WireGuard：下一代内核网络隧道

www.wireguard.com

杰森·A·多南菲尔德
杰森@zx2c4.com

修订草案

抽象的

WireGuard 是一个安全的网络隧道，在第 3 层运行，作为内核虚拟网络实现 Linux 的接口，旨在替换大多数用例的 IPsec，以及流行的用户空间和/或基于 TLS 的解决方案，如 OpenVPN，同时更安全、更高效且更易于使用。虚拟的隧道接口基于一个提议的安全隧道基本原则：对等公钥和隧道源 IP 地址。它使用基于 NoiseIK 的单次往返密钥交换，并使用新颖的计时器状态机机制对用户透明地处理所有会话创建。短的预先共享的静态密钥——Curve25519 点——用于 OpenSSH 风格的相互认证。这除了高度的身份隐藏之外，协议还提供了强大的完美前向保密性。运输速度是使用 ChaCha20Poly1305 认证加密实现的，用于封装数据包数据传输协议。对 IP 绑定 cookie 的改进用于减轻拒绝服务攻击，改进在 IKEv2 和 DTLS 的 cookie 机制上大大增加了加密和身份验证。整体设计允许不分配资源来响应收到的数据包，从系统的角度来看，有多种有趣的 Linux 队列和并行实现技术。最后，WireGuard 可以只需不到 4,000 行代码即可为 Linux 轻松实现，使其易于审核和验证。

本文档永久ID：4846ada1492f5d92198df154f48c3d54205657bc。静态链接：wireguard.com/papers/wireguard.PDF。日期：2020 年 6 月 1 日。这是 e2da747 修订草案。本文的一个版本出现在*Proceedings of the Network 和分布式系统安全研讨会，NDSS 2017*。版权所有 © 2015–2020 Jason A. Donenfeld。版权所有。

内容

第 3 页

1 介绍和动机

在 Linux 中，加密隧道的标准解决方案是 IPsec，它使用 Linux 转换 (“xfrm”) 层。
用户填写内核结构，确定哪个密码套件和密钥，或其他转换，如压缩、
用于遍历子系统的数据包的哪个选择器。通常用户空间守护进程负责
根据密钥交换的结果更新这些数据结构，通常使用 IKEv2 [13]，本身
一个复杂的协议，有很多选择和延展性。复杂性以及数量之多
代码，这个解决方案是可观的。管理员有一套完全独立的防火墙语义
和 IPsec 数据包的安全标签。在将密钥交换层与传输加密分离的同时——
或转换——层是从语义角度明智的分离，类似地，在分离
从网络的角度来看，接口层的转换层是正确的，这种严格正确的分层
方法增加了复杂性并使正确的实施和部署变得令人望而却步。

WireGuard 消除了这些分层分离。而不是 IPsec 和 xfrm 层的复杂性，
WireGuard 只是提供了一个虚拟接口——例如 wg0——然后可以使用标准进行管理
ip(8) 和 ifconfig(8) 实用程序。使用私钥（以及可选的预共享密钥）配置接口后
5.2节中解释的对称密钥) 以及将与之通信的对等方的各种公钥
安全，隧道简单地工作。密钥交换、连接、断开、重新连接、发现等
透明可靠地发生在幕后，管理员无需担心
这些细节。换句话说，从管理的角度来看，WireGuard 接口似乎是
无国籍。然后可以使用防火墙接口的普通基础设施配置防火墙规则，使用
保证来自 WireGuard 接口的数据包将经过身份验证和加密。简单而
简单地说，WireGuard 比 IPsec 更不容易发生灾难性故障和错误配置。这是
然而，重要的是要强调*IPsec 的分层是正确和合理的*；一切都在正确的地方
使用 IPsec，达到学术完美。但是，正如抽象的正确性经常发生的那样，有一个深刻的
缺乏可用性，并且很难实现可验证的安全实施。相反，WireGuard 启动
从有缺陷的分层违规的基础，然后尝试纠正这种合并引起的问题
使用实用的工程解决方案和密码技术来解决现实世界的问题。

另一方面是 OpenVPN，这是一种使用 TLS 的基于用户空间 TUN/TAP 的解决方案。经过

由于它位于用户空间，因此性能非常差——因为必须多次复制数据包
在内核空间和用户空间之间——并且需要一个长期存在的守护进程；OpenVPN 似乎远非无状态
给管理员。虽然 TUN/TAP 接口（例如 tun0）具有上述类似 wg0 的好处，
OpenVPN 也非常复杂，支持大量的 TLS 功能，这暴露了很多
潜在漏洞的代码位。OpenVPN 适合在用户空间实现，因为 ASN.1 和 x509
内核中的解析器历来存在很大问题（CVE-2008-1673、CVE-2016-2053），并且添加了一个
TLS 堆栈只会让这个问题变得更糟。TLS 还带来了一个巨大的状态机，以及一个
源 IP 地址和公钥之间的关联不太清楚。

对于密钥分发，WireGuard 从 OpenSSH 中汲取灵感，其常见用途包括非常
密钥管理的简单方法。通过一组不同的带外机制，两个对等点通常
交换他们的静态公钥。有时它就像 PGP 签名的电子邮件一样简单，有时它是一个复杂的
使用 LDAP 和证书颁发机构的密钥分发机制。重要的是，大多数情况下 OpenSSH
密钥分发完全不可知。WireGuard 紧随其后。两个 WireGuard 对等方交换他们的公钥
通过一些未指定的机制，然后他们就能够进行通信。换句话说，WireGuard 的
对密钥分发的态度是，这是解决该特定问题的错误层，因此
界面非常简单，可以使用任何密钥分发解决方案。作为额外的优势，
公钥只有 32 个字节长，可以很容易地用 Base64 编码表示为 44 个字符，即
可用于通过各种不同的媒介传输密钥。

最后，WireGuard 在密码学上是固执己见的。它故意缺乏密码和协议敏捷性。如果
在底层基元中发现了漏洞，所有端点都需要更新。如继续所示
SSL/TLS 漏洞的洪流，密码敏捷性极大地增加了复杂性。WireGuard 使用一个变体
特雷弗·佩兰的噪音 [23]——在它的开发过程中收到了这个作者的相当多的投入
用于 WireGuard 的论文——用于 1-RTT 密钥交换，使用 Curve25519 [5] 对于 ECDH，
香港发展基金会 [15] 用于扩展 ECDH 结果，RFC7539 [17] ChaCha20的构建[3] 和 Poly1305 [8] 为了
经过身份验证的加密和 BLAKE2s [2] 用于散列。它具有针对拒绝服务的内置保护
攻击，使用新的加密 cookie 机制进行 IP 地址归属。

同样固执己见，WireGuard 仅适用于第 3 层；如下文第2节所述，这是最干净的方法
以确保数据包的真实性和可归因性。作者认为第 3 层是正确的方法
用于桥接多个 IP 网络，并将其强加到 WireGuard 上可以进行许多简化，

第 4 页

产生一个更清晰、更容易实现的协议。它支持 IPv4 和 IPv6 的第 3 层，并且
可以封装 v4-in-v6 和 v6-in-v4。

WireGuard 将这些原则放在一起，专注于简单性和可审计的代码库，同时仍然
速度极快，适用于少量环境。通过结合密钥交换和
第 3 层传输加密为一种机制，并使用虚拟接口而不是转换层，
WireGuard 确实打破了传统的分层原则，追求的是一种兼具两者的可靠*工程*解决方案。
更实用，更安全。在此过程中，它采用了几种新颖的密码和系统解决方案来
实现其目标。

2 加密密钥路由

安全 VPN 的基本原则是在对等点之间建立关联，并且每个 IP 地址都被允许
用作源 IP。在 WireGuard 中，对等点由其公钥（一个 32 字节的 Curve25519 点）严格标识。
这意味着公钥和一组允许的 IP 地址之间存在简单的关联映射。
检查以下*加密密钥路由表*：

配置 1a

接口公钥	**接口私钥**	**监听UDP端口**
HIgo...8ykw	yAnz...fBmk	41414
对等公钥	**允许的源 IP**	
xTIB...p8Dg	10.192.122.3/32、10.192.124.0/24	
TrMv...WXX0	10.192.122.4/32、192.168.0.0/16	
gN65...z6EA	10.10.10.230/32	

接口本身有一个私钥和一个它侦听的 UDP 端口（稍后会详细介绍），然后是一个
同行名单。每个对等点都由其公钥标识。然后每个都有一个允许的源 IP 列表。

当传出数据包在 WireGuard 接口 wg0 上传输时，参考此表以
确定用于加密的公钥。例如，目标 IP 为 10.192.122.4 的数据包
将使用从公钥TrMv...WXX0派生的安全会话进行加密。相反，当 wg0
接收到一个加密的数据包，经过解密和认证后，只有在其源 IP 解析后才会接受它
表中用于解密它的安全会话中使用的公钥。例如，如果一个数据包被解密
来自xTIB...qp8D，仅当解密数据包的源 IP 为 10.192.122.3 或在范围内时才允许
10.192.124.0 到 10.192.124.255；否则它被丢弃。

