A Generic Data Exchange System for F2F Networks

Cyril Soler
Outline

- Overview of Retroshare
- The GXS system
- Decentralize your app!
The Retroshare Project

- Mesh computers using signed TLS over TCP/UDP/Tor/I2P;
- anonymous end-to-end encrypted FT with swarming;
- mail, IRC chat, forums, channels;
The Retroshare Project

- Mesh computers using signed TLS over TCP/UDP/Tor/I2P;
- anonymous end-to-end encrypted FT with swarming;
- mail, IRC chat, forums, channels;
- available on Mac OS, Linux, Windows.
The Retroshare Project

▶ Mesh computers using signed TLS over TCP/UDP/Tor/I2P;
▶ anonymous end-to-end encrypted FT with swarming;
▶ mail, IRC chat, forums, channels;
▶ available on Mac OS, Linux, Windows.
The Retroshare Project

- Mesh computers using signed TLS over TCP/UDP/Tor/I2P;
- anonymous end-to-end encrypted FT with swarming;
- mail, IRC chat, forums, channels;
- available on Mac OS, Linux, Windows.
The Retroshare Project

- Mesh computers using signed TLS over TCP/UDP/Tor/I2P;
- anonymous end-to-end encrypted FT with swarming;
- mail, IRC chat, forums, channels;
- available on Mac OS, Linux, Windows.


Apparently the tool is very nice and generates an output that is compatible with kcachegrind, with negligible CPU overcost. What else can we expect!

How to use it:

1. #> apt-get install libgoogle-perfprofd-dev
2. #> LD_PRELOAD=/usr/bin/libprofiler.so CPUPROFILE=retroshare.prof /retroshare-nogui/src/retroshare
3. #> google-prof -callgrind /retroshare-gui/src/retroshare retroshare.prof > rs.callgrind

Then launch kcachegrind on rs.callgrind.
The Retroshare Project

- Mesh computers using signed TLS over TCP/UDP/Tor/I2P;
- anonymous end-to-end encrypted FT with swarming;
- mail, IRC chat, forums, channels;
- available on Mac OS, Linux, Windows.
The Retroshare Project

History:

- 10 years old.
- 5 main contributors (drbob, csoler, G10H4ck, chris, thunder, ...)
- a few thousands daily users (?)

User experience:

- network bootstrapping is a bit difficult
- lots of options and possibilities, etc.
- once you’re set, you’re pretty much invisible

Code:

- 500,000 lines of C++
- depends on openssl, libcrypto, OpenPGP-SDK (for now)
- backend + UI (Qt / Web)
- channels, forums, email, ... : based on a common generic distribution system
Motivation

Friend-to-Friend network:
- mesh of computers connected by authenticated/encrypted links
- nodes only talk to their trusted neighbors
Motivation

Friend-to-Friend network:

- mesh of computers connected by authenticated/encrypted links
- nodes only talk to their trusted neighbors
- publish/exchange data with any node
- favor interesting content...while preventing flooding, spam, etc.
- provide authentication/anonymity beyond friends
Motivation

Friend-to-Friend network:

- mesh of computers connected by authenticated/encrypted links
- nodes only talk to their trusted neighbors
- publish/exchange data with any node
- favor interesting content...while preventing flooding, spam, etc.
- provide authentication/anonymity beyond friends
Motivation

Friend-to-Friend network:

- mesh of computers connected by authenticated/encrypted links
- nodes only talk to their trusted neighbors

- publish/exchange data with any node
- favor interesting content...while preventing flooding, spam, etc.
- provide authentication/anonymity beyond friends
- be robust to network changes, disconnections, heterogeneity
Generic eXchange System (a.k.a. GXS)

GXS: Asynchronous distribution, authentication, privacy, security of generic data.

