
TOZNY
Application-Layer Encryption basics for Developers
jjones@tozny.com
@syntaxpolice
May 20, 2021

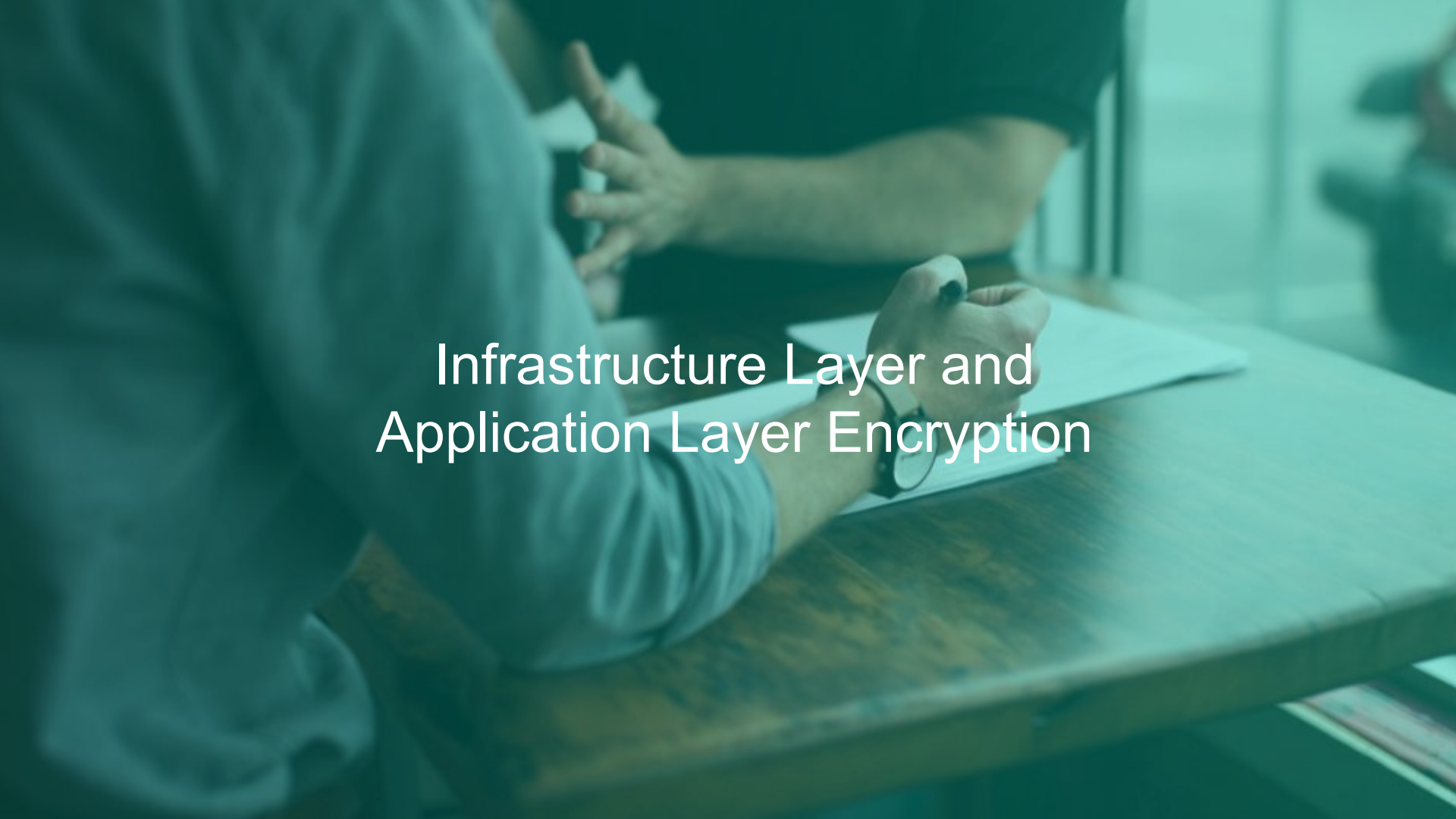
What You Will Get from this Talk

- How to choose between application-layer and infrastructure-layer encryption
- When to use asymmetric and symmetric encryption
- What to do with your keys
- A simple code example
- What this leaves out
- A teaser about quantum computing

You’re going to learn enough to be dangerous

But hopefully enough to get excited

PSA: Don’t roll your own crypto


Infrastructure Layer and
Application Layer Encryption

When We Think of Driving Safety...