有了这个非常简单的原则，管理员可以依赖简单的防火墙规则。例如，一个传入
源 IP 为 10.10.10.230 的接口 wg0 上的数据包可能被认为是来自对等方的真实数据包
的公钥gN65 ... Bz6E。更一般地说，任何到达 WireGuard 接口的数据包都将具有可靠的
真实的源 IP（当然，还要保证传输的完美前向保密）。请注意
这是唯一可能的，因为 WireGuard 严格基于第 3 层。与一些常见的 VPN 协议不同，例如
L2TP/IPsec，在第 3 层使用经过身份验证的对等体标识，强制实施更简洁的网络设计。

对于希望通过另一个 WireGuard 对等点路由*所有*流量的 WireGuard 对等点，
加密密钥路由表可以更简单地配置为：

配置2a

接口公钥	**接口私钥**	**监听UDP端口**
gN65...z6EA	gI6E...fWGE	21841
对等公钥	**允许的源 IP**	
HIgo...8ykw	0.0.0.0/0	

在这里，对等方授权HIgo...f8yk将具有任何源 IP 的数据包以及所有属于
wg0 上的传出将使用与该公钥关联的安全会话进行加密并发送到该公钥

对等方的端点。

2.1 端点和漫游

当然，重要的是，对等方能够在
特定的 Internet 端点。加密密钥路由表中的每个对等体可以*选择*预先指定一个已知的
该对等端点的外部 IP 地址和 UDP 端口。它是可选的原因是如果没有指定

第 5 页

并且 WireGuard 从对等方接收经过正确身份验证的数据包，它将使用外部外部源 IP
用于确定端点的地址。

由于公钥唯一标识对等方，因此加密的 WireGuard 数据包的外部外部源 IP
用于标识对等体的远程端点，使对等体可以在不同的外部IP之间自由漫游，
例如，在移动网络之间，类似于 Mosh [25]。例如，先前的加密密钥
路由表可以扩充为具有对等方的初始端点：

配置 2b

接口公钥	**接口私钥**	**监听UDP端口**
gN65...z6EA	gI6E...fWGE	21841
对等公钥	**允许的源 IP**	**互联网端点**
HIgo...8ykw	0.0.0.0/0	192.95.5.69:41414

然后，这台主机，gN65 ... z6EA，发送加密数据包肥..... f8yk在192.95.5.69:41414。在HIgo...f8yk 之后
收到一个数据包，它更新它的表以了解发送回复数据包的端点是，例如，
192.95.5.64:21841：

配置 1b

接口公钥	**接口私钥**	**监听UDP端口**
HIgo...8ykw	yAnz...fBmk	41414
对等公钥	**允许的源 IP**	**互联网端点**
xTIB...p8Dg	10.192.122.3/32、10.192.124.0/24	
TrMv...WXX0	10.192.122.4/32、192.168.0.0/16	
gN65...z6EA	10.10.10.230/32	192.95.5.64:21841

注意peer的监听端口和发送的数据包的源端口总是一样的，增加很多
简单，同时还确保在 NAT 后面的可靠遍历。并且由于此漫游属性可确保对等方
将拥有最新的外部源 IP 和 UDP 端口，不需要 NAT 来保持会话
长时间打开。（对于必须无限期地保持打开 NAT 会话或状态防火墙的用例，
可以选择将接口配置为定期发送经过身份验证的持久连接。）

这种设计提供了极大的便利和最少的配置。当攻击者具有主动
当然，中间人可以修改这些未经身份验证的外部源 IP，攻击者不会
能够解密或修改任何有效载荷，这仅仅相当于拒绝服务攻击，这已经
只需从这个假定的中间人位置丢弃原始数据包，就可以轻松实现。和，
如第6.5节所述，无法解密并随后回复数据包的主机将很快被遗忘。

3 发送/接收流程

2.1节的漫游设计, 与第2节的加密密钥路由表放在一起，相当于
使用上面的“配置 1”在接口 wg0 上接收和发送数据包时遵循以下流程。

数据包在本地生成（或转发）并准备在传出接口 wg0 上传输：

1. 明文数据包到达 WireGuard 接口 wg0。
2. 检查数据包的目标 IP 地址 192.168.87.21，该地址与对等方TrMv...WXX0匹配。
   （如果它不匹配任何对等点，则将其丢弃，并通过标准 ICMP“无路由到主机”通知发送方
   数据包，以及将 -ENOKEY 返回到用户空间。）
3. 与peer关联的安全会话的对称发送加密密钥和nonce计数器
   TrMv...WXX0用于使用 ChaCha20Poly1305 加密明文数据包。
4. 包含各种字段的标题，在5.4节中解释, 被添加到现在加密的数据包中。
5. 这个报头和加密包一起作为 UDP 包发送到 Internet UDP/IP 端点
   与对等方TrMv...WXX0相关联，导致外部 UDP/IP 数据包包含一个标头作为其有效负载
   和加密的内部数据包。对等端的端点要么是预先配置的，要么是从外部获知的
   最近收到的经过正确身份验证的数据包的外部源 IP 标头字段。（否则，如果没有
   可以确定端点，丢弃数据包，发送 ICMP 消息，并且 -EHOSTUNREACH 是
   返回到用户空间。）

一个UDP/IP包到达主机的UDP端口41414，即wg0接口的监听UDP端口：

第 6 页

1. 包含特定报头和加密有效载荷的 UDP/IP 数据包在正确的端口上接收
   （在这种特殊情况下，端口 41414）。
2. 使用标题（在下面的第5.4节中描述）, WireGuard 确定它与 peer 关联
   TrMv...WXX0的安全会话，检查消息计数器的有效性，并尝试进行身份验证

并使用安全会话的接收对称密钥对其进行解密。如果它无法确定对等体或者
身份验证失败，数据包被丢弃。

3. 由于数据包已经正确认证，使用外层UDP/IP数据包的源IP进行更新
   peer TrMv...WXX0的端点。
4. 一旦数据包有效载荷被解密，接口就有一个明文数据包。如果这不是 IP 数据包，
   它被丢弃。否则，WireGuard 会检查明文内部数据包的源 IP 地址
   在加密密钥路由表中相应地路由。例如，如果解密的源IP
   明文数据包为192.168.31.28，数据包对应路由。但是如果源IP是10.192.122.3，
   数据包没有相应地为*此*对等体路由，并被丢弃。
5、如果明文包没有被丢弃，则插入到wg0接口的接收队列中。

可以将允许的 IP 列表分成两个列表——一个用于检查源地址
传入数据包和一个用于根据目标地址选择对等体的数据包。但是，通过保留这些作为一部分
同一个列表，它允许类似于反向路径过滤的东西。发送数据包时，列表为
根据目的IP查询；当接收到一个数据包时，会参考相同的列表来确定
源 IP 是允许的。然而，而不是询问接收到的数据包的发送对等方是否具有该源 IP
作为其允许的 IP 列表的一部分，它可以提出一个更全局的问题——哪个对等点将被选择
该源 IP 的表，并且该对等体是否与接收到的数据包的对等体相匹配。这强制执行一对一
发送和接收 IP 地址的映射，以便如果从特定对等方接收到数据包，则回复
该 IP 将保证转到同一个对等方。

4 基本用法

在深入研究密码学和实现细节之前，看一个简单的命令可能会很有用
使用 WireGuard 的线路接口，使迄今为止提出的概念更加具体。
考虑一个具有单个物理网络接口 eth0 的 Linux 环境，将其连接到 Internet
公共 IP 为 192.95.5.69。可以添加和配置 WireGuard 接口 wg0 以具有隧道
带有标准 ip(8) 实用程序的 /24 子网中的 IP 地址 10.192.122.3，如左侧所示。加密密钥
然后可以使用 wg(8) 工具以各种方式配置路由表，包括从
配置文件，如右图所示：

添加 wg0 接口

```
$ ip link add dev wg0 type wireguard

$ ip 地址添加 dev wg0 10.192.122.3/24
$ ip route add 10.0.0.0/8 dev wg0

$ ip地址显示
1: lo: <LOOPBACK> mtu 65536
    inet 127.0.0.1/8 范围主机 lo
2: eth0: <广播> mtu 1500
    inet 192.95.5.69/24 范围全局 eth0
3: wg0: <POINTOPOINT,NOARP> mtu 1420
    inet 10.192.122.3/24 范围全局 wg0
```

配置wg0的cryptokey路由表

```
$ wg setconf wg0 configuration-1.conf

$ wg 显示 wg0
接口：wg0
  公钥：HIgo...8ykw
  私钥：yAnz...fBmk
  监听端口：41414
同行：xTIB...p8Dg
  允许的 ips: 10.192.124.0/24, 10.192.122.3/32
同行：TrMv...WXX0
  允许的 ips: 192.168.0.0/16, 10.192.122.4/32
同行：gN65...z6EA
  允许的ips：10.10.10.230/32
  端点：192.95.5.70:54421
$ ip link set wg0 up

$ 平 10.10.10.230
PING 10.10.10.230 56(84) 字节数据。
64 字节：icmp_seq=1 ttl=49 时间=0.01 ms
```

此时，在该系统上向 10.10.10.230 发送数据包将通过 wg0 接口发送数据，
它将使用与公钥gN65...z6EA关联的安全会话加密数据包并发送
通过 UDP 加密和封装数据包到 192.95.5.70:54421。收到来自 10.10.10.230 的数据包时
在 wg0 上，管理员可以确信它来自gN65...z6EA。

