Deniable secure multi party communication

P2P Systems and Security

Markus Teich
teichm@fs.tum.de

Chair for Network Architectures and Services

July 8th, 2014

Content

Introduction

Disclaimer

- All specifications are subject to change!
- No crypto auditing yet
- Not thread safe
- Only tested on GNU/Linux and Mac OS X

Goal

We try to achieve the following properties

- Authenticity
- Integrity
- Confidentiality
- Deniability
- Forward Secrecy
- Consensus

Assumptions

For libgotr to be usable we assume

- reliable, in-order packet transmission
- low latency
- Some more bandwidth for crypto overhead

Burmester-Desmedt GKA 1

Prerequisites

- p prime
- $g \in Z_p^*$
- satisfies DDH

Burmester-Desmedt GKA 2

Every User $U_i, i = 1, 2, \ldots, n$

- selects random $r_i \in Z_p$
- broadcasts $z_i := g^{r_i} \mod p$

Burmester-Desmedt GKA 3

Every $U_i, i = 1, 2, \ldots, n$ **broadcasts**

$$X_i := \left(\frac{z_{i+1}}{z_{i-1}}\right)^{r_i} \mod p$$

Burmester-Desmedt GKA 4

Every $U_i, i = 1, 2, \ldots, n$ computes

$$\begin{aligned} K_i &:= (z_{i-1})^{nr_i} * X_i^{n-1} * X_{i+1}^{n-2} * \cdots * X_{i-2} \mod p \\ &= g^{r_1 r_2 + r_2 r_3 + \cdots + r_n r_1} \mod p \end{aligned}$$

