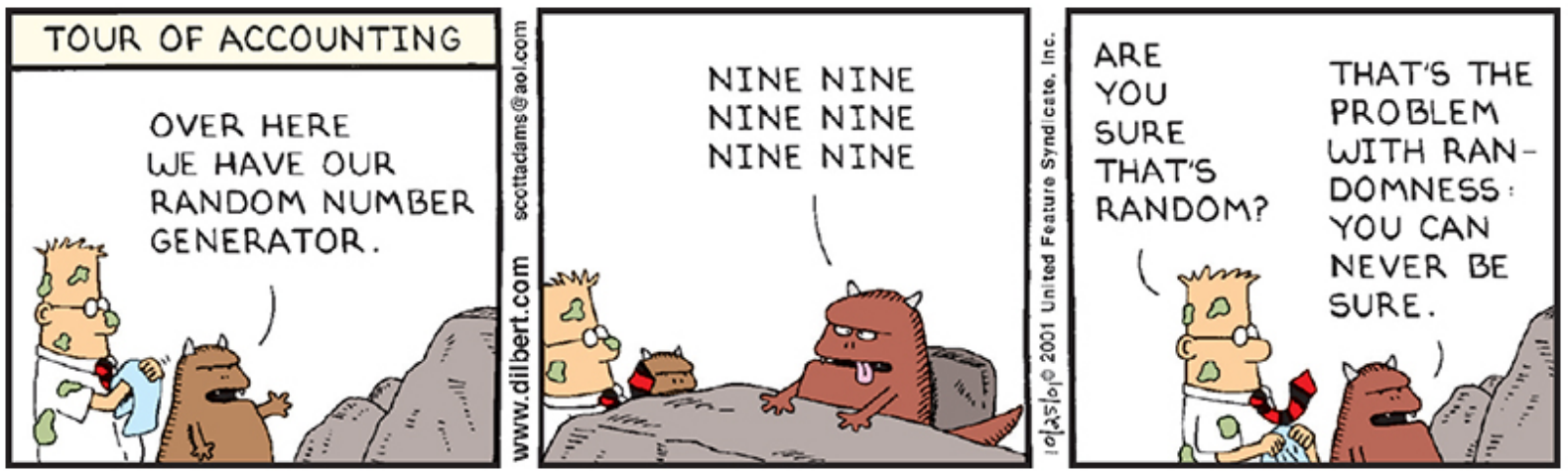
0XFFFFFFFF EVERY TIME IS 0XDEADBEEF —

How a months-old AMD microcode bug destroyed my weekend [UPDATED]

AMD shipped Ryzen 3000 with a serious microcode bug in its random number generator.

