Crypt-Oh No!

4/12/18

Zach Grace, OSCP, GREM, CISSP
@ztgrace

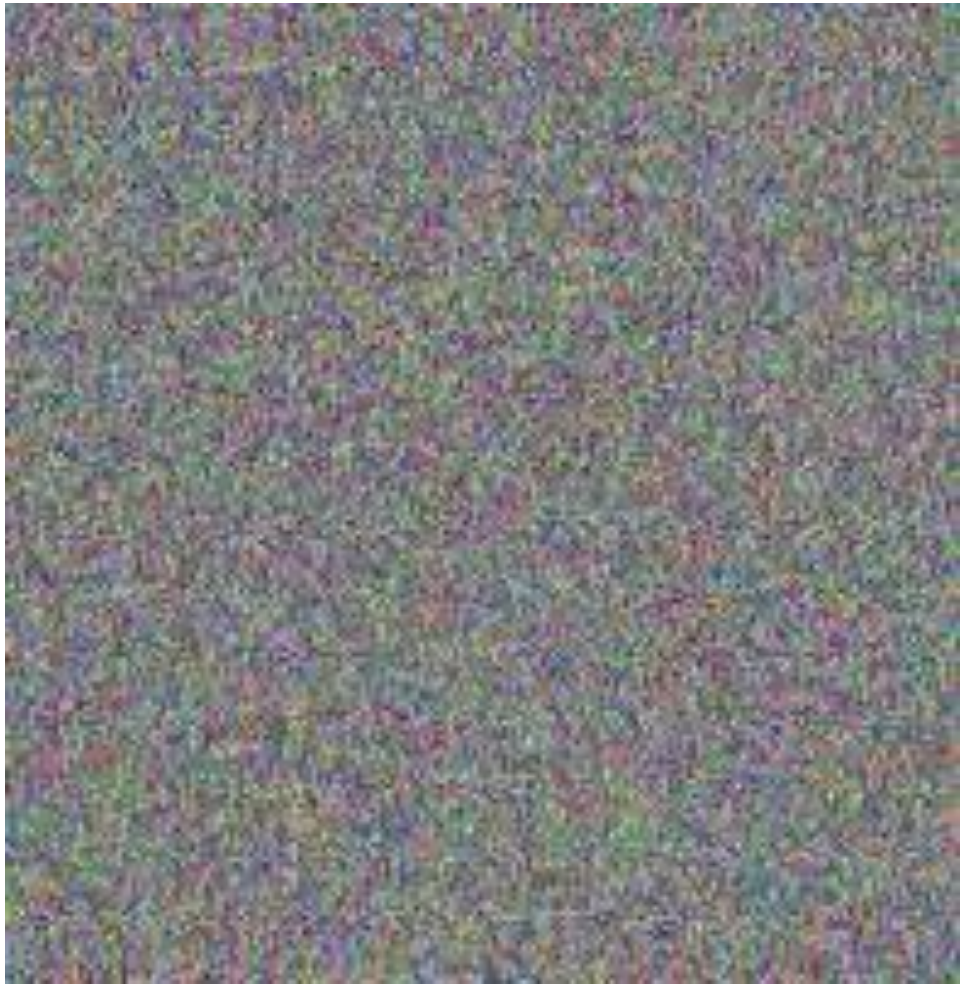
whoami

- Hacker @ Northwestern Mutual
  - Leads pen testing and vulnerability management teams
- Previously
  - 403 Labs/Sikich – Manager of Penetration Testing
  - Metavante/FIS
    - Pen Tester
    - Java Developer
    - Windows/Linux/Solaris/VMware Administrator

DISCLAIMER

This presentation represents my personal opinions and not the official position of NM.

CRYPTOGRAPHY INTENTIONS



CRYPTOGRAPHY REALITY

https://twitter.com/gsuberland/status/885086549620260864

AGENDA

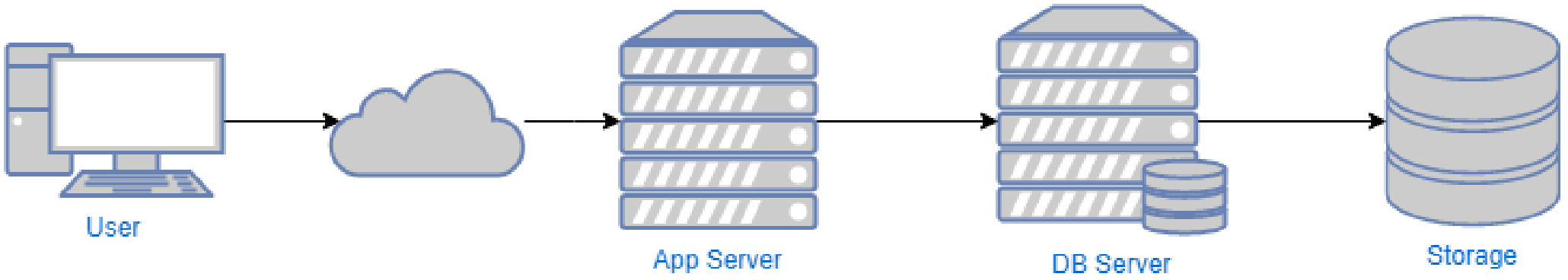
- CIPHER RECOMMENDATIONS
- THREAT MODELING
- STUDY 1: SUBTLE IMPLEMENTATION FLAWS
- STUDY 2: RANDOMNESS
- STUDY 3: ROLL YOUR OWN “ENCRYPTION”
- KEY MANAGEMENT
- RECOMMENDATIONS

