噪声协议框架

特雷弗·佩林 (noise@trevp.net)

修订版 34,2018-07-11,官方/不稳定

内容

一、简介

Noise 是一个基于 Diffie-Hellman 密钥协议的加密协议框架。
噪声可以描述由单个消息和交互组成的协议
协议。

2. 概述

2.1。术语

噪声协议以两方交换握手消息开始。
在这个握手阶段,双方交换 DH 公钥并执行
一系列 DH 操作,将 DH 结果散列为共享密钥。
在握手阶段之后,每一方都可以使用这个共享密钥发送加密
传输消息。

Noise 框架支持握手,其中每一方都有一个长期的
静态密钥对和/或临时密钥对。噪声握手由一种简单的语言描述。这种语言由标记组成,这些标记是

排列成消息模式。消息模式被排列成握手模式。

消息模式是指定 DH 公钥的令牌序列
包括握手消息和执行的 DH 操作
发送或接收该消息时。握手模式指定
包含握手的消息的顺序交换。

握手模式可以通过DH 函数、密码函数、
和散列函数给出具体的噪声协议。

2.2.握手状态机概述

Noise的核心是握手过程中各方维护的一组变量,以及顺序发送和接收握手消息的规则

处理来自消息模式的标记。

每一方都维护以下变量:

- s, e:本地方的静态和临时密钥对（可能是
  空的）。

- rs, re:远程方的静态和临时公钥（可能
  为空）。
- h:一个握手哈希值,它对所有握手数据进行哈希处理。
  被发送和接收。
- ck:散列所有先前DH 输出的链接键。握手完成后,链接密钥将用于派生传输消息的加密密钥。
- k, n:一个加密密钥k （可能为空)和一个基于计数器的随机数n。每当新的 DH 输出导致计算新的ck时,也会计算新的k 。密钥k和 nonce n用于加密静态公钥和握手有效负载。使用k的加密使用一些AEAD密码模式（在 Rogaway [1] 的意义上）,并使用当前的h值作为AEAD 身份验证涵盖的关联数据。

  静态公钥和有效载荷的加密在握手阶段提供了一些机密性和密钥确认。

握手消息由一些 DH 公钥和后跟有效负载组成。
有效负载可能包含应用程序选择的证书或其他数据。
要发送握手消息,发送者指定有效负载并顺序处理来自消息模式的每个令牌。可能的标记是:

- “e”:发送者生成一个新的临时密钥对并将其存储在e变量中,将临时公钥作为明文写入消息缓冲区,并将公钥与旧的 h 一起散列以派生新的 h。
- “s”:发送者将其静态公钥从s变量写入消息缓冲区,如果k不为空,则对其进行加密,并将输出与旧 h 一起散列以派生新 h。
- “ee”、“se”、“es”、“ss”:在发起者的密钥对（静态或临时由第一个字母确定)和响应者的密钥对（静态或临时确定)之间执行 DH由第二个字母）。将结果与旧ck一起散列以得出新的ck 和 k,并将 n 设置为零。

在处理完握手消息中的最终令牌后,发送方然后将有效负载写入消息缓冲区,如果k不为空,则对其进行加密,并将输出与旧 h 一起散列以导出新 h。

举个简单的例子,未经身份验证的 DH 握手由握手模式描述:

-> 和
<- e, ee

发起者发送第一条消息,它只是一个临时公钥。响应者发回自己的临时公钥。然后执行 DH 并将输出散列为共享密钥。

请注意,明文有效负载在第一条消息中发送,在明文临时公钥之后,加密有效负载在响应消息中发送,在明文临时公钥之后。应用程序可以发送它想要的任何有效载荷。

响应者可以发送其静态公钥（在加密下)并通过稍微不同的模式验证自己：

-> 和

<- e,ee,是的,是

在这种情况下,最终的ck和k值是两个 DH 结果的哈希值。由于es令牌指示发起者的临时密钥和响应者的静态密钥之间的 DH,因此发起者成功解密第二条消息的有效负载可用于向发起者验证响应者。

请注意,第二条消息的有效负载可能包含长度为零的明文,但有效负载密文仍将包含身份验证数据（例如身份验证标签或“合成 IV”）,因为加密是 AEAD 模式。

第二条消息的有效负载也可用于为响应者的静态公钥提供证书。

发起者可以发送其静态公钥（在加密下）,并使用带有附加消息的握手模式对自己进行身份验证：

-> 和

<- e, ee, s, is -> s, is

以下部分充实了细节,并添加了一些复杂性。然而， Noise 的核心是这个由变量、令牌和处理规则组成的简单系统,它允许简洁地表达一系列协议。

3.消息格式

所有噪声消息的长度都小于或等于 65535 字节。限制消息大小有几个优点：

- 测试更简单,因为可以轻松测试最大尺寸。
- 降低内存处理错误或整数溢出的可能性。
- 支持大数据流的流式解密和随机访问解密。
- 使封装噪声消息的高级协议能够使用
  有效的 16 位标准长度字段。

由于没有类型或长度字段,因此可以在不解析的情况下处理所有噪声消息。当然,噪声消息可能被封装在一个

包含类型和长度信息的高级协议。噪声消息可能会封装需要某种解析的有效负载,但有效负载由应用程序处理,而不是由噪声处理。

噪声传输消息只是长度小于或等于 65535 字节的 AEAD 密文,由加密的有效载荷和 16字节的身份验证数据组成。详细信息取决于 AEAD 密码函数,例如 AES256-GCM 或 ChaCha20-Poly1305,但通常身份验证数据是附加到密文的 16 字节身份验证标签,或附加到密文的 16 字节合成 IV。

噪声握手消息也小于或等于 65535 字节。它以一个或多个 DH 公钥的序列开始,由其消息模式确定。公钥之后将是单个有效负载,可用于传送证书或其他握手数据,但也可以包含长度为零的明文。

如果静态公钥和有效负载在 DH 操作之前以握手方式发送,则它们将是明文,如果它们发生在 DH操作之后,它们将是 AEAD 密文。 （如果噪声与预共享对称密钥一起使用,则此规则不同;请参阅第 9 节）。与传输消息一样,AEAD 密文将每个加密字段（无论是静态公钥还是有效负载)扩展 16 个字节。

例如,考虑握手模式:

-> 和

<- e, ee, s, is -> s, is

第一条消息由一个明文公钥（“e”)和一个明文有效负载组成（请记住,有效负载在每个消息模式的末尾是隐含的）。

第二条消息由明文公钥（“e”)和加密的公钥（“s”)和加密的有效载荷组成。第三条消息由一个加密的公钥（“s”）和一个加密的有效载荷组成。

假设每个有效负载包含一个长度为零的明文,并且 DH 公钥为56 个字节,则消息大小将为:

1. 56 字节（一个明文公钥和一个明文负载） 2. 144 字节（两个公钥,第二个加密和加密的负载） 3. 88 字节（一个加密的公钥和加密负载）

4. 加密功能

噪声协议用一组具体的DH 函数、密码函数和散列函数来实例化。这些函数的签名定义如下。一些具体的功能在第 12 节中定义。

以下符号将用于算法伪代码：

- ||运算符连接字节序列。 · byte() 函数构造单个字节。

4.1。 DH 功能

噪声取决于以下DH 函数（以及相关的常数）：

- GENERATE_KEYPAIR()：生成新的 Diffie-Hellman 密钥对。 DH 密钥对由public_key和private_key元素组成。 public_key表示将 DH 公钥编码为长度为 DHLEN 的字节序列。public_key编码细节特定于每组 DH函数。

- DH(key_pair, public_key)：在key_pair中的私钥和public_key之间执行Diffie-Hellman 计算，并返回长度为DHLEN 的字节输出序列。为了安全起见，基于此函数的 Gap-DH 问题必须是任何实际的密码分析对手都无法解决的 [2]。

  public_key要么编码某个值，该值是大型素数阶组中的生成器（该值可能具有多个等效编码），要么是无效值。实现必须通过返回一些纯粹是公钥的函数并且不依赖于私钥的输出，或者通过向调用者发出错误信号来处理无效的公钥。 DH 函数可以定义更具体的规则来处理无效值。

- DHLEN = 一个常数，指定公钥和 DH 的字节大小
  输出。出于安全原因，DHLEN 必须为 32 或更大。

4.2.密码函数

噪声取决于以下密码函数：

- ENCRYPT(k, n, ad, plaintext)：使用 32 字节的密码密钥k和一个 8 字节的无符号整数随机数n加密明文，该随机数 n 对于密钥k必须是唯一的。返回密文。加密必须使用“AEAD”加密模式和关联的数据广告（使用[1] 中的术语）并返回与

明文加上 16 个字节用于身份验证数据。如果密钥是秘密的,则整个密文必须与随机密文无法区分（请注意,这是一个附加要求,并非所有 AEAD 方案都必须满足）。

- DECRYPT(k, n, ad, ciphertext):使用32字节的密钥k 、8 字节的无符号整数 nonce n 和相关数据ad 对密文进行解密。返回明文,除非身份验证失败,在这种情况下会向调用者发出错误信号。

- REKEY(k):返回一个新的 32 字节密钥作为 k 的伪随机函数。如果此函数没有专门为某些密码函数集定义，则默认从ENCRYPT(k, maxnonce, zerolen, zeros) 返回前 32 个字节,其中maxnonce等于264-1，zerolen是零长度字节序列, zeros是一个由 32 个字节组成的序列

  零。

4.3.哈希函数

噪声取决于以下散列函数（和相关常数）：

- HASH(data):使用抗冲突加密散列函数散列一些任意长度的数据,并返回 HASHLEN 字节的输出。

- HASHLEN = 一个常数,指定散列输出的字节大小。
  必须是 32 或 64。

- BLOCKLEN = 一个常数,指定散列函数在内部用于划分其输入以进行迭代处理的字节大小。这是使用带有 HMAC 的散列函数所需要的（BLOCKLEN 是 [3] 中的 B）。

Noise 基于上述 HASH() 函数定义了附加函数：

- HMAC-HASH(key, data):使用HASH()函数应用[3] 中的HMAC 。
  此函数仅作为 HKDF() 的一部分调用,如下所示。

- HKDF(chaining_key, input_key_material, num_outputs):采用
  长度为HASHLEN的chaining_key字节序列,以及长度为 0 字节、32 字节或DHLEN字节的input_key_material 字节序列。
  根据 num_outputs 是 2 还是 3 ,返回一对或三个字节序列,每个长度为HASHLEN：

  –设置temp_key = HMAC-HASH(chaining_key, input_key_material)。
  –设置 output1 = HMAC-HASH(temp_key, byte(0x01))。
  –设置 output2 = HMAC-HASH(temp_key, output1 || byte(0x02))。
  –如果 num_outputs == 2 则返回对 (output1, output2)。
  –设置 output3 = HMAC-HASH(temp_key, output2 || byte(0x03))。
  –返回三元组（输出 1、输出 2、输出 3）。

