



# INTRODUCING NOISESOCKET FOR OSSEC

**BY DMITRY DAIN**

CoFounder and CTO

Virgil Security, Inc.

@dmitrydain

# OSSEC CURRENT STATE

You need to reboot every time you add or delete an agent

Agent keys are symmetric keys + counters (what happens when a counter gets corrupted?)

These keys are being copy/pasted and moved around by humans which is insecure

Old Crypto

No Scalable Key management

Hard to provision for IoT devices.  
Scaling is difficult

# WHAT IS NOISESOCKET PROTOCOL?

Simple wire protocol,  
minimal overhead

Built-in padding

Extensible

Ideal for IoT: Sensor Networks,  
Smart Home, Automotive etc.

Perfect for MicroServices  
IoT, Distributed Networks

Developed by Alexey Ermishkin and  
Trevor Perrin

# WHY NOISESOCKET > TLS?

X.509 MUST DIE

Make every TCP Connection  
Secure

Fast crypto primitives by default

Advanced options like 0-RTT, PFS,  
pre-shared keys.

TLS does not support raw public keys.  
Bringing up CA correctly is not something you'd  
like to deal with

Key Management is built-in and  
simple to use

# NOISESOCKET: ADVANCING NOISE FRAMEWORK

Developed by Alexey Ermishkin and Trevor Perrin, a co-author of Signal messenger

Allows arbitrary data being hashed into handshake state (Prologue)

Supports mutual authentication & 0-RTT

Fast

Long-term static keys

Ephemeral keys for perfect forward secrecy

Uses only DH, AEAD (AES-GCM, ChachaPoly), HKDF & Hash (SHA-2, Blake2)

## NOISESOCKET

NoiseSocket is a fundamental building block to start the process of transitioning away from the certificate authorities and centralized trust authorities while simultaneously allowing every server, device, and application on the Internet to have cryptographically authenticated identity and encryption.

# NOISE: SIMPLE STATE MACHINE

## Cipher State

AEAD + nonce

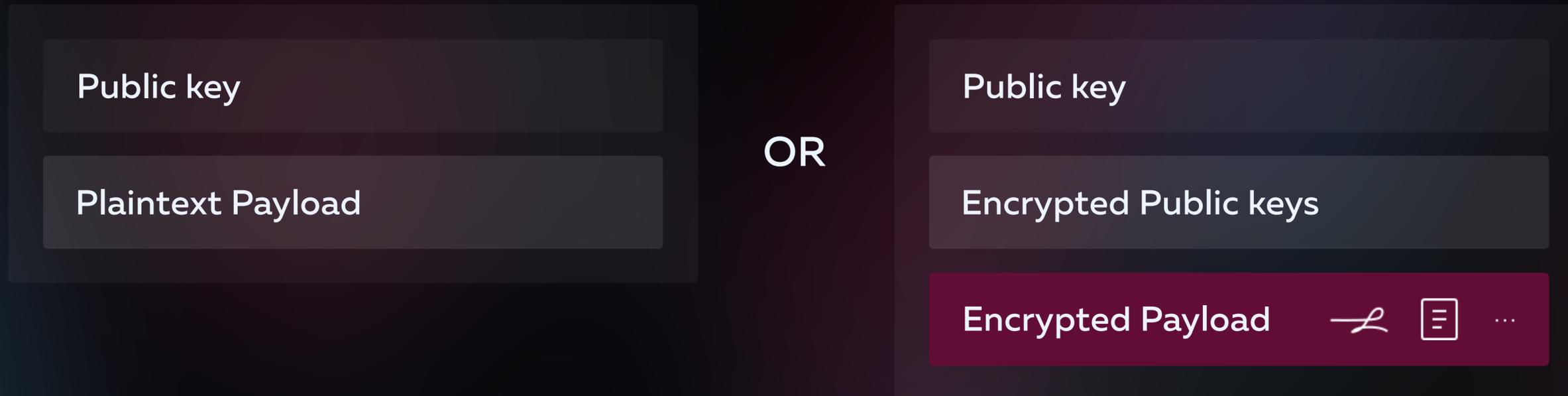
## Symmetric State

Key ratcheting, mixing data into state

## Handshake State

Patterns & tokens  
processing

# HANDSHAKE MESSAGES



- Each message consists of 1 or more public keys + optional payload
- After at least 1 DH all the following data (keys, payload) is encrypted
- New symmetric key after each DH
- Payload may contain signs/certificates/etc. Each message is 64kb max

