WireGuard

FAST, MODERN, SECURE VPN TUNNEL

速くて、現代的な、安全なVPNトンネル

Presented by ジェイソン・ドーネンフェルド

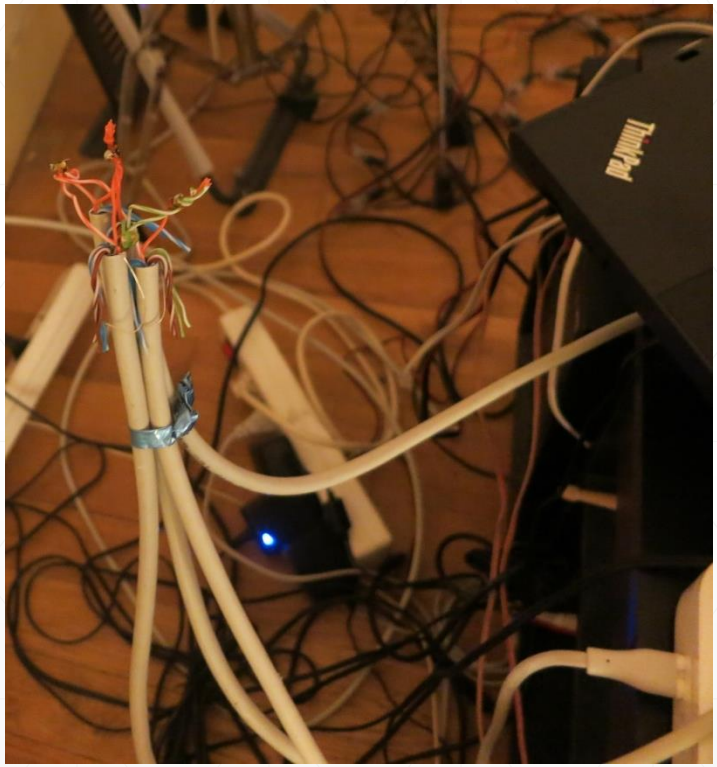
EDGESECURITY

私について

- ZX2C4として知られているジェイソン・ドーネンフェルドは、セキュリティコンサルティング会社のEdge Security (.com)の設立者でもある。
- エクスプロイト、カーネルの脆弱性、暗号の脆弱性の経験がある傍ら、多くの開発経験も有する。
- 私は他の多くのプロジェクトで見つけた暗号と実装の問題を避けるようなVPNを作ることが動機である。

WireGuardとは何か?

- IPv4とIPv6のためのセキュアなレイヤ3ネットワークトンネル。
  - 頑固に。
- Linuxカーネル上で動作するが、クロスプラットフォームの実装も開発中。
- UDP通信。ファイアウォールを突き抜ける。
- 現代的かつ保守的な暗号原則。
- 簡潔さと監査能力を重視。
- 認証はSSHの認証キーに似ている。
- OpenVPNやIPsecと置き換える。