CIPHER RECOMMENDATIONS

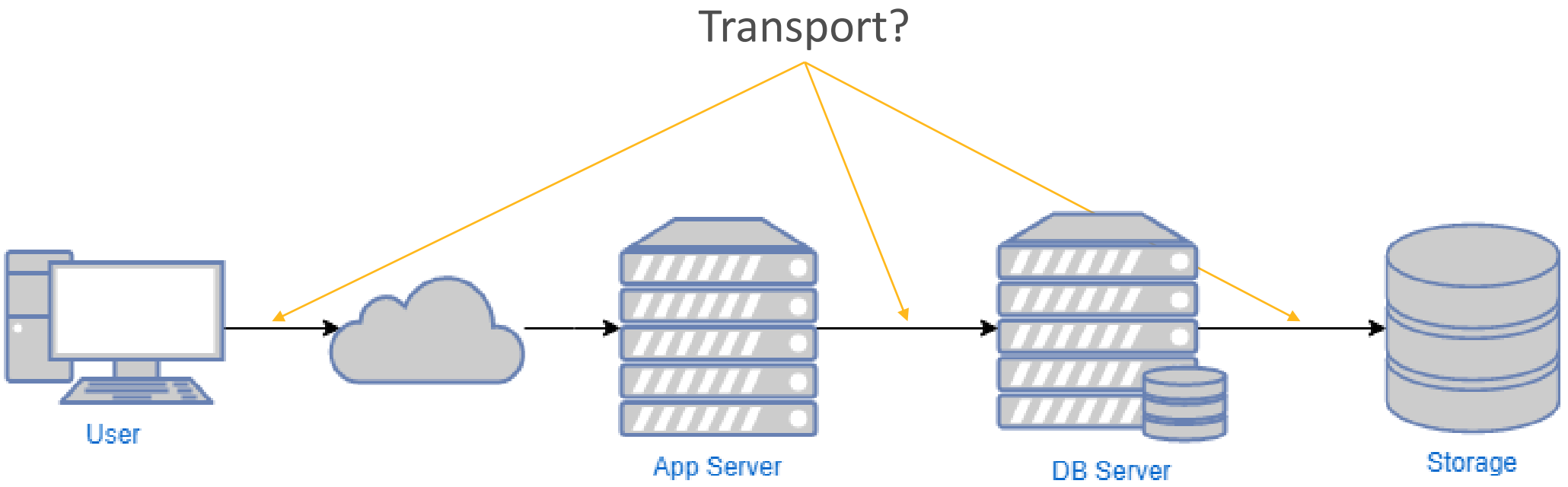
Please read the **Cryptographic Right Answers** for specific cipher recommendations:
http://latacora.singles/2018/04/03/cryptographic-right-answers.html

WHAT ARE YOU TRYING TO PROTECT?

BASIC WEB APP



WHERE TO ENCRYPT?



In Browser?

