number or the device itself)
- Sessions are pretty tightly coupled to devices

These difference allow Snapchat users to:

- Share a device (one device, many users)
- Hop between devices (one user, many devices)
- → All of which lead to **identity churn**

Tightly coupled device to device E2EE protocols

- Can offer stronger assurances that make it less amenable to retry
- Forward secrecy, especially at the recipient level increases retry times
- Yet, we tried!!
- We ran an Axolotl like protocol that had a retry required rate of 1.85%

So, our requirements:

Requirements

- Reduce the churn
  - Securely support multiple users on a given device
  - Support multiple devices for a given user
- Make retries faster!

Introduce the notion of an Account based E2EE

- Private keys are still present **exclusively** on client devices, but,
- Needed a mechanism by which we could perform fast private key to device association changes
- And notions of recipient level forward secrecy, as introduced by Axolotl make retries slower (hurt streaks!) and had to be relaxed

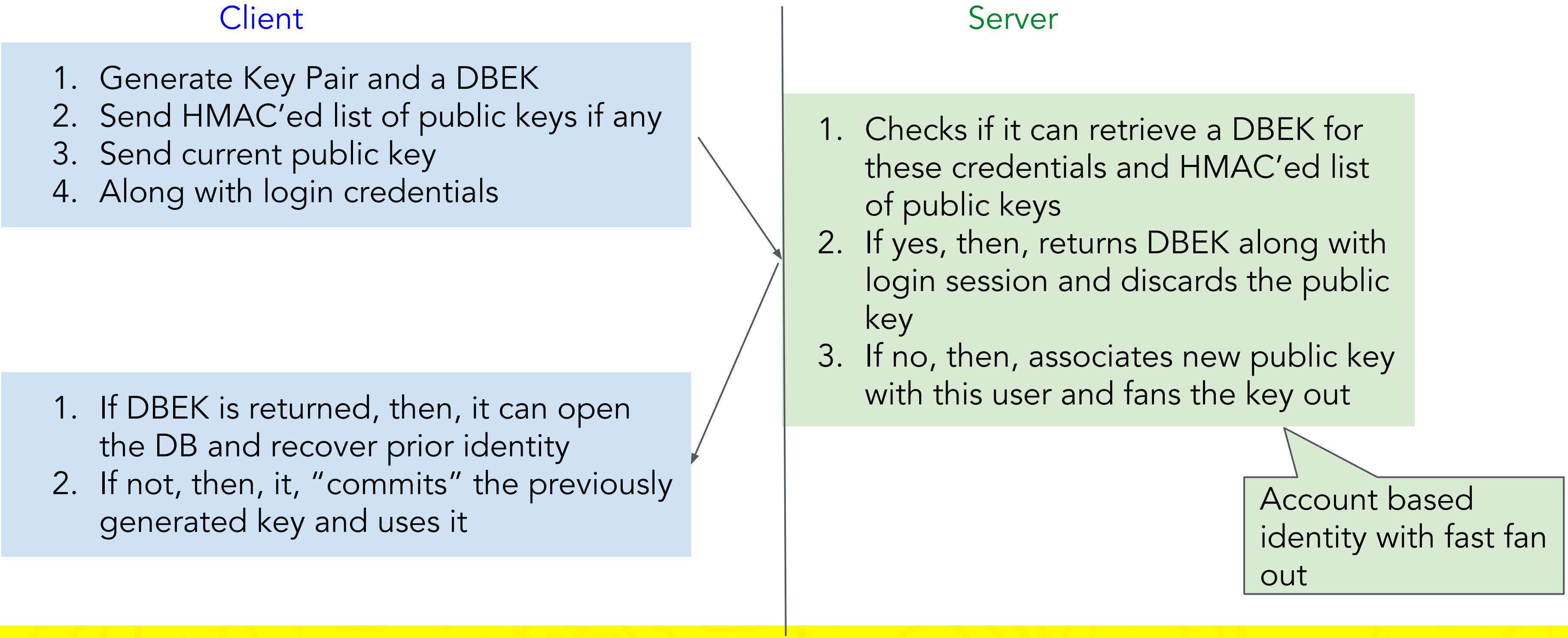
Building blocks - Identity

Post logout secure client DB

- Secrets stored within it can ONLY be recovered when the user is logged in [with help of server: essentially 2-2-secret shared]
  - Create an encrypted database that can be decrypted by keys obtained from the server post login

- No information leakage about the identity of other on device users (e.g. user-id’s or public keys)
  - Use keyed HMAC’s instead of native ID’s or just hashes

How does login work?