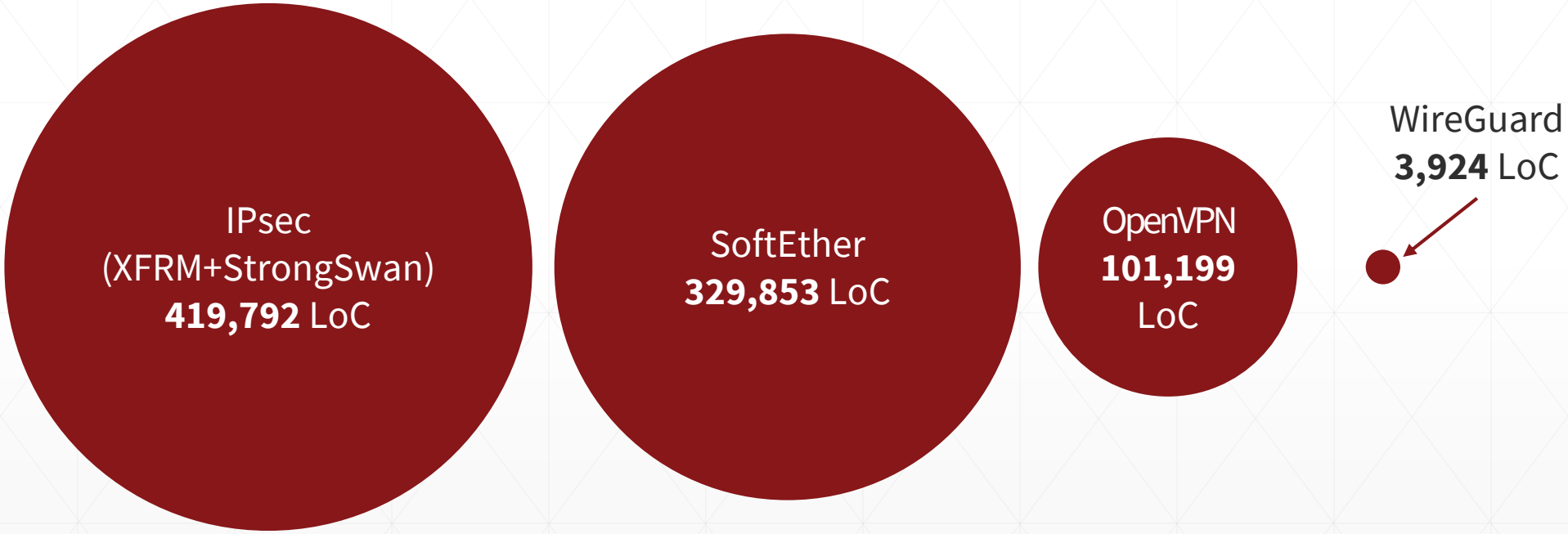


セキュリティ設計の原則１：簡単に監査可能

OpenVPN	Linux XFRM	StrongSwan	SoftEther	WireGuard
101,199 LoC さにらOpenSSL!	13,898 LoC さらにStrongSwan!	405,894 LoC さらにXFRM!	329,853 LoC	**3,924 LoC**

少ないことは良いことだ。

セキュリティ設計の原則1：簡単に監査可能



セキュリティ設計の原則２：簡潔なインターフェース

- WireGuardは通常のネットワークインターフェースを表現する:

```
# ip link add wg0 type wireguard
# ip address add 192.168.3.2/24 dev wg0
# ip route add default via wg0
# ifconfig wg0 …
# iptables -A INPUT -i wg0 …

/etc/hosts.{allow,deny}, bind(), …
```

- eth0やwlan0のように、通常のネットワーク・インターフェースとして構築できるものはすべてwg0上で構築できる。

冒涜！

- WireGuardは冒涜的だ！
- 我々は、IPsecのような90年代ネットワーク技術のいくつかのレイヤリングの仮定を破壊する。
  - IPSecは発信パケットのための「変換テーブル」を含む。それはユーザスペースデーモンによって管理され、鍵交換と変換テーブルの更新が行われる。
- WireGuardに関して、私たちはとても基本的なブロック -- ネットワークインターフェース -- の構築から始め、そこから組み上げる。
- 学術的で綺麗なレイヤ構成は欠いているが、賢い機構を通じて私たちはより一貫性のある何かに到達する。

暗号鍵の経路制御

- **どんなVPNの基本概念でも公開鍵のピアとそれらのピアに使われるIPアドレスとの間の関連付けがある。**
- WireGuardのインターフェースは以下を持っている:
  - 秘密鍵
  - 接続要求待ちのUDPポート
  - ピアのリスト
- ピア:
  - その公開鍵で識別されたもの
  - 関連付けされたトンネルIPのリストを持っている
  - エンドポイントのIPとポートを任意で持つ。

暗号鍵の経路制御

サーバの設定

```
[Interface]
PrivateKey =
yAnz5TF+lXXJte14tji3zlMNq+hd2rYU
IgJBgB3fBmk=
ListenPort = 41414

[Peer]
PublicKey =
xTIBA5rboUvnH4htodjb6e697QjLERt1
NAB4mZqp8Dg=
AllowedIPs =
10.192.122.3/32,10.192.124.1/24

[Peer]
PublicKey =
TrMvSoP4jYQlY6RIzBgbssQqY3vxI2Pi
+y71lOWWXX0=
AllowedIPs =
10.192.122.4/32,192.168.0.0/16
```

クライアントの設定

```
[Interface]
PrivateKey =
gI6EdUSYvn8ugXOt8QQD6Yc+JyiZxIhp
3GInSWRfWGE=
ListenPort = 21841

[Peer]
PublicKey =
HIgo9xNzJMWLKASShiTqIybxZ0U3wGLi
UeJ1PKf8ykw=
Endpoint = 192.95.5.69:41414
AllowedIPs = 0.0.0.0/0
```

