Erik LaBianca
WiserTogether, Inc.

Cryptography for Djangonauts

Background

- Who am I? **Just a developer**
- Should you trust me? **Probably not**
- Should you pay attention anyway? **Probably**

TL;DR

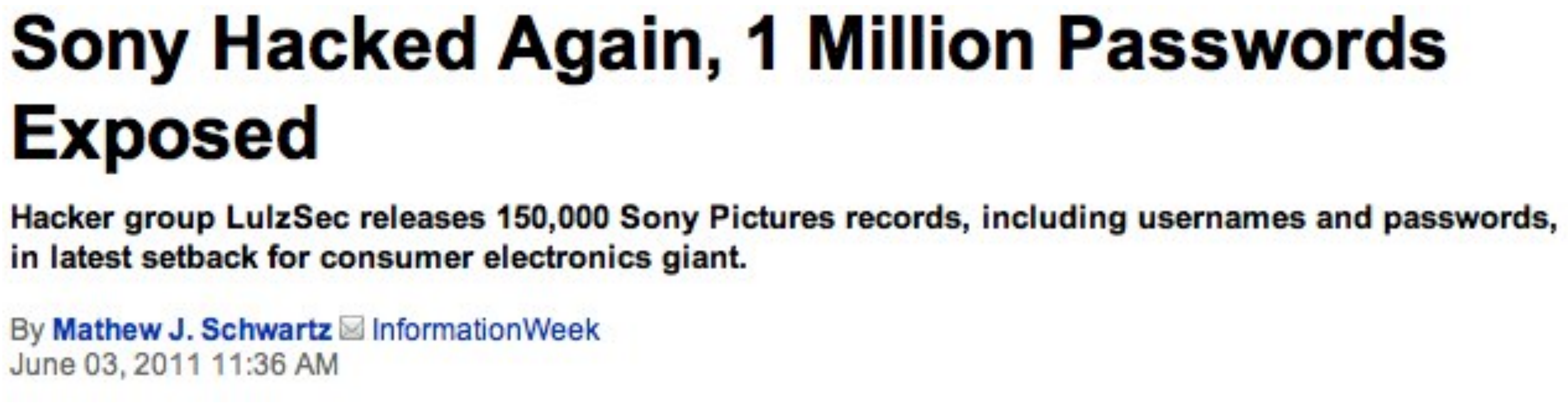
- **Analyze** your risks
- **Don't** write your own
- **Operate** correctly
- **Commit** to keeping up

(please use a password manager)

Why you care

- Doing it **wrong** is easy. And common.
  - OWASP Top-10 A7: Insecure Cryptography

Risk analysis helps



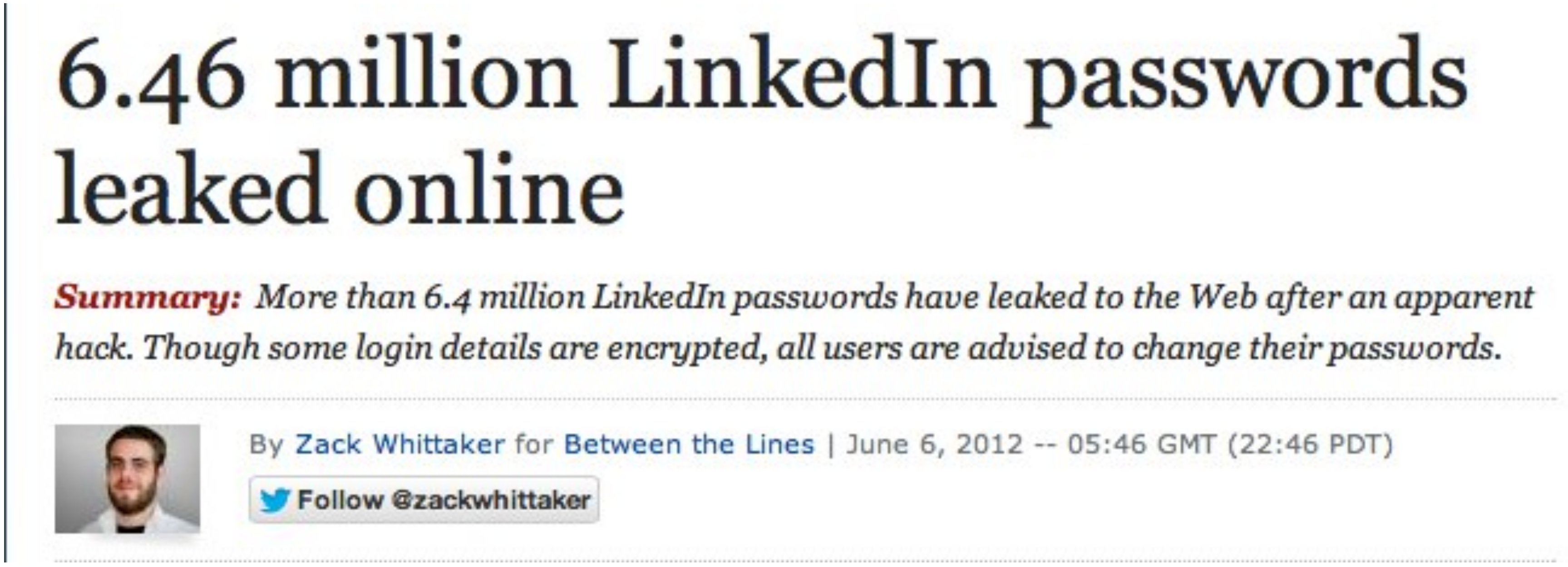
Sony Hacked Again, 1 Million Passwords Exposed