Client

1. Generate Key Pair and a DBEK
2. Send HMAC'ed list of public keys if any
3. Send current public key
4. Along with login credentials

Server

1. Checks if it can retrieve a DBEK for these credentials and HMAC'ed list of public keys
2. If yes, then, returns DBEK along with login session and discards the public key
3. If no, then, associates new public key with this user and fans the key out

Account based identity with fast fan out

Client

1. If DBEK is returned, then, it can open the DB and recover prior identity
2. If not, then, it, "commits" the previously generated key and uses it

Requirements - Status check

- Reduce the churn
    - Securely support multiple users on a given device
    - Support multiple devices for a given user
- Make retries faster!

Building block: Content Upload and when things are perfect!

Content creation and upload - pre E2EE

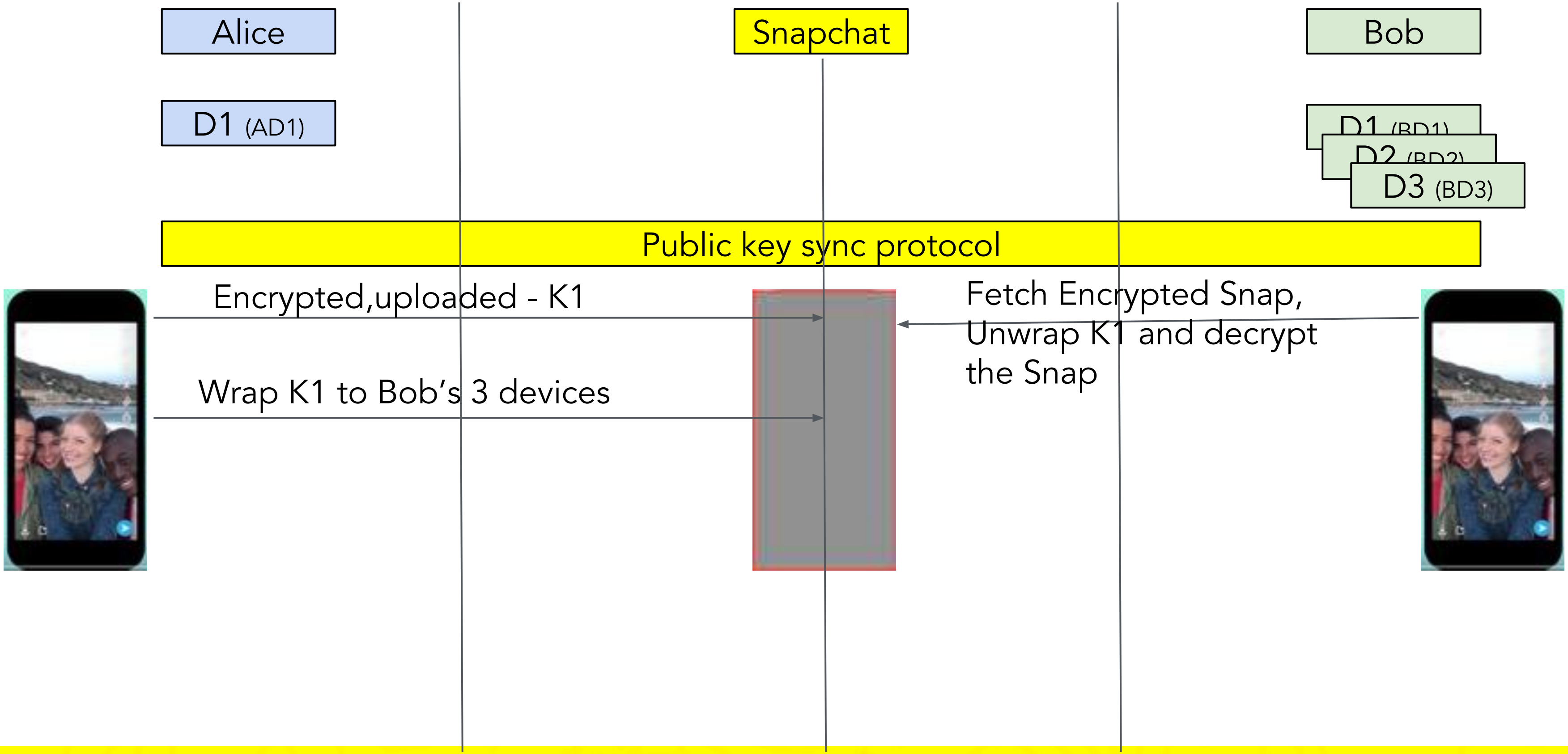
- Content upon creation is encrypted with a key (**CEK**) that is generated on the client