Working principles:

1. subscribers advertise to friends

2. nodes team up to ensure data integrity and spam control

Developers implement their own "services/data" on top of it
GXS Core

GXS core automatically provides:

- local encrypted storage (sqlcipher)
- network sync.
  - accounts for access-restriction, storage-sync time periods, etc
  - multi-chunk transactions
- validation
  - data signatures, spam control, cleaning
GXS Core

GXS core automatically provides:

- local encrypted storage (sqlcipher)
- network sync.
  - accounts for access-restriction, storage/sync time periods, etc
  - multi-chunk transactions
- validation
  - data signatures, spam control, cleaning

Specific services implement:

- private data types (serialization, GUI ↔ GXS types)
- sync. (auto), subscription (manual) and authentication policies
- service specific actions
Services, Groups, Messages, Identities, Circles

Service

Groups

Messages
GXS Primitives

Services, Groups, Messages, Identities, Circles

Service

Groups

Distribution
Authentication

Messages

Signatures
Services, Groups, Messages, Identities, Circles
GXS Primitives

Services, Groups, Messages, Identities, Circles

Service

Groups

Messages

Identities

electron
jolavillette
Quine
Cyril ter
Thunder
jolavillette-ompnt
drob

Distribution
Authentication

Signatures

Circles
Groups and Messages

- versionned, hierarchical data
- meta-data (GXS) + private data (service dependent)

<table>
<thead>
<tr>
<th>Group Meta Data Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group Id</td>
<td>128 bits fingerprint of the public admin key</td>
</tr>
<tr>
<td>Publish time</td>
<td>32-bits integer</td>
</tr>
<tr>
<td>Circle Id</td>
<td>Group Id of parent circle</td>
</tr>
<tr>
<td>Author Id</td>
<td>Group Id of author identity</td>
</tr>
<tr>
<td>Description text</td>
<td>Arbitrary string</td>
</tr>
<tr>
<td>Authentication policy</td>
<td>32-bits flags</td>
</tr>
<tr>
<td>Distribution control flags</td>
<td>8-bits flags</td>
</tr>
<tr>
<td>Admin key</td>
<td>2048-bits RSA <strong>public</strong> key</td>
</tr>
<tr>
<td>Publish key [optional]</td>
<td>2048-bits RSA <strong>public</strong> key</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Message Meta Data Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Id</td>
<td>128 bits hash (meta data + private data)</td>
</tr>
<tr>
<td>Group Id</td>
<td>Id of the parent group</td>
</tr>
<tr>
<td>Publish time</td>
<td>32-bits integer</td>
</tr>
<tr>
<td>Parent Msg Id</td>
<td>Id of parent message</td>
</tr>
<tr>
<td>Orig Msg Id</td>
<td>Id of previous version of message</td>
</tr>
<tr>
<td>Author Id</td>
<td>Group Id of author identity</td>
</tr>
</tbody>
</table>
Pseudo-anonymous identities

- identities are GXS groups in a "Identity" service
- sync-ed on request, identities follow groups/messages
- optionally signed by node key (signature in Group private meta)
- unsigned identities are anonymous beyond friend nodes
Circles

- Circles are GXS groups in a "Circles" GXS service
- subscription and sync are automatic
- membership requires:
  - invitation: list signed by admin key
  - membership request: user msg
- self-restricted circles: only visible to invitee list
Data authentication

- signature schemes of groups and messages
  - groups: admin, author (depends on service auth. flags)
  - messages: author, publish (depends on Group auth. flags)
Data distribution

- synchronization model
  - only compares local times
  - circle restriction $\rightarrow$ data encryption (Anonymized AES+RSA)
Data distribution

- synchronization model
  - only compares local times
  - circle restriction \(\rightarrow\) data encryption (Anonymized AES+RSA)
Reputation management

- Block unwanted content
  - default settings allow enough visibility
  - allow newcomers to bootstrap
  - discourage creation of new identities to spam
Reputation management

- Block unwanted content
  - default settings allow enough visibility
  - allow newcomers to bootstrap
  - discourage creation of new identities to spam

⇒ always receive data, only forward depending on:
  - identity node signature
  - opinions sync-ed from friend nodes (local service)
  - anti-spam policy for the group
Problem: anonymous FT without disclosing data/meta-data to intermediate nodes
File transfer

- Problem: anonymous FT without disclosing data/meta-data to intermediate nodes
- tunnels based on turtle [Matejka 2006]

- no global addressing
- passive tunnel management
- multiple tunnels allowed to the same destination
- anonymity + encryption \(\Rightarrow\) needs a pre-shared key
  - request tunnels using \(H(H(f))\)
  - encryption: chacha20+HMAC with \(H(H(f)\mid tunnel\_id\mid 96\text{-bits IV})\)
So, what now?
Step-by-step example