Hacker group LulzSec releases 150,000 Sony Pictures records, including usernames and passwords, in latest setback for consumer electronics giant.

By Mathew J. Schwartz InformationWeek
June 03, 2011 11:36 AM

(unique passwords are cool)

Roll-your-own is hard



6.46 million LinkedIn passwords leaked online

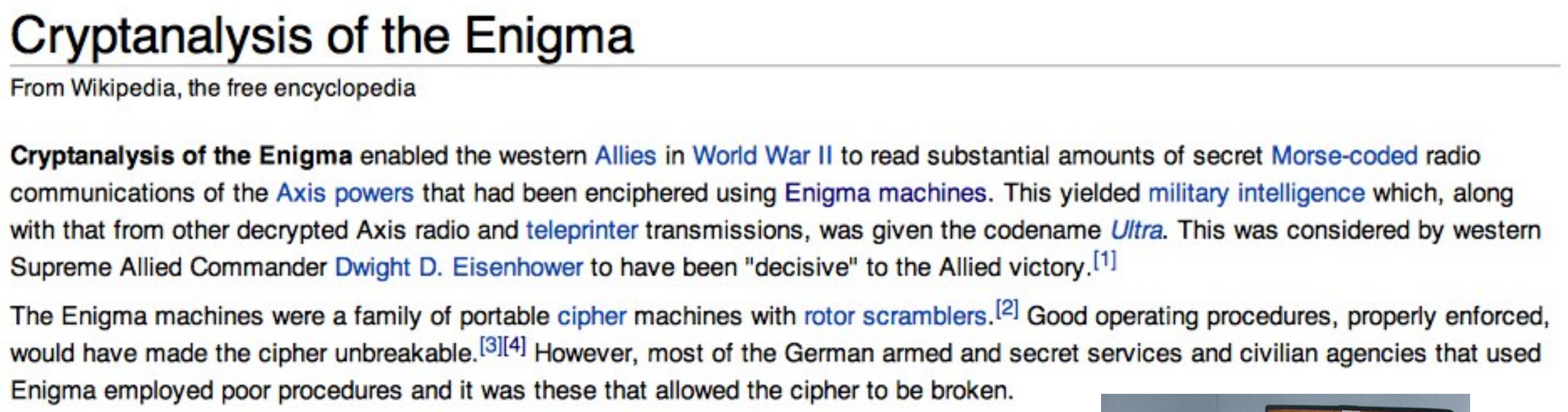
Summary: More than 6.4 million LinkedIn passwords have leaked to the Web after an apparent hack. Though some login details are encrypted, all users are advised to change their passwords.