注意temp_key、 output1、 output2和output3的长度都是HASHLEN字节。另请注意， HKDF()函数只是[4] 中的HKDF ,chaining_key 作为 HKDF 盐,以及零长度的 HKDF 信息。

五、处理规则

为了精确定义处理规则,我们采用了面向对象的术语,并提出了三个“对象”,它们封装了状态变量并包含实现处理逻辑的函数。这三个对象被呈现为一个层次结构:每个更高层的对象都包括它下面的对象的一个实例。从最低层到最高层,对象是:

- CipherState对象包含k和n变量,用于加密和解密密文。在握手阶段,每一方都有一个CipherState,但在传输阶段,每一方都有两个CipherState 对象:一个用于发送,一个用于接收。

- SymmetricState对象包含一个CipherState以及ck和h变量。它之所以如此命名,是因为它封装了 Noise 使用的所有“对称加密”。在握手阶段,每一方都有一个SymmetricState,一旦握手完成就可以删除它。

- HandshakeState对象包含一个 SymmetricState 加上 DH 变量（s、e、rs、re)和一个表示握手模式的变量。在握手阶段,每一方都有一个HandshakeState ,一旦握手完成就可以删除它。

要执行 Noise 协议,您需要Initialize()一个HandshakeState。在初始化期间,您指定握手模式、任何本地密钥对以及您所知道的远程方的任何公钥。在Initialize()之后,您在HandshakeState上调用WriteMessage()和ReadMessage()来处理每个握手消息。如果DECRYPT()或DH() 函数发出任何错误信号,则握手失败并删除 HandshakeState。

处理最终的握手消息会返回两个CipherState对象,第一个用于加密从发起者到响应者的传输消息,第二个用于另一个方向的消息。此时应该删除HandshakeState ,但哈希值 h 除外,它可用于握手后的通道绑定（参见第 11.2 节）。

然后通过在相关的CipherState上调用EncryptWithAd()和DecryptWithAd()对传输消息进行加密和解密,并使用零长度的关联数据。如果DecryptWithAd()由于DECRYPT( )失败而发出错误信号,则丢弃输入消息。应用程序可以选择删除CipherState并在出现此类错误时终止会话,或者可以继续尝试通信。如果EncryptWithAd()或DecryptWithAd()发出信号

由于 nonce 耗尽而导致错误,则应用程序必须删除CipherState 并终止会话。

以下部分详细描述了这些对象。

5.1。密码状态对象

CipherState 可以根据其 k 和 n 变量加密和解密数据：

- k:32 字节的密钥（可能为空）。空是一个特殊的值
  这表明 k 尚未初始化。
- n:8 字节（64 位)无符号整数随机数。

CipherState响应以下函数。应用于n的++后递增运算符表示“使用当前的n值,然后递增它”。最大n值(264-1)保留用于其他用途。如果增加n导致2 64-1,那么任何进一步的 EncryptWithAd()或DecryptWithAd()调用都会向调用者发出错误信号。

- InitializeKey(key):设置k = key。设置 n = 0。
- HasKey():如果k 不为空,则返回true,否则返回false。
- SetNonce(nonce):设置n = nonce。该函数用于处理
  乱序传输消息,如第 11.4 节所述。
- EncryptWithAd(ad, plaintext):如果k不为空,则返回ENCRYPT(k, n++, ad, plaintext)。否则返回明文。
- DecryptWithAd(ad, ciphertext):如果k不为空,则返回DECRYPT(k, n++, ad, ciphertext)。否则返回密文。如果在DECRYPT()中发生身份验证失败,则n不会递增,并且会向调用者发出错误信号。

- Rekey():设置 k = REKEY(k)。

5.2. SymmetricState 对象

SymmetricState对象包含一个CipherState和以下变量：

- ck： HASHLEN 字节的链接键。 · h：
  HASHLEN 字节的散列输出。

SymmetricState 响应以下函数： ·

InitializeSymmetric(protocol_name):采用任意长度的protocol_name字节序列（参见第 8 节）。执行以下步骤：

–如果protocol_name的长度小于或等于HASHLEN字节,则设置h等于protocol_name并附加零字节以形成HASHLEN 字节。否则设置 h = HASH(protocol_name)。

–设置 ck = h。

–调用 InitializeKey(empty)。

· MixKey(input_key_material):执行以下步骤:

–设置 ck, temp_k = HKDF(ck, input_key_material, 2)。
–如果 HASHLEN 为 64,则将 temp_k 截断为 32 字节。
–调用 InitializeKey(temp_k)。

· MixHash(data):设置h = HASH(h || data)。

· MixKeyAndHash(input_key_material):此函数用于处理预共享对称密钥,如第 9 节所述。它执行以下步骤:

–设置 ck, temp_h, temp_k = HKDF(ck, input_key_material, 3)。
–调用 MixHash(temp_h)。
–如果 HASHLEN 为 64,则将 temp_k 截断为 32 字节。
–调用 InitializeKey(temp_k)。

· GetHandshakeHash():返回h。该函数只应在握手结束时调用,即在调用Split()函数之后。

该函数用于通道绑定,如 11.2 节所述

· EncryptAndHash(plaintext):设置ciphertext = EncryptWithAd(h, plaintext),调用MixHash(ciphertext),返回密文。请注意,如果k为空,则EncryptWithAd()调用将设置密文等于明文。

· DecryptAndHash(ciphertext):设置plaintext = DecryptWithAd(h, ciphertext),调用MixHash(ciphertext),返回明文。请注意,如果k为空,则DecryptWithAd()调用将设置明文等于密文。

· Split():返回一对用于加密传输消息的CipherState对象。执行以下步骤,其中zerolen是一个长度为零的字节序列:

–设置 temp_k1, temp_k2 = HKDF(ck, zerolen, 2)。
–如果 HASHLEN 为 64,则将 temp_k1 和 temp_k2 截断为 32 个字节。
–创建两个新的 CipherState 对象 c1 和 c2。
–调用c1.InitializeKey(temp_k1)和c2.InitializeKey(temp_k2)。
–返回对 (c1, c2)。

5.3.握手状态对象

HandshakeState对象包含一个SymmetricState和以下变量,其中任何一个都可能为空。 Empty是一个特殊值,表示变量尚未初始化。

· s:本地静态密钥对 · e:本地临时密钥对 · rs:远程方的静态公钥 · re:远程方的临时公钥

HandshakeState还具有跟踪其角色的变量,以及握手模式的其余部分:

·发起者:指示发起者或响应者角色的布尔值。

· message_patterns:一系列消息模式。每个消息模式都是来自集合(“e”、“s”、“ee”、“es”、“se”、“ss”)的标记序列。 (第 9 节介绍了一个额外的“psk”标记,但我们将其解释推迟到那时。)

HandshakeState 响应以下函数:

· Initialize(handshake_pattern, 发起者, prologue, s, e, rs, re):采用有效的握手模式(参见第 7 节)和指定该方角色为发起者或响应者的发起者布尔值。

采用可能是零长度的序言字节序列,或者可能包含双方想要确认相同的上下文信息(参见第 6 节)。

使用一组 DH 密钥对(s, e)和公钥(rs, re)来初始化局部变量,其中任何一个都可能为空。仅当handshake_pattern使用预消息时才传递公钥(参见第 7 节)。临时值(e, re)通常留空,因为它们是在握手期间创建和交换的;但也有例外(见第10 节)。

执行以下步骤:

–通过组合握手模式和加密函数的名称来派生protocol_name字节序列,如第 8 节中所述。
调用 InitializeSymmetric(protocol_name)。

–调用 MixHash(prologue)。

–将initiator、 s、 e、 rs和re变量设置为对应的论据。

–为来自handshake_pattern的预消息中列出的每个公钥调用一次MixHash() ,并使用指定的公钥作为输入(有关预消息的解释,请参见第 7 节)。如果发起人和

响应者有预消息,发起者的公钥首先被散列。如果在任何一方的预消息中列出了多个公钥,则公钥将按照它们列出的顺序进行散列。

–将message_patterns设置为来自handshake_pattern的消息模式。

· WriteMessage(payload, message_buffer):获取一个长度可能为零的有效负载字节序列,以及一个用于将输出写入的消息缓冲区。执行以下步骤,如果任何EncryptAndHash() 调用返回错误,则中止:

–从message_patterns 中获取并删除下一个消息模式,然后依次处理来自消息模式的每个标记:

对于“e”:将e (必须为空)设置为GENERATE_KEYPAIR()。
将e.public_key 附加到缓冲区。调用MixHash(e.public_key)。

对于“s”:将 EncryptAndHash(s.public_key)附加到
缓冲。

对于“ee”:调用 MixKey(DH(e, re))。 对于“es”:

如果发起者调用MixKey(DH(e, rs)) ,如果响应者调用 MixKey(DH(s, re))。

对于“se”:如果发起者调用MixKey(DH(s, re)) ,如果响应者调用 MixKey(DH(e, rs))。

对于“ss”:调用 MixKey(DH(s, rs))。

–将 EncryptAndHash(payload) 附加到缓冲区。

–如果没有更多消息模式,则通过调用 Split() 返回两个新的 CipherState 对象。

- ReadMessage(message, payload_buffer):采用包含噪声握手消息的字节序列和将消息的明文有效负载写入的payload_buffer 。执行以下步骤,如果任何 DecryptAndHash() 调用返回错误,则中止:

  –从message_patterns 中获取并删除下一个消息模式,然后依次处理来自消息模式的每个标记:

    对于“e”:将re （必须为空)设置为消息中的下一个DHLEN字节。调用 MixHash(re.public_key)。

    对于“s”:如果HasKey() == True,则将temp设置为下一个DHLEN + 消息的 16个字节,否则设置为下一个DHLEN字节。将rs（必须为空)设置为 DecryptAndHash(temp)。

    对于“ee”:调用 MixKey(DH(e, re))。

    对于“es”:如果发起者调用MixKey(DH(e, rs)) ,如果响应者调用 MixKey(DH(s, re))。

    对于“se”:如果发起者调用MixKey(DH(s, re)) ,如果响应者调用 MixKey(DH(e, rs))。

    对于“ss”:调用 MixKey(DH(s, rs))。

  –对消息的剩余字节调用DecryptAndHash()和 将输出存储到 payload_buffer。

  –如果没有更多消息模式,则通过调用 Split() 返回两个新的 CipherState 对象。

6. 序幕

噪声协议有一个序言输入,它允许将任意数据散列到h变量中。如果双方不提供相同的序言数据,握手将由于解密错误而失败。当双方在握手之前进行谈判并希望确保他们对谈判有相同的看法时,这很有用。

例如,假设 Bob 向 Alice 传达他愿意支持的噪声协议列表。然后 Alice 将选择并执行一个协议。
为了确保“中间人”不会编辑 Bob 的列表以删除选项， Alice 和 Bob 可以将该列表作为序言数据。