… We address both the road and the car

Securing the Road

- Stoplights
- Speed limits
- Gentle curves
- Lines on the road
- No passing zones

Encrypting The Infrastructure

- HTTPS / TLS
- VPNs, IPSEC
- Service Mesh
- Full Disk Encryption
- DB Encryption

Securing the Car

- Seatbelts
- Crumple zones
- Airbags
- Horns
- (Better driving)

Securing the Application

- Malware
- Buffer Overflows
- Side Channels
- Broken Authentication
- (Better programming)
- (Not much crypto…)

Do More Application-Layer Encryption

- Use application-layer encryption when:
  - The security should travel with the data
  - You are working across infrastructures
  - You need an extra layer of protection
  - You need to enforce access control with encryption
- It improves privacy, in some cases substantially
- But it’s harder for developers than just implementing HTTPS
- Easy to use encryption is what we do, so look us up
  - See my previous QCon talk or my “developer’s guide to encryption”
  - https://tozny.com/blog/encryption-for-developers/
  - https://www.infoq.com/presentations/encryption-pros-cons

Symmetric and Asymmetric Encryption

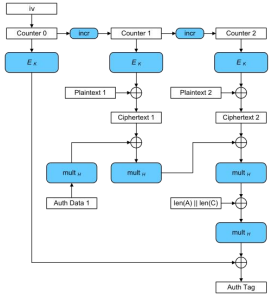
Basic Terminology

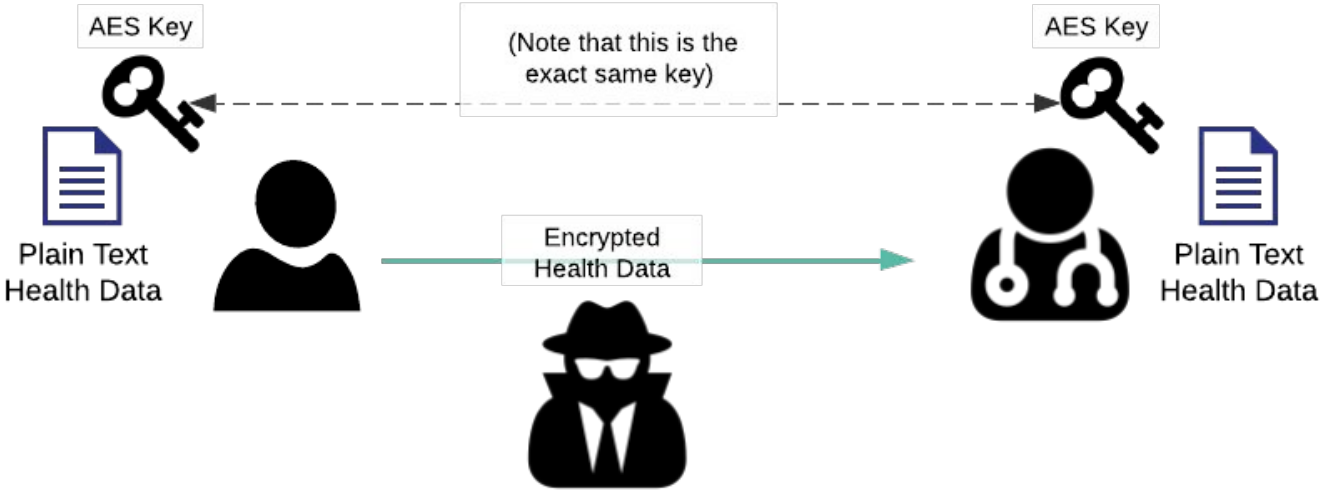
- **Encrypt**: Use a key to make something secret
  - Inputs “plain text” - Outputs “ciphertext”
- **Decrypt**: Use a key to undo encryption and make something readable
  - Inputs “ciphertext” - Outputs “plain text”
- **Sign**: Use a key to prove “integrity”
  - Inputs a THING - Outputs a signature for the THING
- **Verify**: Use a key to check a signature
  - Inputs a THING and its signature - Outputs true only if it matches

Caveat: Because this is a very short talk, I’ll use “sign” and “verify” generally, and not go into details about hashing, tags, DH key exchange, etc.

Symmetric “Bulk” Encryption: AES GCM

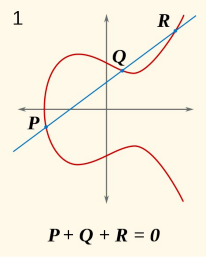
- It’s called “symmetric” because: the same key is used for:
  - Encryption, Decryption, “Signing”, Verification
  - But you need a way to share the key!

