Smoke

An Android Echo Software Application

About

Smoke is an Android communications project. The software is composed of a single multitasking 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 instance, 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 be reset within the Authenticate activity.

Activity Chat

The Chat activity is one of three messaging activities. From this activity, one may message one or more defined participants. If Smoke is connected to at least one network peer, the Send button's text color is set to green. Otherwise, the Send button's text color is set to red.

Before a messaging session may begin between two participants, the participants 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 transferred on the network. Please note the following conversion of the input string: if the input string's length is less than or equal to 64, the input string is converted to Base64(SHA512(string)).

New Window

> Display a new Member Chat activity with the selected participant.

Optional Signatures

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

Purge Session

> Discard the session's 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 Participants status icons.

Activity Fire

The Fire activity is one of three messaging activities. From this activity, one may communicate with one or more groups of anonymous participants. Fire is compatible with Spot-On's Buzz. 256-bit AES-CBC along with SHA-384 HMAC provide the encryption and authentication.

Activity Member Chat

The Member Chat activity is one of three messaging activities. From this activity, one may message one participant.

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.

Context menus may be activated by pressing and holding various widgets. Context-menu items are described below.

Call via McEliece

Exchange private key material via an ephemeral (m = 11, t = 50) McEliece public key pair.

Call via RSA

Exchange private key material via an ephemeral (2048-bit) RSA public key pair.

Copy Text

Place the selected message's text in the clipboard buffer.

Custom Session

Private key material may be generated. The generated material is not transferred on the network.

Delete All Messages

If confirmed, all messages associated with the specified participant are deleted.

Delete Message

If confirmed, the selected message is deleted.

JuggerKnot

Display an input dialog. If the dialog is confirmed, the Juggernaut Protocol is initiated with the specified participant. If the protocol completes successfully, authentication and encryption credentials are created.

Juggernaut

Display an input dialog. If the dialog is confirmed, the Juggernaut Protocol is initiated with the specified participant.

Retrieve Messages

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

Save Attachment

If the selected message contains an image, this option is enabled. Once activated, the attached bitmap is saved. The bitmap is saved to Smoke's data path.

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 Foreground Service option disables or enables a Smoke foreground service. 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.

Participants

A list of participants. A context menu is available.

Password

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

Public Data

Contains the Smoke Alias and Smoke Chat ID. A Smoke Alias is synonymous to an e-mail address. The Smoke Alias is option. If provided, it must contain at least eight characters. Preferably, it should be a unique value. A Smoke Identity is synonymous to a telephone number. Basic public-key data is also displayed in this section.

Share Smoke ID

Share the Smoke ID with a SmokeStack instance via the defined Ozone. Please note that the Smoke Alias, if one is defined, is transformed into a Smoke ID.

Android

Smoke has been successfully tested on Android versions 7.x and 8.x. Versions older than 7.x and 8.x are not supported.

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 decipher 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 widgets. These depictions offer insight into potential system failures.

Custom Session Credentials

Credentials are generated as follows (elongate the first key stream):

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

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 $E(Data, K_e) \parallel HMAC(E(Data, K_e), K_a)$ 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 files 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.

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

Exchanging Private Credentials

The Calling feature allows two parties to exchange private key material. The process of exchanging private credentials is as follows:

1. A participant issues a Call via a selected participant. A new ephemeral McEliece or RSA public-key pair is generated. A signature binding 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 McEliece or 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 unpacks the private key material via the ephemeral private key. The two participants are now paired.

Fiasco Keys

Authentication and encryption key data which are established via the so-called calling mechanism are recorded within the participants_keys database table. Whenever a message from a SmokeStack instance is received, the message's digest is verified using each of the recorded authentication keys. Smoke iterates through the set of Fiasco authentication keys until a correct authentication key is discovered or the search is exhausted. If an authentication key is recovered, the message is deciphered and delivered locally. Newer authentication keys are tested first.

A key pair has a lifetime of 864,000 seconds.

Fire

Fire introduces communication channels 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. Forward secrecy is constituted by the use of ephemeral public keys.

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 previously-established secret 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.

Juggernaut Protocol

Smoke implements the Password Authenticated Key Exchange by Juggling protocol. The data is exchanged within a messaging session. Please note the following conversion of the input string: if the input string's length is less than or equal to 64, the input string is converted to Base64(SHA512(string)). JuggerKnot credentials are generated as follows (elongate the first key stream):

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

Please read https://en.wikipedia.org/wiki/Password_Authenticated_Key_Exchange_by_Juggling for more information.

