

Authenticating with an NKey

The 2.0 version of NATS server introduces a new challenge response authentication option. This challenge response is based on a wrapper we call [NKeys](#). The server can use these keys in several ways for authentication. The simplest is for the server to be configured with a list of known public keys and for the clients to respond to the challenge by signing it with its private key. (A printable private NKey is referred to as seed). This challenge-response ensures security by ensuring that the client has the private key, but also protects the private key from the server, which never has access to it!

Handling challenge response may require more than just a setting in the connection options, depending on the client library.

Go

```
opt, err := nats.NkeyOptionFromSeed("seed.txt")
if err != nil {
    log.Fatal(err)
}
nc, err := nats.Connect("127.0.0.1", opt)
if err != nil {
    log.Fatal(err)
}
defer nc.Close()

// Do something with the connection
```

Java

```
NKey theNKey = NKey.createUser(null); // really should load from somewhere
Options options = new Options.Builder()
    .server("nats://localhost:4222")
    .authHandler(new AuthHandler(){
        public char[] getID() {
            try {
                return theNKey.getPublicKey();
            } catch (GeneralSecurityException|IOException|NullPointerException ex) {
                return null;
            }
        }
    })
    .build();
```

```

    }

    public byte[] sign(byte[] nonce) {
        try {
            return theNKey.sign(nonce);
        } catch (GeneralSecurityException|IOException|NullPointerException ex) {
            return null;
        }
    }

    public char[] getJWT() {
        return null;
    }
})
.build();
Connection nc = Nats.connect(options);

// Do something with the connection

nc.close();

```

JavaScript

```

// seed should be stored and treated like a secret
const seed = new TextEncoder().encode(
    "SUAEL6GG2L2HIF7DUGZJGMRUFKXELGGYFMHF76U02AYBG3K4YLWR3FKC2Q",
);
const nc = await connect({
    port: ns.port,
    authenticator: nkeyAuthenticator(seed),
});

```

Python

```

nc = NATS()

async def error_cb(e):
    print("Error:", e)

await nc.connect("nats://localhost:4222",

```

```

        nkeys_seed="./path/to/nkeys/user.nk",
        error_cb=error_cb,
    )

# Do something with the connection

await nc.close()

```

C#

```

// dotnet add package NATS.Net
using NATS.Net;
using NATS.Client.Core;

await using var client = new NatsClient(new NatsOpts
{
    Url = "127.0.0.1",
    Name = "API NKey Example",
    AuthOpts = new NatsAuthOpts
    {
        NKeyFile = "/path/to/nkeys/user.nk"
    }
});

```

C

```

static natsStatus
sigHandler(
    char          **customErrTxt,
    unsigned char **signature,
    int           *signatureLength,
    const char    *nonce,
    void          *closure)
{
    // Sign the given `nonce` and return the signature as `signature`.
    // This needs to allocate memory. The length of the signature is
    // returned as `signatureLength`.
    // If an error occurs the user can return specific error text through
    // `customErrTxt`. The library will free this pointer.

    return NATS_OK;
}

```

```

}

(...)

natsConnection      *conn      = NULL;
natsOptions          *opts      = NULL;
natsStatus           s          = NATS_OK;
const char           *pubKey     = "my public key.....";

s = natsOptions_Create(&opts);
if (s == NATS_OK)
    s = natsOptions_SetNKey(opts, pubKey, sigHandler, NULL);
if (s == NATS_OK)
    s = natsConnection_Connect(&conn, opts);

(...)

// Destroy objects that were created
natsConnection_Destroy(conn);
natsOptions_Destroy(opts);

```

📄 #3

📄 cocobook-sync 📄 2022-01-19 01:57:33 UTC

📄 cocobook-sync 📄 2024-10-28 14:13:14 UTC