请注意,虽然各方确认他们的序言相同,但他们不会将序言数据混合到加密密钥中。如果输入包含旨在加强加密的秘密数据,则应使用 PSK 握手（参见第 9 节）。

7.握手模式

7.1。握手模式基础知识

消息模式是集合中的一些标记序列（ “e”、“s”、“ee”、“es”、“se”、“ss”、“psk”）。在WriteMessage()
和ReadMessage()中对这些令牌的处理之前已经描述过,除了“ psk”令牌,它将在第 9 节中描述。未来的规范可能会引入其他令牌。

消息前模式是以下标记序列之一：

- “和”
- “s”
- “e, s”
- 空

握手模式包括：

·发起者的消息前模式,代表有关信息
响应者已知的发起者的公钥。

·响应者的消息前模式,代表有关信息
发起者知道的响应者的公钥。

- 实际握手消息的一系列消息模式。

预消息表示在握手之前以某种方式执行的公钥交换,因此这些公钥必须作为预消息“接收者”的 Initialize() 的输入。

第一个实际握手消息从发起者发送到响应者。
下一条消息从响应者发送,下一条从发起者发送,以此类推。

以下握手模式描述了由两种消息模式组成的未经身份验证的 DH 握手：

神经网络：
-> 和
<- e, ee

在下面的握手模式中,发起者和响应者都拥有静态密钥对,握手模式包含三种消息模式：

XX:
-> 和
<- e, ee, s, is -> s, is

握手模式名称是NN和XX。这个命名约定将在 7.5 节中解释。

非空预消息显示为分隔符“...”之前的预消息模式。如果双方都有预先消息，则首先列出发起者，并首先进行哈希处理。在 Initialize() 期间， MixHash()会在任何消息前公钥上调用，如第 5.3 节所述。

以下握手模式描述了发起方预先知道响应方的静态公钥并将其用于“零 RTT”加密的握手：

NK：

<- s

...

-> e, 是 <- e,

ee

在下面的握手模式中，双方都预先知道对方的静态公钥。首先列出发起者的预消息：

KK:

-> 小号

<- s

...

-> e, es, ss <- e, ee,

se

7.2.爱丽丝和鲍勃

在前面显示的所有握手模式中,发起者是左边的一方（用向右箭头发送）,响应者是右边的一方。

但是,在一个复合协议中可能会使用多个噪声协议,其中一个噪声协议中的响应者成为后续噪声协议的发起者。在这种情况下,为了方便术语和符号,我们引入了不同于发起者和响应者角色的Alice和Bob角色的概念。 Alice 将被视为左边的一方（发送带有右箭头的消息）,而 Bob 将被视为右边的一方。

以规范形式（即Alice 发起的形式)编写的握手模式假定发起者是 Alice（最左边的一方）。到目前为止,所有处理规则和讨论都假定了规范形式的握手模式。

然而,握手模式可以通过颠倒箭头和 DH 令牌以Bob 发起的形式编写（例如,将“es”替换为“se”,反之亦然）。
这不会改变握手模式,它只是让并排查看 Alice 发起的和 Bob 发起的握手变得更容易。

以下是上一节中 Bob 发起的握手模式：

神经网络：
<- 和
-> e,ee

XX:
<- 和
-> e, ee, s, se <- s, es

NK：
-> 小号
...
<- e, se ->
e, ee

KK:
<- s
-> 小号
...
<- e, se, ss -> e,
ee, es

有关 Bob 发起的符号的示例,请参阅第 10.2 节。

7.3.握手模式有效性

握手模式必须在以下意义上有效：

1.各方只能在他们的私钥和公钥之间进行DH
具有。

2.每次握手时,各方不得多次发送其静态公钥或临时公钥（即包括预消息在内， “e”和“s”的出现次数不得超过一次,任何一方发送的消息）。

3.各方每次握手不得多次执行 DH 计算（即每次握手不得出现超过一次的“ee”、“es”、 “se”或“ss”）。

4.在远程公钥（静态或临时)和本地静态密钥之间执行 DH 后,本地方不得调用 ENCRYPT() ,除非它也在其本地临时密钥和远程公钥之间执行了 DH。特别是,这意味着（使用规范符号）：

在“se”令牌之后,发起者不得发送握手有效负载或传输有效负载,除非还存在“ee”令牌。

在“ss”令牌之后,发起者不得发送握手有效负载或传输有效负载,除非还有“es”令牌。

在“es”令牌之后,除非还有“ee”令牌,否则响应者不得发送握手有效负载或传输有效负载。

在“ss”令牌之后,除非还有“se”令牌,否则响应者不得发送握手有效负载或传输有效负载。

第一次检查失败的模式显然是无稽之谈。

第二个和第三个检查非法值的冗余传输和冗余计算,以简化实现和测试。

第四次检查有两个目的：

·首先,这是必要的,因为Noise 依赖于涉及临时密钥的DH 输出来随机化共享密钥。未能通过此检查的模式可能会导致灾难性的密钥重用,因为受害者可能会发送使用不包含本地临时密钥贡献的密钥加密的消息（或者本地临时密钥的贡献被来自的无效临时密钥无效)另一方）。

·其次,此检查保证使用临时密钥来提供重要的安全属性,例如前向保密和密钥泄露模拟抵抗。

建议用户仅使用下面列出的握手模式,或经过专家审查以满足上述检查的其他模式。

7.4.单向握手模式

以下握手模式表示支持从发送方到接收方的单向数据流的“单向”握手。这些模式可用于加密文件、数据库记录或其他非交互式数据流。

在单向握手之后,发送者可以发送传输消息流,使用Split()返回的第一个CipherState对它们进行加密。

来自Split()的第二个CipherState被丢弃 - 接收者不得使用它发送任何消息（因为这将违反第 7.3 节中的规则）。

单向模式以单个字符命名,表示发送者静态密钥的状态：

· N =没有发件人的静态密钥 · K = 发件人的静态密钥 收件人已知 · X = 发件人的静态密钥已向收件人发送（“传输”）

纽：
<- s
...
-> e, 是

克：
-> 小号
<- s
...
-> e, 它, ss

X：
<- s
...
-> e, 它, s, ss

N是传统的基于 DH 的公钥加密。其他模式添加了发件人身份验证,其中发件人的公钥要么事先为收件人所知（K）,要么在加密下传输（X）。

7.5。交互式握手模式（基础）

以下握手模式代表交互式协议。这 12种模式被称为基本的交互式握手模式。

基本的交互模式用两个字符命名,表示发起者和响应者的静态键的状态：

第一个字符是指发起者的静态密钥：

· N =发起者没有静态密钥 · K =
发起者的静态密钥响应者已知 · X = 发起者的静态密钥
已发送（“传输”）给响应者 · I = 发起者的静态密钥立即传输给响应者,尽
管身份减少或不存在隐藏

第二个字符指的是响应者的静态键：

· N =响应者没有静态密钥 · K = 响
应者的静态密钥发起者已知 · X = 响应者的静态密钥向
发起者发送（“传输”）

神经网络:

-> 和

<- e, ee

KN:

-> 小号

...

-> 和

<- e, ee, se

NK:

<- s

...

-> e, 是 <- e, ee

KK:

-> 小号

<- s

...

-> e, es, ss <- e, ee, se

恩智浦:

-> 和

<- e,ee,是的,是

克星:

-> 小号

...

-> 和

<- e, ee, is, s, is

XN:

-> 和

<- e, ee -> s, se

在:

-> e, c <- e,

ee, ce

新科:

<- s

...

-> e, is <- e, ee

-> s, is

我:

<- s

...

-> e, es, s, ss <- e, ee, se

XX:

-> 和

<- e, ee, s, is -> s, is

九:

-> e, s <- e,

ee, is, s, is

XX模式是最通用的,因为它支持相互认证和静态公钥的传输。

所有基本模式都允许对握手有效负载进行一些加密:

·发起者预先知道响应者的静态公钥的模式(即以K 结尾的模式)允许零RTT加密,这意味着发起者可以加密第一个握手有效负载。

·所有基本模式都允许对第一个响应有效负载进行半 RTT加密,但加密仅针对以 K 或 I 开头的模式中的发起者静态公钥。

握手有效负载的安全属性通常比传输有效负载实现的最终安全属性弱,因此必须谨慎使用这些早期加密。

在某些模式中,传输负载的安全属性也会有所不同。特别是:以K开头的模式或我有一个警告,即响应者只能保证它发送的传输消息的“弱”前向保密,直到它从发起者接收到传输消息。在收到来自发起者的传输消息后,响应者确信“强”转发

保密。

7.7 节对这些有效载荷安全属性进行了更多分析。

7.6.交互式握手模式(延迟)

上一节中的基本握手模式尽可能早地执行 DH操作以进行身份验证(“es”和“se”)。

可以描述一组额外的握手模式,这些模式将这些身份验证 DH 推迟到下一条消息。为了命名这些延迟握手模式,在基本模式名称中的第一个和/或第二个字符之后使用数字“1”来指示发起者和/或响应者的验证 DH 被推迟到下一个消息。

由于以下几个原因,延迟模式可能很有用:

·发起者可能事先知道响应者的静态公钥,但不希望发送任何 0-RTT 加密数据。

·在某些情况下,推迟身份验证可以改善身份隐藏
握手的属性(参见第 7.8 节)。

· Noise 的未来扩展可能能够用签名或 KEM 密文替换 DH 操作,但只有在发送者正在验证自己(签名)或发送者正在验证接收者(KEM 密文)时才能这样做。因此,每个基本握手模式只能具有每个身份验证 DH

替换为签名或KEM 密文,但延迟变体使两种替换都成为可能。

下面是两个示例,左侧显示基本握手模式,右侧显示延迟变体。完整的 23 种延迟握手模式在附录中。

NK:
<- s
...
-> e, 是 <- e,
ee

NK1:
<- s
...
-> 和
<- e, ee, 是

XX:
-> 和
<- e, ee, s, is -> s, is

X1X:
-> 和
<- e,ee,是的,是
-> 小号
<-se

XX1:
-> 和
<- e, ee, s -> 是, s,
是

X1X1:
-> 和
<- e, ee, s -> es, s

<-se

7.7.有效负载安全属性

下表列出了第 7.4 节中所有单向模式和第 7.5 节中基本模式的噪声握手和传输有效负载的安全属性。每个有效负载都被分配了一个关于提供给接收者的发送者的身份验证程度的“源”属性,以及一个关于提供给发送者的机密程度的“目标”属性。

源属性是:

0.没有认证。此有效载荷可能已由任何一方发送,
包括主动攻击者。

1.发件人身份验证易受密钥泄露模拟 (KCI) 的影响。发件人身份验证基于涉及双方静态密钥对的静态静态 DH （“ss”） 。如果接收者的长期私钥已被泄露,则可以伪造此身份验证。请注意,Noise 的未来版本可能包括签名,这可以改善此安全属性,但会带来其他权衡。