第 7 页

5 协议和密码学

如前所述，为了开始发送加密的封装数据包，1-RTT 密钥交换握手
必须首先发生。发起者向响应者发送消息，响应者向响应者发送消息
发起人。握手后，发起者可以使用共享的对称对发送加密消息
密钥，一个用于发送，一个用于接收，发送给响应者，并遵循来自的第一个加密消息
发起者到响应者，响应者可以开始向发起者发送加密消息。这个订购
限制是需要按照 KEA+C 所述进行确认 [18]，以及允许握手消息
异步处理以传输数据消息。这些消息使用来自噪声的“IK”模式 [23]， 在
除了新颖的 cookie 构造以减轻拒绝服务攻击。该协议的最终结果是
非常健壮的安全系统，它实现了认证密钥交换 (AKE) 安全性的要求 [18],
避免密钥妥协模仿，避免重放攻击，提供完美的前向保密，提供身份
隐藏类似于 SIGMA 的静态公钥 [16]，并具有抵抗拒绝服务攻击的能力。

5.1 沉默是一种美德

WireGuard 的一个设计目标是避免在身份验证之前存储任何状态并且不发送任何响应
到未经身份验证的数据包。没有为未经身份验证的数据包存储状态，也没有生成响应，
WireGuard 对非法同行和网络扫描仪是不可见的。避免了几类攻击
不允许未经身份验证的数据包影响任何状态。更一般地说，可以实现
WireGuard 以一种根本不需要动态内存分配的方式，即使对于经过身份验证的数据包，如
在第7节中解释. 然而，这个属性需要响应者收到的第一条消息
验证发起者。在像这样的第一个数据包中进行身份验证可能会打开响应者
到重放攻击。攻击者可以重播初始握手消息以欺骗响应者重新生成
它的临时密钥，从而使合法发起者的会话无效（尽管不影响保密性）

或任何消息的真实性）。为了防止这种情况，一个 12 字节的 TAI64N [7] 时间戳被包括在内，加密和
已验证，在第一条消息中。响应者跟踪每个对等方收到的最大时间戳，并
丢弃包含小于或等于它的时间戳的数据包。（事实上，它甚至不必是一个准确的
时间戳；它必须是一个对等方单调递增的 96 位数字。）如果响应者重新启动
并丢失此状态，这不是问题：即使可以重放早期的初始数据包，它也可能
不可能中断任何正在进行的安全会话，因为响应者刚刚重新启动，因此没有
要中断的活动安全会话。一旦发起方重新建立与响应方的安全会话
重新启动，发起者将使用更大的时间戳，使前一个无效。这个时间戳确保
攻击者可能不会通过重放攻击破坏发起者和响应者之间的当前会话。（这也是
指的是两个不同的同龄人应该*不*共享私有密钥，因为在这种情况下一个数据包发送到一个可
重播到另一个，然后随后的响应将导致发起者不由自主地从一个漫游
对等另一个。但无论如何，一开始就不应该共享私钥。）从一个实现
来看，TAI64N [7] 非常方便，因为它是 big-endian，允许在两个 12 字节之间进行比较
使用标准 memcmp() 完成的时间戳。由于 WireGuard 不使用签名，为了获得
一定程度的可否认性，第一条消息仅依赖于两个对等方静态密钥的 Diffie-Hellman 结果
验证。这意味着如果他们的任何一个静态密钥被泄露，攻击者将能够
伪造一个初始消息——尽管它无法完成完整的握手——包含最大
时间戳值，从而防止所有未来的连接成功。虽然这可能看起来类似于
传统的密钥妥协模拟漏洞——WireGuard*不易*受到攻击——它实际上是
非常不一样。因为，如果密钥泄露使攻击者能够阻止对等方使用他们被泄露的
再一次，攻击者实际上帮助了对这种妥协的正确响应。如果 TIA64N 的精度
时间戳会造成不合适的信息泄漏，实现可能会截断纳秒的 24 位
时间戳的一部分。

5.2 可选的预共享对称密钥模式

WireGuard 依赖于对等点之间*先验地*交换静态公钥，作为它们的静态身份。这
所有发送数据的保密性依赖于 Curve25519 ECDH 功能的安全性。为了减轻任何未来
随着量子计算的进步，WireGuard 还支持一种模式，在这种模式下，任何对等点都可以*额外地*
在他们之间预先共享一个 256 位对称加密密钥，以便添加一个额外的层
对称加密。这里的攻击模型是攻击者可能会长时间记录加密流量
以期限为基础，希望有一天能够打破 Curve25519 并解密过去的流量。预分享时

第 8 页

从密钥管理的角度来看，对称加密密钥通常很麻烦，而且可能更有可能
被盗的想法是，当量子计算发展到打破 Curve25519 时，这种预先共享的对称
钥匙早就忘记了。而且，更重要的是，在短期内，如果预共享对称密钥是
受到损害，Curve25519 密钥仍然提供足够的保护。代替使用完全
后量子密码系统，在撰写本文时在这里不实用，这种可选的混合方法
用于补充椭圆曲线密码术的预共享对称密钥提供了可靠且可接受的
对极度偏执者的权衡。此外，它允许在 WireGuard 复杂的基础上构建
密钥轮换方案，以实现不同类型的妥协后安全性。

5.3 拒绝服务缓解和 Cookie

计算 Curve25519 点乘法是 CPU 密集型的，即使 Curve25519 是一个极快的曲线
大多数处理器。为了确定握手消息的真实性，Curve25519 乘法
必须计算，这意味着存在拒绝服务攻击的潜在途径。为了抵挡
CPU耗尽攻击，如果响应者——消息的接收者——负载不足，它可能会选择不
处理握手消息（启动或响应握手消息），而是响应
包含 cookie 的 cookie 回复消息。发起者然后使用此 cookie 以重新发送消息
并让响应者在接下来的时间接受它。

响应者维护一个每两分钟改变一次的秘密随机值。cookie 只是
使用这个不断变化的秘密作为 MAC 密钥计算发起者源 IP 地址的 MAC 的结果。这
发起方在重新发送其消息时，使用此 cookie 作为 MAC 密钥发送其消息的 MAC。什么时候
响应者收到消息，如果负载不足，可以选择是否接受和处理
消息基于是否存在使用 cookie 作为密钥的正确 MAC。这种机制关系
从发起者发送到其 IP 地址的消息，提供 IP 所有权证明，允许使用
经典的 IP 速率限制算法（令牌桶等 -有关实现细节，请参阅第7.4节）。

这或多或少是 DTLS 使用的方案 [24] 和 IKEv2 [13]。然而，它存在三个主要缺陷。
首先，如第5.1节所述，我们更愿意保持沉默，不对未经身份验证的消息发送任何回复；
在负载不足时不加选择地发送 cookie 回复消息会破坏此属性。二、曲奇
不应以明文形式发送，因为中间人可以使用它来发送欺诈性消息
被处理的。第三，发起者自己可能会因为被发送欺诈而遭到拒绝服务攻击
cookie，然后它会使用它来计算其消息的 MAC，但没有成功。饼干机制
WireGuard 使用两个 MAC（msg.mac1 和 msg.mac2）修复了这些问题，计算
这将在下面的第5.4.4 节中显示。

对于第一个问题，为了让响应者保持沉默，即使在负载下，所有消息都有一个
使用响应者公钥的第一个 MAC (msg.mac1)。这意味着至少，一个对等方发送一个
消息必须知道它在与谁交谈（通过知道它的公钥），以便引出任何类型的
回复。在负载或未负载下，这个第一个 MAC (msg.mac1) 总是需要存在和有效。
虽然响应者本身的公钥不是秘密的，但在这个攻击模型中它是足够的秘密，在
其目标是确保服务的隐蔽性，因此知道响应者的公钥就足以证明
因为已经知道它的存在。（值得注意的是，第一个 MAC 允许被动攻击者
猜测数据包的目的是哪个公钥，稍微削弱了身份隐藏属性，尽管
正确的猜测不构成密码*证明*，因为没有使用私人材料来生成
苹果电脑。）

同样，为了解决第二个问题——以明文形式发送 MAC——我们应用了一个 AEAD
将随机随机数扩展到传输中的 cookie，再次使用响应者的对称加密密钥
公钥。同样，这里的大多数公共值足以满足我们在拒绝服务攻击中的目的

威胁模型，最后，为了解决第三个问题，我们使用AEAD的“附加数据”字段对cookie中的cookie进行加密
传输以额外验证引发 cookie 的发起消息的第一个 MAC (msg.mac1)
回复消息。这确保了没有中间人位置的攻击者无法发送种子
对发起方的无效 cookie 回复以防止他们使用正确的 cookie 进行身份验证。（攻击者*使用*
一个中间人位置可以简单地丢弃 cookie 回复消息以阻止连接，以便
案例无关紧要，*尽管*仅具有*被动*中间人位置的攻击者确实可以伪造
这些数据包，这与针对 TCP 的拒绝服务攻击没有太大区别。）换句话说，
我们使用 AD 字段将 cookie 回复绑定到启动消息。