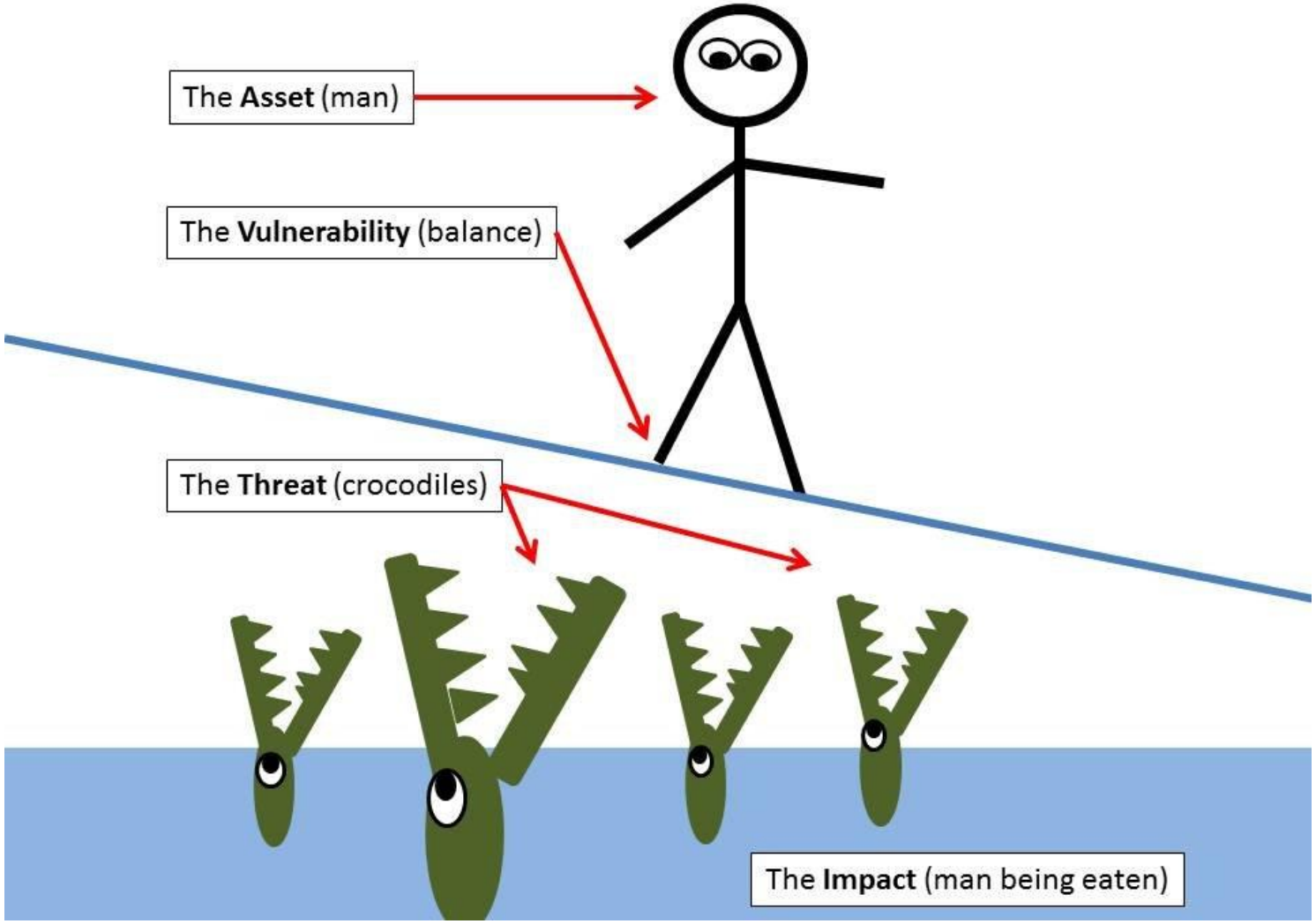
App-Level Encryption?

Database-Level Encryption?

Disk-Level Encryption?