2.抗密钥泄露模拟 (KCI) 的发件人身份验证。发件人身份验证基于发件人的静态密钥对和收件人的临时密钥对之间的临时静态DH （“es”或“se”） 。假设相应的私钥是安全的,则无法伪造此身份验证。

目标属性是：

0.不保密。此有效负载以明文形式发送。

1.加密到临时收件人。此有效负载具有前向保密性,因为加密涉及临时-临时 DH （“ee”）。
但是,发件人尚未对收件人进行身份验证,因此此有效负载可能会发送给任何一方，包括主动攻击者。

2.对已知收件人进行加密,仅对发件人进行前向保密,易受重放攻击。此有效负载仅基于涉及接收者的静态密钥对的 DH 进行加密。如果接收者的静态私钥被泄露,即使是在以后,这个有效载荷也可以被解密。该消息也可以重播,因为接收者没有短暂的贡献。

3.对已知接收者的加密,弱前向保密。此有效负载是基于临时-临时 DH 以及涉及接收者的静态密钥对的临时-静态 DH 加密的。但是,接收者所谓的临时公钥和接收者的静态公钥之间的绑定尚未得到发送者的验证,因此接收者的所谓临时公钥可能是由主动攻击者伪造的。在这种情况下,攻击者稍后可能会破坏接收者的静态私钥来解密有效负载。请注意， Noise 的未来版本可能包括签名,这可以改善此安全属性,但会带来其他权衡。

4.对已知接收者进行加密,如果发送者的私钥已被泄露,则为弱前向保密。此有效负载是基于临时-临时 DH 加密的,并且还基于涉及接收者的静态密钥对的临时-静态 DH。但是,接收者所谓的临时公钥和接收者静态公钥之间的绑定仅根据涉及这些公钥和发送者静态私钥的 DH 进行了验证。因此,如果发送者的静态私钥先前被泄露,则接收者所谓的临时公钥可能是由主动攻击者伪造的。在这种情况下，

攻击者稍后可能会破坏预期收件人的静态私钥
解密有效载荷（这是“KCI”攻击的一种变体,能够
“弱前向保密”攻击）。请注意,未来版本的 Noise 可能
包括签名,这可以提高这种安全性,但会带来
其他权衡。

5.对已知接收者进行加密,强前向保密。这个
有效载荷是基于一个临时的临时 DH 以及一个
ephemeral-static DH 与接收者的静态密钥对。假设
临时私钥是安全的,并且接收者没有主动
由窃取其静态私钥的攻击者冒充,这
有效载荷无法解密。

对于单向握手,下面列出的安全属性适用于握手有效载荷以及传输有效载荷。

对于交互式握手,会列出每次握手的安全属性
有效载荷。传输负载被列为不带图案的箭头。运输
仅当有效负载具有与以前不同的安全属性时才会列出
从同一方发送的握手有效载荷。如果两个传输有效载荷是
列出,第二个的安全属性仅在收到第一个时适用。

	资源	目的地
ñ	0	2
	1	2
X	1	2
-> 和	0	0
<- e, ee	0	1
->	0	1

NK

<- s		
...		
-> e, 是	0	2
<- e, ee	2	1
->	0	5

NX

-> 和	0	0
<- e ,ee ,是的 ,是	2	1
->	0	5

XN

-> 和	0	0
<- e, ee -> s,	0	1
se	2	1
<-	0	5

新康

<- s		
...		
-> e, 是	0	2
<- e, ee	2	1
-> s, se	2	5
<-	2	5

XX

-> 和	0	0
<- e, ee, s, is -> s, is	2	1
	2	5
<-	2	5

KN

-> 小号		
...		
-> 和	0	0
<- e, ee, se	0	3
->	2	1
<-	0	5

KK		
-> 小号		
<- s		
...		
-> e, 它, ss	1	2
<- e, ee, se	2	4
->	2	5
<-	2	5
克星		
-> 小号		
...		
-> 和	0	0
<- e, ee, is, s, is	2	3
->	2	5
<-	2	5
在		
-> e, c <- e,	0	0
ee, ce	0	3
->	2	1
<-	0	5
我		
<- s		
...		
-> e, es, s, ss <- e, ee, se	1	2
	2	4
->	2	5
<-	2	5
九		
-> e, s <- e,	0	0
ee, is, s, is	2	3
->	2	5
<-	2	5

7.8.身份隐藏

下表列出了第 7.4 节中所有单向握手模式和第7.5 节中基本握手模式的身份隐藏属性。此外,我们列出了一些延迟握手模式,它们具有与相应的基本模式不同的身份隐藏属性。

每个模式都分配有描述提供给发起者的静态公钥和响应者的静态公钥的机密性的属性。基本假设是临时私钥是安全的,并且如果各方从他们不信任的另一方收到静态公钥,则双方将中止握手。

本节仅考虑通过握手中的静态公钥字段泄露身份。当然,Noise 参与者的身份可能会通过其他方式暴露,包括有效载荷字段、流量分析或IP 地址等元数据。

相关公钥的属性是:

0. 清晰传输。

1.前向保密加密,但可以被匿名者探测
发起人。

2. 前向保密加密,但发送给匿名响应者。

3.不传输,但被动攻击者可以检查响应者私钥的候选者,并确定候选者是否正确。攻击者还可以将先前记录的消息重播给新的响应者,并通过接收者是否接受该消息来确定两个响应者是否“相同”(即使用相同的静态密钥对)。

4.加密为响应者的静态公钥,无需前向保密。如果攻击者知道响应者的私钥,他们就可以解密发起者的公钥。

5.不传输,但被动攻击者可以检查候选对(响应者的私钥,发起者的公钥),并了解候选对是否正确。

6.加密但前向保密性较弱。一个主动攻击者在没有发起者的静态私钥的情况下伪装成发起者,然后学习发起者的私钥,然后可以解密响应者的公钥。

7.没有被传输,但是主动攻击者在没有发起者的静态私钥的情况下伪装成发起者,然后学习到发起者私钥的候选者,然后可以检查候选者是否正确。

8. 对认证方进行前向保密加密。

9. 主办发行者[illegible]的候选者。响应者

克星	7	6
在	0	-
我	4	3
IK1	0	9
九	0	6

8. 协议名称和修饰符

要为Initialize()生成噪声协议名称,请将 ASCII 字符串“Noise_”与四个下划线分隔的名称部分连接起来,这些部分依次命名握手模式、DH 函数、密码函数,然后是哈希函数。结果名称必须为 255 个字节或更少。

例子:

· Noise_XX_25519_AESGCM_SHA256 ·
Noise_N_25519_ChaChaPoly_BLAKE2s ·
Noise_IK_448_ChaChaPoly_BLAKE2b

每个名称部分必须仅包含字母数字字符(即范围“A” ... “Z”、 “a” ... “z”和“0” ... “ 9”之一中的字符),以及两个特殊字符“+”和“/”。

附加规则适用于每个名称部分,如下所述。

8.1。握手模式名称部分

握手模式名称部分包含握手模式名称以及零个或多个模式修饰符的序列。

握手模式名称必须是仅包含字母字符或数字的大写 ASCII 字符串(例如“XX1”或“IK”)。

模式修饰符指定对握手模式指定的行为的任意扩展或修改。例如,可以将修饰符应用于握手模式,根据某些规则将其转换为不同的模式。 “ psk0”和“fallback”修饰符就是这方面的例子,将在本文档后面定义。

模式修饰符以小写字母数字 ASCII 字符串命名,该字符串必须以字母字符(而非数字)开头。模式修饰符附加到基本模式,如下所述:

添加到基本模式的第一个修饰符只是简单地附加。因此,当添加到“XX”模式中时, “fallback”修饰符会产生“XXfallback”。
其他修饰符用加号分隔。因此,添加“psk0”修饰符将导致名称部分“XXfallback+psk0”,或完整的协议名称,例如“Noise_XXfallback+psk0_25519_AESGCM_SHA256”。

在某些情况下,修饰符的顺序将指定不同的协议。但是,如果某些修饰符的顺序无关紧要,则它们需要按字母顺序排序(这是确保互操作性的任意约定)。

8.2.密码算法名称部分

DH、密码和哈希名称部分的规则是相同的。每个名称部分必须包含一个或多个由加号分隔的算法名称。

每个算法名称必须仅由字母数字字符和正斜杠字符(/) 组成。建议算法名称简短,并且仅在必要时使用“/”字符以避免歧义（例如

“SHA3/256”优于“SHA3256”）。

在大多数情况下,每个名称部分都会有一个算法名称（即没有加号）。多个算法名称仅在模式或修饰符调用时使用。

本文档中的任何模式或修饰符都不需要任何名称部分中的多个算法名称。但是,此功能可能在未来的扩展中很有用。例如,DH 部分可能会使用多个算法名称来指定“混合”后量子前向保密;或者可以为不同的目的指定多个哈希算法。

9. 预共享对称密钥

Noise 提供预共享的对称密钥或PSK模式来支持双方都有 32 字节共享密钥的协议。

9.1。密码函数

PSK 模式使用SymmetricState.MixKeyAndHash()函数将PSK 混合到加密密钥和 h 值中。

请注意， MixKeyAndHash()使用HKDF (..., 3)。 HKDF()的第三个输出用作 k 值,因此如果不使用 k,则可以跳过 k 的计算。

9.2.握手令牌

在 PSK 握手中,允许“psk”令牌在握手模式中出现一次或多次。此标记只能出现在消息模式（而不是消息前模式)中。此令牌通过调用MixKeyAndHash(psk) 进行处理,其中 psk 是应用程序提供的 32 字节秘密值。

在非 PSK 握手中,消息前模式或消息模式中的“e”标记总是导致调用MixHash(e.public_key)。在 PSK 握手中,所有这些调用都跟随着MixKey(e.public_key)。结合下一节中的有效性规则,这可确保基于 PSK 的加密使用使用临时公钥作为随机数随机化的加密密钥。

9.3.有效性规则

为了防止灾难性的密钥重用,使用“psk”令牌的握手模式必须遵循额外的有效性规则：

- 一方在处理“psk”令牌后不得发送任何加密数据,除非它先前已在“psk”令牌之前或之后发送过临时公钥（ “e”令牌） 。

此规则保证从 PSK 派生的k永远不会用于加密,除非它也被MixKey(e.public_key)使用自选的临时公钥随机化。

9.4。模式修饰符

为了指示 PSK 模式和“psk”标记的位置,使用了模式修饰符（参见第 8 节）。修饰符psk0在第一个握手消息的开头放置一个“psk”标记。修饰符psk1、 psk2等在第一个、第二个等握手消息的末尾放置一个“psk”标记。

使用这些修饰符之一的任何模式都必须根据第 9.2 节中的规则处理令牌,并且必须遵循第 9.3 节中的有效性规则。

下表左侧列出了一些未修改的单向模式,右侧列出了推荐的 PSK 模式：

纽： <- s ... -> e, 是	Npsk0： <-s ... -> psk, e, 它
克： -> 小号 <- s ... -> e, 它, ss	Kpsk0：- > 秒 <- s ... -> psk, e, it, ss
X： <- s ... -> e, 它, s, ss	Xpsk1： <- s ... -> e, es, s, ss, psk