Burmester-Desmedt GKA fazit

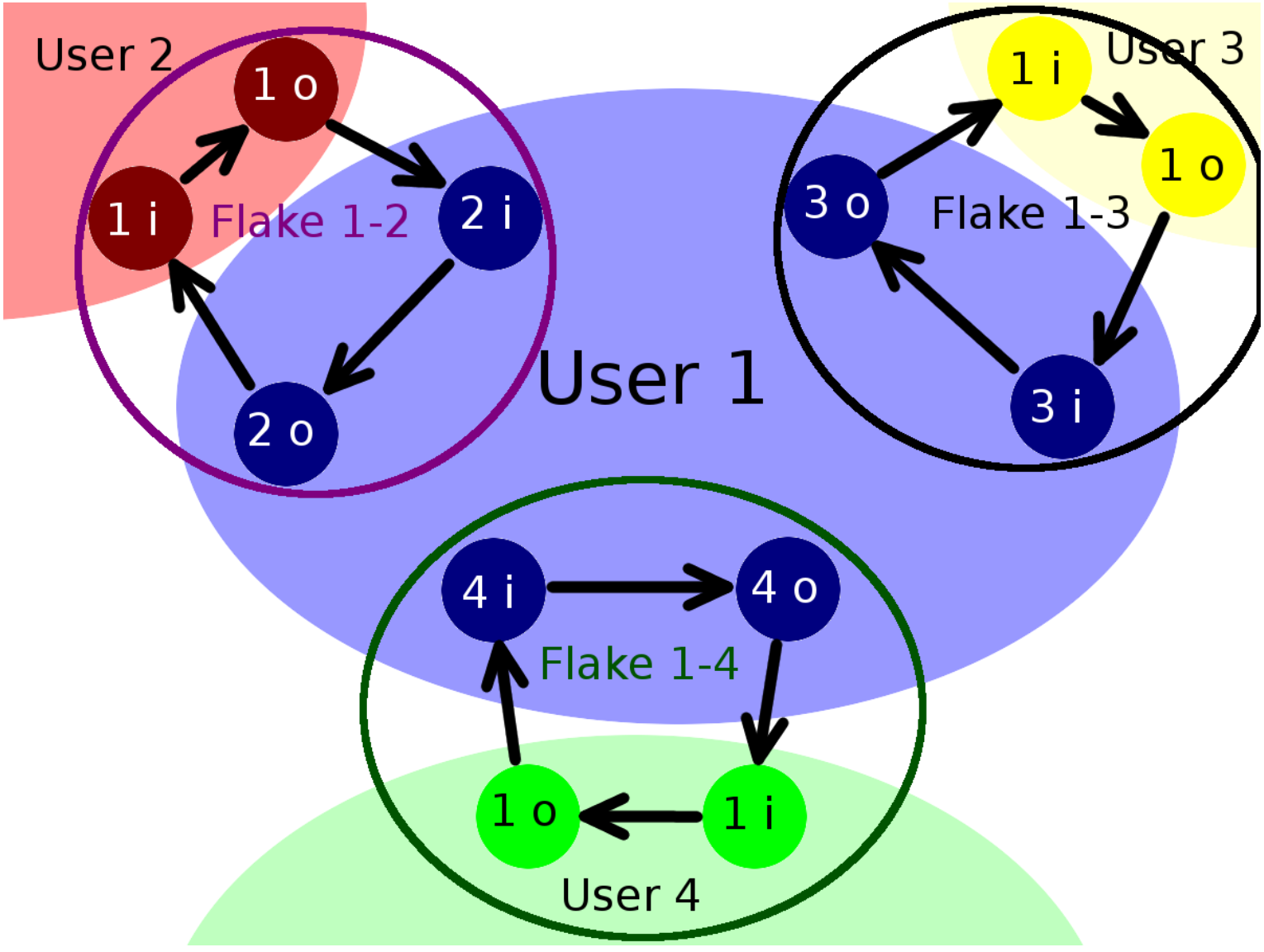
Advantages

- Extended DHE
- Cheap calculations

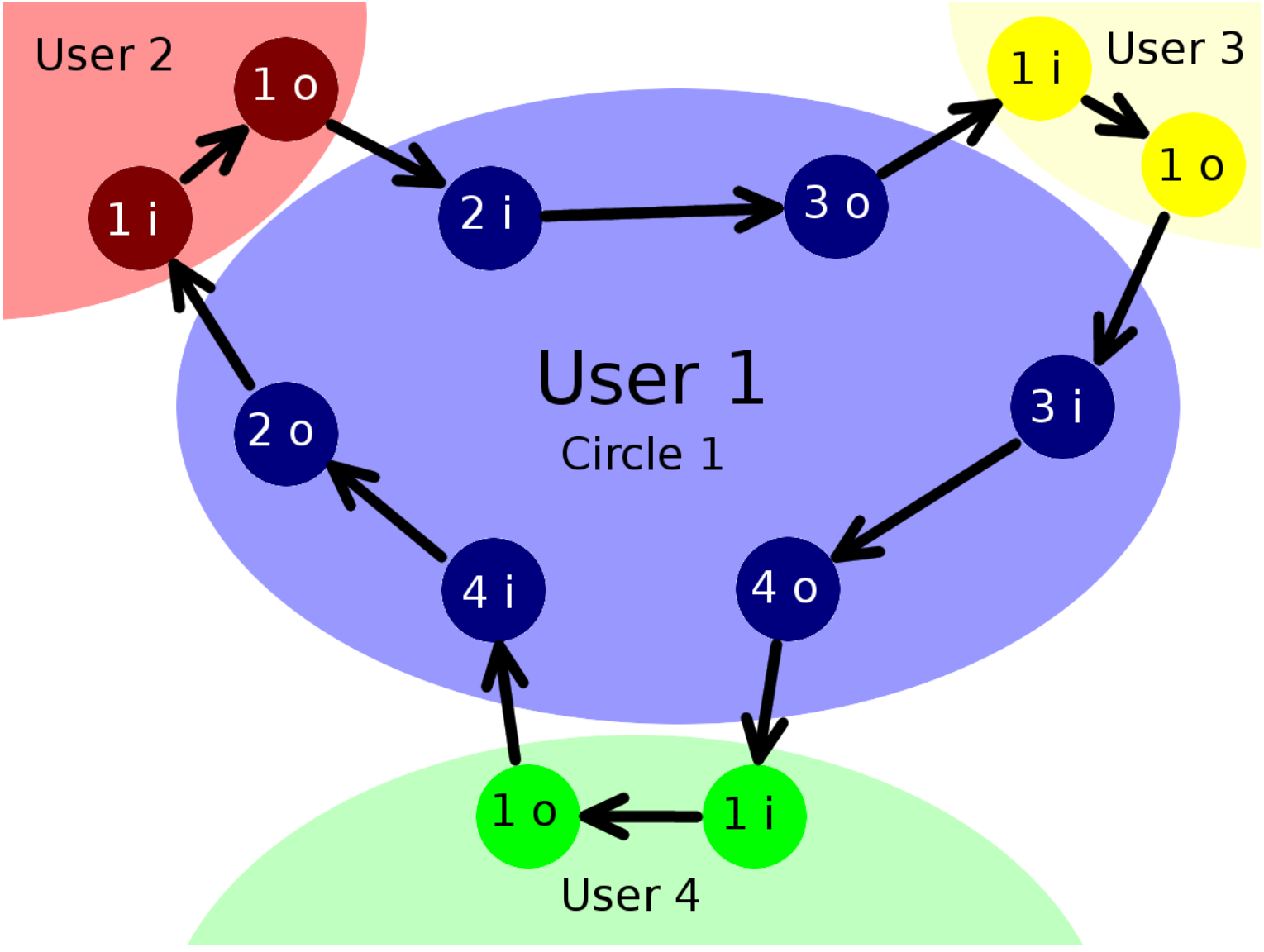