- When the user chooses to share the content with Snapchat, then, the key (**CEK**) is uploaded

- If the user chooses to discard the content, then, the key is never uploaded and content remains inaccessible to Snapchat servers

Change for E2EE

- Wrap **CEK** in an end-to-end encrypted manner
- Persist **CEK** on the client in the post logout secure database until an ACK is received OR the content expires
- Crypto is the easy part: Use a KDF, derive a secret from the pre-shared secret and encrypt and MAC (with AAD) the **CEK**



Alice

Snapchat

Bob

D1 (AD1)

D1 (BD1)

D2 (BD2)

D3 (BD3)

Public key sync protocol

Encrypted,uploaded - K1

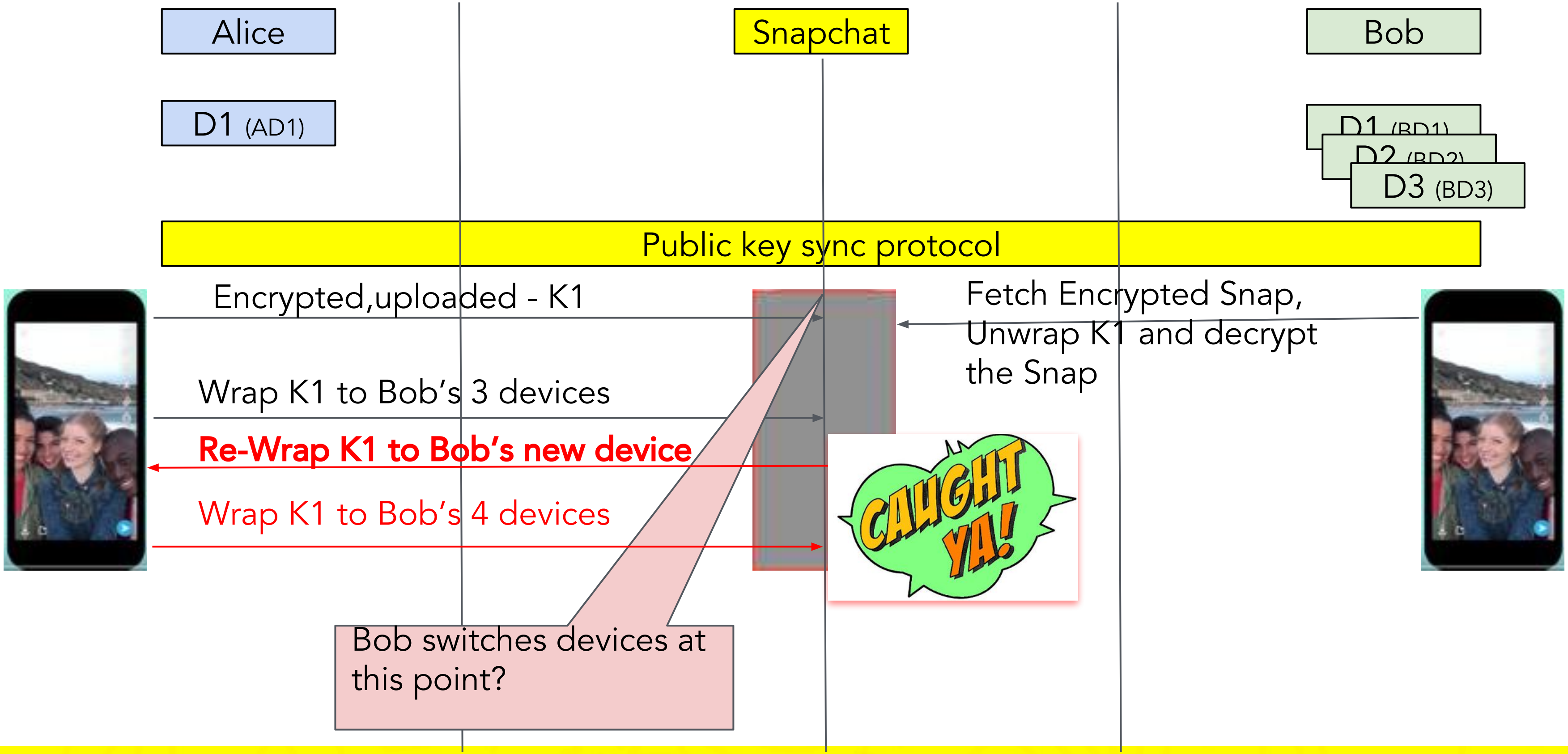
Wrap K1 to Bob's 3 devices

Fetch Encrypted Snap,
Unwrap K1 and decrypt
the Snap

Requirements

- Reduce the churn
  - Securely support multiple users on a given device
  - Support multiple devices for a given user
- Make retries faster!

Building block:
Catch me if you can!
On Sender Side



Alice

Snapchat

Bob

D1 (AD1)

D1 (BD1)

D2 (BD2)

D3 (BD3)