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 = 12, t = 68. Some discussions:

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

Message Structures

This page will detail the various message structures.

AUTHENTICATE

```
[PK Signature] (1)
{
    Random Bytes (1)                                  Variable
}


CALL-HALF-AND-HALF-A

[PK] (1)                      McEliece 352, 608 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 (Base-64)
    \n
    Ephemeral Public Key (3)                          Variable (Base-64)
    \n
    Ephemeral Public Key Type (4)                     1 Byte (Base-64)
    \n
    Sender's Identity (5)                             8 Bytes (Base-64)
    \n
    Sender's Public Encryption Key SHA-512 Digest (6)  64 Bytes (Base-64)
    \n
    [PK Signature] (7)                                Variable (Base-64)
    {
        [PK] (1 ... 2) || [AES-256] (1 ... 6) ||
        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 Smoke Identity.
*/

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

CALL-HALF-AND-HALF-B

```
[PK] (1)                        McEliece 352, 608 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 (Base-64)
     \n
     Ephemeral Public Key (3)                 Variable (Base-64)
     {
          AES-256 Key (1)
          SHA-512 Key (2)
     }
     \n
     Ephemeral Public Key Type (4)            1 Byte (Base-64)
     \n
     Sender's Identity (5)                    8 Bytes (Base-64)
     \n
```

```
        Sender's Public Encryption Key SHA-512 Digest (6)    64 Bytes (Base-64)
        \n
        [PK Signature] (7)                                 Variable (Base-64)
        {
              [PK] (1 ... 2) || [AES-256] (1 ... 6) ||
              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 Smoke Identity.
*/


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


CHAT
[PK] (1)                        McEliece 320, 576 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)
      \n
      Message (3)                                    Variable (Base-64)
      \n
```

```
    Sequence (4)                                 8 Bytes (Base-64)
    \n
    Attachment (5)                               Variable (Base-64)
    \n
    Message Identity (6)                         64 Bytes (Base-64)
    \n
    [PK Signature] (7)                           Variable (Base-64)
    {
         [PK] (1) || [AES-256] (1 ... 6) ||
         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 Smoke Identity.
*/

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

CHAT-RETRIEVAL (Via Ozone)
[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
    {
```

```
                [AES-256] (1 ... 4) (1)
      }
}


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


CHAT-STATUS
[PK] (1)                        McEliece 320, 576 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
      {
            [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 Smoke 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)
     \n
     Key Type (2)                           1 Byte (Base-64)
     \n
     Sender's Smoke Identity (3)            19 Bytes (Base-64)
     \n
     Public Key (4)                         Variable (Base-64)
     \n
     Public Key Signature ((3) || (4) || (6)) (5)    Variable (Base-64)
     \n
     Signature Public Key (6)               Variable (Base-64)
     \n
     Signature Public Key Signature ((3) || (4) || (6)) (7)

                                            Variable (Base-64)
}

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

/*
** The destination is created via the recipient's Smoke 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)
}


JUGGERNAUT
[PK] (1)                        McEliece 320, 576 Bytes, RSA Key Size / 8 Bytes
{
     Sender's Public Encryption Key SHA-512 Digest (1)    64 Bytes
}


[AES-256] (2)
{
     0x03 (1)                                       1 Byte
     A Timestamp (2)                                8 Bytes (Base-64)
     \n
     Payload (3)                                    Variable (Base-64)
     \n
     [PK Signature] (4)                             Variable (Base-64)
     {
          [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 Smoke Identity.
*/


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


MESSAGE-READ
[PK] (1)                       McEliece 320, 576 Bytes, RSA Key Size / 8 Bytes
{
     Sender's Public Encryption Key SHA-512 Digest (1)    64 Bytes
}


[AES-256] (2)
{
     0x02 (1)                                       1 Byte
     Message Identity (2)                           64 Bytes
     [PK Signature] (3)                             Variable
     {
          [PK] (1) || [AES-256] (1 ... 2) ||
          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 Smoke Identity.
*/
```

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

PKP-REQUEST (Via Ozone)
```
[AES-256] (1)
{
     0x01 (1)                                           1 Byte
     A Timestamp (2)                                    8 Bytes
     Destination Smoke Identity (3)                     19 Bytes
     Requested Smoke Identity (4)                       19 Bytes
}


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

SHARE-SMOKE-ID (Via Ozone)
```
[AES-256] (1)
{
     0x02 (1)                                           1 Byte
     A Timestamp (2)                                    8 Bytes
     Smoke Identity (3)                                 19 Bytes
     Temporary Identity (4)                             8 Bytes
}


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

SHARE-SMOKE-ID-CONFIRMATION (Via Ozone)
```
[AES-256] (1)
{
```

```
    0x03 (1)                                          1 Byte
    A Timestamp (2)                                   8 Bytes
    Smoke Identity (3)                                19 Bytes
    Temporary Identity (4)                            8 Bytes
}


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


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


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

Neighbors

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 data-parsing, socket-reading, and socket-writing tasks. TCP neighbors support HTTP and SOCKS proxies. Please note that host translations are not performed via assigned proxies.

