Deniable secure multi party communication

P2P Systems and Security

Markus Teich
teichm@fs.tum.de

Chair for Network Architectures and Services

July 8th, 2014
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
Prerequisites

- $p$ prime
- $g \in \mathbb{Z}_p^*$
- satisfies DDH
Every User $U_i, i = 1, 2, \ldots, n$

- selects random $r_i \in \mathbb{Z}_p$
- broadcasts $z_i := g^{r_i} \mod p$
Every $U_i$, $i = 1, 2, \ldots, n$ broadcasts

$$X_i := \left(\frac{z_i+1}{z_i-1}\right)^{r_i} \mod p$$
Every $U_i, i = 1, 2, \ldots, n$ computes

\[ K_i := (z_{i-1})^{nr_i} \ast X_i^{n-1} \ast X_{i+1}^{n-2} \ast \cdots \ast X_{i-2} \mod p \]

\[ = g^{r_1 r_2 + r_2 r_3 + \cdots + r_n r_1} \mod p \]
Burmester-Desmedt GKA fazit

Advantages

▶ Extended DHE
▶ Cheap calculations

Drawbacks

▶ expensive rekeying
▶ Not hot-plug capable
Hot-pluggable GKA - Flake keys

User 1

Flake 1-2

1o → 2i
2i → 1o

User 2

1i

Flake 1-3

User 3

3i → 1i
1i → 3i

User 4

Flake 1-4

4i → 1i
1i → 4i
Hot-pluggable GKA - Circle keys

User 1
Circle 1

User 2

User 3

User 4
Protocol

Definitions

- $Enc()$ uses EDDHE and includes an HMAC
- $\text{Sig}_{\text{user}}()$ 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

Alice

Choose $DH^A_{pub}$, $DH^A_{sec}$

Bob

Choose $DH^B_{pub}$, $DH^B_{sec}$

$DH^A_{pub}$

$DH^B_{pub}$

$Enc(Sig_A(DH^A_{pub}))$

$Enc(Sig_B(DH^B_{pub}))$
Establish flake key

Alice

Choose $r_{1,2}^A$

Save as $y_{1,2}^A$

Calculate $R_{1,2}^A$

Save as $V_{1,2}^A$

Check $Mac$

Bob

Save as $y_{1,2}^B$

Choose $r_{1,2}^B$

Calculate $R_{1,2}^B$

Check $Mac$

$Enc(\text{Mac}(r_{1,2}^A|y_{1,2}^A|R_{1,2}^A|V_{1,2}^A))$

$Enc(\text{Mac}(r_{1,2}^B|y_{1,2}^B|R_{1,2}^B|V_{1,2}^B))$
Sending a message

\[ \text{Alice} \rightarrow \text{everyone} \]

\[ n - 1 | \text{all } \text{zyWV pairs} | \text{Enc}_G(m|\text{pad}|\text{digest})|\text{Mac}_G(\ldots) \]
Complexity

Joining

\[ O(n) \times 5 \text{ messages to establish circle key} \]
\[ 5 \times \max(\text{RTT}) \text{ round trip times} \]
\[ O(n) \text{ bytes to send and receive} \]

Other user joining

5 messages to establish circle key
5 round trip times
\[ O(1) \text{ bytes to send and receive} \]

Sending a Message

\[ \leq n \text{ messages (structure dependent)} \]
\[ \max(\text{RTT}) \text{ delay} \]
\[ O(n) \text{ bytes} \]
library design

Client

Protocol

libgotr
library design (alternative)
Types

```c
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);
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
To be implemented

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