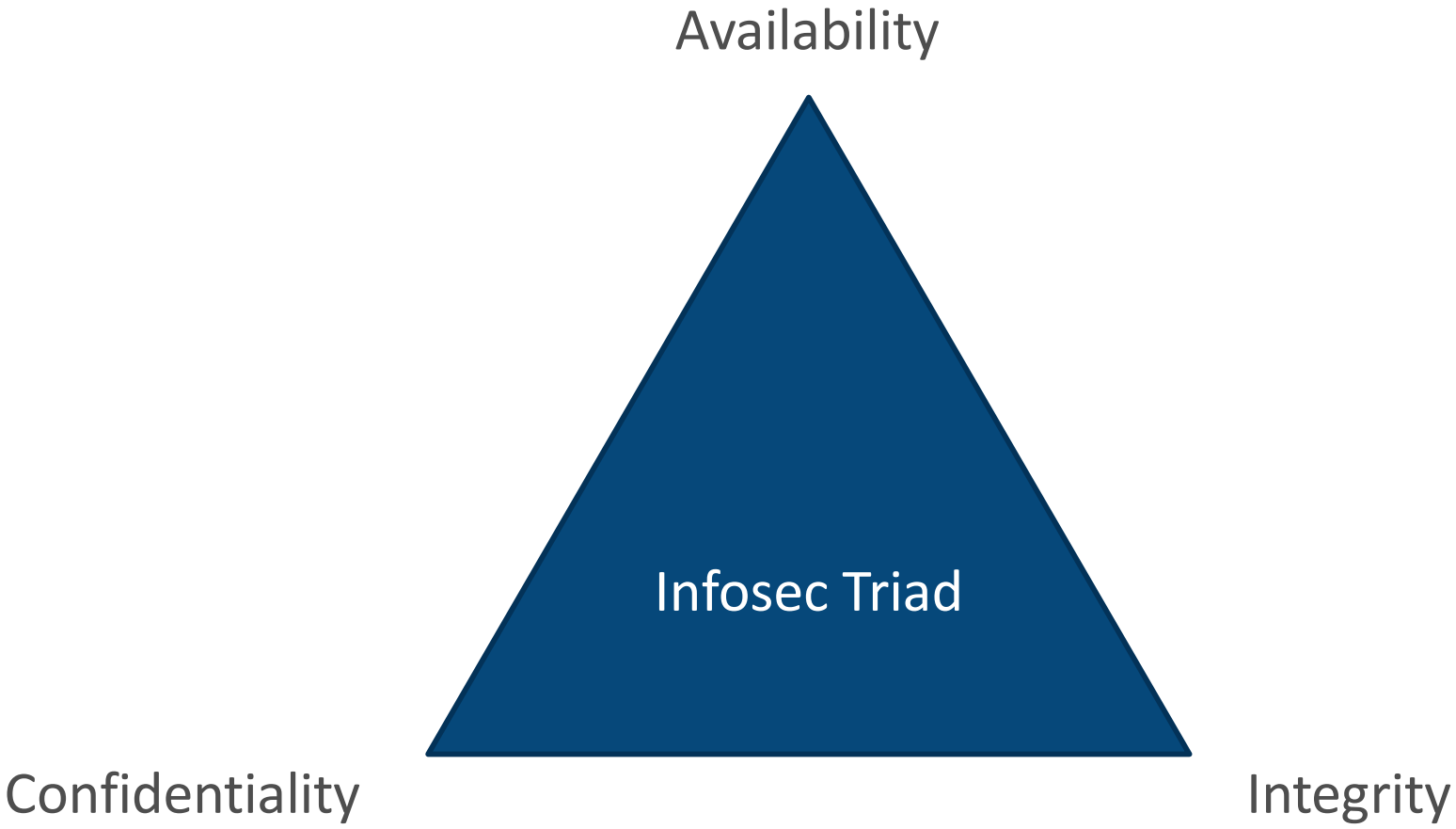
THREAT MODELING

THREAT MODELING



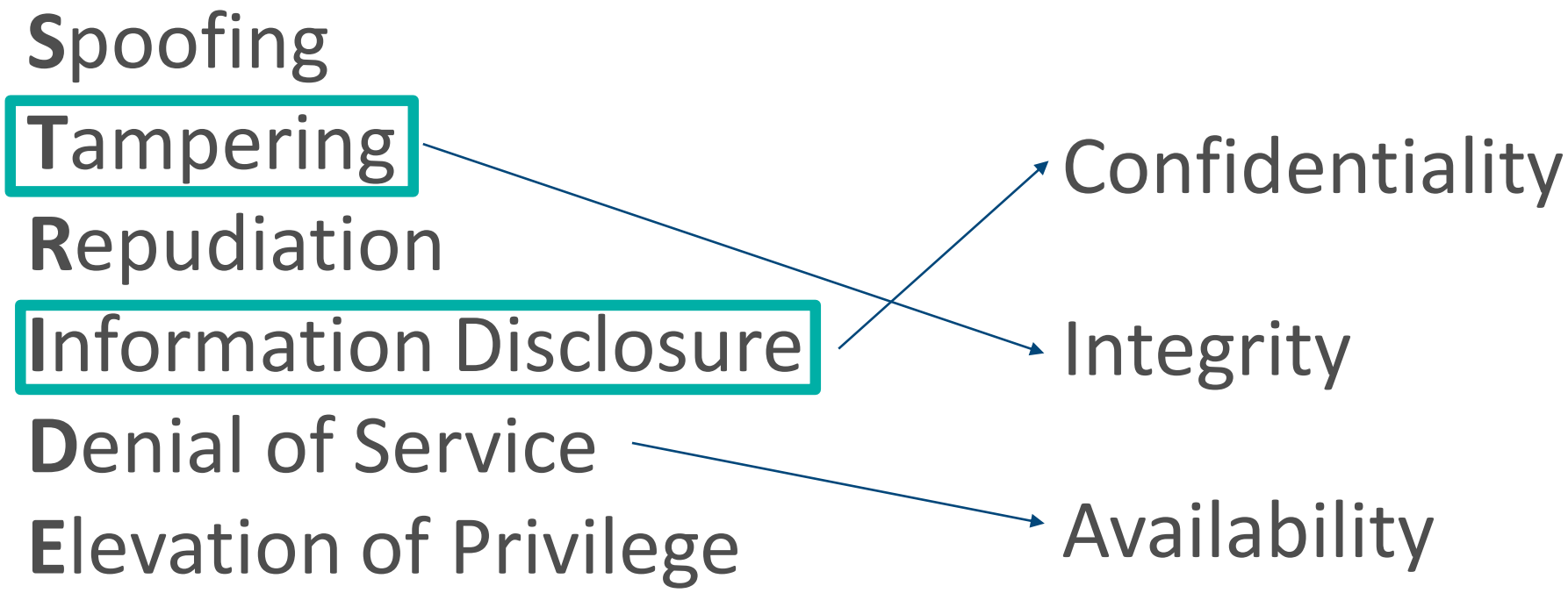
https://twitter.com/cybersecstu/status/978607013067481094?s=21