Initialize Ozone

If enabled, the Ozone credentials will be generated from the specified neighbor values. For example, let's suppose that a SmokeStack is attached to the service bee.service.org:4710. When preparing the neighbor information in Smoke using the aforementioned SmokeStack destination, the Ozone will be initialized to bee.service.org:4710:TCP. In SmokeStack, the Ozone bee.service.org:4710:TCP should also be defined. When completed, the Smoke and SmokeStack instances are artificially paired.

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. A TCP socket is required to connect within 10 seconds. After a connection is established, the SSL/TLS handshake must complete 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 indefinitely. 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 tasks 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 limited.

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. If an Ozone address is defined and the network is available, Smoke will request external messages once per minute.

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. 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

Smoke 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 be activated by pressing and holding on the Participants widget. The contents of the context menu are described below.

Delete (Smoke Identity)

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

Delete Fiasco Keys (Smoke Identity)

> Delete all of the recorded Fiasco keys. The current session keys of the selected participant are not deleted. A confirmation dialog is displayed.

Delete Public Keys (Smoke Identity)

> Delete the Fiasco and public keys of the specified participant. A confirmation dialog is displayed.

New Name (Smoke Identity)

> Assign a new name to the selected participant.

Request Keys via Ozone (Smoke Identity)

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

Share Keys Of (Smoke Identity)

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

Share Smoke ID Of (Smoke Identity)

> The selected participant's Smoke Identity is distributed using the defined Ozone address. An Ozone address must be defined for this option to be enabled.

View Details (Smoke Identity)

> View details of the selected participant.

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.

Private Servers

SmokeStack supports the concept of private servers for TCP clients. A private server will read and write data if and only if a remote peer has been authenticated. The authentication process is as follows:

1. A private server generates a 64-byte stream of random data and concatenates the bytes with the current system time.

2. The server submits the SHA-512 hash of the information generated in the previous step to the remote peer after the SSL/TLS handshake has been completed.

3. A remote peer digitally signs the stream of data and submits the digital signature to the remote server.

4. The remote server inspects the digital signature. If the signature is valid, the remote peer is authenticated.

Please define private servers after the desired participants have been completely defined in SmokeStack.

Smoke Aliases

A Smoke Alias is a unique stream of characters. The minimum length of a Smoke Alias is eight. Similar to e-mail addresses and telephone numbers, Smoke Aliases allow simple pairing of participants. Internally, a Smoke Alias is transformed into a Smoke Identity via the SipHash algorithm. Let's consider a simple pairing scenario:

1. Participant textbrowser@gmail.com assigns the Smoke Alias in the Public Data section of the Settings activity. Once assigned, the participant notifies other participants via e-mail or another form of communication.
2. Notified participants define textbrowser@gmail.com within the Participants section of the Settings activity. The Smoke Alias option must be enabled.
3. Participants notify textbrowser@gmail.com of their aliases.
4. Within new instances, the pairing is automatically initiated once the participants are online. Pairing may also be performed via the Share Keys mechanism.

Please note that a Smoke instance must synchronize itself with a remote server after a new Smoke Alias is assigned within Public Data. Synchronization generally completes in approximately 15 seconds.

Smoke 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. Smoke 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 Smoke identities are generated as follows (elongate the first key stream):

```
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 Smoke 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) form. Sometimes, a release (smoke.apk) form is also distributed. 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.

Task Utilization

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

Time

Time references are included in various message structures. Therefore, it is important that a device's local clock is correct. Smoke also performs numerous internal processes which are time-sensitive.

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 digital signatures are composed of the concatenation of the public encryption and signature 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. Summary:

1. Concatenate the encoded forms of the encryption and the signature public keys.
2. Digitally sign the concatenated product using the private encryption key.
3. Digitally sign the concatenated product using the private signature key.
4. Bundle the two digital signatures.