解决这些问题后，我们可以使用安全的方式添加上述第二个 MAC（msg.mac2）
传输的 cookie 作为 MAC 密钥。当响应者负载不足时，它只会接受以下消息

第 9 页

另外还有这第二个 MAC。

总之，响应者在计算这些 MAC 并将它们与接收到的 MAC 进行比较之后
消息，*必须*始终拒绝带有无效 msg.mac1 的消息，并且在负载不足时*可能会*拒绝带有无效消息的消息
无效的 msg.mac2。如果响应者收到一个带有有效 msg.mac1 但带有无效 msg.mac2 的消息，
并且在负载下，它可能会以 cookie 回复消息进行响应，详见第5.4.7节. 这相当
改进了 DTLS 和 IKEv2 使用的 cookie 方案。

与 HIPv2 相比 [20]，它通过使用 2-RTT 密钥交换和复杂性谜题解决了这个问题，
WireGuard 避开解谜结构，因为前者需要存储状态，而后者需要
发起者和响应者之间的关系不对称。在 WireGuard 中，任一对等点都可能
有动力开始握手。这意味着要求复杂性谜题从
发起者，因为发起者和响应者可能很快就会改变角色，将这种缓解机制变成
拒绝服务漏洞本身。相比之下，我们上面的 cookie 解决方案支持拒绝服务攻击
1-RTT 协议上的缓解措施，同时保持发起者和响应者的角色对称。

5.4 消息

有四种类型的消息，每种类型都以单字节消息类型标识符为前缀，记为 msg.type
以下：

- 第5.4.2节：开始握手过程以建立安全的握手启动消息
  会议。
- 第5.4.3节：结束握手的发起消息的握手响应，之后
  可以建立安全会话。
- 第5.4.7节：对握手发起消息或握手响应消息的回复，解释
  在第5.3节，它传达了一个加密的 cookie 值，用于重新发送被拒绝的
  握手发起消息或握手响应消息。
- 第5.4.6节: 一个封装和加密的 IP 数据包，它使用由协议协商的安全会话。
  握手。

握手的发起者记为下标i，握手的响应者记为
下标r，其中之一表示为下标 *。对于可由任一发起方创建的消息
或响应者，如果创建消息的对等体是发起者，则让 $(m, m)=(i, r)$，如果创建消息的对等体
message 是响应者，让 $(m, m)=(r, i)$。这两个对等点有几个他们在本地维护的变量：

$我^*$	一个 32 位索引，在本地代表另一个对等点，类似于 IPsec 的“SPI”。
$私_{我}^*, S_{酒馆}^*$	静态私钥和公钥值。
$私_{乙}^*, E_{酒馆}^*$	临时私钥和公钥值。
$问$	第5.2节中的可选预共享对称密钥值当预共享密钥模式为 未使用，设置为 0 32。
H^*, C^*	散列结果值和链接键值。
$T_{发送}^*$，$T_{接收}^*$	用于发送和接收的传输数据对称密钥值。
$n_{发送}^*, N_{接收}^*$	用于发送和接收的传输数据消息随机数计数器。

在后面的结构中，使用了几个符号、函数和运算符。二元运算符
表示其操作数的串联，二元运算符 := 表示其右操作数的赋值
到它的左操作数。注释$\bar{N}$返回值（$\bar{N}$ + 16），这是Poly1305认证标签长度
添加到n。ε表示空零长度比特串，$0\ \bar{N}$表示全零（为0x0）长度的位串$\bar{N}$
字节，$\rho\ n$表示长度为n字节的随机位串。让τ被认为是一个临时变量，并让
κ被认为是一个临时加密密钥。除非另有说明，否则所有整数分配都是小端序的。
使用了以下函数和常量：

DH(private key, public key) Curve25519 私钥和公钥的点乘，重新
转动 32 个字节的输出。

DH-Generate() 生成一个随机的Curve25519私钥并推导出其对应的公钥，
返回一对 32 字节的值，（私有的，公共的）。

Aead(key, counter, plain text, auth text) ChaCha20Poly1305 AEAD，如RFC7539 [17],
其随机数由 32 位零组成，后跟计数器的 64 位小端值。

Xaead(key, nonce, plain text, auth text) XChaCha20Poly1305 AEAD，24字节随机
nonce，使用 HChaCha20 实例化 [6] 和 ChaCha20Poly1305。

第 10 页

Hash(input) Blake2s(input, 32)，返回32字节的输出。

Mac(key, input) Keyed-Blake2s(key, input, 16)，BLAKE2s 哈希的键控 MAC 变体函数，返回 16 个字节的输出。

Hmac(key, input) Hmac-Blake2s(key, input, 32)，普通BLAKE2s哈希函数，用于 HMAC 构造，返回 32 字节的输出。

Kdf *n* (key, input) 设置$\tau 0$:= Hmac(key , input) ,$\tau 1$:= Hmac($\tau 0$, 0x1) ,τi := Hmac($\tau 0$,$\tau i-1$ i)，然后返回一个包含 32 个字节值的*n*元组，($\tau 1$,...,τn)。这是香港发展基金会 [15] 功能。

Timestamp() 返回 TAI64N 时间戳 [7] 当前时间，也就是 12 个字节的输出，前 8 个字节是自 1970 年 TAI 以来的秒数的大端整数，最后 4 个字节是从那一秒开始的纳秒数的大端整数。

构造 UTF-8 字符串文字“Noise_IKpsk2_25519_ChaChaPoly_BLAKE2s”，37 字节输出。

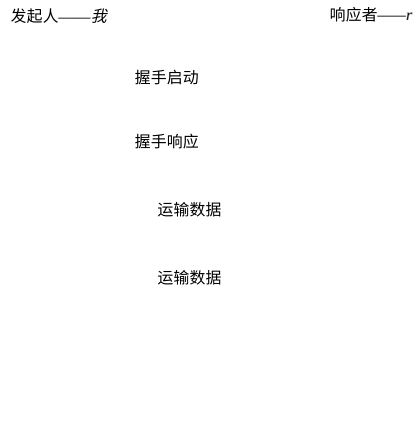
标识符 UTF-8 字符串文字“WireGuard v1 zx2c4 Jason@zx2c4.com”，34 个字节的输出。

Label-Mac1 UTF-8 字符串文字“mac1----”，8 个字节的输出。

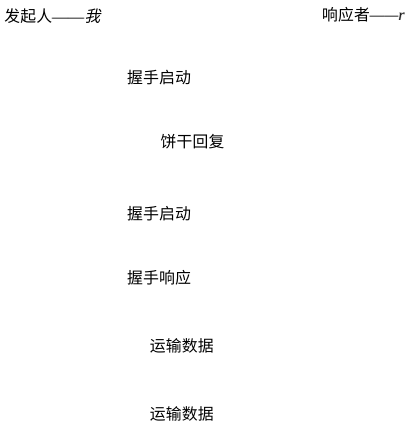
Label-Cookie UTF-8 字符串文字“cookie--”，8 个字节的输出。

5.4.1 协议概述

在大多数情况下，握手将完成在 1-RTT 中，之后传输数据如下：

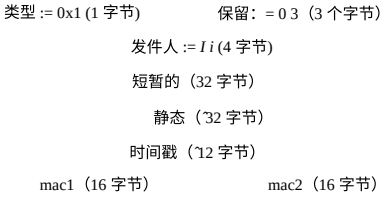


如果一个对等点负载不足，则发送 cookie 回复消息被添加到握手中，以防止拒绝 - 服务攻击：



5.4.2 第一条消息：发起者到响应者

发起者发送此消息，msg：



时间戳字段在第5.1节中解释，以及 mac1 和 mac2 在第5.4.4 节中进一步解释。 *我我*是发送此消息时随机生成 (ρ 4)，并用于将后续回复绑定到由

第 11 页

这条信息。以上剩余字段计算[23] 如下：

C i := 哈希（构造）
H i := 哈希(*C i* 标识符)
H i := 哈希(*H i* *小号酒吧*)
(*E* *私人* *一世* *E* *酒馆* *一世* := DH-生成()
C i := Kdf 1 (*C i* ,*E* *酒馆* *一世*
msg.ephemeral := *E* *酒馆* *一世*
H i := 哈希(*H i* msg.ephemeral)
(*C i* ,κ) := Kdf 2 (*C i* , DH(*E* *私人* *一世* *S酒吧*))
msg.static := Aead(κ, 0 ,*S* *酒馆* *一世* *嗨*)
H i := 哈希(*H i* msg.static)
(*C i* ,κ) := Kdf 2 (*C i* , DH(*S* *私人* *一世* *S酒吧*))
msg.timestamp := Aead(κ, 0 , Timestamp() ,*H i*)
H i := 哈希(*H i* msg.timestamp)

当响应者收到此消息时，它会执行相同的操作，以便其最终状态变量为相同，替换 DH 函数的操作数以产生等效值。（旁注：虽然不是

msg.static 的 Aead(κ,0,Hash(S 酒馆 一世),Hash(H)) 噪音。因此目前不是 WireGuard 的一部分，对上述消息的一个扩展)。额外的哈希确保这条椭圆曲线
点不会直接传输，因此整个握手会有一定程度的限制
非前向秘密后量子安全，前提是公共密钥不是通过其他方式公开的。）

