Smoke

An Android Echo Software Application

About

Smoke is an Android communications research 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.

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. Routinely, expired entries are removed.

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).

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.

Fire

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

```
authentication_key = pbkdf2(sha512(Digest || “sha384”), // Salt
                            Digest,
                            10000,
                            768)                        // Bits (96 Bytes)
authentication_key, destination_key := authentication_key[0 … 31], authentication_key[32 … ]
encryption_key := pbkdf2(Salt,
                         Channel || “aes256” || “sha384”,
                         10000,       // Iteration Count
                         2304)        // Bits (288 Bytes)
encryption_key := encryption_key[0 … 31]
```

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.

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.
- During the key-sharing process, McEliece signatures are not provided.
- Expect degraded performance.
- Initialization processes may require several minutes to complete.

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 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 shared 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.

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.

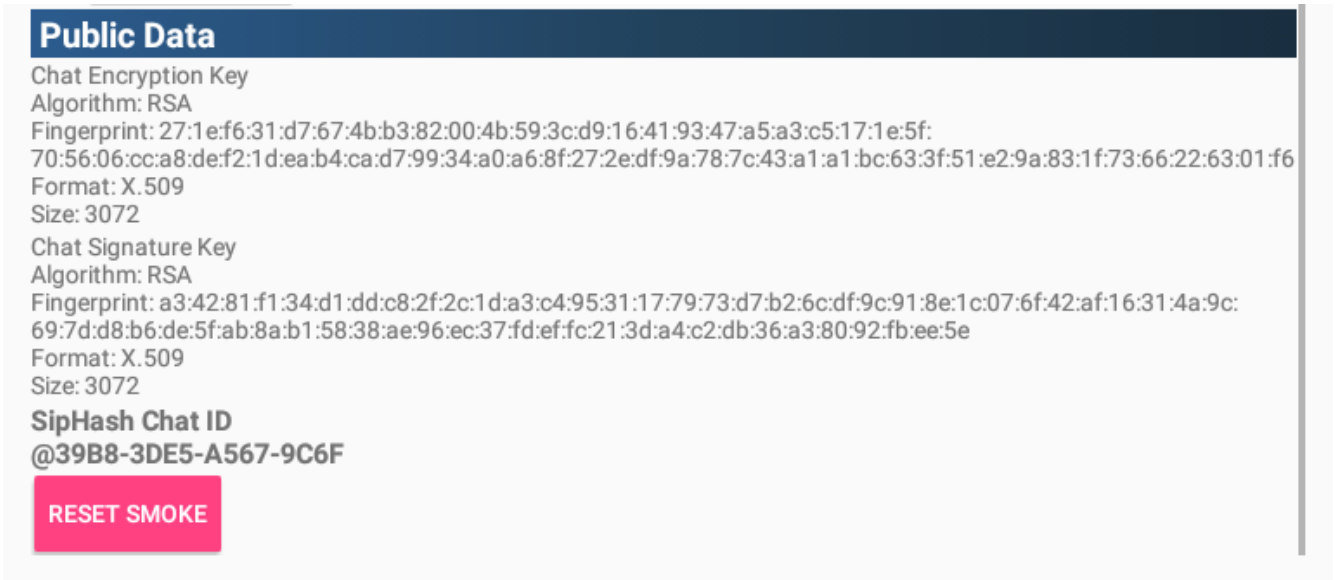
SipHash Identities

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.

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 Settings. 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.