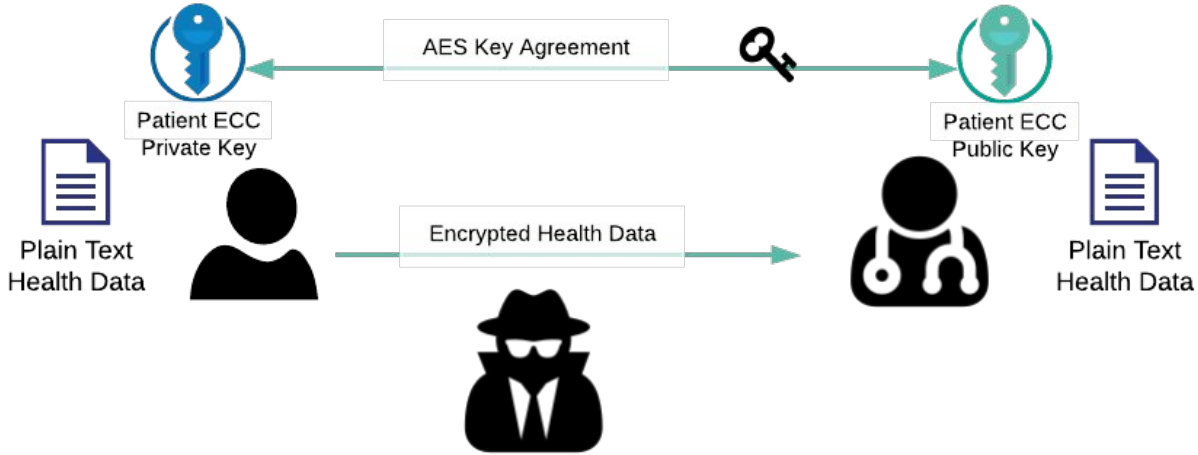




Asymmetric Encryption (PKI): ECC

- It’s called “Asymmetric” because there different keys
  - The “public” part and the “private” part
  - This is usually used for “key agreement” for the symmetric (AES) key



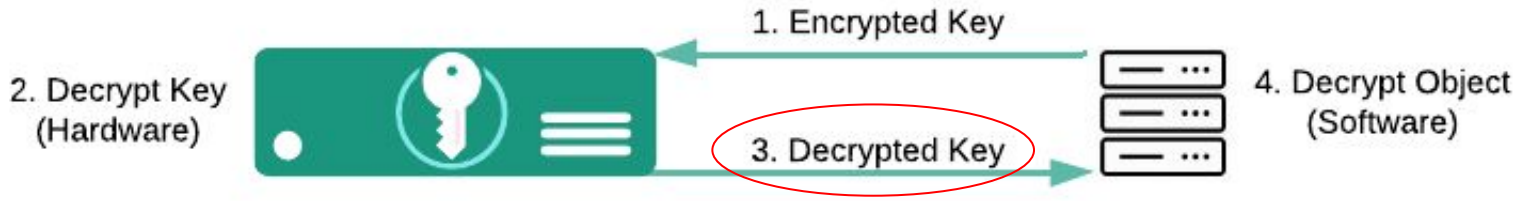


But what do you do with your keys?

- Symmetric keys: Encrypt / exchange them using asymmetric keys
- Asymmetric public key: This is tricky but we won't go into detail!
  - Give it away freely in a way people know it comes from you
  - Certificate Authority, CSR, proves you hold the private key
- Asymmetric private keys: This is the tricky one!
  - Store them in a file encrypted with a password (the password generates a symmetric key!)
  - Hardware Security: e.g. rack-mounted, PIV, or Yubikey-style custom purpose hardware
  - "Keychain" in your OS: e.g. protected with screen lock or biometric (iOS / Android)
  - Secrets managers: e.g. 1Password, Vault, CredStash;

Hardware Key Management

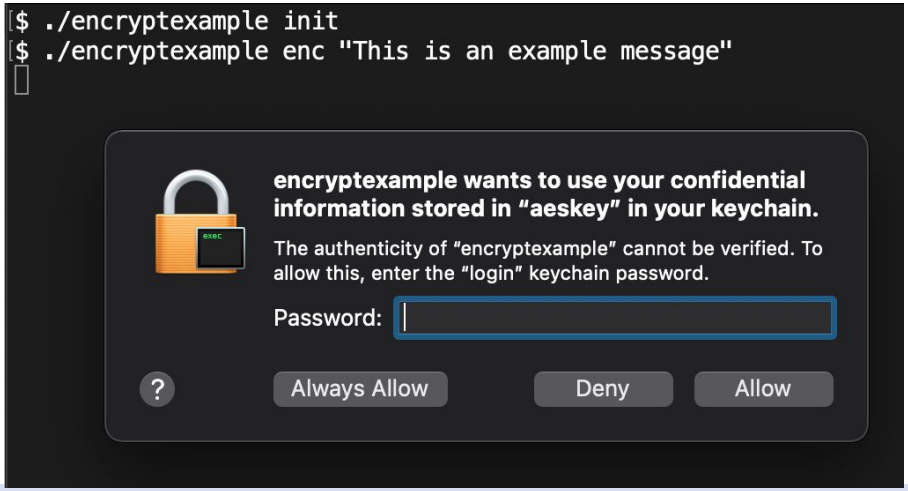
- Encrypt a wrapped secret
  - Perform encryption operations in software
  - Key is visible outside