暗号鍵の経路制御

User spaceはsend()を使ってパケットを送る。

一般的なLinux経路制御表wg0にそれを与えることを決定する。

WireGuardはそれがどのピアのためであるか決定するためにパケットの到達先IPアドレスを調査する。

パケットはピアのセッションキーを用いて暗号化され、ピアのエンドポイントに送信される。

WireGuard UDPソケットはrecv()を使ってエンドポイントのパケットを受信する。

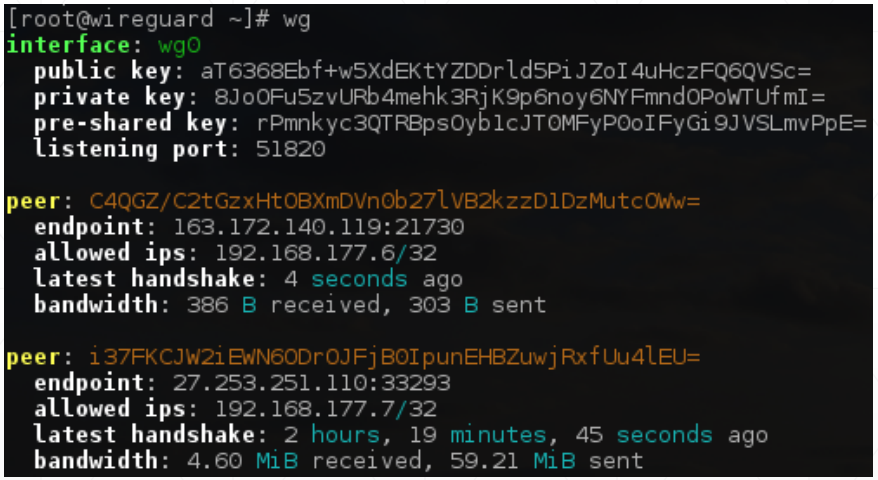
このパケットの復号し、そうすることで、そのピアがどこからなのかを学ぶ。

WireGuardは復号されたパケットの発信元IPを調べて、これがそれを送ったピアと実際に対応するかどうか見ます。

もしそれが対応するなら、パケットは許可されるが、さもなければそれは落とされる。

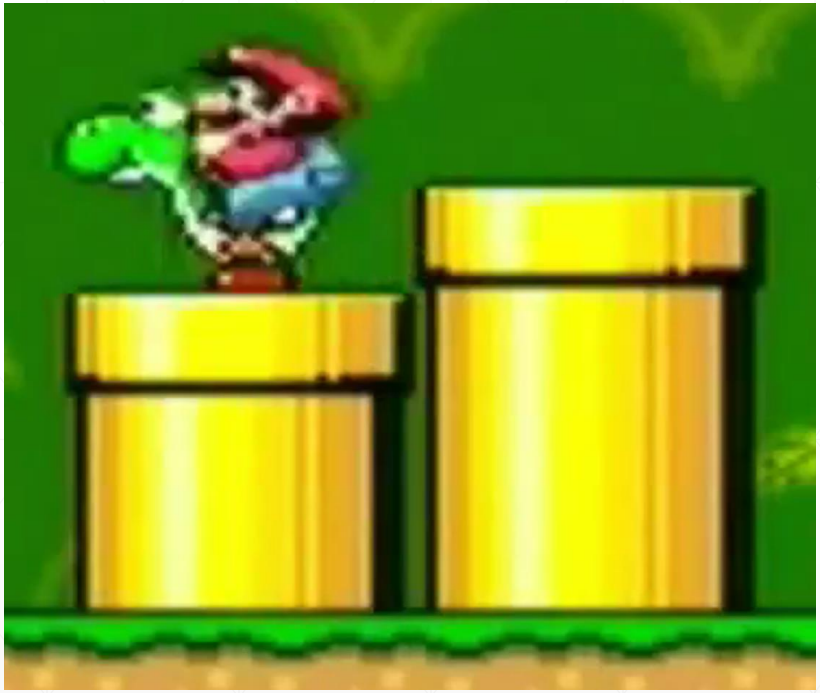
暗号鍵の経路制御

- wg set wg0
  ```
  listen-port 2345
  private-key /path/to/private-key
  peer ABCDEF…
       allowed-ips 192.168.88.3/32
       endpoint 209.202.254.14:8172
  peer XYZWYAB…
       remove
  peer 123456…
       allowed-ips 192.168.88.4/32
       endpoint 212.121.200.100:2456
  ```
- wg setconf < config.file
- wg getconf > config.file
- wg show
- wg genkey > private.key
- wg pubkey < private.key > public.key



暗号鍵の経路制御

- システム管理を非常に単純にする。
- もし、それがインターフェースwg0から来て、かつ、ヨッシーのトンネルIPアドレス192.168.5.17 からであれば、パケットは明らかにヨッシーから来ている。
- iptablesの規則は平易で明確である。