5.4.3 第二条消息：响应者到发起者

响应者在处理来自发起者的第一条消息并应用
相同的操作以达到相同的状态。I_r是在发送此消息时随机生成的 (ρ 4)，并且是
用于将后续回复绑定到由此消息开始的会话，就像上面一样。响应者发送此
留言，留言：

类型 := 0x2 (1 字节)　　保留：= 0 3（3 个字节）

发件人 := I_r (4 字节)　　接收器 := I_i (4 字节)

短暂的（32 字节）

空 ̂0 字节）

mac1（16 字节）　　mac2（16 字节）

字段 mac1 和 mac2 在第5.4.4 节中进一步解释. 以上剩余字段计算[23]
如下：

（E 私法电子酒吧 DH-生成()

C_r := Kdf 1 (C_r ,E^{pub}_r)

msg.ephemeral := E^{pub}_r

H_r := 哈希(H_r msg.ephemeral)

C_r := Kdf 1 (C_r , DH(E^{priv}_r ,E 酒馆 一世))

C_r := Kdf 1 (C_r , DH(E^{priv}_r ,S 酒馆 一世))

(C_r ,τ,κ) := Kdf 3 (C_r ,Q)

H_r := 哈希(H_r τ)

msg.empty := Aead(κ, 0 , ϵ, H_r)

H_r := 哈希(H_r msg.空)

第 12 页

当发起者收到这个消息时，它会做同样的操作，使其最终状态变量相同，
替换 DH 函数的操作数以产生等效值。注意这个握手响应
消息小于握手发起消息，防止放大攻击。

5.4.4 Cookie MAC

在第5.4.2和5.4.3节中，两个握手消息都有msg.mac1和msg.mac2参数。为一个
给定握手消息，msg α表示 msg.mac1 之前 msg 的所有字节，而 msg β表示
msg.mac2 之前的 msg。收到的最新 cookie ～ L ＊秒前由L ＊表示。msg.mac1 和
msg.mac2 字段填充如下：

msg.mac1 := Mac(Hash(Label-Mac1 S 酒馆 m) ,味精α)

如果$L_m = \epsilon$或 ̃ 大号米≥120：

msg.mac2 := 0 16

除此以外：

msg.mac2 := Mac(L_m , msg β)

值 Hash(Label-Mac1 S 酒馆 m) 以上可以预先计算。

5.4.5 传输数据密钥推导

上述两条消息交换完毕后，计算密钥[23] 由发起者和响应者为
发送和接收传输数据消息（第5.4.6节）：

（T 发送 一世 = T 接收 r ,T 接收 一世 = T 发送 r) := Kdf 2 ($C_i = C_r$,ϵ)

ñ 发送 一世 = N 接收 r = N 接收 一世 = N 发送 r := 0

乙私人 一世 = E 酒馆 一世 = E 私有 r = 电子酒吧 r $C_i = C_r := \epsilon$

在最后一行，握手的大多数先前状态从内存中归零（在第7.4节中描述)），但是
值$H_i = H_r$不一定为零，因为它可能在噪声的未来修订中有用[23]。

5.4.6 后续消息：传输数据消息

发起者和响应者交换传输数据消息以交换加密的封装数据包。
封装的内部明文数据包表示为P，长度为P。两个对等点都发送此消息，
留言：

类型 := 0x4 (1 字节)　　保留：= 0 3（3 个字节）

接收器：= I_m（4 个字节）

计数器（8 个字节）

包 (̂ P字节）

其余字段填充如下：

$$P := P \,0\,16\cdot\lceil P/16\rceil - P$$

$$\text{msg.counter} := N_{发送}$$

$$\text{msg.packet} := \text{Aead}(T_{发送}, N_{发送}, P, \epsilon)$$

$$N_{发送} := N_{发送} + 1$$

此消息的接收者使用T_{recv}阅读邮件。请注意，这里没有存储长度值
标头，因为身份验证标签用于确定消息是否合法，而内部 IP
数据包的报头中已经有一个长度字段。封装的数据包本身是零填充的（不修改
IP 数据包的长度字段）在加密之前使流量分析复杂化，尽管零填充应该
切勿将 UDP 数据包大小增加到超过最大传输单元长度。在 msg.packet 之前，有

第 13 页

正好是 16 个字节的头字段，这意味着解密可以就地完成并且仍然可以实现
自然的内存地址对齐，允许在硬件中更容易实现和显着的性能
改进了许多常见的 CPU 架构。这部分是保留零的 3 个字节的结果
字段，使前四个字节作为小端整数一起可读。

msg.counter 值是ChaCha20Poly1305 AEAD的*随机数*，并由接收者跟踪
使用$N_{接收}$。它还可以避免重放攻击。由于 WireGuard 在 UDP 上运行，因此消息可以
有时会出现乱序。出于这个原因，我们使用滑动窗口来跟踪接收到的消息计数器，
我们跟踪接收到的最大计数器，以及接收到的先前消息的窗口，检查
只有在验证了身份验证标签后，使用 RFC2401 附录 C 中详述的算法 [14] 或者
通过 RFC6479 [26]，它使用更大的位图，同时避免位移，实现更极端的数据包重新排序
这可能发生在多核系统上。

5.4.7 负载不足：Cookie 回复消息

如第5.3节所述, 当收到具有有效 msg.mac1 的消息，但 msg.mac2 无效或
过期，对端负载过重，对端可能会发送 cookie 回复消息。I_m由
提示此 cookie 回复消息的消息的 msg.sender 字段，msg：

类型 := 0x3 (1 字节) 保留：= 0 3（3 个字节）

接收器：= I_m（4 个字节）

nonce := ρ 24 (24 字节)

cookie (ℸ16 字节)

秘密变量R_m每两分钟更改为一个随机值，A_m表示
下标的外部IP源地址和UDP源端口，M代表msg.mac1值
这是回复的消息。剩余的加密 cookie 回复字段填充如下：

$$\tau := \text{Mac}(R_m, A_m)$$

$$\text{msg.cookie} := \text{Xaead}(\text{Hash}(\text{Label-Cookie}\, S_{pub\,m}), \text{msg.nonce}, \tau, M)$$

值 Hash(Label-Cookie $S_{pub\,m}$) 以上可以预先计算。通过使用M作为额外的认证
data 字段，我们将 cookie 回复绑定到相关的消息，以防止 peer 被攻击
向他们发送欺诈性 cookie 回复消息。另请注意，此消息小于握手
发起消息或握手响应消息，避免放大攻击。

收到这条消息后，如果它是有效的，这条消息的接收者唯一应该做的就是存储
cookie 以及收到它的时间。第6节中描述的机制将用于
使用这些接收到的 cookie 重新传输握手消息；此 cookie 回复消息本身不应
导致重传。

6 计时器和无状态用户体验

从用户的角度来看，WireGuard 似乎是无状态的。配置接口的私钥，
后跟每个对等体的公钥，然后用户可以简单地正常发送数据包。这
会话状态的维护、完美的前向保密、握手等完全落后于
场景，对用户不可见。虽然历史上类似的自动机制存在缺陷和灾难性，
WireGuard 采用极其*简单的*计时器状态机，其中每个状态并转换到所有相邻的
状态被明确定义，从而产生完全的可靠性。不存在异常状态或状态序列；
一切都被考虑在内。它已在 10 Gb Intranet 以及低带宽测试中取得成功
高延迟跨大西洋商业航空公司互联网。定时器状态机的简单性归功于
事实上只需要 1-RTT 握手，发起者和响应者可以透明地切换角色，
WireGuard 打破了传统的分层，如第1节所述，因此可以使用层内
特征。

第 14 页

6.1 预赛

以下常量用于定时器状态系统：

象征	价值
Rekey-After-Messages	2 60 条消息
拒绝后消息	2 64 − 2 13 − 1 条消息
重新生成密钥	120 秒
拒绝后时间	180 秒
重新加密尝试时间	90 秒
重新加密超时	5秒
保活超时	10 秒

在任何情况下，WireGuard 都不会在每个*Rekey-Timeout*多次发送启动消息。成功接收握手响应消息后创建安全会话（第5.4.3节），和安全会话的年龄是从处理此消息的时间和紧随其后的时间开始计算的传输数据密钥的推导（第5.4.5节）。每当作为结果发送握手启动消息时一个过期定时器，一个额外的抖动量被添加到过期，以防止两个对等点同时重复发起握手。

6.2 传输消息限制

首次建立安全会话后，WireGuard 将尝试通过发送握手启动消息（第5.4.2节），在它发送了*Rekey-After-Messages*传输数据消息之后。

同样，如果对等方是当前安全会话的发起者，WireGuard 将发送握手发起如果在传输传输数据消息之后当前的安全会话开始一个新的安全会话的消息是*Rekey-After-Time*秒，或者如果在收到传输数据消息后，当前的安全会话是 (*Reject-After-Time* − *Keepalive-Timeout* − *Rekey-Timeout*) 秒，它尚未采取行动在这个事件上。这种基于时间的机会性密钥更新仅限于当前会话的发起者，在为了防止“雷群”问题，其中两个对等方可能会尝试在同时。由于被动保活功能，在第6.5节中描述，由一个旧的引发的启动传输传输数据消息*后的*安全会话通常应足以确保新会话创建每个*Rekey-After-Time*秒。但是，对于一个对端已经接收到数据但是没有没有任何数据可以立即发回，*Reject-After-Time*第二个截止日期即将到来早于*Keepalive-Timeout*秒，则由老化的安全会话触发的启动发生在接收路径。