CIA TRIAD

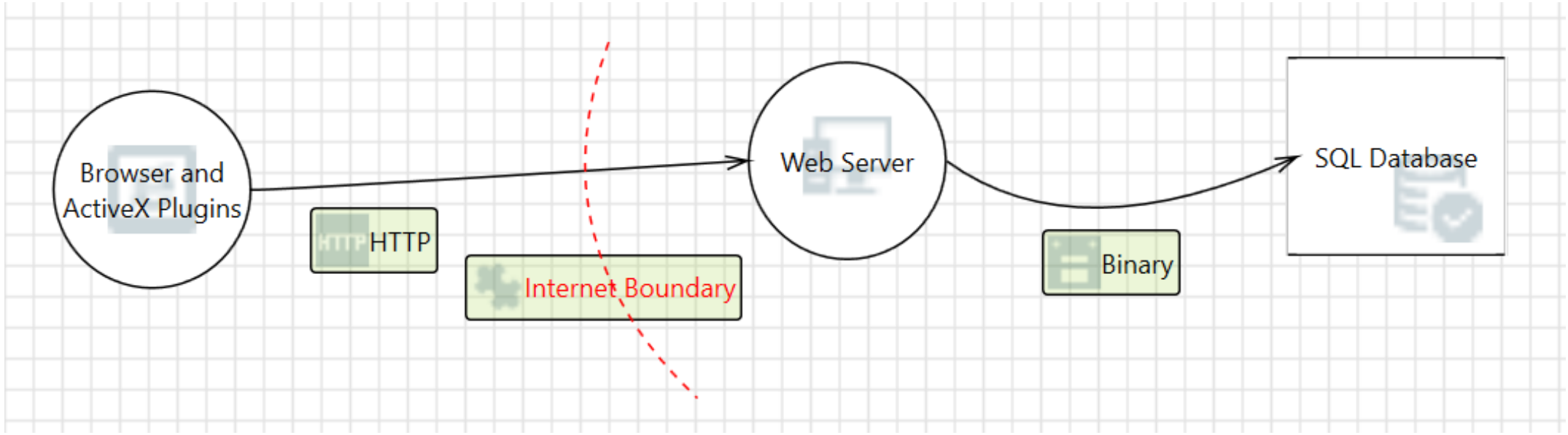


STRIDE

A threat classification model developed by Microsoft.



MICROSOFT'S THREAT MODELING TOOL



PARTIAL LIST OF THREATS

Threat	Category
Cross-Site Scripting	Tampering
Data Flow HTTP is Potentially Interrupted	Denial of Service
Data Flow Sniffing	Information Disclosure
Elevation by Changing the Execution Flow in Web Server	Elevation of Privilege
Elevation Using Impersonation	Elevation of Privilege
Potential Data Repudiation by Web Server	Repudiation
Potential SQL Injection Vulnerability for SQL Database	Tampering

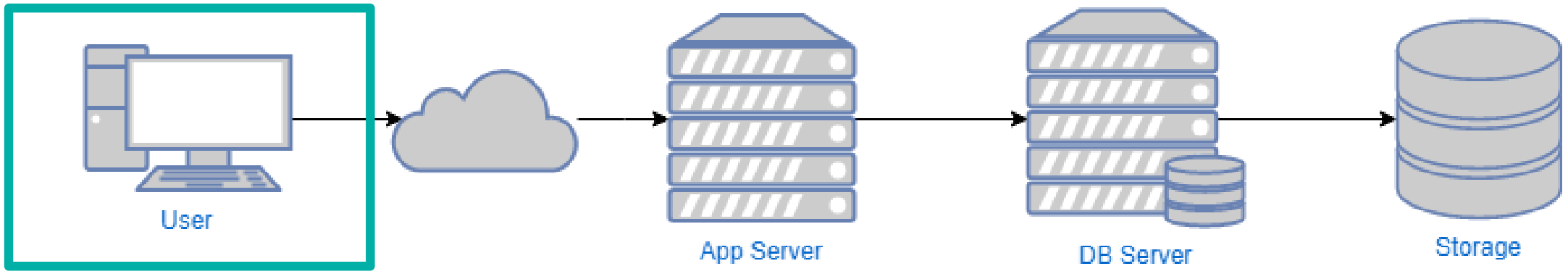
CLIENT-SIDE ENCRYPTION

THREATS

- Data theft
- Sensitive data exposure

USE CASES

- Preventing 3rd party (SaaS) access
- Secure messaging clients
- Cloud-based password managers



