Modern Session Encryption

David Wong
1. KECCAK
2. STROBE
3. NOISE
4. ???

Outline
Sponge Construction

absorbing

00101 01001 01100

01011 10101

squeezing
Duplex Construction

Init

Input  Output  Input  Output  Input  Output

duplexing  \( f \)  duplexing  \( f \)  duplexing  \( f \)
Keyed-mode

init

0 0 0 0 0 0 0

key

⊕

0

duplexing

f

Encryption?

init

0 0 0 0 0 0 0

key

⊕

duplexing

f
Encryption

key

plaintext1 ⊕ ciphertext1

init

duplexing
Authenticated Encryption

key

\[ f \]

init

\( \oplus \)

ciphertext1

\( \oplus \)

plaintext1

tag1

duplexing

duplexing
Sessions

key

init

f

ciphertext1

⊕

plaintext1

duplexing

f

tag1

ciphertext2

⊕

plaintext2

duplexing
2. STROBE

1. KECCAK
Strobe functions

AD

operation = AD
data = 010100…

KEY

operation = KEY
data = 010100…

PRF

operation = PRF
output = 0000…

send_CLR

operation = send_CLR
data = 010100…

recv_CLR

operation = recv_CLR
data = 010100…

send_ENC

operation = send_ENC
plaintext = plaintext
ciphertext = ciphertext

recv_ENC

operation = recv_ENC
ciphertext = ciphertext
plaintext = plaintext

send_MAC

operation = send_MAC
plaintext = plaintext
tag = tag

recv_MAC

operation = recv_MAC
tag = tag
ciphertext = ciphertext

RATCHET

operation = RATCHET
0000…

0000…
Strobe protocol example

```python
def myProtocol:
    myProtocol = Strobe_init("myWebsite.com")
    myProtocol.AD(sharedSecret)
    buffer = myProtocol.send_ENC("GET /")
    buffer += myProtocol.send_MAC(len=16)
    // send the buffer
    // receive a ciphertext
    message = myProtocol.recv_ENC(ciphertext[:-16])
    ok = myProtocol.recv_MAC(ciphertext[-16:])
    if !ok {
        // reset the connection
    }
```

buffer = myProtocol.send_ENC(plaintext1)
buffer += myProtocol.send_MAC(len=16)

// send the buffer

buffer = myProtocol.send_ENC(plaintext2)
buffer += myProtocol.send_MAC(len=16)

// send the buffer

buffer = myProtocol.send_ENC(plaintext3)
buffer += myProtocol.send_MAC(len=16)

// send the buffer

buffer = myProtocol.send_ENC(plaintext4)
buffer += myProtocol.send_MAC(len=16)

// send the buffer
Strobe

- **flexible** framework to support a large number of protocols
- large **symmetric cryptography** library
myHash = Strobe_init("david_wong_hash")
myHash.AD("something to be hashed")
hash = myHash.PRF(outputLen=32)
operation = $AD$
operation = AD

data = 010100...

rate

capacity
operation = AD

rate

data = 010100...

capacity

operation = send_ENC
operation = AD

data = 010100...

operation = send_ENC

rate

capacity
operation = AD
data = 010100...

operation = send_ENC
data = hello
ciphertext

rate

capacity
operation = AD

data = 010100...

operation = send_ENC

data = hello

ciphertext

operation = send_MAC

tag

len = 16
operation = AD
data = 010100...

operation = send_ENC
data = hello
ciphertext

operation = send_MAC
tag
len = 16

rate
capacity

send_AEAD
Strobe

- flexible framework to support a large number of protocols
- large symmetric cryptography library
- fits into tiny IoT devices (~300 lines of code)
- relies on strong SHA-3 standard (SHAKE-compliant)
STROBE protocol framework

Scope
This spec describes the operation of the STROBE framework. It only covers the symmetric portion.
For applications including elliptic curve crypto, see the examples page.

Table of Contents
1. Version
2. Definitions and notation
   2.1. Formatting
3. STROBE Instances
4. STROBE parameters
5. State of a STROBE object
   5.1. Initial state
6. STROBE operations
   6.1. Low-level operations
      6.1.1. AD: Provide associated data
      6.1.2. KEY: Provide cipher key
      6.1.3. CLR: Send or receive cleartext data
      6.1.4. ENC: Send or receive encrypted data
      6.1.5. MAC: Send or receive message authentication code
      6.1.6. PRF: Extract hash / pseudorandom data
      6.1.7. RATCHET: Prevent rollback
   6.2. Operations and flags
outline