Q: So what effort does it take to create e.g. distributed forums?
Step-by-step example

Q: So what effort does it take to create e.g. distributed forums?
A: provide the following (200 lines of code for forums):

▶ service class and authentication policy
▶ private group/message data and group/message creation code
Step-by-step example

Q: So what effort does it take to create e.g. distributed forums?
A: provide the following (200 lines of code for forums):

► service class and authentication policy

```cpp
p3GxsForums::p3GxsForums( RsGeneralDataService *gds, RsNetworkExchangeService *nes, RsGixs* gixs ) :
    RsGenExchange( gds, nes, new RsGxsForumSerialiser(), RS_SERVICE_GXS_TYPE_FORUMS, gixs, forumsAuthenPolicy()),
    RsGxsForums(this), mGenToken(0), mGenActive(false), mGenCount(0)
{
}

guint32_t p3GxsForums::forumsAuthenPolicy()
{
    guint32_t policy = 0;
    guint32_t flag = GXS_SERV::MSG_AUTHEN_ROOT_AUTHOR_SIGN | GXS_SERV::MSG_AUTHEN_CHILD_AUTHOR_SIGN;
    RsGenExchange::setAuthenPolicyFlag(flag, policy, RsGenExchange::PUBLIC_GRP_BITS);
    return policy;
}
```
Step-by-step example

Q: So what effort does it take to create e.g. distributed forums?
A: provide the following (200 lines of code for forums):
  ▶  service class and authentication policy
  ▶  private group/message data and group/message creation code

```cpp
class RsGxsForumMsgItem : public RsGxsMsgItem
{
  public:
    RsGxsForumMsgItem(): RsGxsMsgItem(RS_SERVICE_GXS_TYPE_FORUMS, RS_PKT_SUBTYPE_GXSFORUM_MESSAGE_ITEM) {}
    virtual ~RsGxsForumMsgItem() {}  
    void clear() { mMsg.clear(); }  
  
    virtual void serial_process(RsGenericSerializer::SerializeJob j,RsGenericSerializer::SerializeContext& ctx) 
    {  
        RsTypeSerializer::serial_process(j,ctx,TLV_TYPE_STR_MSG,mMsg.mMsg,"mGroup.Description");
    }  
    
    RsGxsForumMsg mMsg;
};

bool p3GxsForums::createMsg(uint32_t &token, RsGxsForumMsg &msg)
{
    RsGxsForumMsgItem* msgItem = new RsGxsForumMsgItem();
    msgItem->mMsg = msg;
    msgItem->meta = msg.mMeta;
    RsGenExchange::publishMsg(token, msgItem);
    return true;
}
```
Step-by-step example

Q: So what effort does it take to create e.g. distributed forums?
A: provide the following (200 lines of code for forums):
  ► service class and authentication policy
  ► private group/message data and group/message creation code

Comes free with GXS:
  ► advertisement of forums to friend nodes
  ► distribution of posts to subscribed friends
  ► validation of group/message signatures
  ► spam control

Application layer (GUI, lots of Qt):
  ► creating, visualizing forums/posts
  ► editing posts (Meta-data has msg versions)
Q: So what effort does it take to create e.g. distributed forums?
A: provide the following (200 lines of code for forums):
  ▶ service class and authentication policy
  ▶ private group/message data and group/message creation code

Subscribed neighbours
  Unread posts

Post content or forum info

Spam control
  Post author
Develop fully decentralized apps:

- Some ideas...
  - micro-blogging (Twitter)
  - blogs (pictures, comment threads)
  - wiki
  - directory sync
  - calendar+Tasks
  - distributed Git
  - ...

- Our next target: FB style social network
  - user’s page: GXS group
  - page posts: GXS subgroups (allows post-based circle visibility)
  - user’s comments: GXS messages in each post group

- Essentially UI work ;-)  
  - distribution, crypto, ...: already done!
Questions?

Sources: http://github.com/Retroshare/Retroshare
Developers’ blog: http://retroshareteam.wordpress.com
Project website: http://retroshare.net
Technical report: https://hal.inria.fr/hal-01617423
Google Summer of Code 2018
(project ideas here: https://projects.freifunk.net)

Thanks to: