Smoke

An Android Echo Software Application

About..3
Activity Authenticate...4
Activity Chat...5
Activity Fire...7
Activity Settings...8
Android...9
Congestion Control...10
Corrupted Database Values...11
Database Containers..12
Developers..13
Discovery via Cryptography...14
Software Distribution..15
Exchanging Private Credentials...16
Fire..17
Forward Secrecy and SmokeStack...18
Inflate..19
Local Broadcast Manager..20
McEliece CCA2...21
Message Structures..22
Neighbors..29
New Installation..30
Outbound Queues..31
Ozone Address..32
Participants..33
Private Public-Key Server...34
SipHash Identities...35
Software Distribution..36
TCP, UDP Protocols...37
Thread Utilization..38
UDP Datagrams..39
Verifying Public-Key Ownership...40

About

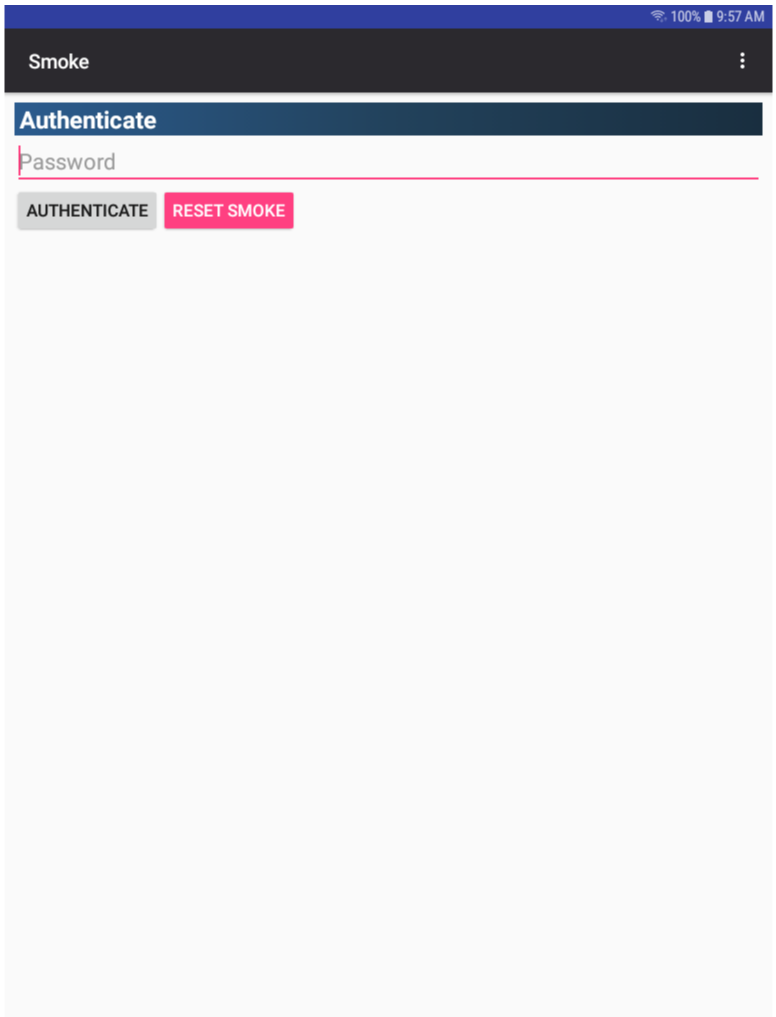
Smoke is an Android communications project. The software is composed of a single multi-threaded application. A companion application, SmokeStack, provides mobile server services.

Software sources are available at https://github.com/textbrowser/smoke and https://github.com/textbrowser/smokestack.

Activity Authenticate

After launching a prepared Smoke installation, the Authenticate activity is displayed. The original password must be provided. If the correct password is provided, essential containers are populated and the kernel is activated. The previously-accessed activity is also activated.

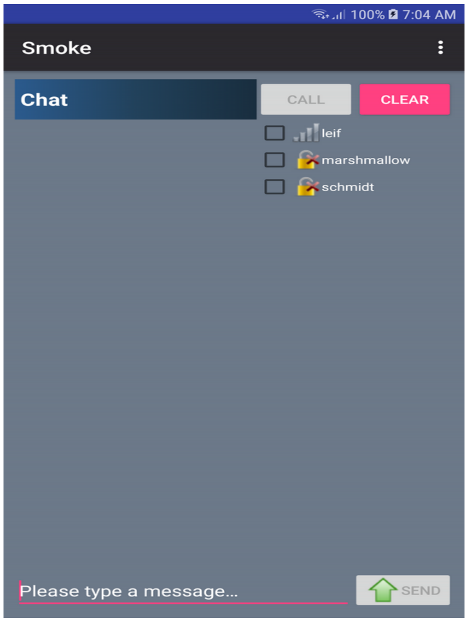
Smoke may also be reset within the Authenticate activity.



Activity Chat

The Chat activity is one of two messaging activities. From this activity, one may message one or more defined participants. If Smoke is connected to at least one network peer, a green network icon is displayed. Otherwise, a red network icon is displayed.

Before a messaging session may begin between two participants, the parties must exchange private key material. Exchanging private key material may be achieved via the Call and Custom Session mechanisms.



A context menu may be activated by pressing and holding on the right-hand Participants widget. Context-menu items are described below.

Custom Session

> Private key material may be generated per the selected participant. The generated material is not transfered on the network.

Optional Signatures

> Messaging and status messages are digitally signed. Signatures may be disabled per participant. Please note that if one party requires signatures and signatures are not provided by the other party, messages will be ignored by the receiving party.

Purge Session

> Discard private key material for the specified participant.

Refresh Participants Table

> Refresh the Participants widget.

Retrieve Messages

> Retrieve messages from SmokeStack instances. An Ozone and an active network must be present for this option to be enabled.

Show Details

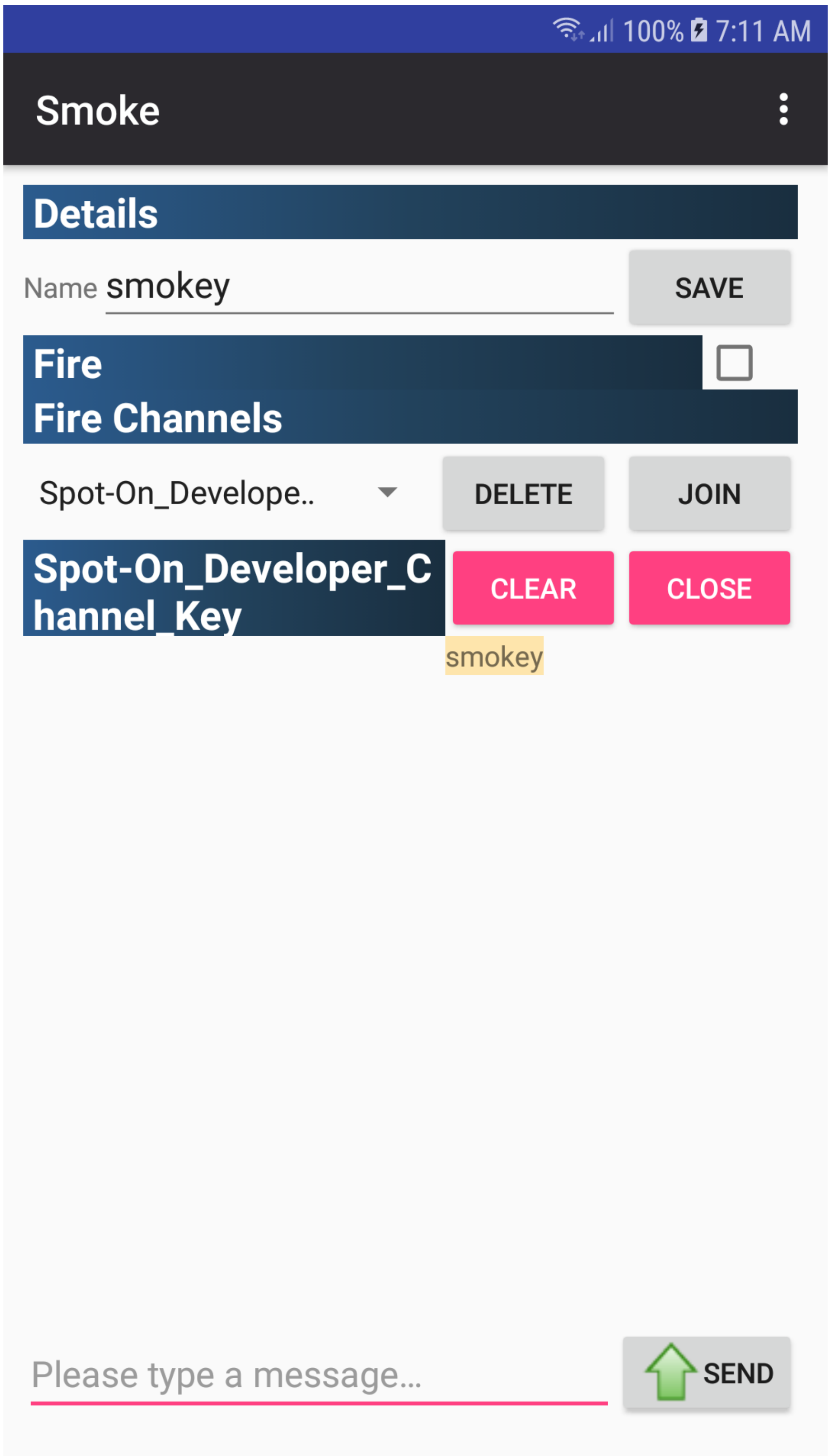
Disable or enable various Participants details.

Show Icons

Disable or enable participant presence icons.

Activity Fire

The Fire activity is one of two messaging activities. From this activity, one may communicate with one or more groups of anonymous participants. If Smoke is connected to at least one network peer, a green network icon is displayed. Otherwise, a red network icon is displayed. Fire is compatible with Spot-On's Buzz. 256-bit AES-CBC along with SHA-384 HMAC address the encryption and authentication requirements.



Activity Settings

The Settings activity contains various configurable items. Smoke may also be reset from this activity. This page will describe miscellaneous portions.

About

> Describes software information, including the Android version of the device. Log clearing may also be performed in this section. The Prefer Active CPU option, if enabled, ensures that the CPU remains active if the screen is turned off.

Ozone

> One Ozone address may be defined in this section. Please refer to the Ozone page for more details.

Password

> Generate new local authentication and encryption keys as well as public and private key pairs. If confirmed, all existing data will be purged.

Public Data

> Contains the SipHash Chat ID. A SipHash Identity is synonymous to an e-mail address. Basic public-key data is also displayed in this section.

Android

Smoke has been successfully tested on Android versions 4.4, 5.0, 5.1, 6.0, 7.0, and 7.1. Android versions 4.4, 5.0, and 5.1 are not officially supported.

According to https://developer.android.com/about/dashboards/index.html, Smoke supports 92.7% of all Android versions.

Congestion Control

Smoke implements a software-based congestion control mechanism. The SipHash algorithm is used for computing digests. Computed digests are stored in an SQLite database table.

Congestion-control items are inspected every 5 seconds. Items older than 60 seconds are discarded.

Corrupted Database Values

Encrypted database values pose an interesting design problem. How should an application depict a faulty database value to the user if the application is unable to properly decrypt an encrypted value? Some software packages ignore the potential problem altogether. Others, delete or hide the corrupted entries; logging the failures in squandered logs. Smoke offers an exceptionally-transparent solution. Damaged database entries are depicted in various containers. These depictions offer insight into potential system failures.

Database Containers

Most of the database fields contain authentically-encrypted values. Some fields contain keyed digests, including keyed digests of binary (false / true) values. Values are stored as $E(Data, K_e) \| HMAC(E(Data, K_e), K_a)$ and $HMAC(Data, K_a)$. 256-bit AES-CBC is used for encrypting data. SHA-512 HMAC is used for data authentication.

Developers

Android Studio is required for development. Please download the application from https://developer.android.com/studio/index.html. Building Smoke may be performed via Studio or a terminal. Please refer to the included Makefile and Makefile.linux for guidance.

Discovery via Cryptography

Cryptographic discovery is a mechanism which allows servers to lighten the computational and data responsibilities of mobile devices.

Shortly after a Smoke instance connects to a SmokeStack service, the Smoke instance shares some non-private material. The material allows a SmokeStack server to transfer messages to their correct destinations. SmokeStack instances routinely distribute gathered, non-expired material to other SmokeStack services, thus creating a network of cooperative SmokeStack faculties.

Cryptographic Discovery assumes a trustworthy network.

To mitigate replay attacks, Smoke offers SmokeStack instances random identity streams during message-retrieval requests. The identity streams self-expire.

Please note that SmokeStack limits the number of client identities to 512.

Exchanging Private Credentials

The Calling feature allows two parties to exchange private key material. Please note that messages which have been recorded in a SmokeStack instance via one set of credentials will not be available if new credentials are established. The process of exchanging private credentials is as follows:

1. A participant issues a Call via a selected participant. A new 2048-bit RSA public-key pair is generated. A signature fastening the two participants is computed. The bundle is then transferred to the recipient.
2. A participant receives the bundle, verifies the included signature, and generates private authentication and encryption keys. The private key material is bundled via the included public RSA key. The participant transfers the signed private key material bundle to the initial participant.
3. The initiating participant receives the private key material, verifies the included signature, and unpackages the private key material via the ephemeral private key. The two participants are now paired.

Fire

Fire introduces communication networks between Smoke and Spot-On. Key generation is described below.

```
authentication_key = pbkdf2(sha512(Digest || “sha384”), // Salt
                            Digest,
                            10000,
                            896)                        // Bits (112 Bytes)
authentication_key, destination_key := authentication_key[0 … 48], authentication_key[48 … ]
encryption_key := pbkdf2(Salt,
                         Channel || “aes256” || “sha384”,
                         10000,      // Iteration Count
                         2304)       // Bits (288 Bytes)
encryption_key := encryption_key[0 … 31]
```

Forward Secrecy and SmokeStack

Smoke includes a mechanism for establishing session-based authentication and encryption keys. This key material is exchanged via ephemeral and permanent public keys. One advantage is that forward secrecy is constituted by the use of ephemeral public keys. One disadvantage is that messages stored on a SmokeStack instance become transitory whenever new authentication and encryption keys are established.

The Forward Fiasco release provides a mechanism for storing secret key pairs for some period of time. After a message-retrieval request has been initiated, a Smoke instance will attempt to uncover the received content using previous keys.

Inflate

Smoke expands text-messaging data to 8192 bytes. If the provided data exceeds 8192 bytes, Smoke expands the provided data by 1024 + mod(data length, 2) bytes. Inflation does not apply to Fire as Fire must remain compatible with Spot-On.

Local Broadcast Manager

Communications between the Kernel and the user interface utilize the Local Broadcast Manager.

McEliece CCA2

As of version Drooling Dragon, Smoke supports McEliece-Fujisaki via BouncyCastle. Parameters are SHA-256, m = 11, t = 50. Some discussions:

- Authentication process may require several minutes to complete.
- Communications between McEliece and RSA are fully functional.
- During the key-sharing process, McEliece signatures are not provided and therefore are not verified.
- Expect degraded performance.
- Initialization processes may require several minutes to complete.

Message Structures

This page will detail the various message structures.

CALL-HALF-AND-HALF-A

```
[PK] (1)                             McEliece 352 Bytes, RSA Key Size / 8 Bytes
{
      AES-256 Key (1)
      SHA-512 Key (2)
}

[AES-256] (2)
{
      0x00 (1)                                                 1 Byte
      A Timestamp (2)                                          8 Bytes
      RSA 2048-Bit Public Key (3)                              294 Bytes
      Sender's Identity (4)                                    8 Bytes
      Sender's Public Encryption Key SHA-512 Digest (5)        64 Bytes
      [PK Signature] (6)                                       Variable Bytes
      {
            [PK] (1 ... 2) || [AES-256] (1 ... 5) ||
            Recipient's Public Encryption Key SHA-512 Digest (1)
      }
}

[SHA-512 HMAC] (3)                   64 Bytes
{
      [PK] || [AES-256] (1)
}

/*
** The destination is created via the recipient's SipHash identity.
*/

[Destination SHA-512 HMAC] (4)                                 64 Bytes
{
      [PK] || [AES-256] || [SHA-512] (1)
```

```
}
```

CALL-HALF-AND-HALF-B

```
[PK] (1)                            McEliece 352 Bytes, RSA Key Size / 8 Bytes
{
      AES-256 Key (1)
      SHA-512 Key (2)
}

[AES-256] (2)
{
      0x01 (1)                                                1 Byte
      A Timestamp (2)                                         8 Bytes
      RSA 2048-Bit Public Key (3)                             256 Bytes
      {
            AES-256 Key (1)
            SHA-512 Key (2)
      }
      Sender's Identity (4)                                   8 Bytes
      Sender's Public Encryption Key SHA-512 Digest (5)       64 Bytes
      [PK Signature] (6)                                      Variable Bytes
      {
            [PK] (1 ... 2) || [AES-256] (1 ... 5) ||
            Recipient's Public Encryption Key SHA-512 Digest (1)
      }
}

[SHA-512 HMAC] (3)                  64 Bytes
{
      [PK] || [AES-256] (1)
}

/*
** The destination is created via the recipient's SipHash identity.
*/
```

```
[Destination SHA-512 HMAC] (4)     64 Bytes
{
      [PK] || [AES-256] || [SHA-512] (1)
}
```

CHAT

```
[PK] (1)                           McEliece 320 Bytes, RSA Key Size / 8 Bytes
{
      Sender's Public Encryption Key SHA-512 Digest (1)    64 Bytes
}

[AES-256] (2)
{
      0x00 (1)                                        1 Byte
      A Timestamp (2)                                 8 Bytes (Base-64)
      Message (3)                                     Variable Bytes (Base-64)
      Sequence (4)                                    8 Bytes (Base-64)
      [PK Signature] (5)                              Variable Bytes (Base-64)
      {
            [PK] (1) || [AES-256] (1 ... 4) ||
            Recipient's Public Encryption Key SHA-512 Digest (1)
      }
}

[SHA-512 HMAC] (3)                 64 Bytes
{
      [PK] || [AES-256] (1)
}

/*
** The destination is created via the recipient's identity.
*/

[Destination SHA-512 HMAC] (4)     64 Bytes
{
      [PK] || [AES-256] || [SHA-512 HMAC] (1)
```

```
}
```

CHAT-RETRIEVAL

```
[AES-256] (1)
{
      0x00 (1)                                               1 Byte
      A Timestamp (2)                                        8 Bytes
      An Identity (3)                                        64 Bytes
      Sender's Public Encryption Key SHA-512 Digest (4)      64 Bytes
      [PK Signature] (5)                                     Variable Bytes
      {
            [AES-256] (1 ... 4) (1)
      }
}

[SHA-512 HMAC] (2)                  64 Bytes
{
      [AES-256] (1)
}
```

CHAT-STATUS

```
[PK] (1)                            McEliece 320 Bytes, RSA Key Size / 8 Bytes
{
      Sender's Public Encryption Key SHA-512 Digest (1)      64 Bytes
}

[AES-256] (2)
{
      0x01 (1)                                   1 Byte
      A Timestamp (2)                            8 Bytes
      Status (3)                                 1 Byte (Ignored)
      [PK Signature] (4)                         Variable Bytes
      {
            [PK] (1) || [AES-256] (1 ... 3) ||
            Recipient's Public Encryption Key SHA-512 Digest (1)
```

```
      }
}

[SHA-512 HMAC] (3)                 64 Bytes
{
      [PK] || [AES-256] (1)
}

/*
** The destination is created via the recipient's identity.
*/

[Destination SHA-512 HMAC] (4)     64 Bytes
{
      [PK] || [AES-256] || [SHA-512 HMAC] (1)
}
```

EPKS

```
[AES-256] (1)
{
      A Timestamp (1)                    8 Bytes (Base-64)
      Key Type (2)                       1 Byte (Base-64)
      Public Key (3)                     Variable Bytes (Base-64)
      Public Key Signature (4)           Variable Bytes (Base-64)
      Signature Public Key (5)           Variable Bytes (Base-64)
      Signature Public Key Signature (6) Variable Bytes (Base-64)
}

[SHA-512 HMAC] (2)                       64 Bytes
{
      [AES-256] (1)
}

/*
** The destination is created via the recipient's SipHash identity.
*/
```

```
[Destination SHA-512 HMAC] (3)            64 Bytes
{
      [AES-256] || [SHA-512 HMAC] (1)
}
```

FIRE-CHAT

```
[AES-256] (1)
{
      0040b (1)         Base-64
      \n
      Name (2)          Base-64
      \n
      ID (3)            Base-64
      \n
      Message (4)       Base-64
      \n
      UTC Date (5)      Base-64
}

[SHA-384 HMAC] (2)                  Base-64
{
      [AES-256] (1)
}

[Destination SHA-512 HMAC] (3)      Base-64
{
      [AES-256] || [SHA-384 HMAC] (1)
}
```

FIRE-STATUS

```
[AES-256] (1)
{
      0040a (1)         Base-64
      \n
```

```
      Name (2)          Base-64
      \n
      ID (3)            Base-64
      \n
      UTC Date (4)      Base-64
}

[SHA-384 HMAC] (2)                Base-64
{
      [AES-256] (1)
}

[Destination SHA-512 HMAC] (3)     Base-64
{
      [AES-256] || [SHA-384 HMAC] (1)
}
```

PKP-REQUEST

```
[AES-256] (1)
{
      0x01 (1)                                  1 Byte
      A Timestamp (2)                           8 Bytes
      Destination SipHash Identity (3)          23 Bytes
      Requested SipHash Identity (4)            23 Bytes
}

[SHA-512 HMAC] (2)                 64 Bytes
{
      [AES-256] (1)
}
```

Neighbors

New neighbors may be defined via the Settings activity. This page will describe the various nuances of network peers.

Smoke offers infinitely-many IPv4 and IPv6 TCP and UDP client definitions. Each network peer includes dedicated and independent parsing-data, read-socket, and write-socket threads. TCP neighbors support HTTP and SOCKS proxies. Please note that host translations are not performed via the defined proxies.

A pull-down menu accompanies each defined neighbor.

Connect

> Instruct Smoke to place the specified neighbor in a connect status. Connection attempts are performed every 2.5 seconds. TCP sockets are required to connect within 10 seconds. After connections are established, SSL/TLS handshakes must be completed within 10 seconds.

Delete

> Display a confirmation prompt. If confirmed, the specified neighbor is scheduled for deletion.

Disconnect

> Instruct Smoke to place the specified neighbor in a disconnect status.

Purge Queue

> Purge the outbound queue of the specified neighbor.

Reset SSL/TLS Credentials

> Reset the locally-stored SSL/TLS credentials of TCP neighbors.

A connected neighbor attempts to read 1 MiB of data from its socket every 100 milliseconds. The read request blocks for at most 2.5 seconds. If data is successfully read, it is appended to an internal buffer. The internal buffer may accumulate at most 8 MiB of data, with the potential of overflow. Parsing of data occurs every 100 milliseconds. Because the parsing and read threads are independent, it’s possible that the internal buffer may temporarily overflow.