3. NOISE

2. STROBE
   1. KECCAK
TLS

- TLS is the **de facto standard** for securing communications
- **complex** specification (TLS 1.3 is 160-page long)
  - supported by other specifications (asn.1, x509, 44 mentioned RFCs …)
- design carrying a lot of **legacy** decisions
- **cryptographic agility** and **complicated** state machine
- **huge** and **scary** libraries (OpenSSL is 700k LOC, 165 CVEs)
  - **cumbersome** configuration…
- often **dangerously** re-implemented (custom implementations)
  - or re-invented (proprietary protocols)
Complexity is the enemy of security
The Noise Protocol Framework

Author: Trevor Perrin (noise@trevp.net)
Revision: 33
Date: 2017-10-04
PDF: noise.pdf

Table of Contents

• 1. Introduction
• 2. Overview
  • 2.1. Terminology
  • 2.2. Overview of handshake state machine
• 3. Message format
• 4. Crypto functions
  • 4.1. DH functions
  • 4.2. Cipher functions
  • 4.3. Hash functions
• 5. Processing rules
  • 5.1. The cipherState object
  • 5.2. The SymmetricState object
  • 5.3. The HandshakeState object
• 6. Prologue
• 7. Handshake patterns
  • 7.1. Pattern validity
  • 7.2. One-way patterns
  • 7.3. Interactive patterns
The Noise Protocol Framework

- no need for certificates or a PKI
- many handshakes to choose from (flexible)
- it’s straight forward to implement (<1k LOC, 18kb for Arduino)
- there are already libraries that you can leverage
- minimal (or zero) configuration
- used by WhatsApp, Slack, the Bitcoin Lightning Network, ...
- if you have a good excuse not to use TLS, Noise is the answer
The crypto functions

- **DH**: X25519 or X448
- **AEAD**: Chacha20-Poly1305 or AES-GCM
- **HASH**: BLAKE2 or SHA-2
Client → ephemeral key → Server
ephemeral key ← Server → Client

handshake
ephemeral key

Client

Diffie-Hellman()

→

keys

Server

Diffie-Hellman()

→

keys

ephemeral key

handshake
ephemeral key

Client

Diffie-Hellman() keys

Server

Diffie-Hellman() keys

handshake

ephemeral key

encrypted data

post-handshake

encrypted data
Client

encrypted data

encrypted data

post-handshake

e

e

handshake

keys

keys

Server
$\rightarrow e$

$\leftarrow e, \text{ ee}$
Tokens

- **e**: ephemeral key
- **s**: static key
- **ee**: $DH$(client ephemeral key, server ephemeral key)
- **es**: $DH$(client ephemeral key, server static key)
- **se**: $DH$(client static key, server ephemeral key)
- **ss**: $DH$(client static key, server static key)
- **psk**: pre-shared key
## Handshake Patterns

<table>
<thead>
<tr>
<th></th>
<th>N(rs):</th>
<th>K(s,rs):</th>
<th>X(s,rs):</th>
<th>NN():</th>
<th>NK(rs):</th>
<th>NX(rs):</th>
</tr>
</thead>
<tbody>
<tr>
<td>←</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>→ e</td>
<td>s</td>
<td>e</td>
</tr>
<tr>
<td>→</td>
<td>s</td>
<td>s</td>
<td>s</td>
<td>e, ee</td>
<td>s</td>
<td>e, ee, s, es</td>
</tr>
<tr>
<td>→</td>
<td>e, es</td>
<td>e, es, s, ss</td>
<td>e, es</td>
<td>e, es</td>
<td>e, ee</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>XN(s):</th>
<th>XK(s, rs):</th>
<th>XX(s, rs):</th>
<th>KN(s):</th>
<th>KK(s, rs):</th>
</tr>
</thead>
<tbody>
<tr>
<td>→</td>
<td>e</td>
<td>s</td>
<td>e</td>
<td>s</td>
<td>s</td>
</tr>
<tr>
<td>←</td>
<td>e, ee</td>
<td>e, ee, s, es</td>
<td>e, ee, s, es</td>
<td>e</td>
<td>e, es, ss</td>
</tr>
<tr>
<td>→</td>
<td>s, se</td>
<td>e, es</td>
<td>s, se</td>
<td>e</td>
<td>e, ee, se</td>
</tr>
<tr>
<td>←</td>
<td>s, se</td>
<td>e, ee</td>
<td>s, se</td>
<td>e</td>
<td>e, ee, se</td>
</tr>
</tbody>
</table>

...
\[\text{NX}(rs):\]
\[\rightarrow e\]
\[\leftarrow e, \text{ee}, s, \text{es}\]
NX(rs):
→ e
← e, ee, s, es
\[ \text{NX}(rs) : \]

\[ \rightarrow e \]

\[ \leftarrow e, ee, s, es \]