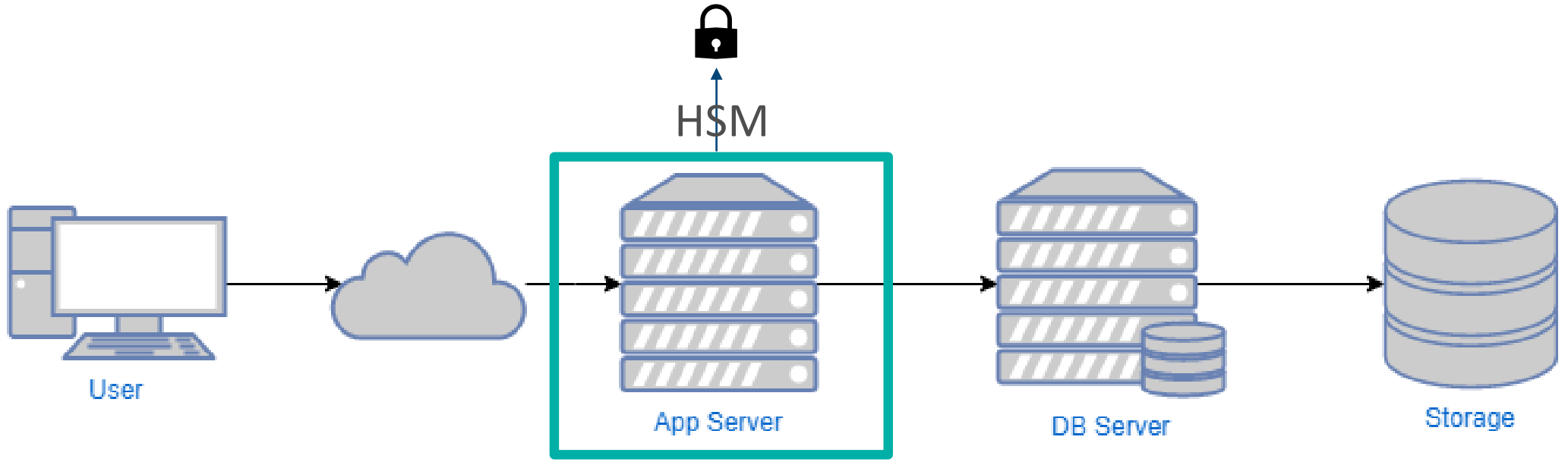
APP SERVER

THREATS

- Database attacks, i.e. SQL injection (can steal the data, but not read it)
- Unauthorized access by admins

USE CASES

- Web Apps/APIs
- Fine grained access control



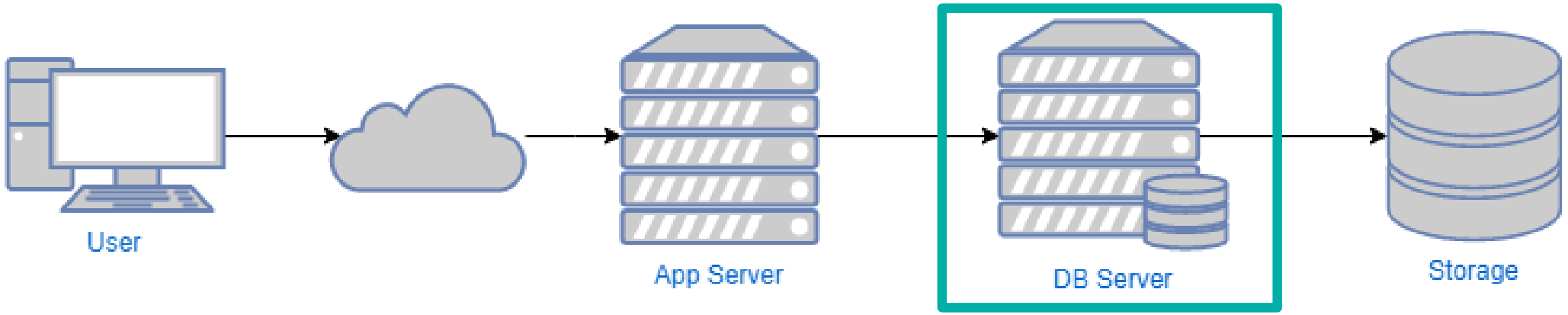
DATABASE-LEVEL ENCRYPTION

THREATS

- Theft/accidental exposure*
- Unauthorized data access

USE CASES

- Lost/stolen hard drives
- Segregation of duties
- Check box security (TDE)



TRANSPARENT DATA ENCRYPTION

…As mentioned earlier, it's important to identify the remaining attack surface even if TDE is enabled. A few to highlight are: SQL injection attacks, cross-site scripting that hijacks an administrator's permissions, code flaws that expose vulnerabilities, attacks by a high-privileged user with sysadmin privileges, attacks by an OS or machine administrator ("box admin"), or attacks by anyone who has physical access to the machine or has access to the OS image and the data files, just to name a few scenarios where attackers could obtain access through elevation of privileges. These are vulnerabilities that TDE does *not* cover, and you'll need to protect against these by other means.

https://blogs.msdn.microsoft.com/sqlsecurity/2016/10/05/feature-spotlight-transparent-data-encryption-tde/