By Zack Whittaker for Between the Lines | June 6, 2012 -- 05:46 GMT (22:46 PDT)

Follow @zackwhittaker

(try out a password manager)

The details are hard

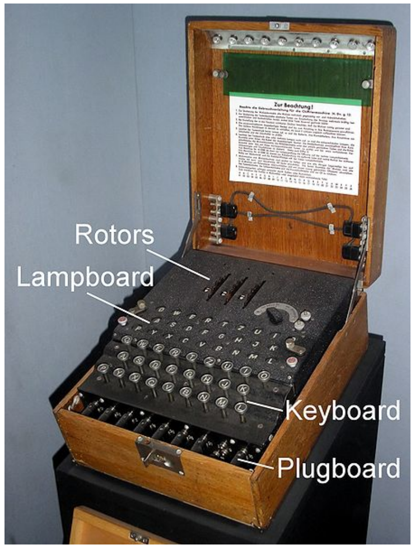


Cryptanalysis of the Enigma

From Wikipedia, the free encyclopedia

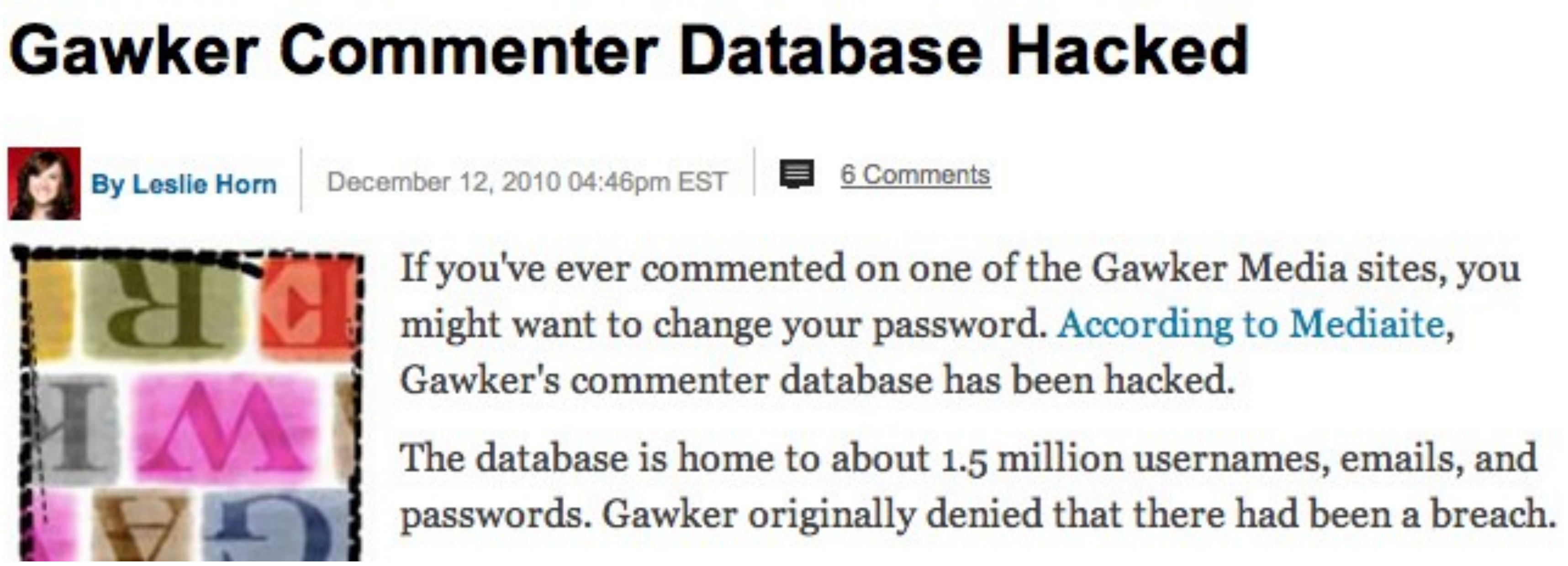
Cryptanalysis of the Enigma enabled the western Allies in World War II to read substantial amounts of secret Morse-coded radio communications of the Axis powers that had been enciphered using Enigma machines. This yielded military intelligence which, along with that from other decrypted Axis radio and teleprinter transmissions, was given the codename *Ultra*. This was considered by western Supreme Allied Commander Dwight D. Eisenhower to have been "decisive" to the Allied victory.[1]

The Enigma machines were a family of portable cipher machines with rotor scramblers.[2] Good operating procedures, properly enforced, would have made the cipher unbreakable.[3][4] However, most of the German armed and secret services and civilian agencies that used Enigma employed poor procedures and it was these that allowed the cipher to be broken.



- Extra Credit:
  - http://en.wikipedia.org/wiki/Cryptanalysis_of_the_Enigma

Keeping up is hard



Gawker Commenter Database Hacked

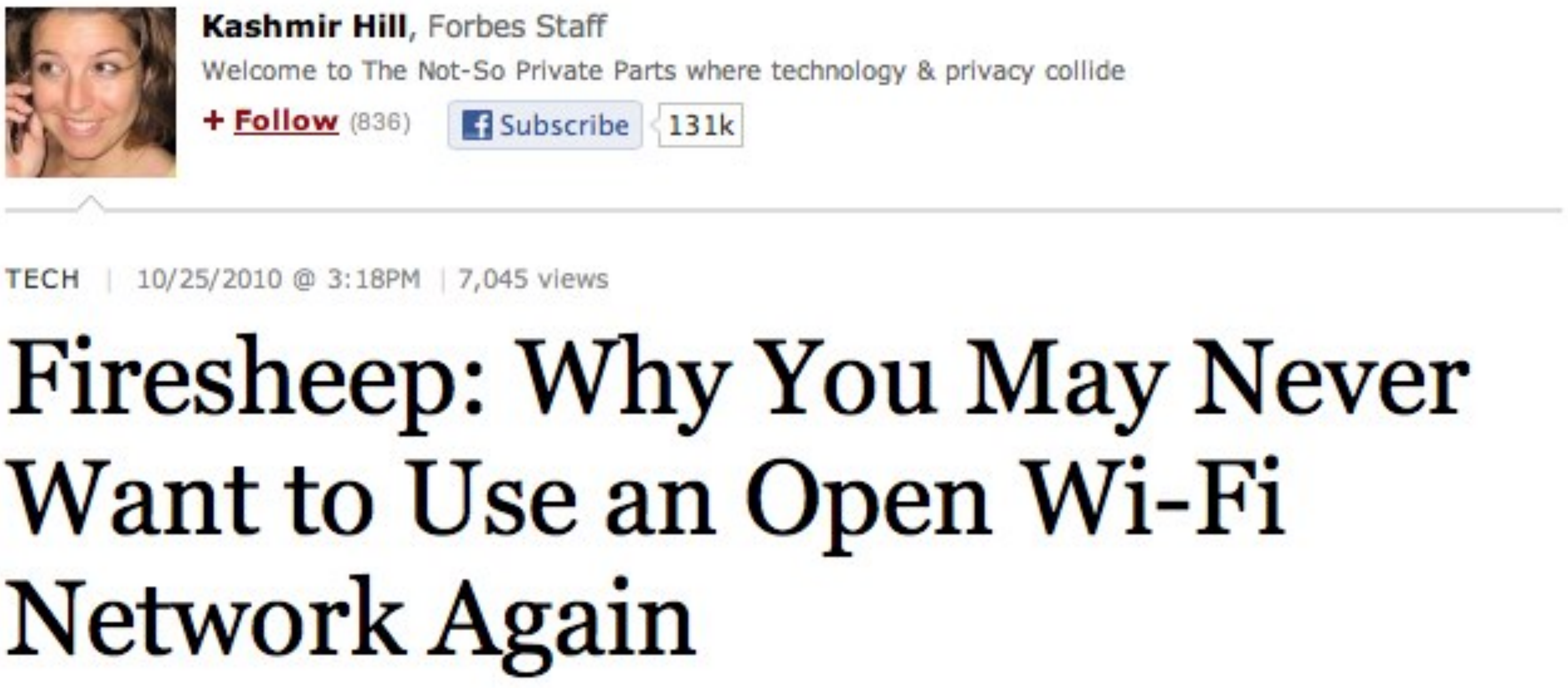
By Leslie Horn | December 12, 2010 04:46pm EST | 6 Comments

If you've ever commented on one of the Gawker Media sites, you might want to change your password. According to Mediaite, Gawker's commenter database has been hacked.

The database is home to about 1.5 million usernames, emails, and passwords. Gawker originally denied that there had been a breach.

(please use a password manager)

Attacks are mechanized



Kashmir Hill, Forbes Staff

Welcome to The Not-So Private Parts where technology & privacy collide

TECH | 10/25/2010 @ 3:18PM | 7,045 views

Firesheep: Why You May Never Want to Use an Open Wi-Fi Network Again

(please use a password manager)

Show of Hands

- How many of you have:
  - Used hashlib, md5sum, or another hash function?
  - Set up truecrypt, luks, filevault, bitlocker, or another symmetric cryptography system?
  - Configured a web server to serve HTTPS, or another SSL/TLS service?
  - Used PGP or S/MIME?
  - Configured a Certificate Authority?