セキュリティ設計の原則２：簡潔なインターフェース

- 管理者にはインターフェースがステートレスに見える。
- wg0, wg1, wg2とインターフェースを追加し、ピアを設定すると即座にパケットの送信ができる。
- エンドポイントはモッシュの中のように歩き回る。
  ※モッシュ：人々が密集した状態で無秩序に体をぶつけあうこと。
- アイデンティティはSSHと同じく、静的な公開鍵のみである。
- セッション状態、接続その他もろもろは管理者には見えない。

セキュリティ設計の原則３：静的割付け、保護された状態、決められたヘッダ長

- 動作するWireGuardのために要求されるすべての状態は設定中に割り当てられる。
- 受け取ったパケットに応答して動的に割り当てられるメモリはない。
  - 全ての脆弱性の種類を取り除く。
- 全てのパケットヘッダは固定されたフィールドを持ち、構文解析を必要としない。
  - 他のすべての脆弱性の種類を取り除く。
- 認証されていないパケットに応答して変更されるステートはない。
  - さらに他のすべての脆弱性の種類を取り除く。

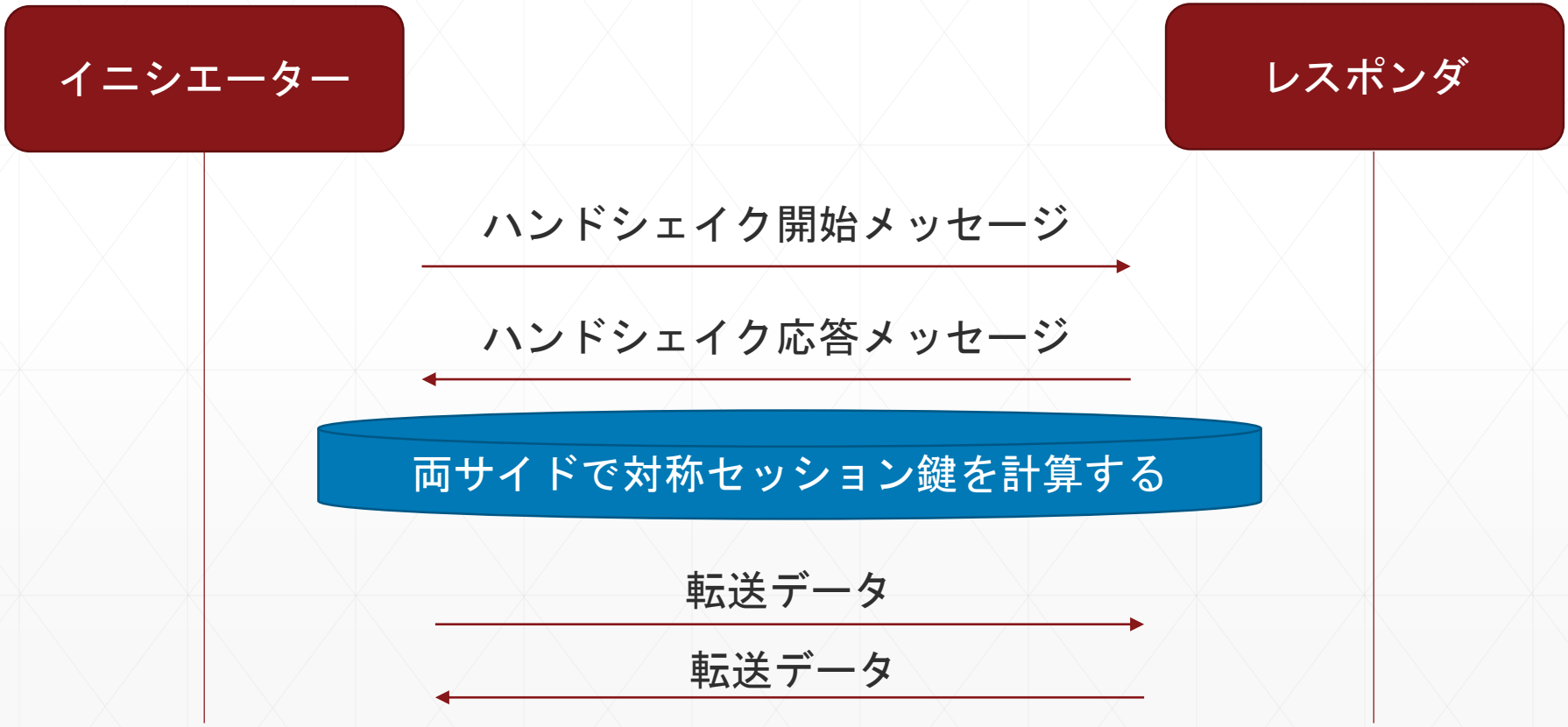
セキュリティ設計の原則４：ステルス

- WireGuardのいくつかのアスペクトはカーネルルートキットプロジェクトから生じた。
- 認証されていないいかなるパケットにも応答すべきでない。
- スキャナとサービス検索を妨げる。
- サービスは正しく暗号化されたパケットにのみ応答する。
- 全くおしゃべりでない。
  - それらがデータ交換を行わなかったとき、両者のピアは沈黙する。