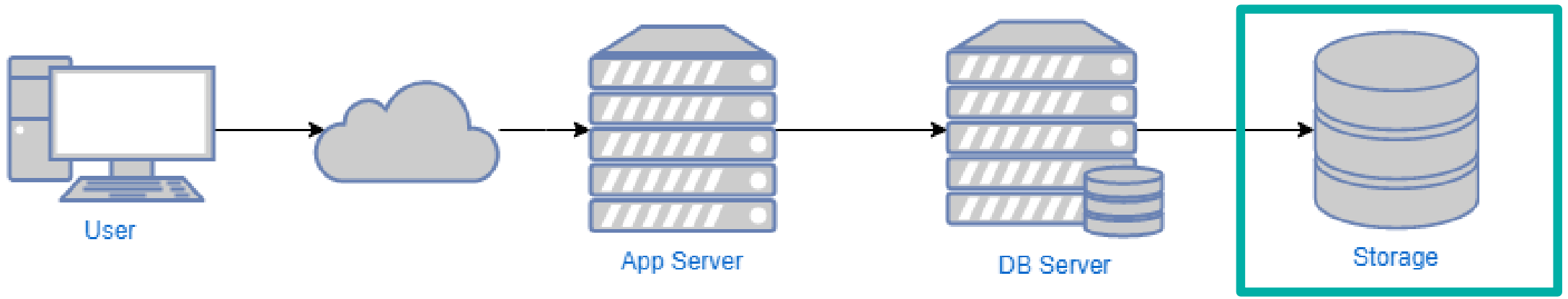
MEDIA-LEVEL ENCRYPTION

THREATS

- Accidental exposure
- Theft

USE CASES

- Cloud Storage: AWS S3 & EBS
- Lost/stolen devices



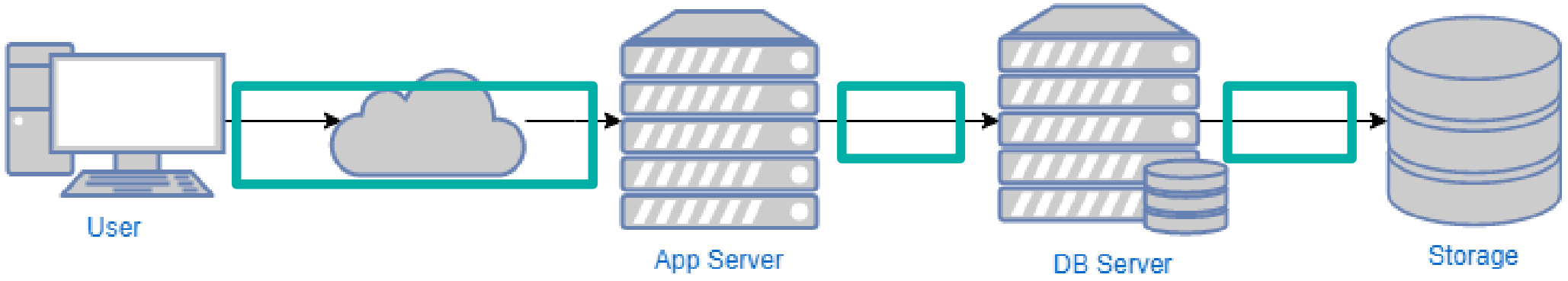
TRANSPORT-LEVEL ENCRYPTION

THREATS

- Man-in-the-middle attacks
- Network sniffing
- BGP hijacking

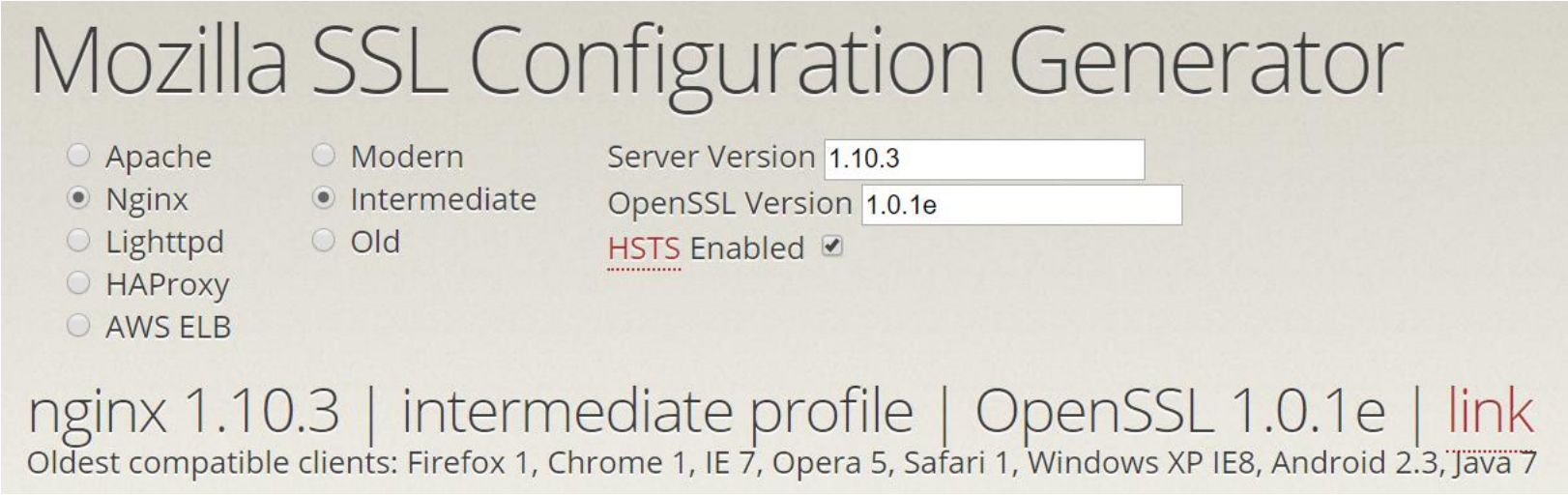
USE CASES

Umm…just encrypt in transport everywhere and use mutual authentication where possible.