Each neighbor object includes two internal queues, Echo and real-time queues. Echo queues allow Smoke to echo internal data from local neighbor to local neighbor. This mechanism must be enabled via the Echo option. Each Echo queue may contain at most 256 messages. Please note that the Echo mechanism may burden a device. A neighbor will echo data if it discovers that the data is not intended for it. Calling, Chat statuses, Fire statuses, and SmokeStack message-retrieval requests utilize real-time queues. Real-time queues are not bound.

Various per-neighbor statistics are included in the Settings activity. Also included are per-neighbor descriptive errors.

New Installation

After launching a new installation of Smoke, some initial settings are required.

Encryption

Public-key algorithm. McEliece-Fujisaki and 3072-bit RSA are supported.

Iteration Count

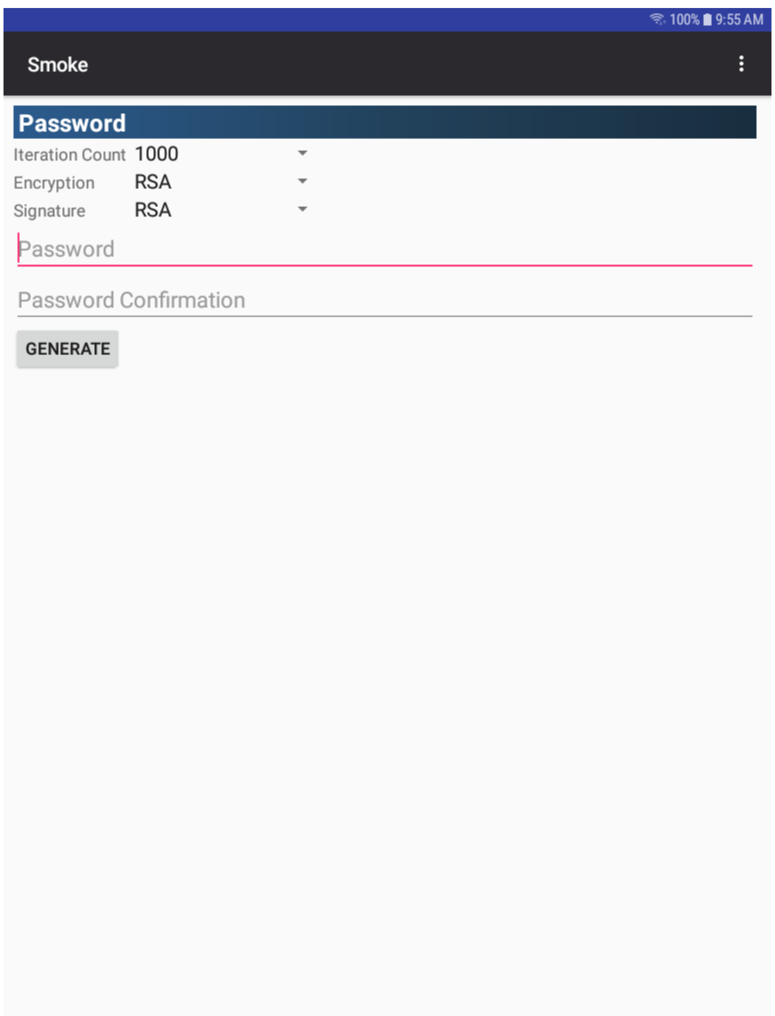
Local authentication and encryption keys are generated via PBKDF2. The function requires an iteration count. If the selected value exceeds 7500, a confirmation prompt is displayed.

Password

At least one character is required.

Signature

Public-key digital signatures. 384-bit ECDSA and 3072-bit RSA are supported.



Outbound Queues

Smoke offers near-real-time communications. As network services may be unreliable, certain outbound messages are enqueued in an SQLite database table. Each network peer is assigned a separate queue. Messages are dequeued in a timely manner and placed onto the network. Calling messages, retrieval of offline messages, and status messages are considered expendable and are therefore written to network sockets regardless of network availability.

Please note that peers which are in disconnected status-control states are ignored during the enqueue processes.

Ozone Address

An Ozone address may be assigned via the Settings activity.

An Ozone address is a pseudo-private string which identifies a virtual entity. Smoke and SmokeStack utilize Ozones as a means of retrieving and storing offline messages and public-key pairs. Smoke supports one Ozone while SmokeStack supports infinitely many. Ozone addresses must be exchanged separately. Retrieved messages are only meaningful within the context of a session. It is possible for multiple Smoke parties to house distinct Ozones if common SmokeStack instances are aware of the distinct Ozone addresses.

Please note that public Ozone addresses will introduce denial of service vulnerabilities.

Participants

Participant SipHash Identities may be defined within the Participants section of the Settings activity. After defining a participant, local public-key pairs may be shared manually. An automatic process distributes key pairs to participants which have not been paired. A context menu may also be activated by pressing and holding on the Participants widget. The contents of the context menu are described below.