Drawbacks

- expensive rekeying
- Not hot-plug capable

Hot-pluggable GKA - Flake keys



Hot-pluggable GKA - Circle keys

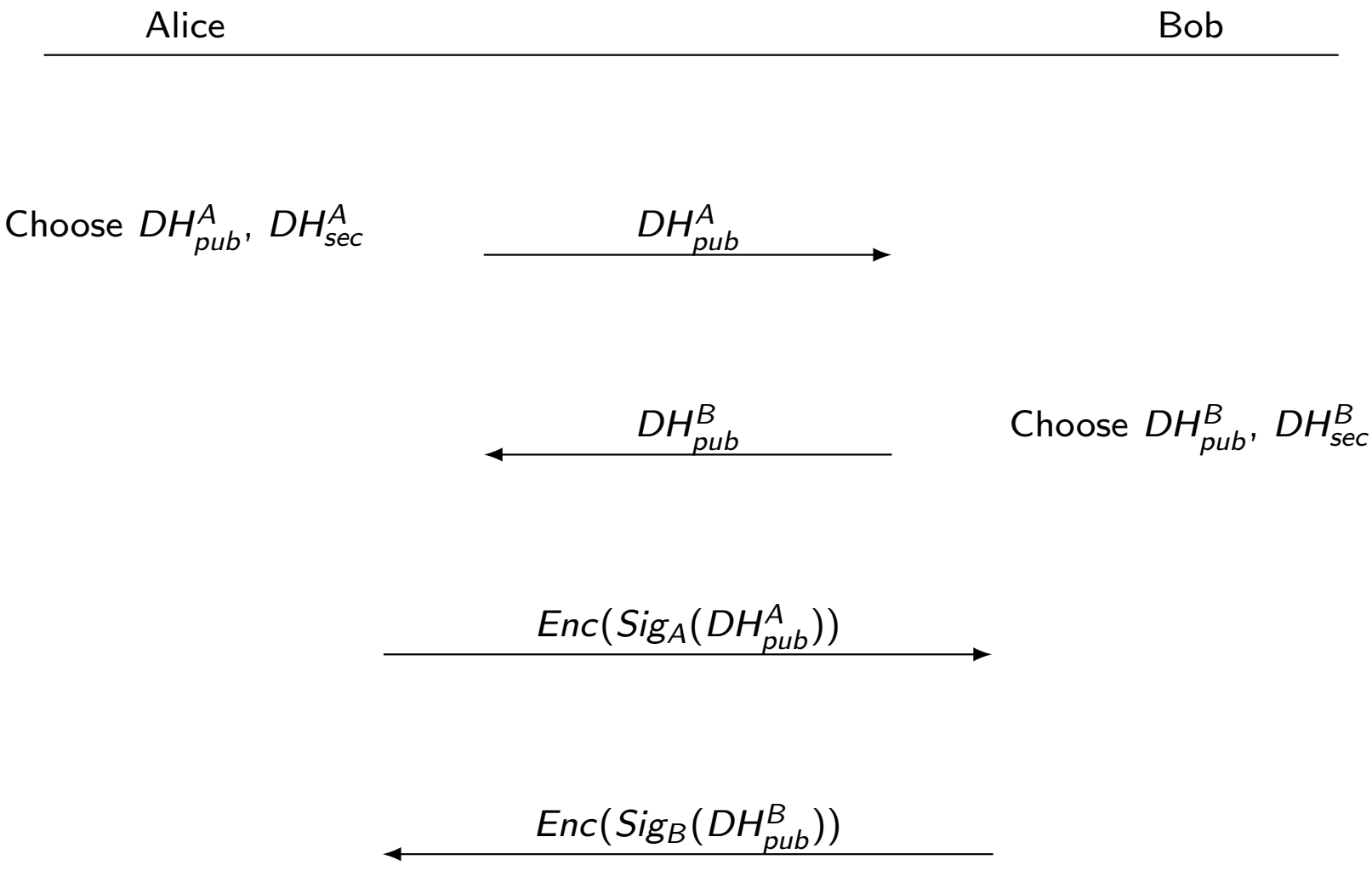