Public key sync protocol

Encrypted,uploaded - K1

Wrap K1 to Bob’s 3 devices

Re-Wrap K1 to Bob’s new device

Wrap K1 to Bob’s 4 devices

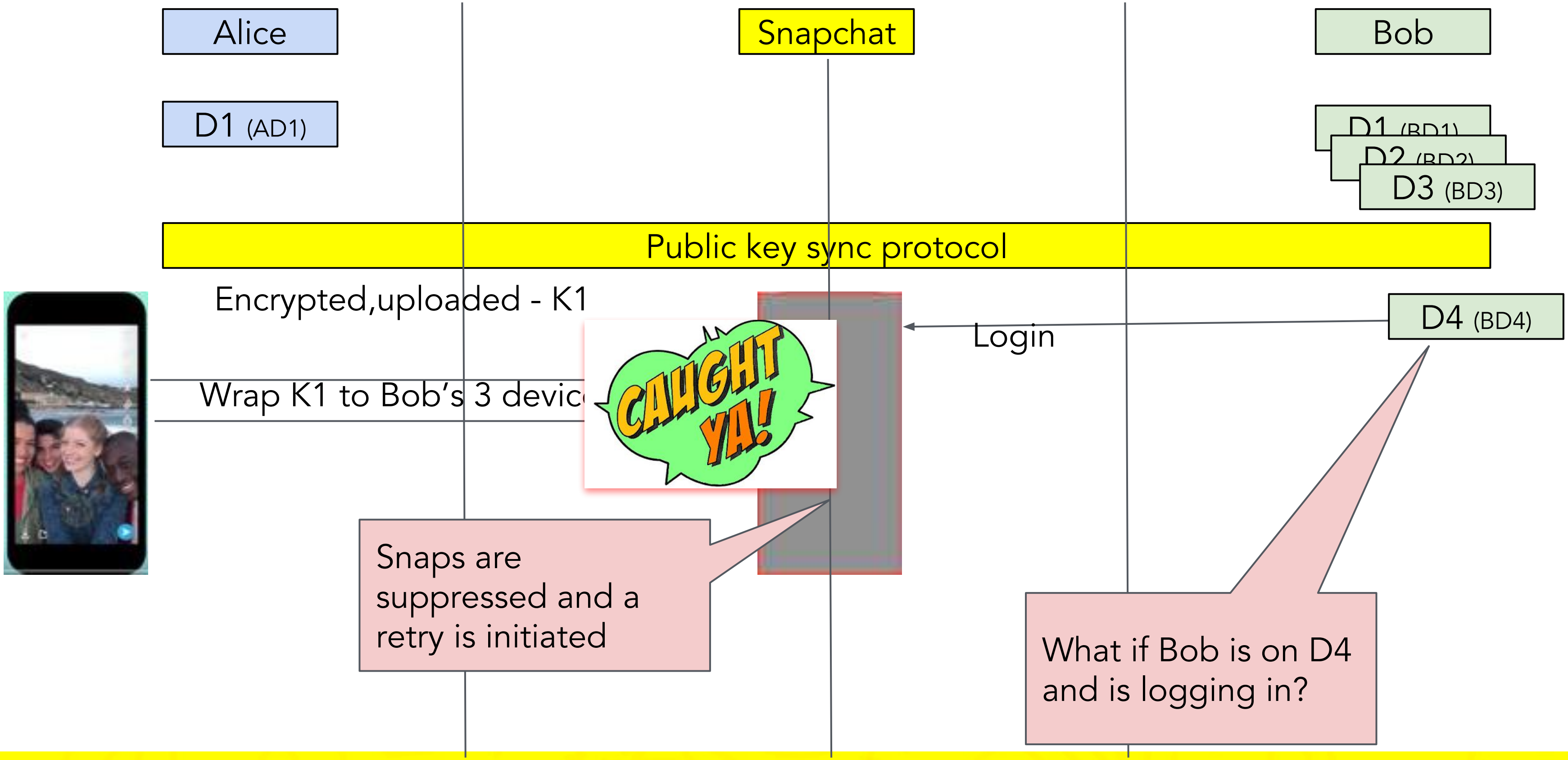
Fetch Encrypted Snap, Unwrap K1 and decrypt the Snap

Bob switches devices at this point?

Requirements - Status Check

- Reduce the churn
    - Securely support multiple users on a given device
    - Support multiple devices for a given user
- Make retries faster!

Building block:
Catch me if you can!
On Recipient Side



Alice

Snapchat

Bob

D1 (AD1)

D1 (BD1)

D2 (BD2)

D3 (BD3)