在*Reject-After-Messages*传输数据消息之后或在当前安全会话被*Reject-After-Time*秒，以先到者为准，WireGuard 将拒绝发送或接收更多传输数据使用当前安全会话发送消息，直到通过 1-RTT 握手创建新的安全会话。

6.3 密钥轮换

大约每隔*Rekey-After-Time*秒创建新的安全会话（这更有可能在*Rekey-After-Messages*传输数据消息发送之前发生），由于传输消息上述第6.2节中描述的限制. 这意味着安全会话不断轮换，创建一个新的每次使用临时对称会话密钥，以实现完美的前向保密。但是，请记住，在发起人之后接收握手响应消息（第5.4.3节），响应者不能发送传输数据消息（第5.4.6节) 直到它收到来自发起者的第一个传输数据消息。而且，进一步，运输使用前一个安全会话加密的数据消息可能会在新的安全会话结束后传输被创建。由于这些原因，WireGuard 将当前的安全会话保存在内存中，以前的安全会话会话，以及未确认会话的下一个安全会话。每次新的安全会话被创建，现有的轮换到“前一个”槽，而新的占据“当前”槽，对于发起者和响应者，“下一个”时隙被间隙使用，直到握手被确认。这然后丢弃“previous-previous”一个，并将其内存归零（有关内存的讨论，请参见第7.4节）归零）。如果在 (*Reject-After-Time* × 3) 秒后没有创建新的安全会话，则当前安全会话，前一个安全会话，以及可能的下一个安全会话被丢弃并归零，此外任何可能的部分完成的握手状态和临时密钥。

第 15 页

6.4 握手发起重传

用户第一次通过 WireGuard 接口发送数据包时，无法立即发送数据包，因为当前会话不存在。因此，在对数据包进行排队后，WireGuard 会发送握手启动消息（第5.4.2节）。

发送握手发起消息后，由于首包条件，或由于限制第6.2节的条件, 如果握手响应消息（第5.4.3节) 之后没有收到*Rekey-Timeout*秒，构建一个新的握手发起消息（带有新的随机临时键）并发送。在放弃之前*尝试*重新启动*Rekey-Attempt-Time*秒，尽管这当对等方明确尝试发送*新的*传输数据消息时，计数器会重置。至关重要未来的工作包括调整*Rekey-Timeout*值以使用指数退避，而不是当前的固定值。

6.5 被动保活

最重要也是最优雅的，WireGuard 实现了一个被动的 keepalive 机制来确保会话保持活动状态并允许两个对等方被动地确定连接是否已失败或已断开连接。如果对等方已收到有效认证的传输数据消息（第5.4.6节），但没有任何数据包本身发送回*Keepalive-Timeout*秒，它发送一个*keepalive 消息*。保持连接消息很简单具有零长度封装的加密内部数据包的传输数据消息。由于所有其他运输数据

消息包含 IP 数据包，其最小长度为 min(IPv4 header，IPv6 header)，这个
keepalive 消息可以通过具有零长度封装数据包的简单优点轻松区分。（笔记
消息的 msg.packet 字段实际上长度为 16，即 Poly1305 的长度 [8] 验证
标签，因为零长度明文仍然需要验证，即使没有什么可以加密。）

这种被动保持连接仅在对等方没有任何内容可发送时发送，并且仅在以下情况下发送
另一个对等方正在向它发送经过身份验证的传输数据消息。这意味着当双方都
交换传输数据消息时，网络链接将保持沉默。

因为发送的每条传输数据消息都需要某种形式的回复——要么是一个自然生成的
根据封装数据包的性质或这个保活消息——我们可以确定安全会话
如果传输数据消息尚未收到 (*Keepalive-Timeout* +
Rekey-Timeout) 秒，在这种情况下，握手启动消息将发送到无响应的对等方，
每*Rekey-Timeout*秒一次，如第6.4节, 直到成功重新创建安全会话或直到
*Rekey-Attempt-Time*秒已过。

6.6 与 Cookie 回复系统的交互

如第5.3和5.4.7 节所述, 当对等点处于负载下时，握手启动消息或握手
可能会丢弃响应消息并发送 cookie 回复消息。收到 cookie 回复消息后，
这将使对等方能够使用有效的 msg.mac2 发送新的启动或响应消息，该消息不会被
丢弃，对等方 *不*应该立即重新发送现在有效的消息。相反，它应该简单地存储
从 cookie 回复消息中解密 cookie 值，并等待*Rekey-Timeout*到期
重试握手发起消息的计时器。这可以防止潜在的带宽生成滥用，以及
有助于缓解首先需要 cookie 回复消息的负载条件。

7 Linux 内核实现

Linux 内核中 WireGuard 的实现有几个目标。首先，它应该简短而简单，所以
审计和审查安全漏洞的代码不仅容易，而且令人愉快；WireGuard 是
用*不到 4,000 行代码实现*（不包括加密原语）。其次，它必须非常
速度快，使其在性能上与 IPsec 竞争。第三，它必须避免分配和其他资源
响应传入数据包的密集分配。第四，它必须像本地一样流畅地集成
可以使用现有的内核基础架构和用户空间期望、工具和 API。第五，它必须是
可构建为外部内核模块，无需对核心 Linux 内核进行任何更改。WireGuard 是
不仅仅是一个从未发布过实验室代码的学术项目，而是一个旨在
生产就绪的实现。

第 16 页

7.1 排队系统

WireGuard 设备驱动程序具有向内核表明它支持通用分段卸载的标志
(GSO)、分散收集 I/O 和硬件校验和卸载，总而言之，这意味着内核将处理
WireGuard 的“超级数据包”，即远远超过 MTU 大小的数据包，已被预先排队
上层，例如 TCP 或 TCP 和 UDP 软木塞系统。这允许 WireGuard 批量操作
出站数据包组。将数据包分成≤MTU 大小的块后，WireGuard 尝试加密，
封装并通过 UDP 一次性发送所有这些，缓存路由信息，以便它只需要
每组数据包计算一次。这也具有减少缓存未命中的非常重要的效果：
等到超级数据包的所有单个数据包都被加密和封装后才能将它们传递出去
到网络层，非常复杂和 CPU 密集型的网络层保存指令，中间
CPU 缓存中的变量和分支预测，在许多情况下使发送性能提高 35%。

同样，如第6.4节所述, 有时传出的数据包必须排队直到握手完成
成功地。当数据包最终能够被发送时，现有排队数据包的整个队列都是
将其视为单个超级数据包，以便从与上述相同的优化中受益。

最后，为了防止不必要的分配，所有数据包转换都是*就地*完成*的*，避免
复制的需要。这不仅适用于就地发生的数据加密和解密，而且
也用于使用 sendfile(2) 发送的某些用户空间数据和文件；这些是使用这个零拷贝超级处理的
数据包排队系统。

排队系统的未来工作可能涉及将 WireGuard 与 FlowQueue [12]-
代码 [21] 调度算法。

7.2 软中断与并行

xfrm 层与 WireGuard 相比，其优点是不需要在 softirq 中进行加密，
这使它具有更大的灵活性。然而，*就是*在做加密处理的先例
接口层的 softirq：用于无线 WPA 加密的 mac802111 子系统。WireGuard，正在
进行加密的虚拟接口在架构上与无线接口没有太大不同
同层加密。虽然在实践中它确实工作得很好，但它不是并行的。为此，该
内核的 padata 系统用于并行化并发工作者加密和解密操作
用于所有 CPU 和 CPU 内核的利用率。同样，数据包校验和可以与此并行计算
方法。但是，在发送数据包时，它们必须按顺序发送，这意味着不能简单地将每个数据包
加密后立即发送。幸运的是，padata API 将操作划分为并行步骤，
接着是一个有序的串行步骤。这也有助于并行解密，其中消息计数器
必须按照数据包到达的顺序检查和递增，以免不必要地拒绝它们。为了
为了减少延迟，如果超级数据包中只有一个数据包并且其长度小于 256 字节，或者如果
只有一个 CPU 核在线，数据包在 softirq 中处理。

同样，握手发起和响应消息以及 cookie 回复消息在单独的
并行低优先级工作线程。如第5.3节所述, ECDH 操作是 CPU 密集型的，所以它是
重要的是，大量的握手工作不会垄断 CPU。低优先级后台工作队列
用于此异步握手消息处理。

7.3 基于 RTNL 的虚拟接口和容器化

为了与现有的 ip(8) 实用程序和基于 netlink 的 Linux 用户空间集成，内核的 RTNL
层用于注册虚拟接口，在内核内部称为“链接”。这很容易访问
ip-link(8) 和 ip-set(8) 访问的内核 API。用于配置接口私钥和公钥
对等方的密钥和端点，最初使用RTM_SETLINK RTNL 消息，但事实证明这太
有限的。事实证明，简单地实现一个基于 ioctl(2) 的 API，传递一系列结构，要简洁得多
来回。虽然这种方法非常干净，但 Linux 网络堆栈正在朝着完全
基于 Netlink 的配置 API，因此最终选择了 Generic Netlink 协议。独立的用户空间
工具 wg(8) 用于通过 Netlink 进行通信，未来计划涉及集成此功能
直接进入 ip(8)。