# NOISESOCKET: HANDSHAKE MESSAGES

2 BYTES LENGTH

## NEGOTIATION DATA

2 bytes: Version

1 byte: Cipher

1 byte: Pattern

1 byte: DH

1 byte: Hash

2 BYTES LENGTH

## NOISE MESSAGE

Negotiation data and its length goes into Noise Prologue

# NOISESOCKET: TRANSPORT MESSAGE STRUCTURE

2 BYTE PACKET LENGTH

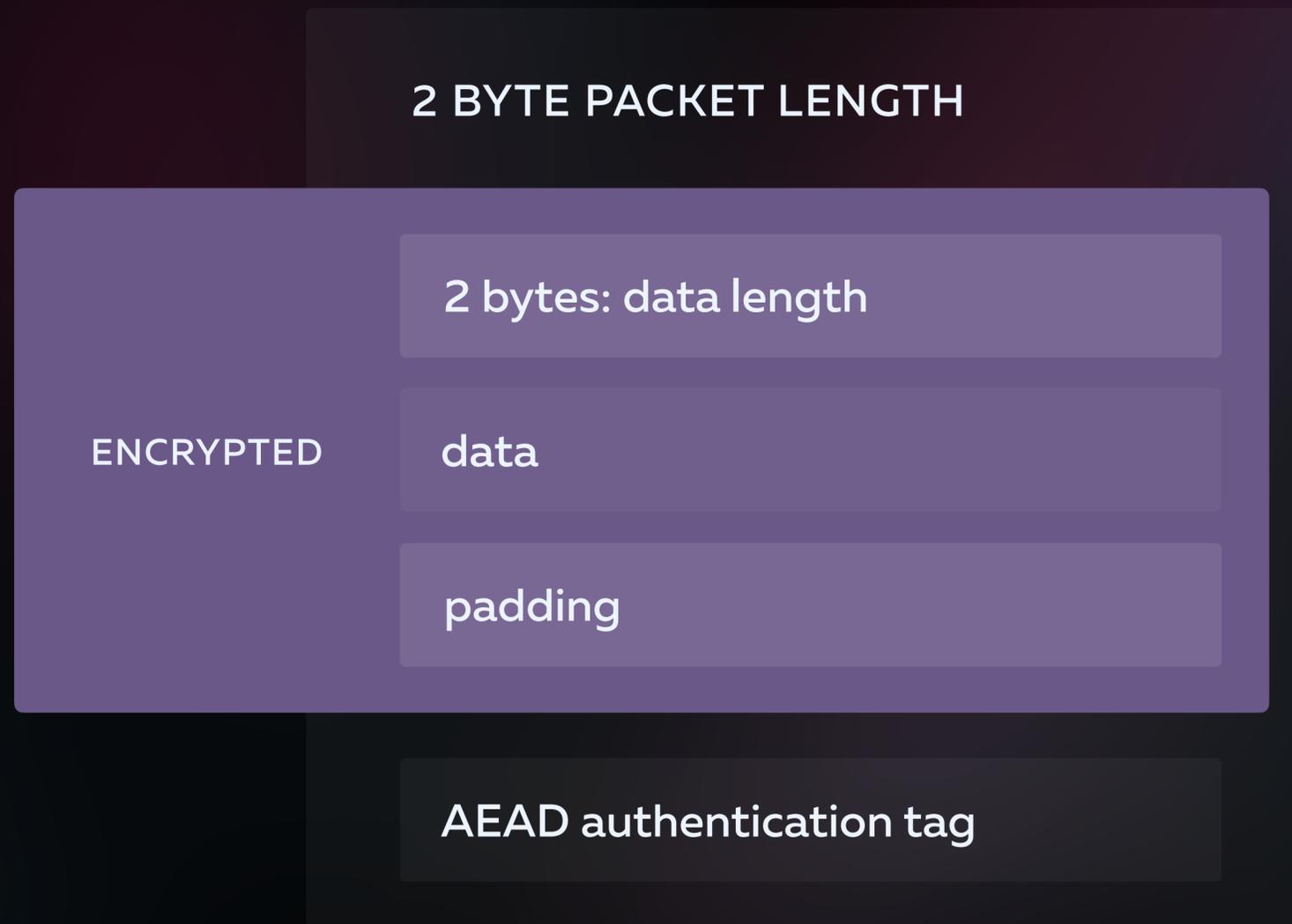
2 bytes: data length

data

padding

AEAD authentication tag

# TRANSPORT MESSAGE STRUCTURE



If no padding is used, data length = length of actual data

# HTTP OVER NOISE IN GO LANGUAGE

## SERVER

```
server := &http.Server{...}  
1, err:= noisesocket.Listen(":12888", serverKeys);  
server.Serve(1);
```

## CLIENT

```
transport := &http.Transport{  
    DialTLS: func(network, addr string) (net.Conn, error) {  
        return noisesocket.Dial(addr, clientKeys, nil)  
    },  
}  
cli := &http.Client{Transport: transport,}  
req, err := http.NewRequest("POST", "https://localhost:12888/...", body)  
resp, err := cli.Do(req)
```