Analyze your Risks

- Inventory your **Assets**
  - Data (PII/PHI?)
  - Systems
- Identify your **Vulnerabilities**
  - Lost Backups
  - Lost Laptops
  - Compromised Systems
  - Insecure Networks
  - Employees and Customers
- Analyze **Controls**
  - Destruction (or stop collecting)
  - Locked safe
  - Cryptography

"All I'm saying is *now* is the time to develop the technology to deflect an asteroid."

Extra Credit:
- http://csrc.nist.gov/publications/nistpubs/800-30/sp800-30.pdf
- ISO 27005

Cryptographic Hash Functions

Plaintext → Fixed Length Hash

→

4aef0ceec93c3c95b09e39674527bd
22364808c29390db01fd63d163

```
>>> import hashlib
>>> hashlib.sha224('Plaintext').hexdigest()
'95c7fbca92ac5083afda62a564a3d014fc3b72c9140e3cb99ea6bf12'

$ openssl dgst -sha224 ~/Pictures/hashbrowns.png
SHA224(~/Pictures/hashbrowns.png)=
4aef0ceec93c3c95b09e39674527bd22364808c29390db01fd63d163
```

Cryptographic Hash Properties

- No Keys
- Easy to compute the hash value (digest) of any message
- Very hard to
  - generate a message for a known hash value
  - modify a message without changing the hash
  - find two messages with the same hash
- Used for
  - Verifying integrity of files or messages
    - Django Session and Cookie signing
    - SSL / TLS / HTTPS - Keyed Hashing for Message Authentication (HMAC)
  - Password verification (caveats apply!)
    - django.contrib.auth
  - Reliable identification of unique files (git, hg)
  - Pseudorandom bit generation
- Extra Credit:
  - http://en.wikipedia.org/wiki/Cryptographic_hash_function
  - http://tools.ietf.org/html/rfc2104

Symmetric Encryption Algorithms

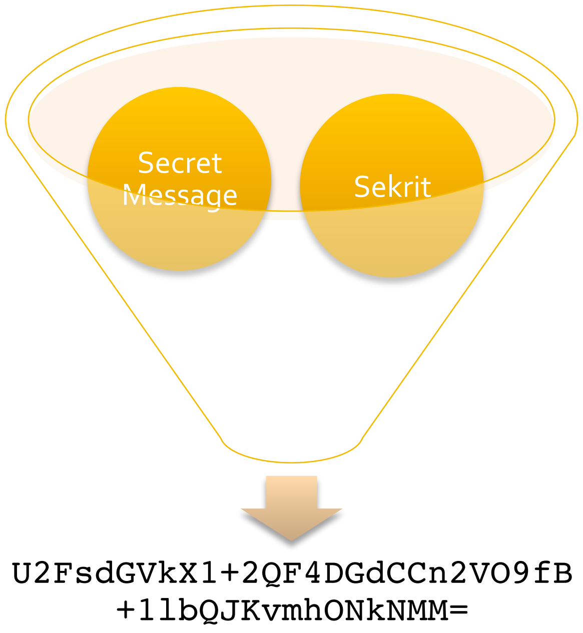
- AKA "Secret Key Encryption"
- 2-way (encrypt and decrypt)
- 1 key (must be shared, and kept secret)
- Fast

```
$ echo "Secret Message" |
openssl enc -aes-256-cbc -k
Sekrit -a

U2FsdGVkX1+2QF4DGdCCn2VO9fB
+1lbQJKvmhONkNMM=

$ echo
"U2FsdGVkX1+2QF4DGdCCn2VO9fB
+1lbQJKvmhONkNMM=" | openssl enc
-d -aes-256-cbc -k Sekrit –a

Secret Message
```