Delete (SipHash Identity)

> Delete the selected participant. A confirmation dialog is displayed.

Request Keys via Ozone (SipHash Identity)

> Submit a public-key request to SmokeStack instances via the selected SipHash Identity. An Ozone address must be defined for this option to be enabled.

Share Keys (SipHash Identity)

> The selected participant's public-key pair is distributed using the specified SipHash Identity. If a public-key pair does not exist for the specified participant, the option is disabled.

Private Public-Key Server

In addition to housing messages, SmokeStack also serves as a private public-key server. A SmokeStack administrator is responsible for coordinating the storage of public-key pairs of participants. Participants may request public-key pairs of specific participants via Ozone addresses.

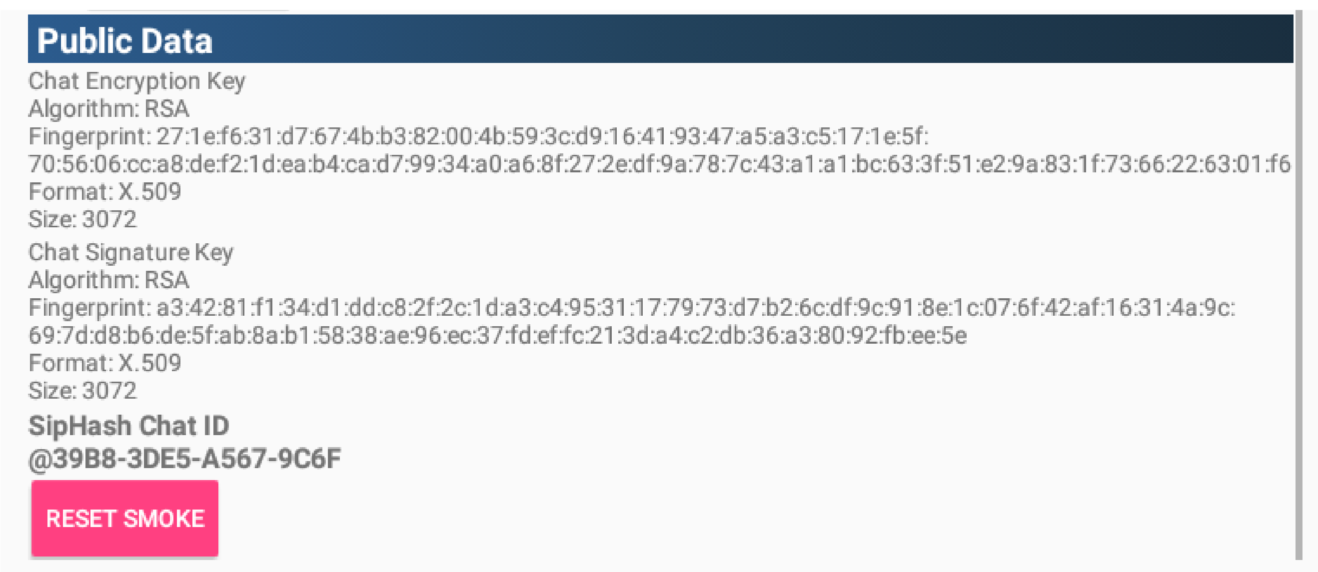
SipHash Identities

Exchanging public-key pairs is often an involved process. Smoke implements the pseudo-random function SipHash so as to simplify the process. The SipHash function generates outputs of 8 bytes (16 characters hexadecimal). These short strings are easily memorized and/or distributed via other communications applications. SipHash identities are generated as follows:

```
id := siphash(public-encryption-key || public-signature-key,
              pbkdf2(sha512(public-encryption-key || public-signature-key), // Salt
                     public-encryption-key || public-signature-key,
                     4096, // Iteration Count
                     128)) // Bits (16 Bytes)
```

Non-confidential authentication and encryption key streams from SipHash identities are generated as follows:

```
keystream1 := pbkdf2(sha512(id), // Salt
                     id,
                     4096,       // Iteration Count
                     160)        // Bits (20 Bytes)
keystream2 := pbkdf2(sha512(id), // Salt
                     keystream1,
                     1,          // Iteration Count
                     768)        // Bits (96 Bytes)
```



The transport keys which are generated from SipHash identities may be used for exchanging public-key data via the Echo Public-Key Share (EPKS) protocol.

It is impossible to avoid SipHash collisions as there are infinitely-many inputs and a limited number of outputs.