TLS MADE EASY

+



STUDY 1: SUBTLE IMPLEMENTATION FLAWS

KEY GENERATION

```
1   "use strict";
2
3   import crypto from 'crypto-pbkdf2';
4   import sjcl from 'sjcl';
5
6   let salt = 'B78C39D1 F52E8A02';
7   const ITERATIONS = 4000;
8   const KEY_SIZE = 256;
9
10  function generateKey(pwd) {
11
12      let hash = crypto.PBKDF2(pwd, salt, ITERATIONS, KEY_SIZE);
13      console.log('hash: ' + hash.toString());
14      let key = sjcl.misc.pbkdf2(hash.toString(), salt, ITERATIONS, KEY_SIZE).join(' ');
15      console.log('key: ' + key);
16
17      return key;
18  }
19
20
```

static salt

enough iterations?

same func, different libs?

Cryptopals Rule: Always operate on raw bytes, never on encoded strings. Only use hex and base64 for pretty-printing.

changes key value

Logging the keys. Ouch.

STUDY 2: RANDOMNESS

RANDOMNESS

```go
const (
    characters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890"
    label      = "kanali"
)

apiKey := createRandByteSlice(32)

ciphertext, err := rsa.EncryptOAEP(sha256.New(), cryptoRand.Reader, publicKey, apiKey, []byte(label))
c.Println(string(key))

func createRandByteSlice(length int) []byte {
    // generate new seed based on time
    mathRand.Seed(time.Now().UTC().UnixNano())
    // construct random string of characters
    s := make([]byte, length)
    for i := range s {
        s[i] = characters[mathRand.Intn(len(characters))]
    }
    return s
}
```

Deterministic “random” number generator

“randomly” chooses a character from the *characters* string

- Limits key space to 62^{32} ($3.1*10^{33}$)
- Rewritten to provide 256^{16} ($3.4*10^{38}$)
- Approx 100000x increase in key space

https://github.com/northwesternmutual/kanalictl

STUDY 3: PLEASE DON’T ROLL YOUR OWN “ENCRYPTION”…

UNNAMED CONSUMER DEVICE LOGIN FUNCTION

```
function encode(input)
{
    var text = "";
    var possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
    var len = input.length;
    var lenn = input.length;
    for( var i=1; i <= (320-len); i++ )
    {
        if (0 == i % 7 && len > 0)
          text += input.charAt(--len);
        else if (i == 123)
        {
          if (lenn < 10)
            text += "0";
          else
            text += Math.floor(lenn/10);
        }
        else if (i == 289)
          text += lenn%10;
        else
          text += possible.charAt(Math.floor(Math.random() * possible.length));
    }
    return text;
}
```

"encode" function from PORC@MadSec

OUTPUT OF THE ENCODE FUNCTION

encode("aaaaaa")

Output:

v17o7B**a**wkqZwf**a**NAJIRK**a**g2hIzu**a**6p3tjn**a**4m7iy9**a**hrD5SS5VUiNUyrTkriwcvGktv4lxX
p2yEWWbWEp0pKB04kcK2MzczyxwtCDzSQ0uDSPh2d2hCkLcSTV1**0**hr2xSelmvsMUgGlY
8xqVMXC4M082MYJte6QPp87lsB8CeJqh1eu1wZBphqtxJBLgfusBwEC4ZUj6byRB4IiSGI6j
eyjdWPjb66v3gChUyERFnVU5ipOb4vr7jYHCxm0sQ7XoMkUyX8J4UPMbtWpzN7w881M
FnNoqC4fOW**6**0kCIeyNY95muoS6zFholqosTRIh5vb0

KEY MANAGEMENT

KEY MANAGEMENT

- Secrets should **never** be stored in plaintext, especially in repos
- Rotate your keys
- Use a vaulting solution! (seriously, just some something)
  - Hardware Security Module (HSM) /Hardware Security Appliance (HSA)
  - Amazon KMS
  - Azure Key Vault
  - Conjour
  - Ansible Vault

RECOMMENDATIONS

APPROACH

1. Build a threat model of your application
2. Use your threat model to inform cryptographic choices
3. Implement appropriate crypto choices per Cryptographic Right Answers
4. Design proper key management and rotation as part of the implementation
5. Use cryptographically secure psuedo-random number generators (CSPRNG)
6. Always operate on raw bytes (no strings)
7. Profit!

REFERENCES AND RESOURCES

- Cryptographic Right Answers (http://latacora.singles/2018/04/03/cryptographic-right-answers.html)
- How To Safely Generate A Random Number - https://sockpuppet.org/blog/2014/02/25/safely-generate-random-numbers/
- Cryptopals - https://cryptopals.com/
- Serious Cryptography - https://nostarch.com/seriouscrypto
- Mozilla Server-Side Encryption - https://wiki.mozilla.org/Security/Server_Side_TLS

THANK YOU

Northwestern Mutual®