Public key sync protocol

Encrypted,uploaded - K1

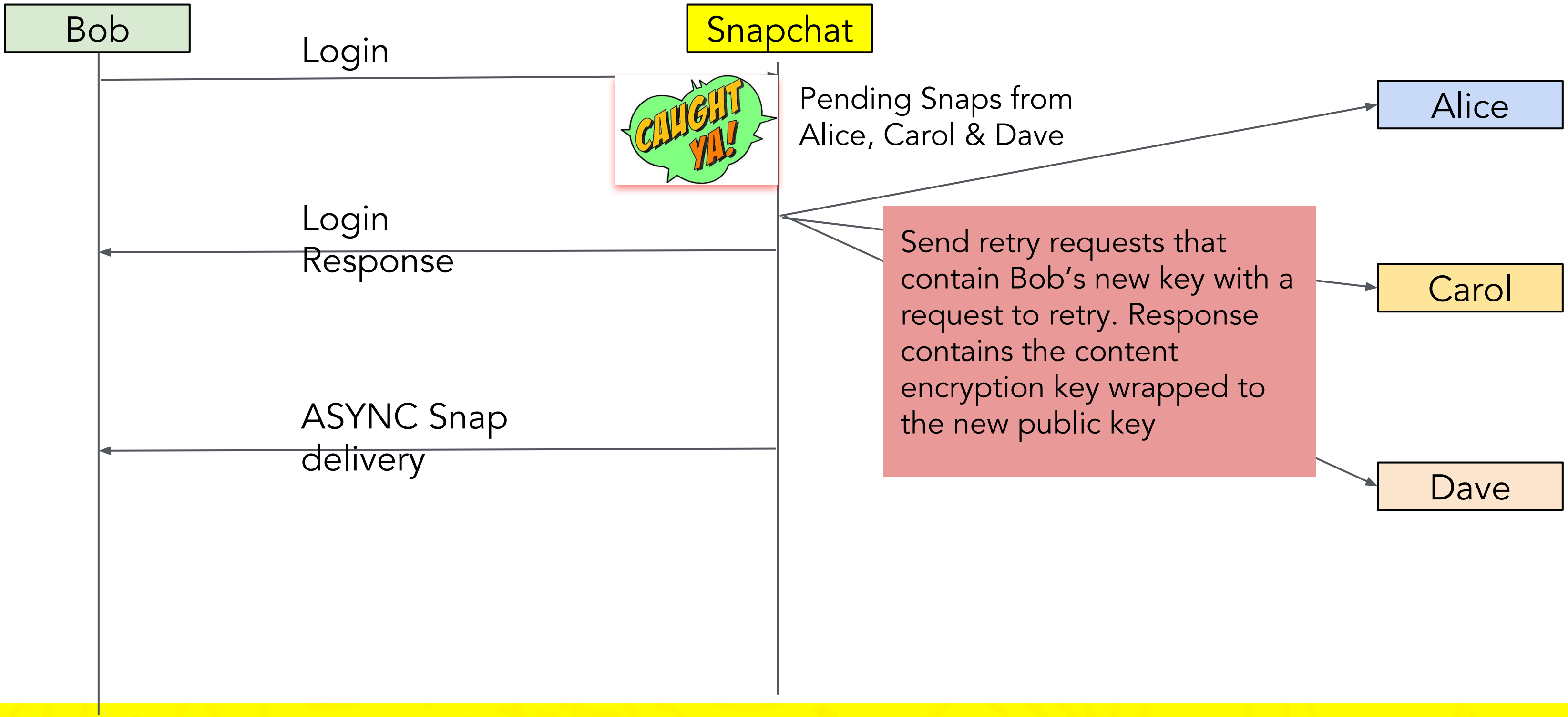
Wrap K1 to Bob’s 3 devic

Login

D4 (BD4)

Snaps are suppressed and a retry is initiated

What if Bob is on D4 and is logging in?



Bob

Snapchat

Login

Pending Snaps from
Alice, Carol & Dave

Alice

Login Response

Send retry requests that contain Bob’s new key with a request to retry. Response contains the content encryption key wrapped to the new public key

Carol

ASYNC Snap delivery

Dave

Retry Mechanisms

- A regular message that is obtained on next app open
- Or a push notification to make it more instantaneous

What about the security of the push notification?

- Push notifications are not completely in our control - rely on Apple or Google for ultimate delivery to the user

- The push notification contains the public key to re-wrap to. So, integrity of this message is paramount

- We encrypt the public key with a key that is known ONLY to the logged in user and Snapchat's servers

- Google had published a blog post in July 2018 on related work; we're proud to have implemented it across both platforms in July of 2017

Requirements - Status Check

- Reduce the churn
  - Securely support multiple users on a given device
  - Support multiple devices for a given user
- Make retries faster!

Where are we?

Launched

- Launched to 100% of users running compatible versions on Jan 12th **2018. Billions of <u>1/1</u> Snaps / day!**

- Retry rate is about 0.1% with retry times of:
    - p50 - 3.5 seconds
    - p80 - 1 minute

Extensions

- Periodic forward secrecy (essentially re-key)
- Sender to ‘other’ sender devices (as added recipients) to reduce loss rate
- Desire to extend to other 1/1 message types - text and group chat
- In the Trevor Perrin spectrum of EtA vs AtE, we are so far “E only”. Desire to add “A” via peer authenticity and/or Key Transparency

Summary

- Even if there isn’t a strong coupling between identities and devices, we demonstrated a mechanism by which one can achieve end to end encryption
  - Making the account the focal point of the identity
  - Caching users’ last devices
  - Secure caching of sensitive data that is unlocked upon successful authentication