Protocol

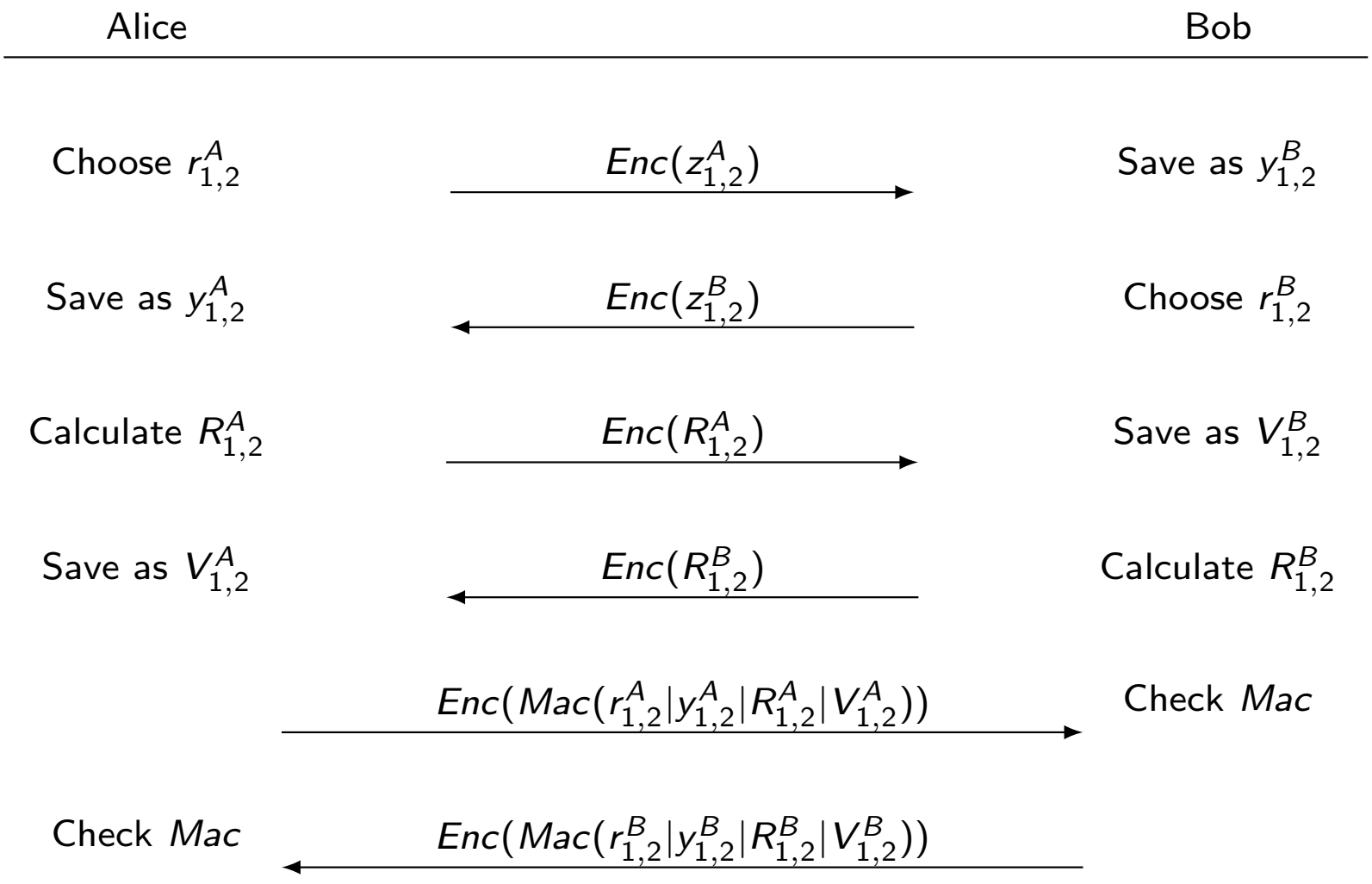
Definitions

- $Enc()$ uses EDDHE and includes an HMAC
- $Sig_{user}()$ uses long term EDDSA keys
- $Mac()$ is an HMAC with the flake key
- $Enc_G()$ uses a key k_1 derived from the circle key
- $Mac_G()$ uses a key k_2 derived from the circle key

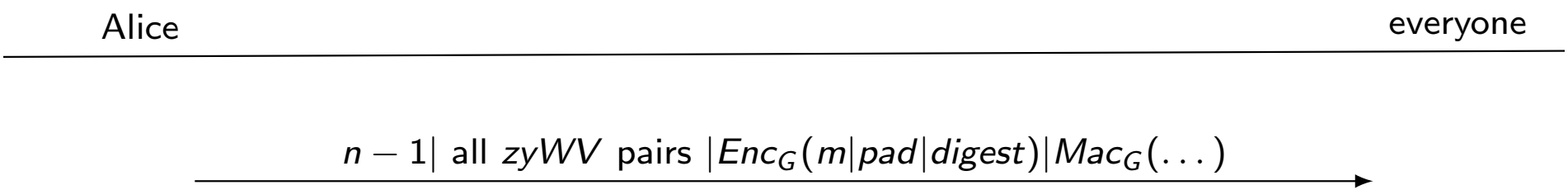
Establish secure pair channel



Establish flake key



Sending a message



Complexity

Joining

$O(n) * 5$ messages to establish circle key

$5 * max(RTT)$ round trip times

$O(n)$ bytes to send and receive

Other user joining

5 messages to establish circle key

5 round trip times

$O(1)$ bytes to send and receive

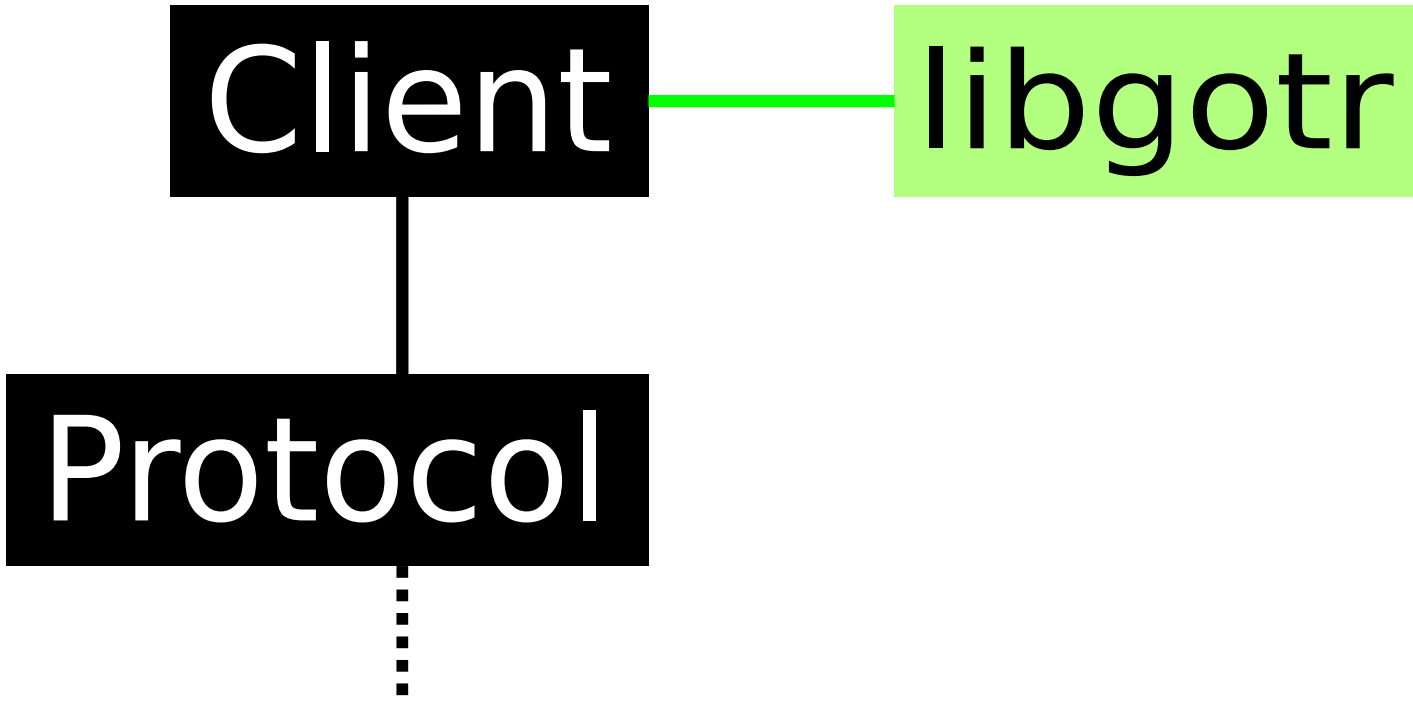
Sending a Message

$<= n$ messages (structure dependent)

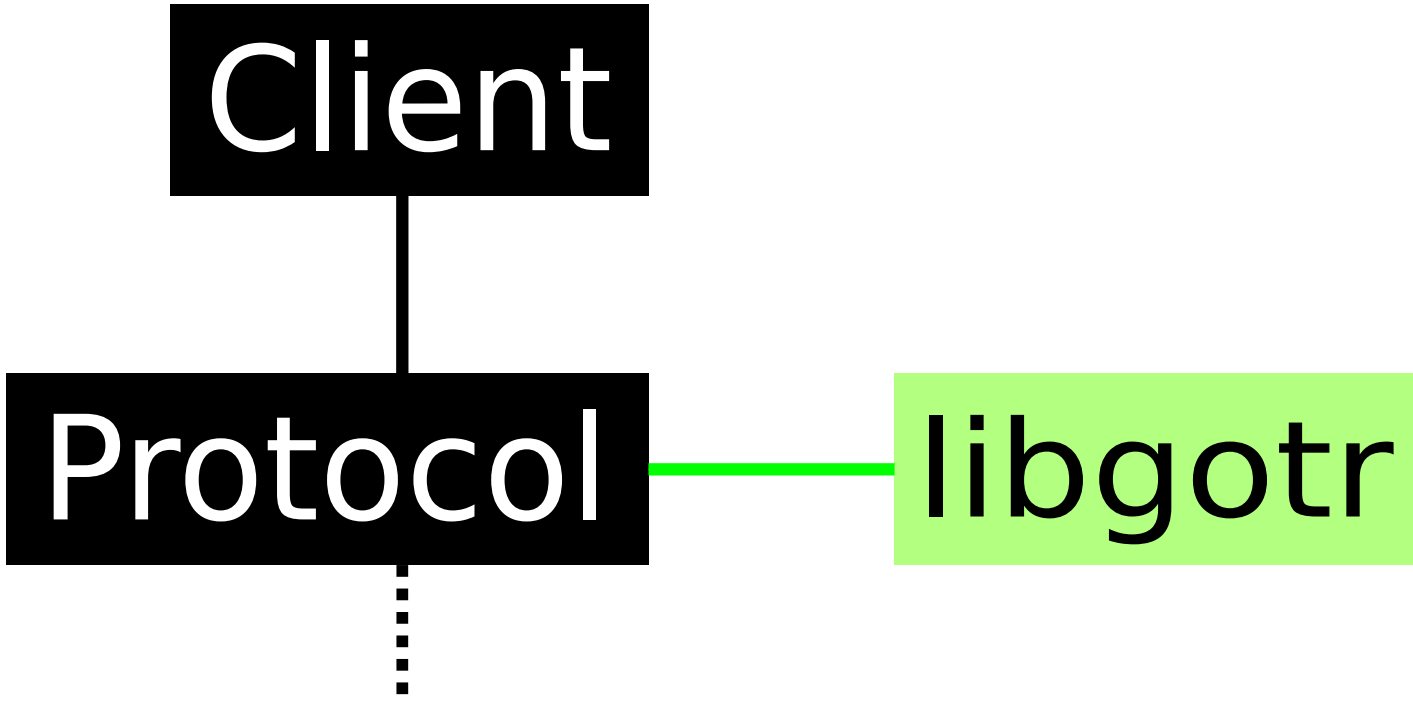
$max(RTT)$ delay

$O(n)$ bytes

library design



library design (alternative)



Types

```
struct gotr_chatroom;
struct gotr_user;

typedef int (*gotr_cb_send_all)(
    void *room_closure,
    const char *b64_msg);
typedef int (*gotr_cb_send_user)(
    void *room_closure,
    void *user_closure,
    const char *b64_msg);
typedef void (*gotr_cb_receive_user)(
    void *room_closure,
    void *user_closure,
    const char *plain_msg);
```

Managing

```
struct gotr_chatroom *gotr_join(
    gotr_cb_send_all send_all,
    gotr_cb_send_user send_user,
    gotr_cb_receive_user receive_user,
    const void *room_closure,
    const char *privkey_filename);
struct gotr_user *gotr_user_joined(
    struct gotr_chatroom *room,
    void *user_closure);
void gotr_keyupdate(
    struct gotr_chatroom *room);
void gotr_leave(struct gotr_chatroom *room);
```

Messaging

```
int gotr_send(
    struct gotr_chatroom *room,
    char *plain_msg);
int gotr_receive(
    struct gotr_chatroom *room,
    char *b64_msg);
struct gotr_user *gotr_receive_user(
    struct gotr_chatroom *room,
    struct gotr_user *user,
    void *user_closure,
    char *b64_msg);
```

Demo

Client

- UDS based
- Multiple Personality Disorder
- Only one chatroom

Current Status

What already works

- Client
- Long term key generation and storage
- Flake key generation

Future Work

To be implemented

- Circle key generation
- Protocol Messages
- Useful client (plugin)