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. The Send button is enabled if at least one participant is selected and a writable neighbor (not necessarily network-ready) is available.

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 over 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 encryption and authentication.

The Send button is enabled if at least one network-ready neighbor is available.

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. The Send button is enabled if a session is established and if a writable neighbor (not necessarily network-ready) is available.

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 McEliece (m = 11, t = 50) public-key pair.

Call via RSA

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

Copy Text

: Place the selected message’s text into the clipboard buffer.

Custom Session

: Private key material may be generated. The generated material is not transferred over 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 Credentials

: 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.

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.

Optional Steam

Disable or enable Steam sharing for the specified participant. Active Steams will not adhere to this setting as Steam packet bundles do not include explicit participant information.

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.

Activity Settings

The Settings activity contains various configuration 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. If confirmed, all existing data will be purged. A new public and private key pair may also be generated.

Public Data

> Contains the Smoke Alias and Smoke ID. A Smoke Alias is synonymous to an e-mail address. The Smoke Alias is optional. 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 are 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.

Activity Smokescreen

Lock or unlock the Smoke application.

Activity Steam

The Steam activity allows for the transferring of files to Steam participants and/or anonymous destinations. Received files are stored in the Downloads folder.

Empty files are considered valid, however, they will not be transferred.

Please note that Simple Steams (files transferred to anonymous destinations) will be streamed sequentially in the order in which they were registered for transfer.

An Asynchronous World

The world of Smoke is asynchronous. Smoke itself is asynchronous. Processes that are otherwise asynchronous are sometimes synchronized. As Smoke does not include connections between peers, it functions relatively well in a chaotic, disconnected environment.

Steam (file sharing) introduces two reliable communications processes: key establishment and packet distribution. These processes replicate TCP in a chaotic, disconnected environment.

Texting also provides reliable delivery of deliverable messages. A deliverable message is a message which will be accepted by a participant if the message's timestamp is within some tolerance. Timestamps mitigate replay attacks. Messages which are not deliverable are discarded.

Android

Smoke has been successfully tested on Android versions 7.x, 8.x, and 9.x. Versions older than 7.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 65 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 (stretch the first key stream):

```
keystream1 := pbkdf2(sha512(string), // Salt
                     string,
                     4096,       // Iteration Count
                     160)        // Bits (20 Bytes)
keystream2 := pbkdf2(sha512(string), // Salt
                     base64(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) || 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 novel 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 instances offer 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.

Exporting / Importing Participant Personalities

Smoke provides an elegant, yet tedious, process for exporting and importing participant credentials. Available in the Settings activity, participant credentials may be shared with SmokeStack instances. Simply select individual participants and share the identifiers followed by the public-key personalities. The important process is similarly simple. After participant identifiers have been recorded, request public-key personalities via selected identifiers.

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. The 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.

Future-Proofing Software

Future-proofing software is an involved task. This section will document some of the problems.

CBC and CTR modes suffer from birthday attacks. It is recommended that keys be used conservatively in these modes. Consider using AES-GCM-SIV.

Cipher and hash algorithms should be interchangeable either through options or simple software changes.

Latest versions of TLS should be applied.

Public-key-based applications should consider the constraints imposed by the algorithms. For example, RSA-2048 with v1.5 padding can encrypt at most 245 bytes.

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 are 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 (stretch the first key stream):

```
keystream1 := pbkdf2(sha512(key material), // Salt
                     base64(key material),
                     4096,       // Iteration Count
                     160)        // Bits (20 Bytes)
keystream2 := pbkdf2(sha512(key material), // Salt
                     base64(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 a Local Broadcast Manager instance.

McEliece CCA2

Smoke supports McEliece-Fujisaki and McEliece-Pointcheval via BouncyCastle. Parameters are SHA-256, m = 11, m = 12 (Fujisaki only), t = 50, t = 68 (Fujisaki only). 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.

Super McEliece-Fujisaki is also somewhat supported: m = 13, t = 118.

Message Structures

This section will detail the various message structures.

AUTHENTICATE

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

CALL-HALF-AND-HALF-A

```
[PK] (1)
{
      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)
{
      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 (Ignored) (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)
{
      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)
{
      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)                           Variable (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)
{
      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)
{
      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)
}
```

MESSAGE-READ (Via Ozone)

```
[AES-256] (1)
{
      0x04 (1)                                                  1 Byte
      A Timestamp (2)                                           8 Bytes
      Message Identity SHA-512 Digest (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)
}
```

PKP-REQUEST (Via Ozone)

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

[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)                                        Variable
      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)                                        Variable
      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)
```

}

STEAM-KEY-EXCHANGE-A