请注意,建议X 使用psk1修饰符。这是因为X传输发起者的静态公钥。因为 PSK 通常是成对的,所以响应者可能无法确定 PSK,直到它解密了发起者的静态公钥。因此,psk1 在这里可能比 psk0 更有用。

按照类似的逻辑,我们可以定义最有可能的交互式 PSK 模式:

神经网络:
-> 和
<- e, ee

神经网络:
-> 和
<- e, ee

NK:
<- s
...
-> e, 是 <- e,
ee

NK:
<- s
...
-> e, 是 <- e,
ee

恩智浦:
-> 和
<- e,ee,是的,是

XN:
-> 和
<- e, ee -> s,
se

HHпск0:
-> пск, e <- e,
ee

NNpsk2：-
> e
<- e, ee, psk

NKpsk0:
<- s
...
-> psk, e, 是 <- e, ee

NKpsk2:
<- s
...
-> e, 是 <- e,
ee, psk

NXpsk2: ->
e
<- e, ee, s, is, psk

XNpsk3：-
> e
<- e, ee -> s,
se, psk

新科：
<- s
...
-> e, 是
<- e, ee
-> s, se

XK psk3：
<- s
...
-> e, 是
<- e, ee
-> s、se、psk

XX：
-> 和
<- e, ee, s, is -> s, is

XXpsk3: ->
e
<- e, ee, s, es -> s, se, psk

KN：
-> 小号
...
-> 和
<- e, ee, se

KNpsk0：-
> 小号
...
-> psk, e <- e,
ee, se

KN：
-> 小号
...
-> 和
<- e, ee, se

KNpsk2：
-> 小号
...
-> 和
<- e, ee, se, psk

KK：
-> 小号
<- s
...
-> e, es, ss <- e, ee,
se

KKpsk0：
-> 小号
<- s
...
-> psk, e, es, ss <- e, ee, se

KK：
-> 小号
<- s
...
-> e, es, ss <- e, ee,
se

KKpsk2：
-> 小号
<- s
...
-> e, es, ss <- e, ee,
se, psk

克星：

-> 小号

...

-> 和

<- e, ee, is, s, is

在：

-> e, s <- e,

ee, se

在：

-> e, c <- e,

ee, ce

我：

<- s

...

-> e, es, s, ss <- e, ee, se

我：

<- s

...

-> e, es, s, ss <- e, ee, se

九：

-> e, s <- e,

ee, is, s, is

KXpsk2：

-> 小号

...

-> 和

<- e, ee, se, s, es, psk

INpsk1: ->

e, s, psk <- e, ee, se

INpsk2: ->

e, s <- e, ee,

se, psk

IKpsk1：

<- s

...

-> e, es, s, ss, psk <- e, ee, se

IKpsk2：

<- s

...

-> e, es, s, ss <- e, ee, se,

psk

IXpsk2: ->

e, s <- e, ee,

se, s, es, psk

上面的列表并没有穷尽所有可以使用这些修饰符形成的模式。特别是，这些 PSK 修饰符中的任何一个都可以安全地应用于任何先前命名的模式，从而产生像IKpsk0、 KKpsk1甚至XXpsk0+psk3 这样的模式，这些都没有在上面列出。

这仍然没有穷尽所有可以在这些修饰符之外使用“psk”标记的方式（例如，将“psk”标记放置在消息模式的中间）。定义额外的 PSK 修饰符超出了本文档的范围。

10. 复合协议

10.1。复合协议的基本原理

到目前为止,我们假设 Alice 和 Bob 希望执行由发起者（Alice)选择的单个噪声协议。然而,在收到 Alice 的第一条消息后,Bob 可能希望切换到不同的噪声协议的原因有很多。例如：

- Alice 可能选择了基于密码、DH 函数的噪声协议，
  或 Bob 不支持的握手模式。
- Alice 可能基于Bob 的静态公钥或 PSK 的过期版本发送了“零 RTT”加密初始消息。

处理这些场景需要一个复合协议,其中 Bob从 Alice 选择的初始噪声协议切换到新的噪声协议。在这样的复合协议中,发起者和响应者的角色将互换　Bob将成为新噪声协议的发起者,而 Alice 将成为响应者。

复合协议引入了显着的复杂性,因为 Alice 需要宣传她开始使用的噪声协议和她能够切换到的噪声协议,并且双方必须协商安全过渡。

这些细节在很大程度上超出了本文档的范围。但是,为了举例说明如何构建复合协议并提供一些构建块,以下部分定义了一个回退修饰符并展示了如何使用它来创建噪声管道复合协议。

Noise Pipes 支持XX模式,但也允许 Alice 缓存 Bob 的静态公钥并尝试使用 0-RTT 加密进行 IK 握手。

如果 Bob 无法解密 Alice 的初始IK消息,他将切换到
XXfallback模式,本质上允许各方完成XX握手,就好像 Alice 发送了XX初始消息而不是IK初始消息

信息。

10.2.后备修饰符

回退修饰符通过将Alice 的初始消息转换为 Bob 必须通过其他方式（例如,通过来自 Alice的初始IK消息)接收的预消息,将 Alice 发起的模式转换为 Bob 发起的模式。

在此转换之后,握手模式的其余部分被解释为Bob 发起的握手模式。

例如,下面是应用于XX以产生XXfallback的fallback修饰符：

XX:

-> 和

<- e,ee,是的,是

-> s, se

XX后备：

-> 和

...

<- e, ee, s, is -> s, is

请注意,回退只能应用于 Alice 发起形式的握手模式,其中 Alice 的第一条消息能够被解释为前消息（即它必须是“e”、“s”或“e、s”）.

10.3.零 RTT 和噪声协议

用于零 RTT 加密的典型复合协议涉及三种不同的
噪声协议：

·如果Alice 不拥有启用零RTT 加密的有关Bob 的存储信息,或者不希望使用零RTT 握手,则使用完整协议。

·零RTT 协议允许对初始消息中的数据进行加密。 ·如果 Bob 无法解密 Alice 的第一个协议,则他触发交换协议

零 RTT 握手消息。

Bob在收到第一条消息时必须有某种方法来区分完整和零 RTT 情况。如果 Alice 进行了零 RTT 尝试,则必须有某种方法让她在收到响应时区分零 RTT 和切换情况。

例如,每个握手消息之前可以有一些协商数据,例如类型字节（参见第 13 节）。该数据本身不是噪声消息的一部分,而是表明正在使用哪个噪声协议。

10.4.噪音管

本节定义了Noise Pipe复合协议。以下握手模式满足上一节中讨论的完整、零 RTT 和切换角色,因此可用于提供具有简单零 RTT 选项的完整握手：

XX:

-> 和

<- e, ee, s, is -> s, is

我：

<- s

...

-> e, es, s, ss <- e, ee, se

XX后备：

-> 和

...

<- e, ee, s, is -> s, is

如果双方之前没有通信，则使用XX模式进行完整握手，之后 Alice 可以缓存 Bob 的静态公钥。

IK 模式用于零 RTT 握手。

如果 Bob 未能解密初始 IK 消息（可能是由于更改了他的静态密钥），则XXfallback模式用于交换握手。

10.5。握手无法区分

各方可能希望对窃听者隐藏他们正在执行的握手类型。例如，假设各方正在使用噪声管道，并且想要隐藏他们是否正在执行完整握手、零 RTT 握手或回退握手。

这很容易：

·前三个消息的有效负载可以用随机字节填充到恒定大小，而不管执行哪个握手。

- Bob 将尝试将第一条消息解密为IK消息，并将
  如果解密失败，切换到 XXfallback。
- ·发送IK初始消息的 Alice 可以使用试用解密来
  区分使用 IK 或 XXfallback 的响应。
- ·尝试完全握手的 Alice 将发送一个临时公钥，然后是随机填充，并将使用XXfallback来处理响应。
  请注意，不使用XX ，因为服务器无法通过试用解密将XX消息与失败的 IK 尝试区分开来。

这使噪声临时公钥保持清晰。临时公钥是随机选择的 DH 公共值，但它们通常具有足够的结构，即使窃听者无法区分不同的握手，窃听者也可能怀疑各方正在使用噪声。为了使短暂与随机字节序列无法区分，可以使用 Elligator [5] 等技术。

11. 高级功能

11.1。虚拟键

考虑一个协议，如果响应者请求，发起者将验证自己。这可以被视为发起者根据响应者的第一次握手有效负载中的某些值在 NX 和 XX 等模式之间进行选择。

噪声并不直接支持这一点。相反，这可以通过始终执行XX 来模拟。如果不请求身份验证，发起者可以通过发送一个虚拟的静态公钥来模拟NX情况。虚拟公钥的值无关紧要。

这种技术很简单，因为它允许使用单一的握手模式。它也没有透露从消息大小或计算时间中选择了哪个选项。可以扩展它以允许XX模式支持任何身份验证排列（仅发起者、仅响应者、两者或无）。

类似地，虚拟 PSK （例如全零的 PSK)将允许协议可选地支持 PSK。

11.2.通道绑定

各方可能希望执行噪声协议，然后使用签名、密码或其他方式在应用层执行身份验证。

为了支持这一点，Noise 库可能会在握手完成后调用GetHandshakeHash() ，并将返回的值作为唯一标识 Noise 会话的握手哈希公开给应用程序。

然后，各方可以签署握手散列，或将其与密码一起散列，以获得具有“通道绑定”属性的身份验证令牌：该令牌不能被具有不同会话的接收方使用。

11.3.重新生成密钥

各方可能希望使用单向函数定期更新其密码状态密钥，以便密码状态密钥的妥协不会解密较旧的消息。

定期更新密钥也可用于减少在单个密码密钥下加密的数据量（这对于好的密码通常并不重要,尽管请注意第 14 节中关于 AESGCM 数据量的讨论）。

为此,Noise 支持可在CipherState上调用的Rekey()函数。

是否以及何时执行重新生成密钥取决于应用程序。例如：

·应用程序可能会执行连续的密钥更新,即在发送或接收每条传输消息后重新加密相关的密码状态。这很简单,并且可以很好地保护较旧的密文,但对于更改密钥代价高昂的实现可能会很困难。

·应用程序可能会在加密状态之后自动对其进行重新加密
用于发送或接收一些消息。

·应用程序可能会根据任意标准选择重新生成密钥,在这种情况下,它们会通过发送消息向对方发出信号。

应用程序必须自行做出这些决定;没有指定 rekey 行为的模式修饰符。

请注意,rekey 仅更新 cipherstate 的k值,它不会重置cipherstate 的n值,因此如果发送264或更多传输消息,执行 rekey 的应用程序仍必须执行新的握手。