RTNL 子系统允许在网络命名空间之间移动 WireGuard 虚拟接口。这个
允许在一个命名空间中创建发送和接收套接字（用于外部 UDP 数据包），而
接口本身保留在另一个命名空间中。例如，docker(1) 或 rkt(1) 容器来宾可以具有作为
它唯一的网络接口是一个 WireGuard 接口，实际的外部加密数据包从
主机上的真实网络接口，在容器内外创建端到端的经过身份验证的加密。

第 17 页

7.4 数据结构和原语

虽然 Linux 内核已经包含两个精心设计的路由表实现——一个 LC-trie [22] 用于 IPv4
和 IPv6 的 radix trie——它们与 FIB 路由层密切相关，并且根本不可重用于其他
使用。为此，开发了一个非常小的路由表。作者已成功实施
加密密钥路由表作为分配路由表 [11]，一个 LC-trie [22]，以及一个标准的基数树，其中
每一个都提供了足够但略有不同的性能特征。最终的简单性
古老的 radix trie 是首选，具有良好的性能特征和实现它的能力
无锁查找，使用 RCU 系统 [19]。每次传出数据包通过 WireGuard 时，
使用此表查找目标对等方，每次传入数据包到达 WireGuard 时，其有效性
通过查阅此表进行检查，因此性能实际上在这里很重要。

对于所有握手启动消息（第5.4.2节），响应者必须查找解密后的静态公共
启动器的密钥。为此，WireGuard 采用了一个哈希表，该表使用了极快的 SipHash2-4 [1] 苹果电脑
具有秘密的功能，以便上层可以为 WireGuard 接口提供公钥
更复杂的密钥分发方案，无法挂载哈希表碰撞拒绝服务攻击。

虽然 Linux 内核的加密 API 拥有大量原语集合，并且打算在多个
不同的系统，API 引入了不必要的复杂性和分配。WireGuard 的多次修订
使用具有不同集成技术的加密 API，但最终，使用具有直接、
事实证明，非抽象的 API 更干净，资源占用更少。堆栈和堆压力都是
通过直接使用加密原语，而不是通过内核的加密 API 来减少。加密货币
当以多方面的方式使用多个键时，API 也使得避免分配变得极其困难
噪音要求。在撰写本文时，WireGuard 附带了 ChaCha20Poly1305 的优化实现，用于
各种英特尔架构矢量扩展，以及针对 ARM/NEON 和 MIPS 的实现。
在运行时选择硬件支持的最快实现，浮点单元为
机会主义地使用。加密操作的所有临时密钥和中间结果都被清零
使用后的内存，以保持完美的前向保密并防止各种潜在的泄漏。
必须特别通知编译器有关此显式归零的信息，以便不会优化“死区”
out，为此内核提供了 memzero_explicit 函数。

与加密原语相比，现有的基于令牌桶哈希的速率限制的内核实现，
用于在 cookie IP 归属后负载不足时限制握手启动和响应消息的速率
发生，在 WireGuard 中已经非常少且易于重用。WireGuard 使用 Netfilter hashlimit
匹配器。

7.5 FIB 考虑

为了避免路由循环，提出了对 Linux 内核的一项更改——目前由作者发布到
Linux 内核邮件列表 [9]—允许排除接口的 FIB 路由查找。这样一来，
内核的路由表可能有 0.0.0.0/1 和 128.0.0.0/1，对于 0.0.0.0/0 的组合覆盖，同时
更具体的，发送到wg0接口。然后，可以路由 WireGuard 对等点的各个端点
如果 wg0 不存在，则使用 FIB 查找将返回的设备，即通过实际 0.0.0.0/0
路线。或者更一般地说，当查找将数据包路由到特定对等端点的正确接口时，
将返回*保证*不是 wg0的接口路由。这比当前更可取
需要将 WireGuard 对等端点的显式路由添加到内核路由表时的情况
WireGuard 绑定的路由具有优先权。这项工作正在进行中。

上面提到的另一种方法是使用网络命名空间来完全隔离 WireGuard 接口
和路由表来自物理接口和路由表。一个命名空间将包含 WireGuard
接口和带有默认路由的路由表，用于通过 WireGuard 接口发送所有数据包。另一个
命名空间将包含各种物理接口（以太网设备、无线电等）
使用其通常的路由表。WireGuard 接口的传入和传出 UDP 套接字将*位于*
第二个物理接口命名空间，而不是第一个 WireGuard 接口命名空间。这样，发送的数据包
WireGuard 接口命名空间在那里加密，然后使用位于物理中的套接字发送
接口命名空间。这可以防止所有路由循环并确保完全隔离。进程生活在
WireGuard 接口命名空间将作为其唯一的网络手段 WireGuard 接口，防止
任何潜在的明文数据包泄漏。

第 18 页

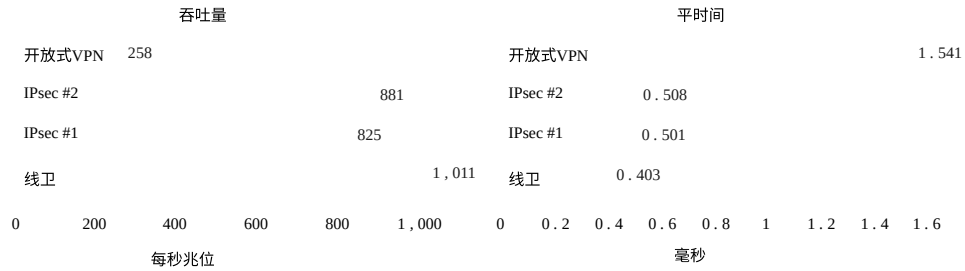
7.6 潜在的用户空间实现

为了让 WireGuard 得到广泛采用，比我们目前用于 Linux 的实现更多
内核必须写。作为下一步，作者计划实现一个跨平台的低速用户空间
使用安全且高速的语言（如 Rust、Go 或 Haskell）实现基于 TUN 的实现。

8 性能

WireGuard 在两种模式下与 IPsec 和 OpenVPN 一起进行了基准测试，在 Intel
Core i7-3820QM 和带有 Intel 82579LM 和 Intel I218LM 千兆以太网卡的 Intel Core i7-5200U
分别，结果平均超过三十分钟。结果非常有希望：

协议	配置
线卫	256 位 ChaCha20、128 位 Poly1305
IPsec #1	256 位 ChaCha20、128 位 Poly1305
IPsec #2	256 位 AES，128 位 GCM
开放式VPN	256 位 AES、HMAC-SHA2-256、UDP 模式



在这两个指标上，WireGuard 都优于 OpenVPN 和两种 IPsec 模式。CPU 为 100%
OpenVPN 和 IPsec 的吞吐量测试期间的利用率，但未完全用于测试
WireGuard，表明 WireGuard 能够使千兆以太网链路完全饱和。

虽然 AES-NI 加速的 AES-GCM IPsec 密码套件似乎优于 AVX2 加速
ChaCha20Poly1305 IPsec 密码套件，随着未来的芯片增加向量指令的宽度——例如 upcom-
AVX512——预计随着时间的推移，ChaCha20Poly1305 的性能将超过 AES-NI [4]。ChaCha20Poly1305
特别适合在软件中实现，不受旁道攻击，效率很高，在
与AES相反，因此对于没有专用AES指令的嵌入式平台，ChaCha20Poly1305也将
性能最好。

此外，由于实现的简单性，WireGuard 已经优于两种 IPsec 密码套件
并且没有开销。OpenVPN 和 WireGuard 之间的巨大差距是意料之中的，无论是在
ping 时间和吞吐量，因为 OpenVPN 是一个用户空间应用程序，这意味着增加了延迟
调度器的开销以及在用户空间和内核空间之间多次复制数据包的开销。

9 结论

在不到 4,000 行中，WireGuard 证明可以拥有安全的网络隧道
简单实施，性能极佳，利用最先进的密码学，并保持容易
管理。简单性使其可以非常容易地独立验证并在广泛的范围内重新实现
平台的多样性。所使用的加密结构和原语确保了广泛的高速
设备的多样性，从数据中心服务器到手机，以及可靠的安全属性
未来。易于部署也将消除目前许多常见和灾难性的陷阱
在许多 IPsec 部署中都可以看到。大约在 Ferguson 和 Schneier 介绍它的时候描述 [10],
“IPsec 令我们非常失望。考虑到 [*原文如此*] 工作人员的素质和时间
花在它上面，我们期待一个更好的结果。[...] 我们对 IPsec 的主要批评是它的复杂性。”

第 19 页

相比之下，WireGuard 专注于简单性和可用性，同时仍然提供可扩展且高度安全的
系统。通过对未经身份验证的数据包保持沉默，不进行任何分配并通常保持
资源利用率降到最低，可部署在网络外缘，作为可信赖的
可靠的接入点，它不会轻易向攻击者展示自身，也不会提供可行的攻击目标。这
加密密钥路由表范式易于学习，将促进安全的网络设计。该协议是基于
基于密码学上合理和保守的原则，使用很好理解但现代的密码原语。
WireGuard 是从实用的角度设计的，旨在解决现实世界的安全网络问题。