- Perform encryption operations in hardware
  - Key is not visible outside
  - Can be more expensive

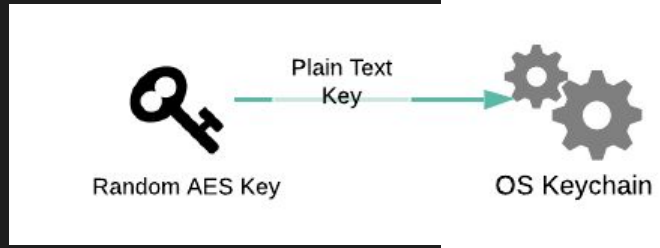

A Simple Code Example

```
$ ./encryptexample init
$ ./encryptexample enc "This is an example message"
Ciphertext: 8960c26a0f59c5f53d50a11bf577157cd6156753a4e4b85a8c205d60dfd20272dd3dd574f1dc22c9298e
None: be73c5ea1739e46a1eb08960
$ ./encryptexample enc "This is an example message"
Ciphertext: c082e774dbcc243018ae615aae603ae9109512a812f82b12a6daf7b474bc2662776b26846e6f0bbf6e1e
None: 9611eacc21eeee58cb4e8c7f
$ ./encryptexample dec c082e774dbcc243018ae615aae603ae9109512a812f82b12a6daf7b474bc2662776b26846e6f0bbf6e1e 9611eacc21eeee58cb4e8c7f
This is an example message
$
```



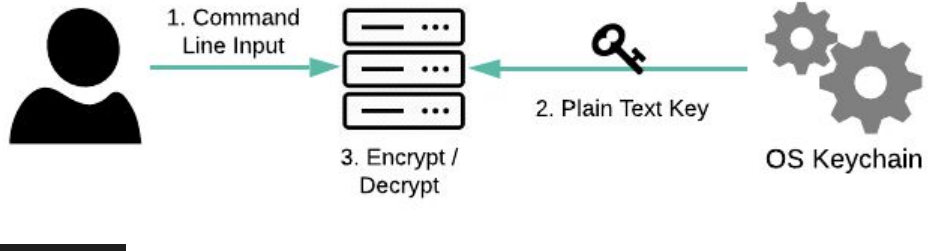
Code Example: Init


```go
func storeKey(keyToStore []byte) error {
    item := keychain.NewGenericPassword(service, account, label, keyToStore, accessGroup)
    item.SetSynchronizable(keychain.SynchronizableNo)
    item.SetAccessible(keychain.AccessibleWhenUnlocked)
    err := keychain.AddItem(item)
    if err != keychain.ErrorDuplicateItem {
        // error except duplicate
        return err
    }
    return nil
}

func initKey() error {
    key := make([]byte, 32)
    if _, err := io.ReadFull(rand.Reader, key); err != nil {
        panic(err.Error())
    }
    return storeKey(key)
}
```



Code Example - Encrypt / Decrypt


```go
func fetchKey() ([]byte, error) {
    key, err := keychain.GetGenericPassword(service, account, label, accessGroup)
    return key, err
}

func encrypt(key []byte, plaintextStr string) {
    plaintext := []byte(plaintextStr)

    block, err := aes.NewCipher(key)
    if err != nil {
        panic(err.Error())
    }

    // Never use more than 2^32 random nonces
    nonce := make([]byte, 12)
    if _, err := io.ReadFull(rand.Reader, nonce); err != nil {
        panic(err.Error())
    }

    aesgcm, err := cipher.NewGCM(block)
    if err != nil {
        panic(err.Error())
    }

    ciphertext := aesgcm.Seal(nil, nonce, plaintext, nil)
    fmt.Printf("Ciphertext: %x\n", ciphertext)
    fmt.Printf("None: %x\n", nonce)
}
```



Summary

What this leaves out

- This is a 20 minute talk: You now know enough to be dangerous
- Lots about Keys
  - What to do with the public key: Certificate Authorities
  - Secure key generation and random numbers
  - Choosing a key length, generating keys from passwords
- Encryption for privacy
- FIPS compliance, libSodium
- Hashing, password storage
- When to sign plaintext vs. ciphertext
- All the different versions of AES, Nonce, initialization vectors
- How crypto systems actually get attacked

A teaser about quantum computing

- Quantum computers might be able to break asymmetric encryption “soon”
  - That makes key exchange hard
- What does “soon” mean?
  - Depends on how long you want your data to be secret
  - And how long it takes to standardize and adopt a solution
- What are we going to do about this
  - NIST competition to find new algorithms
  - Use AES 256 and pre-shared keys
  - Or don’t worry about it!

Thank You!

Application-Layer Encryption basics for Developers

ijones@tozny.com
@syntaxpolice