11.4.乱序传输消息

在某些用例中,噪声传输消息可能会丢失或乱序到达（例如,当通过 UDP 发送消息时）。为了处理这个问题,应用程序协议可以发送用于加密每个传输消息的n值以及该消息。在接收到这样的消息时,接收者将使用接收到的 n 值在接收 CipherState 上调用 SetNonce() 函数。

这样做的接收者必须跟踪接收到的解密成功的n 个值,并拒绝任何重复该值的消息,以防止重放攻击。

请注意,有损和无序消息传递会引入许多其他问题（包括无序握手消息和拒绝服务风险）,这些问题超出了本文档的范围。

11.5。半双工协议

在一些应用协议中,各方严格交替发送消息。
在这种情况下,噪声可用于半双工模式 [6],其中Split()返回的第一个CipherState用于加密两者中的消息

方向， Split()返回的第二个CipherState未使用。这允许进行一些小的优化，因为Split()只需计算单个输出CipherState，并且双方只需要在传输阶段存储单个 CipherState。

必须非常小心地使用此功能。特别是，如果协议没有严格交替并且双方使用相同的CipherState和 nonce 值加密不同的消息，这将是一个灾难性的安全故障。

12. DH 函数、密码函数和散列函数

12.1. 25519 DH 功能

- GENERATE_KEYPAIR()：返回一个新的 Curve25519 密钥对。
- DH(keypair, public_key)：执行 Curve25519 DH 函数（又名[7] 中的“X25519”）。无效的公钥值将产生所有的输出
  零。

  或者，允许实现检测产生全零输出的输入并发出错误信号。不鼓励这种行为，因为它增加了复杂性和实现差异，并且不会提高安全性。这种行为是允许的，因为它可能与某些软件的行为相匹配。

- DHLEN = 32

12.2. 448 DH 功能

- GENERATE_KEYPAIR()：返回一个新的 Curve448 密钥对。
- DH(keypair, public_key)：执行 Curve448 DH 函数（在 [7] 中又称为“X448”）。无效的公钥值将产生全零的输出。

  或者，允许实现检测产生全零输出的输入并发出错误信号。不鼓励这种行为，因为它增加了复杂性和实现差异，并且不会提高安全性。这种行为是允许的，因为它可能与某些软件的行为相匹配。

- DHLEN = 56

12.3. ChaChaPoly 密码函数

- ENCRYPT(k, n, ad, plaintext) / DECRYPT(k, n, ad, ciphertext):来自 [8] 的 AEAD_CHACHA20_POLY1305 。 96 位随机数是通过对 32 位零进行编码,然后对 n 进行小端编码形成的。 （ChaCha20 的早期实现使用 64 位随机数;通过这些实现,可以兼容将 n直接编码为没有 32 位零前缀的 ChaCha20 随机数）。

12.4. AESGCM 密码函数

- ENCRYPT(k, n, ad, plaintext) / DECRYPT(k, n, ad, ciphertext):带有 GCM 的 AES256，来自 [9],密文附加了 128 位标签。
  96 位随机数是通过对 32 位零进行编码,然后对 n 进行大端编码来形成的。

12.5。 SHA256 哈希函数

- HASH（输入）:来自[10] 的SHA-256。
- HASHLEN = 32 · BLOCKLEN = 64

12.6。 SHA512 哈希函数

- HASH（输入）:来自[10] 的SHA-512。
- HASHLEN = 64 · BLOCKLEN = 128

12.7。 BLAKE2s 哈希函数

· HASH(input):来自 [11] 的 BLAKE2,摘要长度为 32。 · HASHLEN = 32 · BLOCKLEN = 64

12.8. BLAKE2b 哈希函数

· HASH(输入):来自 [11] 的 BLAKE2b,摘要长度为 64。 · HASHLEN = 64 · BLOCKLEN = 128

13. 申请职责

基于噪声构建的应用程序必须考虑几个问题：

·选择加密函数：建议将25519 DH 函数用于典型用途，但448 DH 函数可能会提供额外的安全性，以防针对椭圆曲线密码术开发密码分析攻击。

448 DH 函数应与 512 位散列一起使用，如SHA512或BLAKE2b 。 25519 DH 函数可以与SHA256或BLAKE2s 等 256 位散列一起使用，尽管512 位散列可能会提供额外的安全性，以防针对较小的散列函数开发密码分析攻击。

AESGCM很难在软件中以高速和恒定时间实现。

·可扩展性：建议应用程序对所有消息的有效负载使用可扩展的数据格式（例如 JSON、Protocol Buffers）。这确保了将来可以添加被旧实现忽略的字段。

·填充：建议应用程序对所有允许填充的加密消息的有效负载使用数据格式。这允许实现避免泄漏有关消息大小的信息。根据前面的要点，使用可扩展的数据格式可能就足够了。

·会话终止：应用程序必须考虑到一系列噪声传输消息可能会被攻击者截断。应用程序应在传输有效负载内包含显式长度字段或终止信号，以表示交互式会话的结束或单向传输消息流的结束。

·长度字段：应用程序必须处理噪声消息的任何帧或附加长度字段，考虑到噪声消息的长度可能高达 65535 字节。如果需要显式长度字段，建议应用程序在每条消息之前添加一个 16 位大端长度字段。

·协商数据：应用程序可能希望在握手之前和/或在每个握手消息之前支持某些协商数据的传输。协商数据可能包含诸如噪声协议的版本信息和标识符之类的东西。例如，一个简单的方法是在每个噪声握手消息之前发送一个单字节类型字段。更灵活的方法可能会发送可扩展的结构，例如 protobuf。协商数据引入了显着的复杂性和安全风险，例如回滚攻击（见下一节）。

14. 安全考虑

本节收集了各种安全注意事项：

·身份验证：具有静态公钥的噪声协议验证参与者是否拥有相应的私钥，但由应用程序确定远程方的静态公钥是否可接受。这样做的方法包括签署公钥的证书（并且可以在握手有效负载中传递）、预先配置的公钥列表或“固定”/“密钥连续性”方法，其中各方记住他们遇到的公钥并检查是否同一方在未来提供相同的公钥。

·会话终止：防止攻击者截断传输消息流是应用程序的职责。请参阅上一节。

·回滚：如果各方基于一些先前的协商（未包括在序言中)来决定噪声协议，那么回滚攻击可能是可能的。这是复合协议的一个特殊风险，如果噪声握手之前是双方之间的通信，则需要特别注意。

·静态密钥重用：与噪声一起使用的静态密钥对应与单个散列算法一起使用。密钥对不应在 Noise 之外使用，也不应与多个哈希算法一起使用。可以接受将静态密钥对与不同的噪声协议一起使用，前提是它们都使用相同的哈希算法。 （在 Noise 之外重用 Noise 静态密钥对需要非常仔细的分析，以确保使用不会相互损害，并保留安全证明）。

· PSK 重用：与噪声一起使用的 PSK 应与单个散列算法一起使用。 PSK 不应在噪声之外使用，也不应与多种哈希算法一起使用。

·临时密钥重用：噪声协议中的每一方都必须在发送任何加密数据之前发送一个新的临时公钥。绝不能重复使用临时密钥。违反这些规则可能会导致灾难性的密钥重用。这是第 7 节中的模式和第 7.3 节中的有效性规则背后的一个基本原理。这也是为什么单向握手只允许来自发送者而不是接收者的传输消息的原因。

·滥用公钥作为秘密：使用带有消息前公钥的模式并假设成功的握手意味着对方知道公钥可能很诱人。不幸的是，情况并非如此，因为将公钥设置为无效值可能会导致可预测的 DH 输出。例如， Noise_NK_25519发起者可能会发送一个无效的临时公钥，以导致已知的全零 DH 输出，尽管不知道响应者的静态公钥。如果各方想要使用共享密钥进行身份验证，则应将其用作 PSK。

·通道绑定:根据 DH 功能,恶意方可能会通过将公钥设置为导致可预测的 DH 输出的无效值（如上一个项目符号)来参与派生相同共享密钥的多个会话.也可以将公钥设置为等效值,从而为不同的输入产生相同的 DH 输出。这就是为什么更高级别的协议应该使用握手哈希(h)来进行唯一的通道绑定,而不是ck,如第 11.2 节所述。

·增加随机数:使用相同的密钥k重复使用n的随机数值进行加密将是灾难性的。实现必须仔细遵循随机数的规则。由于整数溢出,不允许 nonce 回零,并保留最大 nonce 值。这意味着各方不得发送超过264-1的传输消息。

·协议名称:与Initialize()一起使用的协议名称必须唯一标识与它一起使用的每个密钥（无论是临时密钥对、静态密钥对还是 PSK)的握手模式和加密函数的组合。如果使用相同的协议名称但使用不同的加密操作集重复使用相同的密钥,则可能会发生不良交互。

·预共享对称密钥:预共享对称密钥必须是具有 256 位熵的秘密值。

·数据量:随着使用单个密钥加密的数据量增加， AESGCM密码函数的安全性逐渐降低。
因此,各方不应发送超过256字节（约 72 PB)的由单个密钥加密的数据。如果可以发送如此大量的数据,则应选择不同的密码函数。

·散列冲突:如果攻击者可以在序言数据或握手散列上发现散列冲突,他们可能能够执行“脚本冲突”攻击,诱使各方对握手数据有不同的看法。

重要的是使用具有抗冲突哈希函数的噪声,并在任何弱点处替换哈希函数。

·实施指纹识别:如果此协议用于匿名方的设置,则应注意实施在所有情况下的行为相同。这可能需要强制执行处理无效 DH 公钥的确切行为。

15. 基本原理

本节收集了各种设计原理。

15.1.密码和加密

密码密钥和 PSK 是 256 位,因为:

·在考虑密码分析安全余量、时间/内存权衡、多密钥攻击、密钥更新和量子攻击时,256 位是密码密钥的保守长度。

·预共享密钥长度是固定的,以简化测试和实施,并防止用户错误地使用低熵密码作为预共享密钥。

随机数是 64 位,因为:

· 一些密码只有 64 位随机数（例如 Salsa20）。

· ChaCha20的初始规范和实现中使用了 64 位随机数,因此噪声随机数可以与这些实现一起使用。

· 64 位使得整个 nonce 可以很容易地被视为一个整数,并且递增。

· 96 位随机数（例如在 RFC 7539 中)是一个令人困惑的大小,不清楚是否随机随机数是可以接受的。

密文中的认证数据（即认证标签或合成IV)为 128 位,因为:

·某些算法（例如 GCM)在以下情况下会比理想 MAC 失去更多的安全性截断。

·噪声可用于各种环境,包括攻击者可以收到有关身份验证数据猜测是否正确的快速反馈。

· 单个固定长度比支持可变长度标签更简单。

密文必须与随机无法区分,因为:

·这使得噪声协议更容易与随机填充（用于长度隐藏）、抗审查的“不可指纹”协议或隐写术一起使用。但是请注意,除非使用诸如 Elligator [5] 之类的技术,否则临时密钥很可能与随机密钥区分开来。