10 致谢

WireGuard 的诞生得益于许多人的出色建议和指导，尤其是：Trevor Perrin、
Jean-Philippe Aumasson、Steven M. Bellovin 和 Greg Kroah-Hartman。

参考

[1] 让-菲利普·奥马松和丹尼尔·J·伯恩斯坦。“密码学进展 - INDOCRYPT 2012: 13th

印度密码学国际会议，印度加尔各答，2012 年 12 月 9-12 日。会议录”。在：
编。作者：Steven Galbraith 和 Mridul Nandi。文档 ID：b9a943a805fbfc6fde808af9fc0ecdfa。柏林，
海德堡：Springer Berlin 海德堡，2012 年。SipHash：快速短输入 PRF，第 489-508 页
978-3-642-34931-7。doi：10.1007/978-3-642-34931-7_28。网址：https://cr.yp.to/siphas
20120918.pdf（引自第17页）。

[2] 让-菲利普·奥马松等人。“BLAKE2：更简单、更小、像 MD5 一样快”。在：*第 11 届会议纪要
应用密码学和网络安全国际会议*。ACNS'13。加拿大 AB 班夫：
Springer-Verlag，2013 年，第 119-135 页。isbn：978-3-642-38979-5。doi：10.1007/978-3-642-38980-1_8。网址：
https://blake2.net/blake2.pdf（引自第3页）。

[3] 丹尼尔·J·伯恩斯坦。“ChaCha，Salsa20 的变体”。在：*国资委 2008 年*。文档编号：4027b5256e17b97968
42e6d0f68b0b5e。2008. 网址：https://cr.yp.to/chacha/chacha-20080128.pdf（引自第3页）。

[4] 丹尼尔·J·伯恩斯坦。*CPU 针对视频游戏进行了优化*。网址：https://moderncrypto.org/mail-
archive/noise/2016/000699.html（引自第18页）。

[5] 丹尼尔·J·伯恩斯坦。“Curve25519：新的 Diffie-Hellman 速度记录”。在：*公钥密码学——PKC
2006 年*。埃德。由 Moti Yung 等人撰写。卷。3958. 计算机科学讲义。文档 ID：4230efdfa673
480fc079449d90f322c0。柏林，海德堡：Springer-Verlag Berlin Heidel
978-3-540-33852-9。doi：10.1007/11745853_14。网址：https://cr.yp.t
（引用第3页）。

[6] 丹尼尔·J·伯恩斯坦。*扩展 Salsa20 nonce*。文档 ID：c4b172305ff16e1429a48d9434d50e8a。
2011. 网址：https://cr.yp.to/snuffle/xsalsa-20110204.pdf（引自第9页）。

[7] 丹尼尔·J·伯恩斯坦。*TAI64*、*TAI64N 和 TAI64NA*。url: https://cr.yp.to/libtai/tai64.html (cit. on
第7，10）。

[8] 丹尼尔·J·伯恩斯坦。“Poly1305-AES 消息认证代码”。在：*快速软件加密：第 12 名
国际研讨会，FSE 2005，巴黎，法国，2005 年 2 月 21 日至 23 日，修订的精选论文*。卷。3557。
计算机科学讲义。文档 ID：0018d9551b5546d97c340e0
第 32-49 页。doi：10.1007/11502760_3。网址：https://c
15）。

[9] 杰森 A. 多南菲尔德。*flowi{4,6}_oif 的倒数：flowi{4,6}_not_oif*。网址：http://lists.openwall.net/
netdev/2016/02/02/222（引自第17页）。

[10] 尼尔斯·弗格森和布鲁斯·施奈尔。*IPsec 的密码评估*。技术。代表
安全公司，2000 年。doi：10.1.1.33.7922。网址：https://www
paper-ipsec.pdf（引自第18页）。

[11] 日口洋一。*分配路由表：一个快速免费的基于多位 Trie 的路由表*。2002. 网址：
https://github.com/hariguchi/art/blob/master/docs/art.pdf（引自第17页）。

[12] Toke Hoeiland-Joergensen 等。*FlowQueue-CoDel 数据包调度程序和活动队列管理
算法*。RFC。互联网工程任务组，2016 年 3 月，p。23. 网址：https://tools.ietf.org/html/draft-
ietf-aqm-fq-codel-06（引自第16页）。

[13] C.考夫曼等。*Internet 密钥交换协议第 2 版*。RFC 5996。RFC 编辑器，2010 年 9 月。网址：
http://www.rfc-editor.org/rfc/rfc5996.txt（引文上页3，8）。

第 20 页

[14] 斯蒂芬·肯特和兰德尔·阿特金森。*IP 的安全架构*。RFC 2401。RFC 编辑，1998 年 11 月，
页。57. 网址：http://www.rfc-editor.org/rfc/rfc2401.txt（引用第13页）。

[15] 雨果·克劳奇克。“密码学的进步 - CRYPTO 2010：第 30 届年度密码学会议，圣诞老人
美国加利福尼亚州芭芭拉，2010 年 8 月 15-19 日。会议录”。在：编辑。塔尔·拉宾。柏林、海德堡：施普林格
柏林海德堡，2010 年。密码提取和密钥派生：HKDF 计划，第 631 页–
648. isbn：978-3-642-14623-7。doi：10.1007/978-3-642-14623-7_34。网址：https://eprint.iacr.or
264.pdf（引文上页3，10）。

[16] 雨果·克劳奇克。“SIGMA：认证 Diffie-Hellman 的‘SIGn-and-MAC’方法及其
在 IKE 协议中使用”。在：*密码学进展 - CRYPTO 2003，第 23 届国际年会
密码学会议，美国加利福尼亚州圣巴巴拉，2003 年 8 月 17-21 日，会议录*。卷。2729.
计算机科学讲义。斯普林格，2003 年，第 400-425 页。doi：10.1007/978-3-540-45146-4_24。
网址：http://www.iacr.org/cryptodb/archive/2003/CRYPTO/1495/1495.pdf（引用第7页）。

[17] 亚当·兰利和约夫·尼尔。*ChaCha20 和 Poly1305 用于 IETF 协议*。RFC 7539。RFC 编辑，五月
2015年网址：http://www.rfc-editor.org/rfc/rfc7539.txt（引文上页3，9）。

[18] 克里斯汀·劳特和安东·米佳金。“公钥密码学 - PKC 2006：第 9 届国际会议
公钥密码学的理论与实践，纽约，纽约，美国，2006 年 4 月 24 日至 26 日。会议录”。在：
编。由 Moti Yung 等人撰写。柏林，海德堡：Springer Berlin Heidelberg，2006 年。KEA的安全分析
经过身份验证的密钥交换协议，第 378-394 页。isbn：978-3-540-33852-9。doi：10.1007/11745853_25。
网址：http://research.microsoft.com/en-us/um/people/klauter/pkcspringer.pdf（引用第7页）。

[19] Paul E. McKenny 等。“读取-复制更新”。在：*渥太华 Linux 研讨会*。2002 年 6 月，第 338-367 页。网址：
http://www.rdrop.com/~paulmck/RCU/rcu.2002.07.08.pdf（引自第17页）。

[20] R. Moskowitz 等。*主机身份协议版本 2*。RFC 7401。RFC 编辑，2015 年 4 月。网址：http://
www.rfc-editor.org/rfc/rfc7401.txt（引自第9页）。

[21] 凯瑟琳·尼科尔斯和范·雅各布森。“控制队列延迟”。在：*社区。ACM* 55.7（2012 年 7 月
第 42-50 页。编号：0001-0782。doi：10.1145/2209249.2209264。网址：http://doi.ac
2209336（引自第16页）。

[22] Stefan Nilsson 和 Gunnar Karlsson。“使用 LC 尝试的 IP 地址查找”。在：*IEEE Journal on Selected
通信领域*17.6（1999 年 6 月），第 1083-1092 页。刊号：0733-8716。doi：10.1109/49.772439。网址：
https://www.nada.kth.se/~snilsson/publications/IP-address-lookup-using-LC-tries/text.pdf (cit. on
页。17）。

[23] 特雷弗·佩林。*噪声协议框架*。2016. 网址：http://noiseprotocol.org/noise.pdf（引自
第3，7，11，12）。

[24] E. Rescorla 和 N. Modadugu。*数据报传输层安全版本 1.2*。RFC 6347。RFC 编辑器，
2012 年 1 月。网址：http://www.rfc-editor.org/rfc/rfc6347.txt（引用第8页）。

[25] 基思·温斯坦和哈里·巴拉克里希南。“Mosh：移动客户端的交互式远程外壳”。在：*USENIX
年度技术会议*。马萨诸塞州波士顿，2012 年 6 月。网址：https://mosh.mit.edu/mosh-paper.pdf
（引用第5页）。

[26] 张向阳和 Tina Tsou。*无位移位的 IPsec 抗重放算法*。RFC 6479。RFC

编辑，2012 年 1 月，第 9. 网址：http : //www.rfc-editor.org/rfc/rfc6479.txt（引用第13页）。