セキュリティ設計の原則５：頑丈な暗号

- 我々は、トレヴァー・ペリンの雑音プロトコル・フレームワークを利用する。– noiseprotocol.org
  - WireGuard開発から多くのフィードバックを受けて開発された。
  - 特別仕様で書かれた、カーネルのためのNoiseIKの非常に限定された実装。
- Perfect forward secrecy(PFS) — 2分ごとに新しい鍵へ
- 危殆化鍵による成りすまし攻撃を回避する。
- アイデンティティの隠蔽
- 認証された暗号
- ネットワークパケット再構成中のリプレイ攻撃の防止。
- 最新の基本要素：Curve25519, Blake2s, ChaCha20, Poly1305, SipHash2-4
- 暗号アジリティがない！

原則3+4+5 → 鍵交換


イニシエーター
レスポンダ
ハンドシェイク開始メッセージ
ハンドシェイク応答メッセージ
両サイドで対称セッション鍵を計算する
転送データ
転送データ

原則３＋４＋５ → 鍵交換

- 2つのピアがデータを交換するために、それらは静的公開鍵から一時的な対称暗号セッション鍵を得られなければならない。
- 鍵交換は、頑丈な暗号、静的割当て、状態保護、固定されたヘッダ長、ステルス性の原則を守るように特にうまく設計されている。
- 1つのピアは交換のイニシエーター、もう一つのピアはレスポンダ。
- イニシエーターとレスポンダはいつでも役割を交代できる。
- 1-RTT
- どちらの側でも、新しいセッション鍵を得るために、ハンドシェイクをやり直すことができる。
- 不正なハンドシェイクメッセージは無視され、ステルスの原則が維持される。

鍵交換：Diffie-Hellmanの復習

```
private A = random()
public A = derive_public(private A)

private B = random()
public B = derive_public(private B)
```

DH(private A, public B) == DH(private B, public A)

鍵交換：NoiseIK

- 一方のピアがイニシエーター；他方がレスポンダ
- それぞれのピアは、静的な長期間鍵ペアという、アイデンティティを持っている。
- 新しいハンドシェイク毎に、各ピアは一時的な鍵ペアを生成する。
- われわれが求めるセキュリティ仕様は、2つの静的な鍵ペアと2つの一時的な鍵ペアの組合せ上でDH()を計算することによって達成される。
- Session keys = Noise(

```
DH(ephemeral, static),
DH(static, ephemeral),
DH(ephemeral, ephemeral),
DH(static, static)
)
```

- 最初の3つのDH()は “triple dh”を作り、1-RTT内に最後のDH()が最初のメッセージの認証を許可する。

鍵交換：NoiseIK - イニシエーター→レスポンダ

- イニシエーターはレスポンダの静的な長期公開鍵を知っていることから始める。
- イニシエーターはレスポンダに以下のものを送る：
  - 平文の一時的な公開鍵。
  - 以下の結果の（間接的な）鍵を使って、認証され、暗号化されたイニシエーターの公開鍵：

    ```
    DH(initiator's ephemeral private, responder's static public) ==
    DH(responder's static private, initiator's ephemeral public)
    ```

    - これを復号した後、レスポンダはイニシエーターの公開鍵を知る。
    - レスポンダの静的秘密鍵の管理を必要とするから、レスポンダだけがこれを復号できる。
  - 以下と同様に上記の計算の（間接的な）結果の鍵を使って認証され、暗号化された単調増加カウンター（たいていはまさにTAI64Nのタイムスタンプ）：

    ```
    DH(initiator's static private, responder's static public) ==
    DH(responder's static private, initiator's static public)
    ```

    - このカウンタは再送DoS攻撃に対して防御する。
    - イニシエーターが管理する自身のプライベートキーを検証することを認証する。
    - 最初のメッセージの認証 – static-static DH().