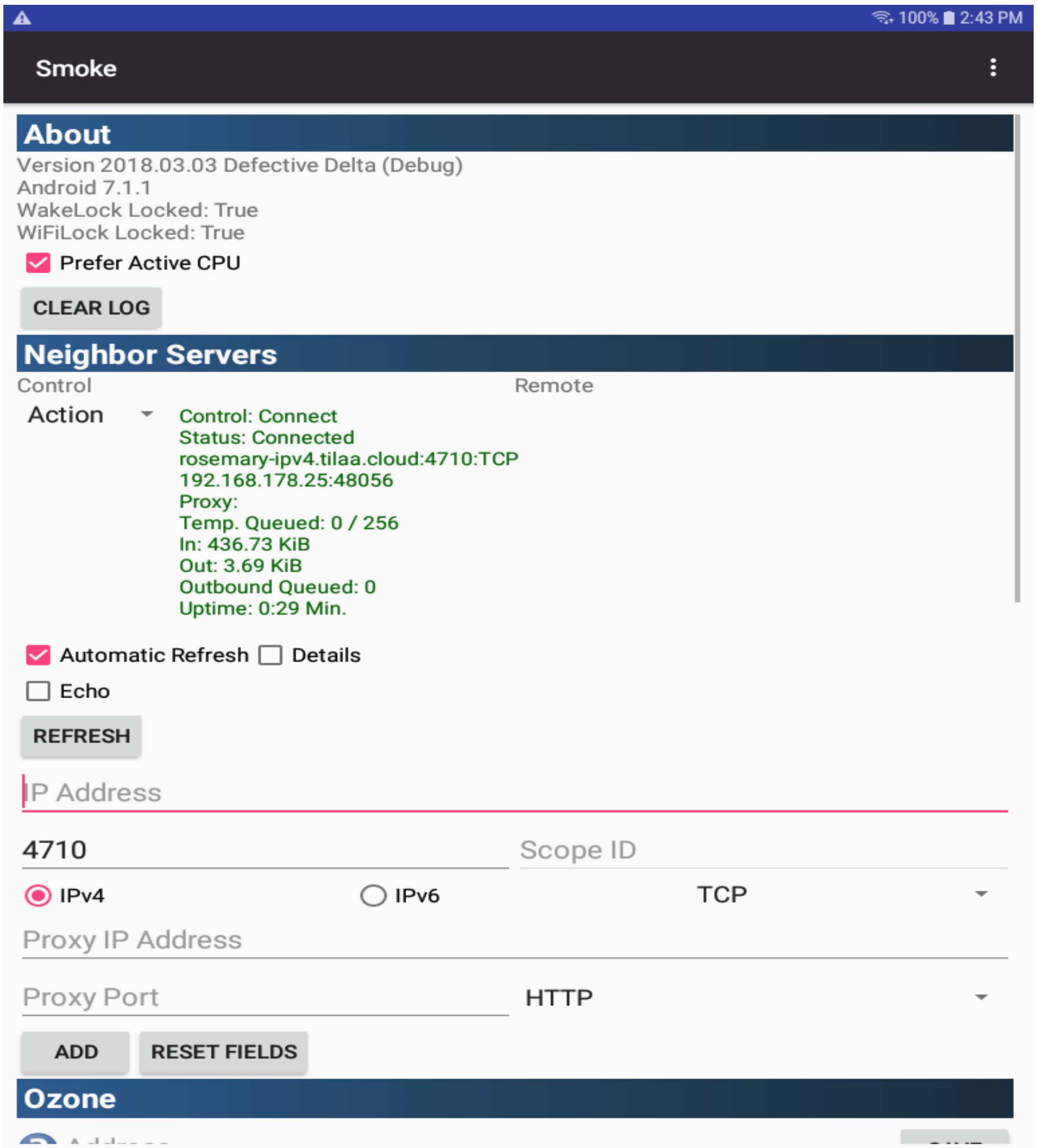
Software Distribution

Smoke is distributed in debug (smoke-debug.apk) and release (smoke.apk) forms. The release bundle is signed and includes the source.

TCP, UDP Protocols

Smoke supports both the TCP and UDP network protocols. Multicast and unicast UDP varieties are provided. Multiple clients may be defined via the Settings activity. A limit on the number of clients is not imposed. When defining neighbors, one may define SmokeStack and/or Spot-On neighbors. SmokeStack, the companion application of Smoke, offers mobile server services as well as message and public-key storage.

Example UDP multicast address: 239.255.43.21.



Thread Utilization

Smoke is an extremely-threaded application. For example, the Kernel object utilizes 7 threads while a single Neighbor object spawns 3 threads. Various threads are also defined in the activities.

UDP Datagrams

Outbound UDP messages are partitioned into 576-byte datagrams. For example, a 15000-byte message will be partitioned into 27 datagrams.

Verifying Public-Key Ownership

Before initiating an exchange of public-key pairs, Smoke generates digital signatures using the private keys of the encryption and signature public keys. The signatures are included in the EPKS bundle. A receiving Smoke instance verifies the signatures and accepts the public-key pairs if the signatures are valid. McEliece signatures are not included and are therefore not verified.