Rekey 默认使用 nonce 264-1加密,因为:

·使用AESGCM和ChaChaPoly可以有效地计算rekey（ “加密”只需要应用密码,并且可以跳过认证标签的计算）。

Rekey 不会将 n 重置为零,因为：

· 保持n 不变很简单。

·如果密码有一个弱点,即重复更新密钥会产生一个密钥循环,那么让n提前将避免灾难性地重复使用相同的 k 和 n 值。

·让n提前对加密的总数设置一个界限，
可以使用一组派生键来执行。

AESGCM 数据量限制为256字节,因为：

·这是252个 AES 块（每个块为 16 个字节）。该限制基于生日冲突的风险,用于排除明文猜测。
攻击者可以排除对256字节明文进行随机猜测的概率小于百万分之一（大约为(252 * 252) / 2128）。

密码随机数对于AESGCM是大端,对于ChaCha20 是小端,因为：

· ChaCha20 在内部使用小端块计数器。

· AES-GCM 在内部使用大端块计数器。

· 在密码中使用一致的字节序是有意义的。

15.2.散列函数和散列

推荐的散列函数系列是 SHA2 和 BLAKE2,因为：

· SHA2 广泛可用,并且经常与AES 一起使用。

· BLAKE2 速度快,与ChaCha20 相似。

支持 256 位和 512 位的哈希输出长度,因为：

· 256 位散列在 128 位安全性下提供足够的抗冲突性
等级。

·与 SHA-512 和 BLAKE2b 相比,256 位哈希（SHA-256 和 BLAKE2s)在处理较小的输入（由于更小的块大小)时需要更少的 RAM 和更少的计算。

· SHA-256 和 BLAKE2 在 32 位处理器上比在内部使用 64 位操作的较大散列更快。

MixKey() 设计使用 HKDF,因为：

· HKDF 是众所周知的,HKDF “链”在其他地方也以类似的方式使用
协议（例如 Signal、IPsec、TLS 1.3）。

- HKDF 有一个已发表的分析[12]。
- HKDF 在每个MixKey()输入之间应用多层散列。
  这种“额外”的散列可能会减轻散列函数弱点的影响。

HMAC 与所有散列函数一起使用,而不是允许散列使用更专业的函数（例如键控 BLAKE2）,因为：

- HKDF 需要使用 HMAC,以及 [12] 中的一些 HKDF 分析
  取决于 HMAC 的嵌套结构。
- HMAC 广泛用于Merkle-Damgard 哈希,例如SHA2。 Keccak 和 BLAKE 等 SHA3候选者必须适合HMAC。因此,HMAC 应该适用于所有广泛使用的散列函数。
- HMAC 应用嵌套散列来处理每个输入。这种“额外”的散列可能会减轻散列函数弱点的影响。
- HMAC（和HKDF)是广泛使用的结构。如果在散列函数中发现一些弱点,密码分析者可能会在 HMAC 和 HKDF 的上下文中分析该弱点。
- 一致地应用HMAC 很简单,并且在使用不同的散列函数时避免了具有不同密码分析属性的定制设计。
- HMAC 易于构建在散列函数接口之上。如果仅使用底层散列不能实现更专业的功能（例如键控 BLAKE2） ,则不能保证在散列函数可用的任何地方都可用。

使用MixHash()而不是直接通过MixKey() 发送所有输入,因为：

- MixHash() 比 MixKey() 更有效。
- MixHash()产生一个可能对更高级别有用的非秘密值
  级协议,例如用于通道绑定。

h 值散列握手密文而不是明文,因为：

- 这确保h是一个非秘密值,可用于通道绑定或其他目的而不会泄露秘密信息。
- 这为密文的延展性提供了更强的保证。

15.3.其他

建议使用大端长度字段,因为：

- 长度字段可能通过解析大端“网络字节顺序”是传统的代码来处理。

· 一些密码在内部使用大端（例如GCM、SHA2）。 · 虽然

Curve25519、Curve448 和ChaCha20/Poly1305 确实使用little-endian，但它们很可能由专门的库处理，因此没有强烈的理由支持与它们保持一致。

会话终止留给应用程序，因为：

·在噪声中提供终止信号对应用没有多大帮助，因为应用程序仍然必须正确使用信号。

·对于具有自己的终止信号的应用，在噪声中具有第二个终止信号可能会造成混淆而不是有用。

不使用显式随机随机数（如 TLS“随机”字段），因为：

· 一次性临时公钥使显式随机数变得不必要。

·显式随机数允许重用临时公钥。然而，重用短暂（定期更换）更复杂，需要安全的时间源，在短暂妥协的情况下安全性较低，并且仅提供小的优化，因为可以以DH 操作成本的一小部分来完成密钥生成.

· 显式随机数会增加消息大小。

·显式随机数使“后门”加密实现更容易，例如通过修改RNG，使密钥恢复数据通过随机数字段泄露。

16. 知识产权

噪声规范（本文档）特此置于公共领域。

17. 致谢

噪音的灵感来自：

· Dan Bernstein 等人[13]、[14] 的NaCl 和CurveCP 协议。 · Hugo Krawczyk [15]、[16] 的 SIGMA 和 HOMQV 协议。 · Ian Goldberg 等人的 Ntor 协议 [17]。 · Mario Di Raimondo 等人对OTR 的分析[18]。 · Caroline Kudla 和Kenny Paterson 对Simon Blake-Wilson 等人[19]、[20] 的“协议4”的分析。 · Mike Hamburg 提出的基于海绵的协议框架，该框架

导致频闪[21]，[22]。 · 双棘轮
算法[23] 中使用的KDF 链。

关于规格和设计的一般反馈来自：Moxie Marlinspike、Jason Donenfeld、Rhys Weatherley、Mike Hamburg、David Wong、Jake McGinty、Tiffany Bennett、Jonathan Rudenberg、Stephen Touset、Tony Arcieri、Alex Wied、Alexey Ermishkin、Olaoluwa Osuntokun、Karthik巴尔加万和纳迪姆·科比西。

有用的编辑反馈来自：Tom Ritter、Karthik Bhargavan、David Wong、Klaus Hartke、Dan Burkert、Jake McGinty、Yin Guanhao、Nazar Mokryn skyi、Keziah Elis Biermann、Justin Cormack、Katriel Cohn-Gordon 和 Nadim Kobeissi。

关于密钥推导设计的有用意见和反馈来自：Moxie Mar linspike、Hugo Krawczyk、Samuel Neves、Christian Winnerlein、JP Aumasson和 Jason Donenfeld。

PSK 方法很大程度上是由 Jason Donenfeld基于他在 WireGuard 中使用 PSK 的经验而激发和设计的。

延迟模式是与 Justin Cormack 讨论的结果。附录中的模式推导规则也来自 Justin Cormack。

延迟模式的安全属性表由来自 Nadim Kobeissi 的 Noise Explorer 工具导出。

密钥更新设计得益于与 Rhys Weatherley、Alexey Ermishkin 和 Olaoluwa Osuntokun 的讨论。

BLAKE2 团队（特别是 JP Aumasson、Samuel Neves 和 Zooko）提供了关于将 BLAKE2 与 Noise 结合使用的有益讨论。

Jeremy Clark、Thomas Ristenpart 和 Joe Bonneau 对早期版本提供了反馈。

18. 附录

18.1.延迟模式

下表在右列中列出了所有 23 种延迟握手模式,在左列中列出了它们相应的基本握手模式。有关基本模式和延迟模式的解释,请参见第 7 节。

NK:

<- s

...

-> e, 是 <- e, ee

NK1:

<- s

...

-> 和

<- e, ee, 是

恩智浦:

-> 和

<- e,ee,是的,是

NX1:

-> 和

<- e, ee, s

-> 是

XN:

-> 和

<- e, ee -> s, se

X1N:

-> 和

<- e, ee

-> 小号

<-se

新科：
<- s
...
-> e, 是
<- e, ee
-> s, se

X1K：
<- s
...
-> e, 是
<- e, ee
-> 小号
<-se

XK1：
<- s
...
-> 和
<- e, ee, is -> s, is

X1K1：
<- s
...
-> 和
<- e, ee, 是
-> 小号
<-se

XX：
-> 和
<- e,ee,是的,是
-> s, se

X1X：
-> 和
<- e,ee,是的,是
-> 小号
<-se

XX1：
-> 和
<- e, ee, s -> 是, s,
是

X1X1：
-> 和
<- e, ee, s -> es, s

<-se

KN:

-> 小号

...

-> 和

<- e, ee, se

KK:

-> 小号

<- s

...

-> e, es, ss <- e, ee, se

K1N:

-> 小号

...

-> 和

<- e, ee

-> 瑟

K1K:

-> 小号

<- s

...

-> e, 是 <- e,

ee

-> 瑟

KK1:

-> 小号

<- s

...

-> 和

<- e, ee, is, is

K1K1:

-> 小号

<- s

...

-> 和

<- e, ee, 是

-> 瑟

克星：

-> 小号

...

-> 和

<- e, ee, is, s, is

K1X：

-> 小号

...

-> 和

<- e,ee,是的,是

-> 瑟

KX1：

-> 小号

...

-> 和

<- e, ee, se, s

-> 是

K1X1：

-> 小号

...

-> 和

<- e, ee, s -> 是,是

在：

-> e, c <- e,

ee, ce

I1N：

-> e, s <- e,

ee

-> 瑟

我：
<- s
...
-> e, 它, s, ss
<- e, ee, se

九：
-> e, s <- e,
ee, is, s, is

I1K：
<- s
...
-> e, 是, s
<- e, ee
-> 瑟

IK1：
<- s
...
-> e, s <- e,
ee, is, is

I1K1：
<- s
...
-> e, s <- e,
ee, 是
-> 瑟

I1X：
-> e, s <- e,
ee, s, 是
-> 瑟

IX1：
-> e, c <- e,
ee, ce, c
-> 是

I1X1：
-> e, s <- e,
ee, s -> se, es

18.2.延迟模式的安全属性

下表列出了噪声握手的安全属性和
传输上一节中所有延迟模式的有效负载。这
安全属性使用第 7.7 节中的符号进行标记。

	资源	目的地
NK1		
<- s		
...		
-> 和	0	0
<- e, ee, 是	2	1
->	0	5
NX1		
-> 和	0	0
<- e, ee, s	0	1
-> 是	0	3
->	2	1
<-	0	5
X1N		
-> 和	0	0
<- e, ee	0	1
-> 小号	0	1
<-se	0	3
->	2	1
X1K		
<- s		
...		
-> e, 是 <- e,	0	2
ee	2	1
-> 小号	0	5
<-se	2	3
->	2	5
<-	2	5