鍵交換：NoiseIK – レスポンダ→イニシエーター

- イニシエーターの一時的な公開鍵と同じように、このポイントのレスポンダはイニシエーターの事前の最初のメッセージから静的公開鍵を学習する。
- レスポンダは以下のものをイニシエーターに送信する：
  - 一時的な平文の公開鍵。
  - 空のバッファ、以下と同様に事前のメッセージで計算した（間接的な）結果の鍵を使って認証され、暗号化された：

    ```
    DH(responder's ephemeral private, initiator's ephemeral public) ==
    DH(initiator's ephemeral private, responder's ephemeral public)
    ```

    そして

    ```
    DH(responder's ephemeral private, initiator's static public) ==
    DH(initiator's static private, responder's ephemeral public)
    ```

    - レスポンダが管理する自身の秘密鍵を検証することを認証する。

鍵交換：セッションの導出

- 先の2つのメッセージ（イニシエーター→レスポンダと、レスポンダ→イニシエーター）の後、イニシエーターとレスポンダの両者は以下のDH()計算に結びつく何かを持っている：
  - `DH(initiator's ephemeral private, responder's static public) == DH(responder's static private, initiator's ephemeral public)`
  - `DH(initiator's static private, responder's static public) == DH(responder's static private, initiator's static public)`
  - `DH(initiator's ephemeral private, responder's ephemeral public) == DH(responder's ephemeral private, initiator's ephemeral public)`
  - `DH(initiator's static private, responder's ephemeral public) == DH(responder's ephemeral private, initiator's static public)`
- 上記から、それらは対称の認証された暗号セッション鍵を得ることができる—1つは送信用、1つは受信用。
- イニシエーターがそれらの鍵を使って最初のメッセージを送信するとき、レスポンダはイニシエーターが応答メッセージを理解したという確証を受け取った後、データをイニシエーターに送ることができる。

鍵交換

- 暗号は少し複雑にしたかもしれない、しかし、とても限定的な一まとまりの基本操作を使う:
  - 楕円曲線Diffie–Hellman、ハッシング、認証された暗号。
- そして、まだちょうど1–RTT。
- 実際に実装を行うのが非常に簡単であり、OpenSSLやStrongSwanにあるような複雑な混乱はきたさない。
- X.509やASN.1の証明書は必要とせず:まさにSSHと同様に、両者は非常に短い(32バイト)base64にコード化された公開鍵を交換する。

```
zx2c4@thinkpad WireGuard/src $ cloc noise.c
-------------------------------------------------------------------------------
Language     blank       comment           code
-------------------------------------------------------------------------------
C               87            39            441
-------------------------------------------------------------------------------
```

タイマー：ステートフルプロトコルのためのステートレスインターフェース

- 先に述べたように、WireGuardはユーザスペース（あなたが自身のために設定したピアとその動作）にはステートレスを表現する。
- 一連のタイマーは内部でユーザには見せないようにしてセッション状態を管理する。
- すべての状態機械の遷移は説明されている、だから、未定義状態に遷移することはない。
- イベントベース。

タイマー

ユーザスペースがパケットを送る。	・もし、120秒以内にセッションが確立されなければ、ハンドシェイクのイニシエーションを送信する。
5秒後にハンドシェイクの応答がない。	・ハンドシェイクイニシエーションの再送する。
入ってくるパケットの認証に成功	・もし、我々がその時間の間に送信するべきものを持っていなければ、10秒後に暗号化された空のパケットを送信する。
15秒後に入ってくるパケットの認証の成功はない。	・ハンドシェイクイニシエーションを送信する。

セキュリティの原則６：サービス拒否攻撃の耐性

- ハッシュ化と対称暗号は速い、しかし、公開鍵暗号は遅い。
- 我々は楕円曲線Diffie-Hellman (ECDH)に最も速い曲線のCurve25519を使用する。
- DH()を計算することを要求する機械を圧倒する。
  - OpenVPNの脆弱性！
- UDPはこの困難を作り出す。
- WireGuardは “cookies”を使うことでこの問題を解決する。

Cookies: TCP-like

- 対話:
  - イニシエーター:このDH()を計算する。
  - レスポンダ:あなたの呪文は”karaage”である。呪文を付けて再び私に尋ねよ。
  - イニシエーター:私の呪文は”karaage”である。このDH()を計算せよ。
- IPアドレスの所有者を証明する。しかし、状態を保存することなしにIPアドレスの(送受信)割合を制限することはできない。
  - セキュリティ設計の原則に違反、動的割当てはしない！
- いつでもメッセージに応答する。
  - セキュリティ設計の原則に違反、ステルス性！
- 呪文を遮ることができる。

Cookies: DTLS-likeとIKEv2-like

- 対話:
  - イニシエーターはこのDH()を計算する。
  - レスポンダ:あなたの呪文は“cbdd7c…bb71d9c0”である。呪文を付けて再び私に尋ねなさい。
  - イニシエーター:私の呪文は”cbdd7c…bb71d9c0”である。このDH()を計算せよ。
- “cbdd7c…bb71d9c0” == MAC(レスポンダ秘密鍵, イニシエーターのIPアドレス)
  ここで、レスポンダ秘密鍵は2, 3分ごとに更新される。
- IPアドレスの所有者を状態保存なしに証明する。
- いつでもメッセージに応答する。
  - セキュリティ設計の原則に違反:ステルス性!
- 呪文は遮ることができる。
- 大量の偽物の呪文によってイニシエーターをDoS状態にできる。