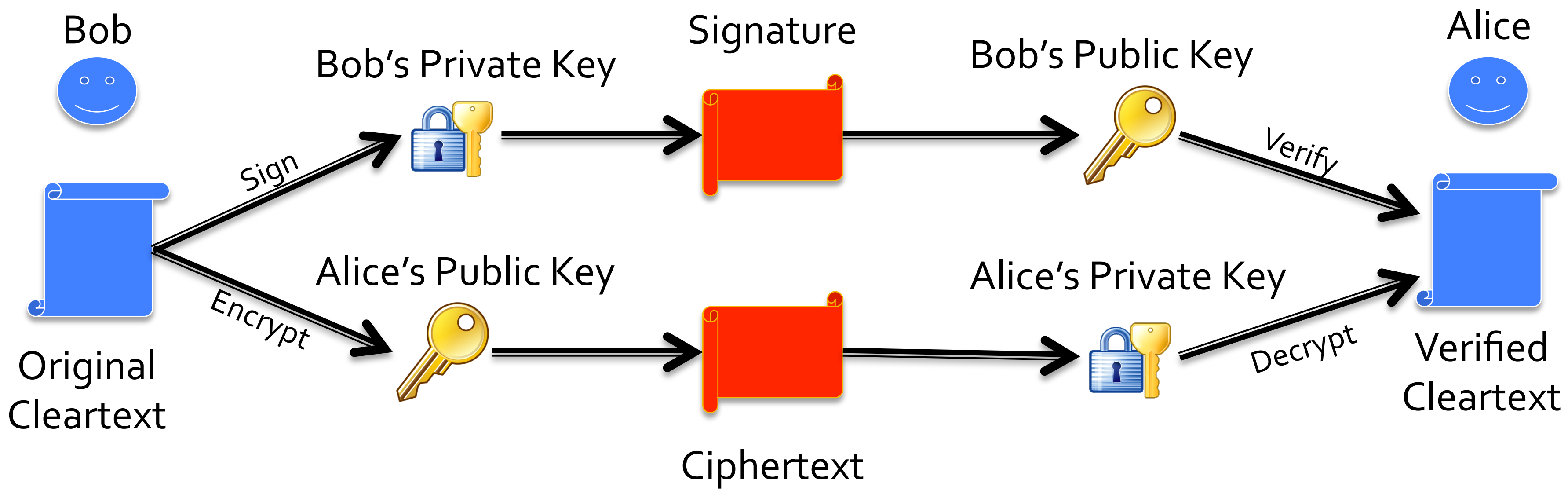


Symmetric Encryption Properties

- Reversible (both encrypt and decrypt)
- Requires a Shared Secret
- Uses
  - Encrypting files, backups, etc
  - Encrypting file systems (filevault, bitlocker, truecrypt, luks)
  - Encrypting transmission (SSL, TLS, IPSec)
- Algorithms
  - DES (out of date)
  - One Time Pad (inconvenient)
  - AES (NIST certified, hardware optimized)
  - Blowfish
- Implementations
  - M2Crypto (OpenSSL Wrapper)
  - PyCrypto (Pythonic)

Public Key Cryptography

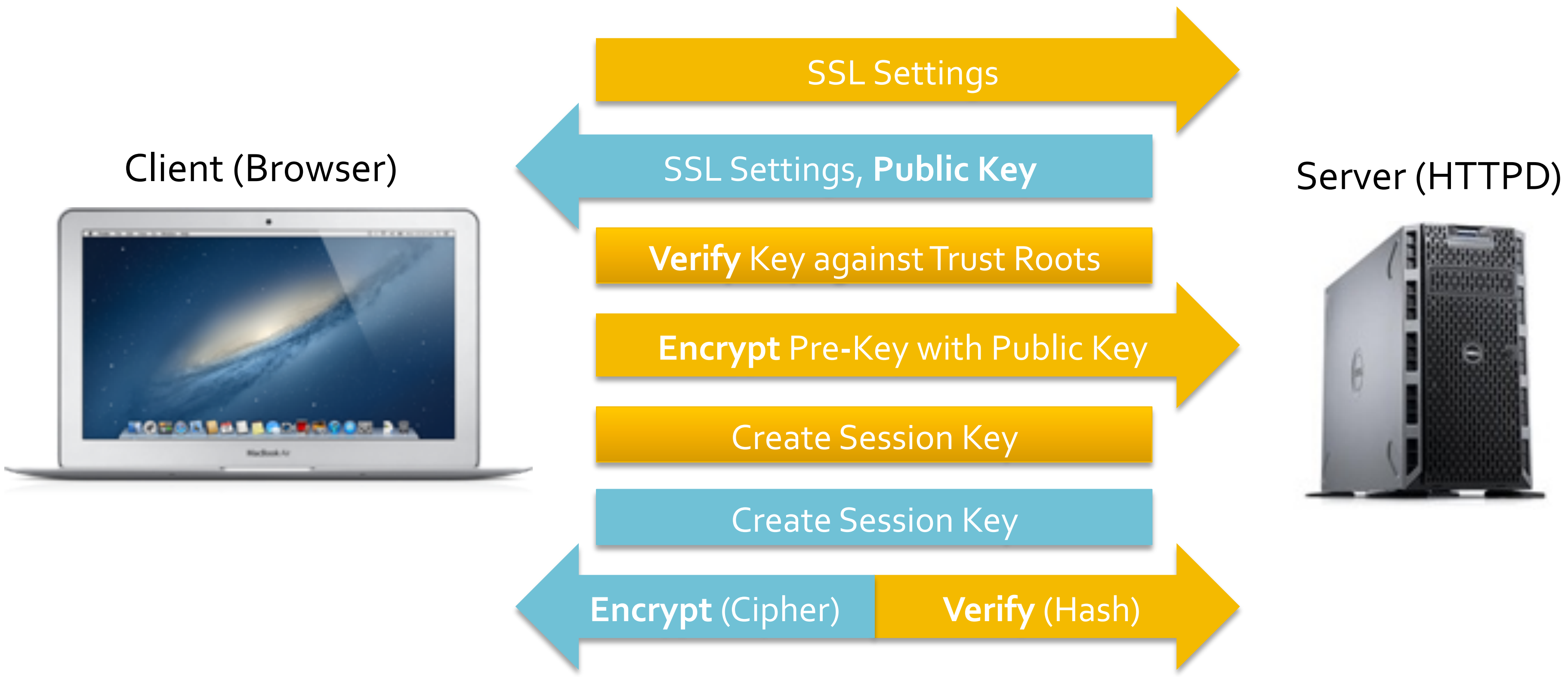
- Asymmetric
- N-way (encrypt, decrypt, sign and verify for N parties)
- 2+ keys (public and private for each party



Asymmetric Encryption Properties

- Lots of Complex Keys
- Slow
- Algorithms
  - RSA, DSA
- Uses
  - Key Validation
    - Certificate Authorities, Web of Trust
  - Key Exchange
    - SSL, TLS, HTTPS
  - Secure Asynchronous Transfer
    - S/MIME, PGP/MIME, PGP

Putting it all together: HTTPS



Extra Credit:

- http://tools.ietf.org/html/rfc5246
- http://technet.microsoft.com/en-us/library/cc781476

Doing it Right: Table Stakes

- Django does Crypto right
  - Use Django 1.4 if you can
  - Keep settings.SECRET_KEY a secret
- Enable HTTPS
- Enforce use of HTTPS via redirects
- Inform Django you're using HTTPS
  - Check request.is_secure
  - Set settings.SESSION_COOKIE_SECURE=True
  - Set settings.CSRF_COOKIE_SECURE=True
  - Set settings.SECURE_PROXY_SSL_HEADER

Doing it Right: The Hard Part

- Protect private data via SKC
- Support encrypted payloads via PKC.
  - How will you unlock the secret keys?
- Use full-disk (boot volume) encryption
  - How will you provide the symmetric key?
- Extra Credit
  - FIPS certified implementations
  - FIPS / NIST configurations

User Passwords: Naïve Code

```
from hashlib import sha224

users = ([1, 'bob', 'secret'],
         [2, 'alice', 'sekrit'],
         [3, 'eve', 'secret'])
for user in users:
    user[2] = sha224(user[2]).hexdigest()[:8]
print users

$ python naive_hash.py
([1, 'bob', '95c7fbca'],
 [2, 'alice', '034f4966'],
 [3, 'eve', '95c7fbca'])
```

Please do not do this!

- Same password results in the same hash. Bad!
  - Entire list can be bruteforced in one pass.

User Passwords: Done Right

```
# see django/contrib/auth/hashers.py

from django.utils.crypto import (pbkdf2,
  constant_time_compare, get_random_string)

def encode(password, salt=None, iterations=10000):
    if not salt:
        salt = get_random_string()
    hash = pbkdf2(password, salt, iterations)
    hash = hash.encode('base64').strip()
    return "%s$%d$%s$%s" % ('pbkdf2', iterations,
                            salt, hash)

def verify(password, encoded):
    alg, iterations, salt, hash = encoded.split('$', 3)
    encoded_2 = encode(password, salt, int(iterations))
    return constant_time_compare(encoded, encoded_2)

for user in users:
    user[2] = encode(user[2])
```