```
[PK] (1)
{
      AES-256 Key (1)
      SHA-512 Key (2)
}

[AES-256] (2)
{
      0x04 (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
      File Digest (5)                                           32 Bytes (Base-64)
      \n
      File Identity (6)                                         48 Bytes (Base-64)
      \n
      File Name (7)                                             Variable (Base-64)
      \n
      File Size (8)                                             8 Bytes (Base-64)
      \n
      Sender's Public Encryption Key SHA-512 Digest (9)         64 Bytes (Base-64)
      \n
      [PK Signature] (10)                                       Variable (Base-64)
      {
            [PK] (1 ... 2) || [AES-256] (1 ... 9) ||
            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)
}
```

STEAM-KEY-EXCHANGE-B

```
[PK] (1)
{
      AES-256 Key (1)
      SHA-512 Key (2)
}

[AES-256] (2)
{
      0x05 (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
      File Digest (5)                                        32 Bytes (Base-64)
      \n
      File Identity (6)                                      48 Bytes (Base-64)
      \n
      File Name (Empty) (7)                                  Variable (Base-64)
      \n
      File Size (8)                                          8 Bytes (Base-64)
      \n
      Sender's Public Encryption Key SHA-512 Digest (9)      64 Bytes (Base-64)
      \n
      [PK Signature] (10)                                    Variable (Base-64)
      {
            [PK] (1 ... 2) || [AES-256] (1 ... 9) ||
            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)
}
```

STEAM-SHARE-A

```
[PK] (1)
{
      File Identity (1)                                    48 Bytes
}

[AES-256] (2)
{
      0x06 (1)                                             1 Byte
      A Timestamp (2)                                      8 Bytes
      File Offset (3)                                      8 Bytes
      File Packet (4)                                      Variable
}

[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)
}
```

STEAM-SHARE-B

```
[PK] (1)
{
      File Identity (1)                                    48 Bytes
```

```
}

[AES-256] (2)
{
      0x07 (1)                                          1 Byte
      A Timestamp (2)                                   8 Bytes
      File Offset (3)                                   8 Bytes
}

[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)
}
```

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.

Non-TLS

Allows the neighbor to observe traditional socket operations.

Passthrough

Passthrough neighbors are special full-duplex connections which Smoke utilizes for distributing data to non-Smoke destinations. Data which is received on passthrough connections is echoed directly to other non-passthrough neighbors if echoing is enabled.

A menu accompanies each defined neighbor.

Connect

> Instruct Smoke to place the specified neighbor in a connected status. Connection attempts are performed every 3.5 seconds. A TCP socket is required to connect within 10 seconds. After a connection is established, the SSL/TLS handshake must complete within 20 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 disconnected status.

Purge Queue

> Purge the outbound queue of the specified neighbor.

Reset SSL/TLS Credentials

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

A connected neighbor attempts to read 1 MiB of data from its socket every 100 milliseconds. The read request blocks indefinitely. Data are 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 by 1024^2 – 1 bytes.

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 are 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, McEliece-Pointcheval, and 4096-bit RSA are supported.

Iteration Count

: Local authentication and encryption keys are generated via Argon2id or PBKDF2. The functions require an iteration count. If the selected value exceeds 10 for Argon2id or 7500 for PBKDF2, a confirmation prompt is displayed.

Password

: At least one character is required.

Signature

: Public-key digital signatures. 384-bit ECDSA and 4096-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 disposable 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. It is possible for multiple Smoke parties to house distinct Ozones if common SmokeStack instances are aware of the distinct Ozone addresses.

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

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.

Performance Considerations

Smoke is a multi-tasking process and several of its internal operations are performed in separate tasks, thus allowing the main thread to remain as responsive as possible.

Chat Activity

- Reading of participant information occurs in a separate task. UI elements are repainted on the main thread.

Database

- Recording of new Steam objects occurs in a separate task as computing file digests is not necessarily timely.
- Removal of neighbor outbound queue data is performed by a separate task.

Fire Activity

- Each channel is assigned a task for monitoring participants.

Kernel

- Automatic requesting of SmokeStack messages is performed in a separate task.
- Neighbor objects are prepared in a separate task.
- Network status information is gathered in a separate task and reported to the main thread.
- Outbound messages (Chat, Fire, Juggernaut, Message Retrieval Request, Share Smoke Identity, Steam Key Exchange) are prepared in a separate task.
- Participant calling keys are generated in a separate task.
- Public-key publication is performed in a separate task.
- Purging of congestion control data, expired Juggernaut credentials, and expired participant key streams is performed in a separate task.
- Purging of expired temporary identifiers is performed in a separate task.
- Status message broadcasting is performed in a separate task.
- Steam objects are prepared in a separate task.

Member Activity

- Network status information is gathered in a separate task and presented to the UI via the main thread.

Miscellaneous

- A special database cursor is maintained for rapid access to data for the Member Chat activity. The cursor is synchronized in various logical regions.

Neighbor

- Accumulated data are parsed in a separate task.
- Data are written to the network in a separate task.

- Neighbor statistics and statuses are prepared in a separate task.
- Network data are read in a separate task.
- Purging of neighbor queues is performed in a separate task.

Settings Activity

- Neighbor information is gathered in a separate task and UI elements are repainted on the main thread. Purging of malformed outbound data and participants is also performed in this task.

State

- Participant elements for various interface widgets are gathered in a separate task.

Steam Activity

- Task for notifying adapter.

Steam Key Exchange

- A separate task is responsible for distributing and parsing key data.
- Public-key pairs are generated in a separate task.

Steam Reader

- Files are read and distributed in separate tasks.

Steam Writer

- A single writer records packets to respective files. The writer contains a separate task for recording status information.

Time

- Time server is queried in a separate task.

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 disregard non-authentication data until 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 data 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. The server will repeatedly submit unique information every 10 seconds until the peer has authenticated itself.
3. The remote peer retrieves a stream of 64 random bytes as well as its signature key digest. It digitally signs the 64 random bytes, the signature key digest, and the original stream of random data and submits the 64 random bytes, the signature key digest, and the digital signature to the remote server. Please note that SmokeStack servers are conceptually indistinguishable from one another. Therefore, remote peers do not provide SmokeStack identifiers during this step.
4. The server reviews the two random-byte streams for uniqueness. If the two byte streams are dissimilar, it validates the digital signature. If the digital signature is valid and the two random-byte streams are dissimilar, the remote peer is authenticated.

Please define private servers after the desired participants have been completely defined in SmokeStack. This is required because SmokeStack instances must be in possession of public-key pairs.

Please note that multiple devices may contain identical Smoke instances. Thus, several identical Smoke instances may authenticate themselves with a given SmokeStack instance.

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 vanya@nasa.gov 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 vanya@nasa.gov within the Participants section of the Settings activity. The Smoke Alias option must be enabled.
3. Participants notify vanya@nasa.gov of their aliases.
4. Within new instances, the pairing process 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.

A Smoke identity is generated as follows:

