Streamable Encryption

Aleksandar Hrusanov

Mentored by Simone Colombo and David Lazar

DEDIS lab @ EPFL
21.01.2021

Motivation: working with large ciphertexts

Safe:

```
gpg -d largefile.enc > largefile.tmp
tar xz < largefile.tmp
```

Slow and wastes space

Unsafe:

```
gpg -d largefile.enc | tar xz
```

Contributions

Secretstream, a Go library for streaming cryptography

- Ported from libsodium
- Working to merge upstream into `x/crypto`: **Go issue #43774**

Sage, a new tool for safe, streamable cryptography:

```
./sage largefile.enc | tar xz
```

Challenge 1: ciphertext tampering

Unsafe:

```
gpg -d largefile.enc | tar xz
```

Bitflips in the ciphertext will be detected by GPG…
...*but only at the end,* when it might be too late:

```
gpg -d script.enc | sh
```

Solution: secretstream

Classic secretbox API

```
Seal(file) → ctxt

Open(ctxt) → file, ok
```

secretstream: split file into chunks and authenticate along the way

```
file = (chunk1, chunk2, …, chunkN)

Push(chunk1) → ctxt1; Push(chunk2) → ctxt2; ...

Pull(ctxt1) → (chunk1, ok); Pull(ctxt2) → (chunk2, ok); ...
```

golang.org/x/crypto/secretstream

- Based on ChaCha20-Poly1305 AEAD scheme
  - Constant-time, hardware-independent, fast
- Two interfaces: *Encryptor* & *Decryptor*
- Full test suite with > 90% code coverage
- Known Answer Tests ensure compatibility with Libsodium

Challenge 2: sender authentication

Safe:

```
gpg --verify largefile.enc.sig && gpg -d ...
```

Slow and not streamable!

How do we ensure a stream comes from a trustworthy source?

Authenticating streams in various contexts

Stream to yourself

```
Encrypt(privateKey, msg)
```

Stream to a friend

```
Encrypt(passphrase, msg)
```

```
Encrypt(myPrivateKey, theirPublicKey, msg)
```

Stream anonymously

```
Encrypt(theirPublicKey, msg)?
```

This doesn't seem possible to do safely!
(attacker can substitute the ciphertext)

Anonymous streaming is unsafe

Stream anonymously

```
Encrypt(theirPublicKey, msg)?
```

This doesn't seem possible to do safely!
(attacker can substitute the ciphertext)

The popular Age tool supports this unsafe mode of operation:

```
age -r recipient.pub largefile
```

Conclusion: avoid streaming anonymous ciphertexts

Sage, a better tool for file encryption

- Simple CLI tool for file encryption, inspired by Age
- Uses secretstream under the hood
- Key generation functionality
  - X25519 Diffie-Hellman key-agreement protocol
- Support authenticated streams in various contexts (work-in-progress)
- Tries to avoid unsafe use cases (work-in-progress)
- .sage format expands on .age format

Related work

STREAM

- Streaming encryption algorithm published in a paper [HRRV15]
- No widely established API
- Used in Age, but no official implementations

Secretstream

- Well-established API
- Supported by many languages
- Supports ratcheting for forward secrecy

.sage File Format

```
-> X25519 encode(X25519(ephemeral secret, basepoint))
encrypt[HKDF[salt, label](X25519(ephemeral secret, public key))](file key)
+++ encode(stream_header)
=== [encode(additional_data)]
--- encode(HMAC[HKDF["", "header"](file key)](file header))
SECRETSTREAM[HKDF[nonce, "payload"](file key)](plaintext)
```

DEMO

Conclusion

Open-source contribution provides a high-level API for streaming encryption in Go

Sage is an MVP of a file encryption tool built on top of it which supports safe streaming and avoids unsafe use cases

Future work

- Extend Sage to support other use cases
- Explore workarounds to the to anonymous streaming problem