XK1

<- s		
...		
-> 和	0	0
<- e, ee, 是	2	1
-> s, se	2	5
<-	2	5

X1K1

<- s		
...		
-> 和	0	0
<- e, ee, 是	2	1
-> 小号	0	5
<-se	2	3
->	2	5
<-	2	5

X1X

-> 和	0	0
<- e,ee,是的,是	2	1
-> 小号	0	5
<-se	2	3
->	2	5
<-	2	5

XX1

-> 和	0	0
<- e, ee, s -> 是, s,	0	1
是	2	3
<-	2	5
->	2	5

X1X1

-> 和	0	0
<- e, ee, s -> es, s	0	1
	0	3
<-se	2	3
->	2	5
<-	2	5

K1N

-> 小号		
...		
-> 和	0	0
<- e, ee	0	1
-> 瑟	2	1
<-	0	5

K1K

-> 小号		
<- s		
...		
-> e, is <- e,	0	2
ee, is	2	1
-> 瑟	2	5
<-	2	5

KK1

-> 小号		
<- s		
...		
-> 和	0	0
<- e, ee, is, is	2	3
->	2	5
<-	2	5

K1K1

-> 小号		
<- s		
...		
-> 和	0	0
<- e, ee, 是	2	1
-> 瑟	2	5
<-	2	5

K1X

-> 小号		
...		
-> 和	0	0
<- e ,ee ,是的 ,是	2	1
-> 瑟	2	5
<-	2	5

KX1

-> 小号		
...		
-> 和	0	0
<- e, ee, se, s	0	3
-> 是	2	3
<-	2	5
->	2	5

K1X1

-> 小号		
...		
-> 和	0	0
<- e, ee, s -> 是 ,是	0	1
	2	3
<-	2	5
->	2	5

I1N

-> e, s <- e,	0	0
ee	0	1
-> 瑟	2	1
<-	0	5

I1K

<- s		
...		
-> e, is, s <- e, ee	0	2
	2	1
-> 瑟	2	5
<-	2	5

IK1		
<- s		
...		
-> 和,s	0	0
<- e, ee, is, is	2	3
->	2	5
<-	2	5
I1K1		
<- s		
...		
-> e, s <- e,	0	0
ee, 是	2	1
-> 瑟	2	5
<-	2	5
I1X		
-> e, s <- e,	0	0
ee, s, 是	2	1
-> 瑟	2	5
<-	2	5
IX1		
-> 和,s	0	0
<- e, ee, se, s	0	3
-> 是	2	3
<-	2	5
->	2	5
I1X1		
-> e, s <- e,	0	0
ee, s -> se, es	0	1
	2	3
<-	2	5
->	2	5

18.3.模式推导规则

以下规则用于导出单向、基本和延迟握手模式。

首先,填充由模式名称定义的消息前内容。

接下来通过应用下表中匹配的第一条规则来填充发起者的第一条消息。然后删除匹配规则并重复此过程,直到无法应用更多规则。如果这是一种单向模式,那么它现在已经完成。

否则,以相同的方式填充响应者的第一条消息。一旦无法应用更多响应者规则,则切换到发起者的下一条消息并重复此过程,切换消息直到任何一方都无法应用更多规则。

发起人规则:

1. 发送“e”。
2. 如果“e”已经发送和接收,则执行“ee”。
3. 如果“s”已发送,并且“e”已接收,则执行“se”。如果发起者身份验证被延迟,则在应用它的第一条消息中跳过此规则,然后将发起者身份验证标记为非延迟。
4. 如果“e”已发送,并且“s”已接收,则执行“es”。如果响应者身份验证被延迟,则针对它应用的第一条消息跳过此规则,然后将响应者身份验证标记为非延迟。
5. 如果“s”已经发送和接收,并且“es”已经执行,则执行“ss”,并且这是第一条消息,并且不延迟发起者认证。
6. 如果这是第一条消息并且发起者是“I”或单向“X”,则发送“s”。
7. 如果这不是第一条消息并且发起者是“X”,则发送“s”。

回复规则:

1. 发送“e”。
2. 如果“e”已经发送和接收,则执行“ee”。
3. 如果“e”已发送,并且“s”已接收,则执行“se”。如果发起者身份验证被延迟,则在应用它的第一条消息中跳过此规则,然后将发起者身份验证标记为非延迟。
4. 如果“s”已发送,并且“e”已接收,则执行“es”。如果响应者身份验证被延迟,则针对它应用的第一条消息跳过此规则,然后将响应者身份验证标记为非延迟。
5. 如果响应者是“X”,则发送“s”。

18.4.更改日志

修订版 34：

·增加了官方/不稳定标记;不稳定只指新的延迟模式,本文档的其余部分被认为是稳定的。

·澄清 DH() 定义,使标识元素为无效值（不是发电机）,因此可能会被拒绝。

·阐明了 AEAD 方案的密文不可区分性要求并增加了理由。

· 阐明了散列消息前公钥的顺序。

· 为清晰起见重写了握手模式的解释。

· 添加了新的有效性规则以禁止重复相同的DH 操作。

·阐明了有关临时密钥和密钥重用的复杂有效性规则。

·从模式符号中删除了带括号的键列表,因为它是重复的丹特。

· 添加了延迟模式。

·将“Authentication”和“Confidentiality”安全属性重命名为“Source”和“Destination”以避免混淆。

· [安全]添加了一个新的身份隐藏属性,并更改了身份隐藏属性 3 以讨论身份相等检查攻击。

·将“备用模式”概念替换为 Bob 发起的模式表示法。

·为清晰起见,重写了关于复合协议和管道的部分,包括更清晰地区分“切换协议”和“回退模式”。

·不再强调“类型字节”的建议,并添加了更一般的讨论的谈判数据。

· [安全]添加了有关静态密钥重用和 PSK 重用的安全注意事项。

· 在附录中添加了模式推导规则。

19. 参考文献

[1] P. Rogaway，“Authenticated-encryption with Associated-data”，第 9 届 ACM 计算机和通信安全会议论文集，2002 年。http://web.cs.ucdavis.edu/~rogaway/papers/ad .pdf

[2] Okamoto, Tatsuaki 和 Pointcheval, David，“间隙问题：密码方案安全性的一类新问题”，第4 届公钥密码学实践与理论国际研讨会论文集：公钥密码学， 2001. https://www.di.ens.fr/~pointche/Documents/Papers/2001_pkc.pdf [3] H. Krawczyk、M. Bellare 和 R. Canetti，“HMAC：用于消息身份验证的键控哈希”。互联网工程任务组； RFC 2104（信息）； IETF，1997 年 2 月。 http://www.ietf.org/rfc/rfc2104.txt [4] H.

Krawczyk 和 P. Eronen，“基于 HMAC 的提取和扩展密钥导出函数 (HKDF)”。互联网工程任务组； RFC 5869（信息）； IETF，2010 年 5 月。 http://www.ietf.org/rfc/rfc5869.txt

[5] DJ Bernstein、M. Hamburg、A. Krasnova 和 T. Lange，“Elligator：与均匀随机字符串无法区分的椭圆曲线点。”密码学 ePrint档案，报告 2013/325，2013。http: //eprint.iacr.org/2013/325

[6] Markku-Juhani O. Saarinen，“超越模式：从加密海绵排列构建安全记录协议”。密码学 ePrint 档案，报告 2013/772，2013。http: //eprint.iacr.org/2013/772 [7] A. Langley、M. Hamburg 和 S. Turner，“用于安全的椭圆曲线”。

互联网工程任务组； RFC 7748（信息）； IETF，2016 年 1 月。 http://www.ietf.org/rfc/rfc7748.txt [8] Y. Nir 和 A. Langley，“IETF 协议的 ChaCha20 和 Poly1305”。互联网工程

任务组； RFC 7539（信息）； IETF，2015 年 5 月。网址： //www.ietf.org/rfc/rfc7539.txt

[9] MJ 德沃金，“SP 800-38D。分组密码操作模式建议：Galois/Counter Mode (GCM) 和 GMAC，”美国国家标准与技术研究院，盖瑟斯堡，马里兰州，美国，2007 年。http： //nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38d.pdf

[10] NIST，“FIPS 180-4。安全哈希标准 (SHS)，”美国国家标准与技术研究院，盖瑟斯堡，马里兰州，美国，2012 年。http： //csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf

[11] M.-J.沙里宁和 J.-P。 Aumasson，“BLAKE2 加密哈希和消息验证码 (MAC)”。互联网工程任务组； RFC 7693（信息）； IETF，2015 年 11 月。 http://www.ietf.org/rfc/rfc7693.txt [12] H. Krawczyk，“ 密码提取和密钥派生：hkdf方案 。”密码学 ePrint 档

案，报告 2010 /264，2010。http://eprint.iacr。

组织/2010/264

[13] DJ Bernstein、T. Lange 和 P. Schwabe，“NaCl:网络和密码学图书馆”。 https://nacl.cr.yp.to/

[14] DJ Bernstein，“CurveCP:互联网的可用安全性。”。 https://曲线cp.org

[15] H. Krawczyk，“SIGMA： SIGN-and-MAC Approach to Authenticated Diffie-Hellman and Its Use in the IKE Protocols”,密码学进展 - CRYPTO 2003,2003。http ://webee.technion .ac.il/~hugo/sigma.html [16] S. Halevi 和 H. Krawczyk，“一次性

HMQV 和非对称密钥包装”。密码学 ePrint 档案,报告 2010/638,2010。http ://eprint。iacr.org/2010/638 [17] I. Goldberg、D. Stebila 和 B. Ustaoglu，“密钥交换协议中的匿名性和单向认证”,设计、代码和密码学,卷。 67,没有。 2,2013 年 5 月。http ://cacr.uwaterloo.ca/

techreports/2011/cacr2011-11.pdf [18] M. Di Raimondo、R. Gennaro 和 H. Krawczyk，“Secure Off-the-record Messaging”,2005 年 ACM 电子社会隐私研讨会论文集， 2005 年。http://www.dmi .unict.it/diraimondo/web/wp-content/上传/论文/otr.pdf [19] C.

Kudla 和 KG Paterson，“密钥协议协议的模块化安全证明”,密码学进展 - ASIACRYPT 2005:第 11 届密码学和信息安全理论与应用国际会议,2005 年。http://www。 isg.rhul.ac.uk/~kp/ModularProofs.pdf [20] S. Blake-Wilson、D. Johnson 和 A. Menezes，“关键协议协议及其安全性分析”,载于 Crytography and Coding:第 6 届 IMA 国际会议 Cirencester,英

国,1997 年 12 月 17-19 日,论文集,1997 .http： //citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.387

[21] M. Hamburg，“密钥交换和 DuplexWrap 类协议”。 Noise@moderncrypto.org 邮件列表,2015 年。https: //moderncrypto.org/邮件存档/噪音/2015/000098.html [22] Mike Hamburg，“STROBE 协议框架”。密码学 ePrint档案，

报告 2017/003,2017。http: //eprint.iacr.org/2017/003

[23] T. Perrin 和 M. Marlinspike，“双棘轮算法”,2016 年。https://whispersystems.org/docs/specifications/doubleratchet/