# SIMPLE!

# NOISESOCKET: USE CASES

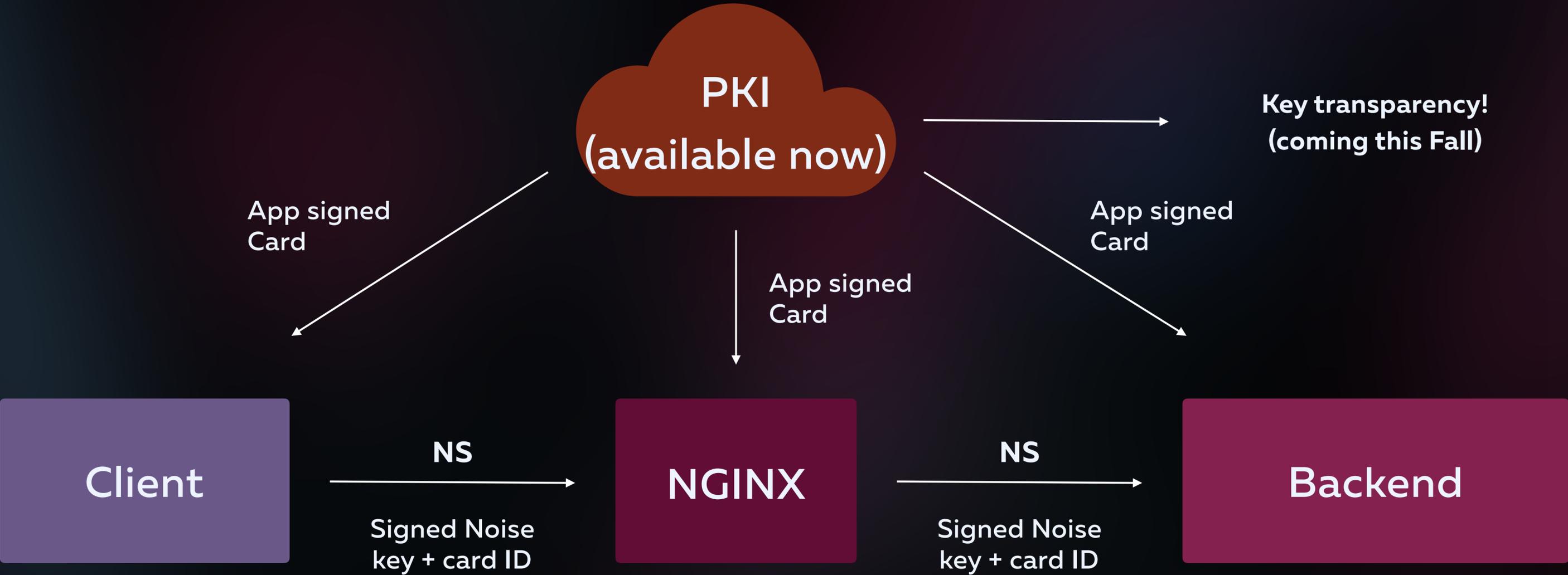


NoiseSocket is natural for p2p where NodeID is often a public key. No need for signatures/etc



We have a Go library for that!

# DISTRIBUTED VERIFIABLE TRUST: VIRGIL USE CASE



Each node verifies key signature & card's app signature to build TRUST



# BENCHMARKS



KEEP ALIVE OFF

TLS: 90 RPS

NoiseSocket: 261 RPS

# BENCHMARKS



KEEP ALIVE OFF

TLS: 820.74 RPS

NoiseSocket: 1263.33 RPS

# NOISESOCKET: CODE SIZE

## ARM:

BLAKE2s: 1800 bytes

ChaChaPoly: 2700 bytes

Curve25519: 10200 bytes

Ed25519: 10200 bytes

RNG: 1300 bytes

NoiseSocket: 3000-5000 bytes

Total: ~26kb

## AVM:

BLAKE2s: 1028 bytes

ChaChaPoly: 2024 bytes

Curve25519: 10240 bytes

Ed25519: 10240 bytes

RNG: 2040 bytes

NoiseSocket: 3000-5000 bytes

Total: ~30kb

# OSSEC 3 REPORT CARD

You need to reboot every time you add or delete an agent ✓

Agent keys are symmetric keys + counters (what happens when a counter gets corrupted?) ✓

These keys are being copy/pasted and moved around by humans which is insecure ✓

Old Crypto ✓

No Scalable Key management ✓

Hard to provision for IoT devices. Scaling is difficult ✓

## What's Next

- Server registration (Use additional utility + possibly HSM)
- Need to decide what Network backend we should use.  
(libuv - fast but a lot of changes to OSSEC code,  
or Noisesocket over OSSEC networking)
- OSSEC 3 Release support



# THANK YOU!

## RESOURCES

[github.com/noisesocket/spec](https://github.com/noisesocket/spec)

[VirgilSecurity.com](https://www.virgilsecurity.com)

[noiseprotocol.com](https://noiseprotocol.com)

Special thanks to Rhys Weatherley and David Wong for helpful  
discussion