```
id := siphash(alias,
              pbkdf2(sha512(alias), // Salt
                            alias,
                            4096, // Iteration Count
                            128)) // Bits (16 Bytes)
```

Smoke Identities

Exchanging public-key pairs is often an involved process. Smoke implements the pseudo-random function SipHash to simplify the process. The SipHash function generates outputs of 128 bits (16 bytes). A Smoke identity is 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 a Smoke identity 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
                     base64(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.

Smoke Pipes (Simple Steams)

Piping through Smoke allows for the transfer of data from Smoke devices to network-capable, non-Smoke devices. The process is as follows:

1. Define a passthrough network interface in the Settings activity. Optionally, disable or enable TLS. If TLS is enabled, it is expected that the defined endpoint supports TLS.
2. Prepare the endpoint service on the destination device. In this example: nc -l 192.168.178.15 4710 > output.
3. In the Steam activity, select a single file and specify the destination as Other (Non-Smoke). Tag the file for transfer. Repeat as often as desired.
4. Resume each file.
5. The first file to be tagged is the first file to be transferred.

Using commands such as head and tail, it's possible to partition the output file into separate files. Data may also be piped to multiple endpoints.

A concrete example follows.

1. In a console: "nc -k -l 192.168.178.15 4710 > output". If necessary, disable the firewall or prepare specific firewall rules on the destination device.
2. Define the non-TLS passthrough 192.168.178.15:4710 in Smoke's Settings activity.
3. Prepare 3 image files for distribution in Smoke.
4. Activate the Rewind & Resume All Steams context-menu option in the Steam activity.
5. Once the 3 files have been transferred, observe the file sizes of each Steam.
6. In a console: "head -c sizeof(file a) output > file1".
7. In a console: "sha256 file1". The digest must match the digest provided by Smoke.
8. In a console: "tail -c "$((sizeof(file a) + sizeof(file b)))" output | head -c sizeof(file a) > file2".
9. In a console: "sha256 file2". The digest must match the digest provided by Smoke.
10. In a console: "tail -c sizeof(file c) output > file3".
11. Finally: "sha256 file3". The digest must match the digest provided by Smoke.