Password Hashing: Done Right

```
$python password_hash.py
([1, 'bob',   'secret'],
 [2, 'alice', 'sekrit'],
 [3, 'eve',   'secret'])

([1, 'bob',
   'pbkdf2$10000$cNTDFLN3M6wQ$' \
   'YaLSp47KyS197VKNkAD6A0LYO2ZSc2EcWb07b7NBw+M='],
 [2, 'alice',
   'pbkdf2$10000$w7JZjGibBuvf$’ \
   'dVlM9aP8b5SCf/hJwqB47nDBIBbKw955yJfN+82BFV0='],
 [3, 'eve',
   'pbkdf2$10000$P4X6u9IL2a9P$’
   '2EGFbYELD1azOK3Xhon6s9rW9sRs2LZP9xLp9ekbvIU='])
```

- Bob and eve’s passwords hash to radically different values
- The algorithm and counter is stored in the password string so it can be updated in the future
- The random salt is stored so we can still verify successfully
- Extra Credit:
  - Add HMAC: check out https://github.com/fwenzel/django-sha2

Example: Hashed Record Lookup

```
from hashlib import sha224
salt = 'aNiceLongSecret'
users = ([1, 'bob', '123456789'],
         [2, 'alice', '123456780'],
         [3, 'eve', '123456781'])
for user in users:
    user[2] = sha224(salt + user[2]).hexdigest()[:8]
```

- Better than nothing.
  - Makes brute-force infeasible without the salt value
  - Salt should be stored separately from values
  - Still allows you to “look up” values by their hashed value, such as an ID#.

Example: Symmetric Encryption

```
from base64 import b64encode, b64decode
from M2Crypto.EVP import Cipher

from django.utils.crypto import get_random_string

def encrypt(key, iv, cleartext):
    cipher = Cipher(alg='aes_256_cbc', key=key, iv=iv, op=1) # 1=encode
    v = cipher.update(cleartext) + cipher.final()
    del cipher # clean up c libraries
    return b64encode(v)

def decrypt(key, iv, ciphertext):
    data = b64decode(ciphertext)
    cipher = Cipher(alg='aes_256_cbc', key=key, iv=iv, op=0) # 0=decode
    v = cipher.update(data) + cipher.final()
    del cipher  # clean up c libraries
    return v

(key, iv) = ('nicelongsekretkey', get_random_string(16))
ciphertext = encrypt(key, iv, 'a very long secret  1message')
cleartext = decrypt(key, iv, ciphertext)
```

Example: Shared Secret SSO

```
from django.contrib.auth import models, login, logout, authenticate
from django.core.urlresolvers  import reverse
from django.http import HttpResponseRedirect
from django.utils import simplejson as json
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def sso_token_handler(request):
    init_vector = request.GET.get('iv', None)
    token = request.GET.get('token', None)
    token_data = json.loads(decrypt('sekrit', init_vector, token))
    user = User.objects.get(token_data['user'])
    if user is None:
        user = create_user(token_data)
    authuser = authenticate(user=user)
    login(request, authuser)
    return HttpResponseRedirect(reverse('home'))
```

Final Thought: Key Management

- How will you make keys available to your application?
- Keys on local disk
    - Useful for encrypting backups
    - Useful for encrypting transmission
    - Not so useful for encryption-at-rest
- Keys on physical device (smartcard or HSM)
  - Great idea! Good luck in the “cloud”.
- Keys in memory
  - Still potentially exploitable, but requires compromise of a running machine.
  - How do they get there?
    - Must be provided at boot or initialization time somehow

Examples: Public Key Encryption

```
import gnupg
gpg_handler_parameters = { 'use_agent': True, }
gpg_handler = gnupg.GPG(**gpg_handler_parameters)
ciphertext = gpg_handler.encrypt(
        'Secret Message',
        'erik.labianca@gmail.com',
        always_trust=True,
        )

cleartext = gpg_handler.decrypt(ciphertext)
```

Exercise:

- Configure gpg keyring and unlock secret keys with gpg-agent
- Load symmetric keys via PGP

Questions?

WISER together® | Use healthcare wisely.

Work With Us

- http://wisertogether.com/careers/
- https://github.com/WiserTogether/

Contact Me

- mailto:erik.labianca@gmail.com
- https://twitter.com/easel/
- https://github.com/easel/
- https://easel.github.com/talks/django-cryptography.pdf

Other Resources

- http://www.kyleisom.net/downloads/crypto_intro.pdf
- http://www.garykessler.net/library/crypto.html
- http://code.google.com/p/python-gnupg/
- http://chandlerproject.org/Projects/MeTooCrypto
- https://www.dlitz.net/software/pycrypto/