Cookies: HIPv2-likeとBitcoin-like

- 対話:
  - イニシエーター:このDH()を計算せよ。
  - レスポンダ:ビットコインを採掘し、私に尋ねよ！
  - イニシエーター:私は一生懸命働いてビットコインを見つけた。このDH()を計算せよ。
- 作業の証明。
- もし、パズルがDH()よりも難しければ、DoSとの戦いに頑健である。
- しかしながら、イニシエーターとレスポンダはCPUのオーバヘッドを招くことなく役割を交代することができない。
  - そのクライアントごとに作業の証明をしなければならないサーバを想像せよ。

Cookies: WireGuardの変形

- 各ハンドシェイクメッセージ（イニシエーターとレスポンダ）は2つのmacを持つ：mac1とmac2。
- mac1は以下のように計算される：
  `HASH(responder_public_key || handshake_message)`
  - もしこのmacが不正また消失なら、メッセージは無視されるだろう。
  - 応答を引き出すために、イニシエーターがレスポンダのイデンティティ鍵を知らなければならないことを確実にする。
    - ステルス性を確実にする—セキュリティ設計の原則。
- もしレスポンダが負荷状態(DoS攻撃を受けていない)になければ、普通に続行する。
- もしレスポンダが負荷状態(DoS攻撃を受けている)なら…

Cookies: WireGuardの変形

- もしレスポンダが負荷状態(DoS攻撃を受けている)なら、以下の計算をしたcookieを付加したものを返す:

```
AEAD(
   key=HASH(responder_public_key || salt),
   additional_data=handshake_message,
   MAC(key=responder_secret, initiator_ip_address)
)
```