The Scripts directory contains a program for partitioning a received aggregate into individual components.

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 may include the source.

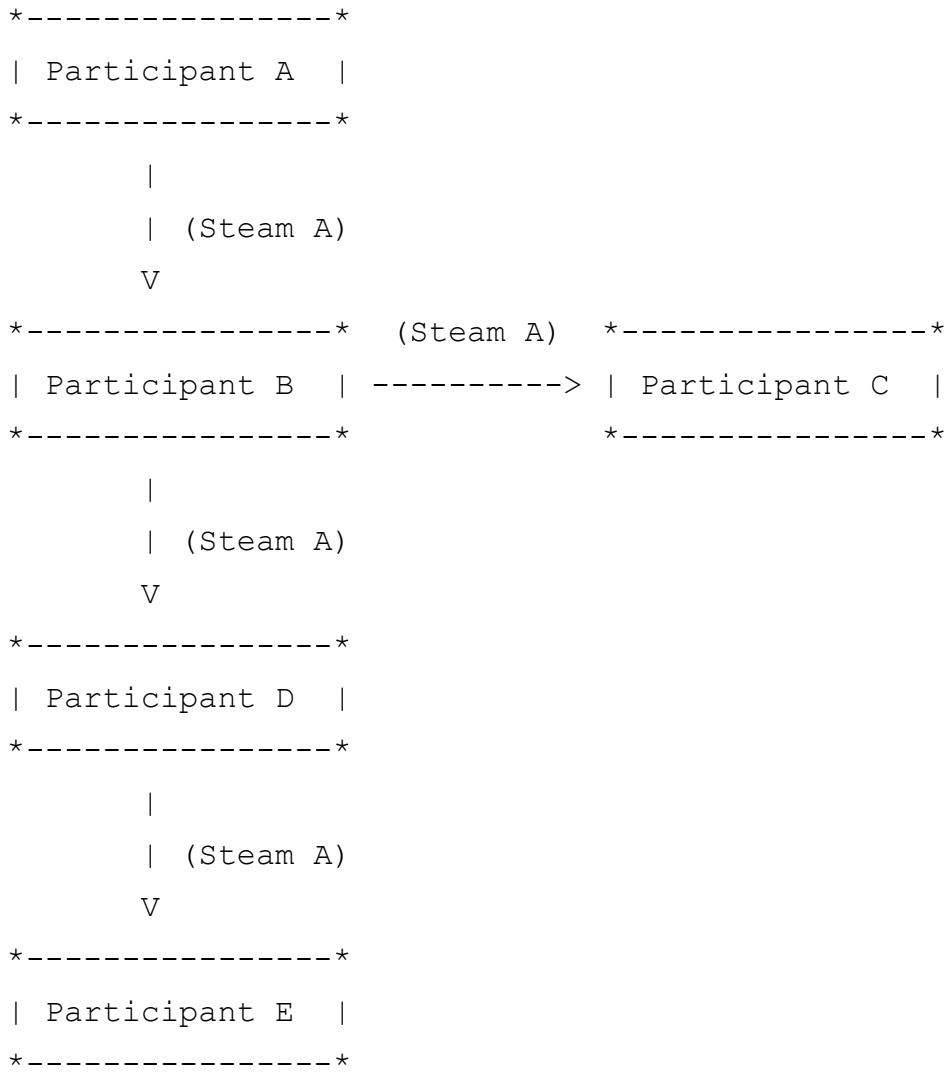
Steam Ephemeral Key Exchange

Please note that this protocol is a partial forward secrecy key exchange as the generator of the ephemeral public-key pair temporarily records the pair in an SQLite database. The ephemeral key-pair is removed from the database after a response is received from the destination participant. The destination participant removes the ephemeral public key after the first packet is recorded.

1. Generate an ephemeral public-key pair. A public-key pair is generated per file. Do not generate a public-key pair if one already exists or if private keys (4) have been recorded.
2. Locally record the ephemeral public-key pair.
3. Transfer the ephemeral public key to the destination participant repeatedly until private keys are received.
4. Destination receives the ephemeral public key and generates private keys. Do not generate private keys if private keys already exist.
5. Locally record the private keys, unless the private keys exist.
6. Destination encrypts the private keys via the ephemeral public key and submits the results to the source participant.
7. Source participant receives the bundle and deciphers the private keys via the ephemeral private key.
8. Source participant records the private keys (4).
9. Source participant deletes the ephemeral public-key pair from the local SQLite database.
10. Destination participant the deletes ephemeral public key after the first packet of the Steam is recorded.

Steamrolling

Steamrolling is the process of real-time broadcasting inbound Steams. Let’s review an example.



Participants C, D, and E shall receive Steam A data as participant B receives and writes Steam A data. If the stream of bytes between A and B is halted, the halting event will percolate throughout the network.

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.

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.

Smoke shall notify the operator if the device’s Unix time differs from the Unix time of an external source by approximately five seconds. Notifications, if enabled, will occur every thirty seconds.

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 recipient 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.