Client \rightarrow Server

\[ e_{\text{public}} \]

\[ \text{payload1} \]
\[
\text{NX}(rs): \\
\rightarrow e \\
\leftarrow e, ee, s, es
\]
\[ \text{NX}(rs): \]
\[ \to e \]
\[ \gets e, \ e e, \ s, \ es \]
\[ \text{NX}(rs): \]
\[ \rightarrow e \]
\[ \leftarrow e, ee, s, es \]

Client

```
epublic
```

```
payload1
```

Server

```
re_{public}
```

```
E_{K1}(rs)
```

\[ \text{NX}(rs): \]

\[ \rightarrow e \]

\[ \leftarrow e, ee, s, es \]
\[
\text{NX}(rs):
\]
\[
\rightarrow e
\]
\[
\leftarrow e, ee, s, es
\]

Diagram:
- Client
- Server
- \(e_{\text{public}}\)
- \(\text{payload1}\)
- \(r_{e\text{public}}\)
- \(E_{K1}(rs)\)
- \(E_{K2}(\text{payload2})\)
1. **Initialization**
   - `e_{public}`
   - `payload1`
   - `re_{public}`
   - `E_{k1}(rs_{public})`
   - `E_{k2}(payload2)`

2. **CipherState**
   - `n=0`
   - `k`

3. **HandshakeState**
   - `DH(e, re)`
   - `DH(e, rs)`
   - `HKDF`
   - `HASH`
   - `GENERATE_KEYPAIR()`
   - `“Noise_NX_25519_AESGCM_SHA256”`

4. **SymmetricState**
   - `ck`
GENERATE_KEYPAIR()

Initialization

<table>
<thead>
<tr>
<th>e_{\text{public}}</th>
<th>payload1</th>
</tr>
</thead>
<tbody>
<tr>
<td>re_{\text{public}}</td>
<td>re_{\text{public}}</td>
</tr>
<tr>
<td>DH(e, re)</td>
<td>rs_{\text{public}}</td>
</tr>
<tr>
<td>E(rs_{\text{public}})</td>
<td>tag1</td>
</tr>
<tr>
<td>DH(e, rs)</td>
<td>rs_{\text{public}}</td>
</tr>
<tr>
<td>E(payload2)</td>
<td>tag2</td>
</tr>
</tbody>
</table>
libdisco

libdisco is a modern plug-and-play secure protocol and a cryptographic library implemented in 1000 lines of code in Golang. It offers different ways of securely connecting peers together, as well as different cryptographic primitives for all of an application's needs.

Warning

libdisco is experimental. It still has not been thoroughly reviewed and thus should not be used in production.

libdisco is a library built by merging the Noise protocol framework and the Strobe protocol framework. This means that it supports a subset of Noise's handshakes while offering the cryptographic primitive Strobe has to offer. In other words, you can use libdisco to securely connect peers together, or to do basic cryptographic operations like hashing or encrypting.
700,000 LOC

OpenSSL

disco-c

libdisco (go)

1,000 LOC

2,000 LOC

4,000 LOC

DiscoNet* (C#)

* implementation by Artyom Makarov
Trust Graph of Disco

- DISCO
- STROBE
- KECCAK-F
- X25519
Trust Graph of biased SSL/TLS

- AES-GCM
- HKDF
- HMAC
- X25519
- ECDSA
- SHA-256
- TLS 1.3
Trust Graph of SSL/TLS

- TLS 1.3
  - CHACHA20-POLY1305
  - AES-CCM
  - AES-GCM
  - HKDF
  - HMAC
  - SHA-256
  - SHA-384
  - SHA-512

- TLS 1.2
  - TLS 1.1
  - X.509
  - RSA-PSS or RSA-PKCS#1 v1.5
  - ECDHE
  - DH
  - ECDH

- TLS 1.1
  - TLS 1.2
  - X.509
  - RSA-PSS or RSA-PKCS#1 v1.5
  - ECDHE
  - DH
  - ECDH

- X.509
  - DER

- ASN.1
  - DER

- DH
  - secp256r1
  - secp384r1
  - secp521r1

- ECDH
  - ed448
  - ed25519
  - secp256r1
  - secp384r1
  - secp521r1
The state of Disco

- **Disco** is a draft specification extending Noise *(experimental)*
- **Noise** is a stable draft *(rev34)*
- **Strobe** is alpha *(v1.0.2)*
- **⚠** Disco and the implementations are still *experimental*
  - need more eyes, more interoperability testing, etc.
  - looking to formally prove handshakes with Tamarin
the disco is at
www.discocrypto.com

I write about crypto
www.cryptologie.net

follow me on
twitter.com/cryptodavidw

(and I work here)