- mac2は以下のように計算される:

```
MAC(key=cookie, handshake_message)
```

  - もしそれが妥当なら、負荷時であってもメッセージは続行される。

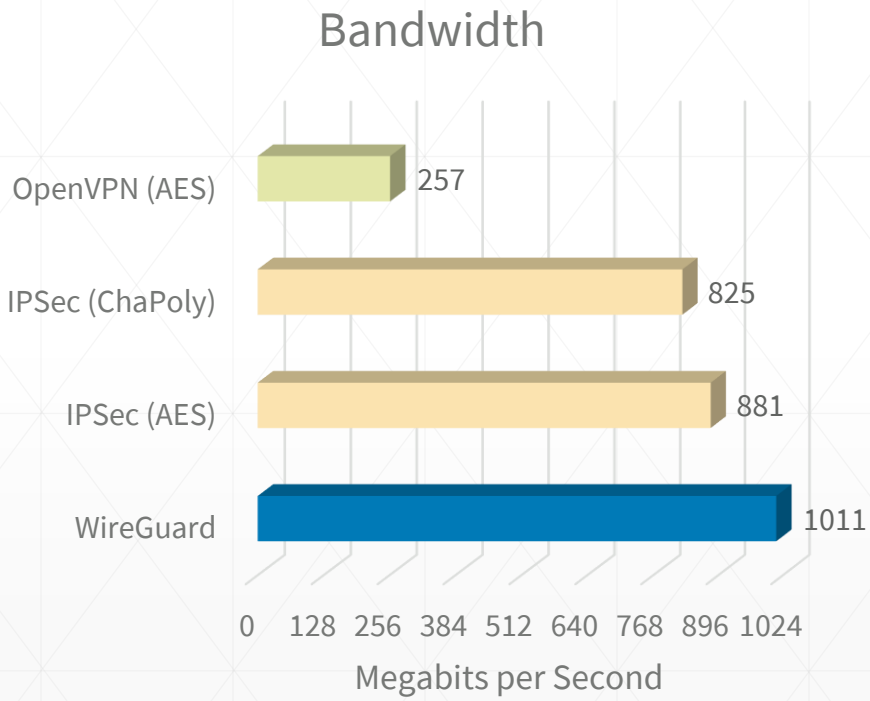
Cookies: WireGuardの変形

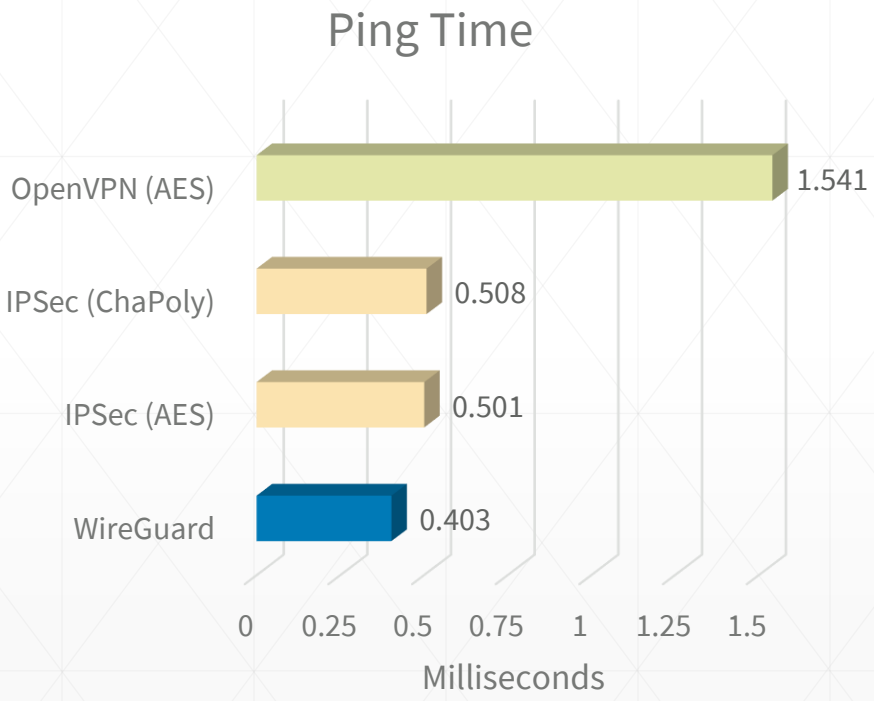
- IPアドレスがアトリビュートされると、通常のトークンバケット率の制限が適用される。
- ステルス状態は持続する。
- Cookieは、すでに同様のエクスチェンジが行えない他者により傍受されることはない。
- 暗号化されたcookieは元のハンドシェークメッセージを「追加データ」パラメータとして使うため、イニシエーターはDoS攻撃を食らわない。
  - いずれにせよ、攻撃者はDoS攻撃を実施するためのMITMポジションをすでに持たなければならない。

パフォーマンス

- カーネルスペースに置かれることは速くて低遅延であることを意味する。
  - ユーザスペースとカーネルスペースに2重のパケットコピーをする必要がない。
- ChaCha20Poly1305はほとんど全てのハードウェアでとても速く、安全である。
  - AES-NIも明らかに速い。しかし、IntelとARMのベクトル演算がより幅広になり、場合によってはChaChaはAES-NIと争ってもより良く実行できる。
  - 特別なハードウェアなしにAESは安全(キャッシュタイミング攻撃を避ける)に、かつ、効率的に実装することが極めて難しい。
  - ChaCha20はほとんどの一般的な目的のプロセッサに効果的に実装可能である。
- WireGuardの簡潔な設計はオーバヘッドを減らすことを、従って良いパフォーマンスを得ることを意味する。
  - 少ないコード→速いプログラム？いつでもじゃない、しかし、このケースでは確かである。

パフォーマンス：測定


Bandwidth
OpenVPN (AES)
257
IPSec (ChaPoly)
825
IPSec (AES)
881
WireGuard
1011
0 128 256 384 512 640 768 896 1024
Megabits per Second


Ping Time
OpenVPN (AES)
1.541
IPSec (ChaPoly)
0.508
IPSec (AES)
0.501
WireGuard
0.403
0 0.25 0.5 0.75 1 1.25 1.5
Milliseconds

再び：WireGuardは簡潔で、速くて、安全である。

- 4000行未満のコード。
  - 部屋にいるすべての人が簡単に監査できる。
- 基本的なデータ構造で簡単に実装されている。
- WireGuardの設計は実際に安全なコーディングパターンに適合している。
- 最小の状態は維持し、動的割当てはない。
- ステルス性と最小のアタックサーフェース。
- 頑丈な暗号の基礎。
- 安全なトンネルの基本的な特性：ピアとピアのIPの関係。
- とても効率的－クラス最高。
- 普通のネットワークデバイスによる簡潔な標準インターフェース
- 頑固に。

Demo

More Information

WireGuard

- Main website:
  www.wireguard.io
- Source code:
  $ git clone https://git.zx2c4.com/WireGuard
- Mailing list:
  lists.zx2c4.com/mailman/listinfo/wireguard
  wireguard@lists.zx2c4.com

Jason Donenfeld

- Personal website:
  www.zx2c4.com
- Company website:
  www.edgesecurity.com
- Email:
  Jason@zx2c4.com