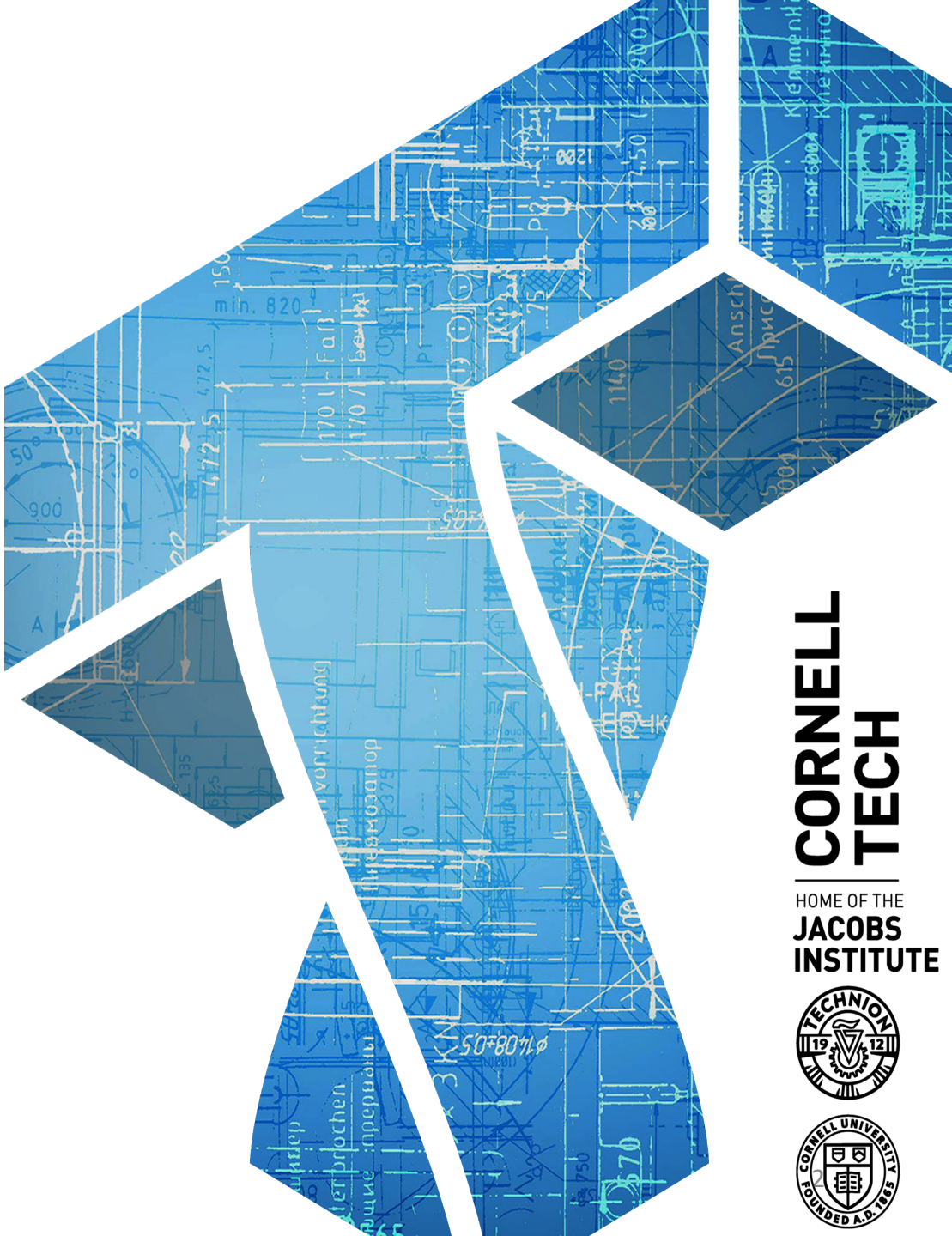
JIM SALTER - 10/29/2019, 7:00 AM



CS 5435: Asymmetric cryptography (part 2)

Instructor: Tom Ristenpart

https://github.com/tomrist/cs5435-fall2019



CORNELL TECH

HOME OF THE JACOBS INSTITUTE

Where we're at

- Last time: key exchange, key transport, public-key encryption, forward secrecy, Diffie-Hellman, man-in-the-middle attacks

- Today's lecture
  - Digital signatures
  - Public-key infrastructure (PKI)

TLS handshake for Diffie-Hellman Key Exchange

Client

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Pick random Ns

← ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

Check CERT using CA public verification key

Check σ

← CERT = (pk_s , signature over it)

Pick random x

$X = g^x \bmod p$

← $p , g , X , \quad \sigma = \text{Sign}(sk_s, p \,||\, g \,||\, X)$

Pick random y

$Y = g^y \bmod p$

Y →

$PMS = g^{xy} \bmod p$

$PMS = g^{xy} \bmod p$

ChangeCipherSpec,

{ Finished, PRF(MS, “Client finished” || H(transcript)) } →

ChangeCipherSpec,

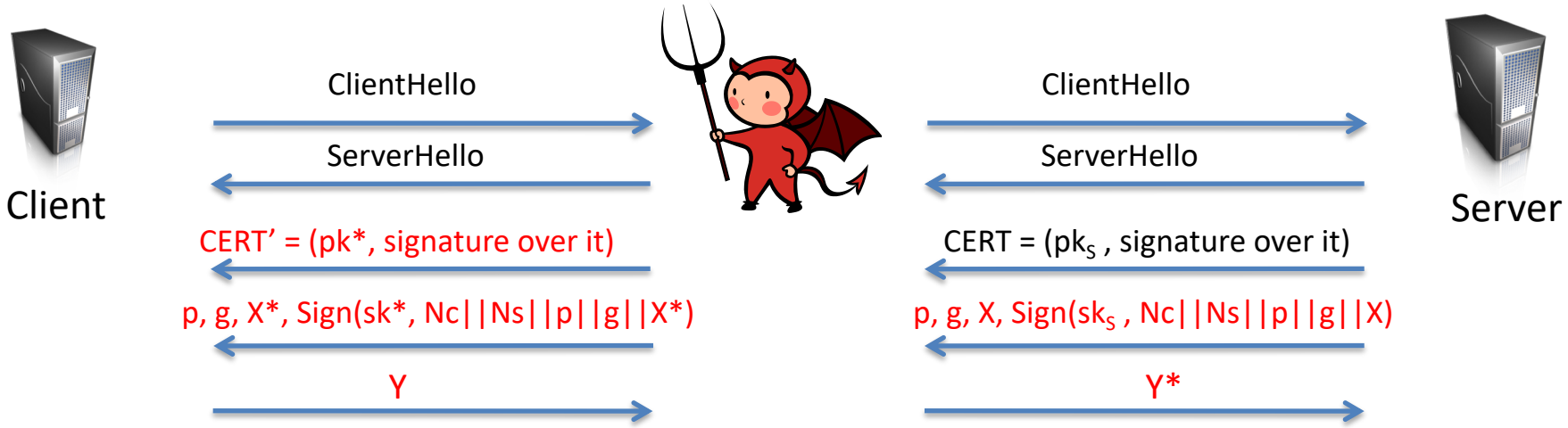
{ Finished, PRF(MS, “Server finished” || H(transcript’)) } ←

Bracket notation means contents encrypted

MS <- PRF(PMS, “master secret” || Nc || Ns)

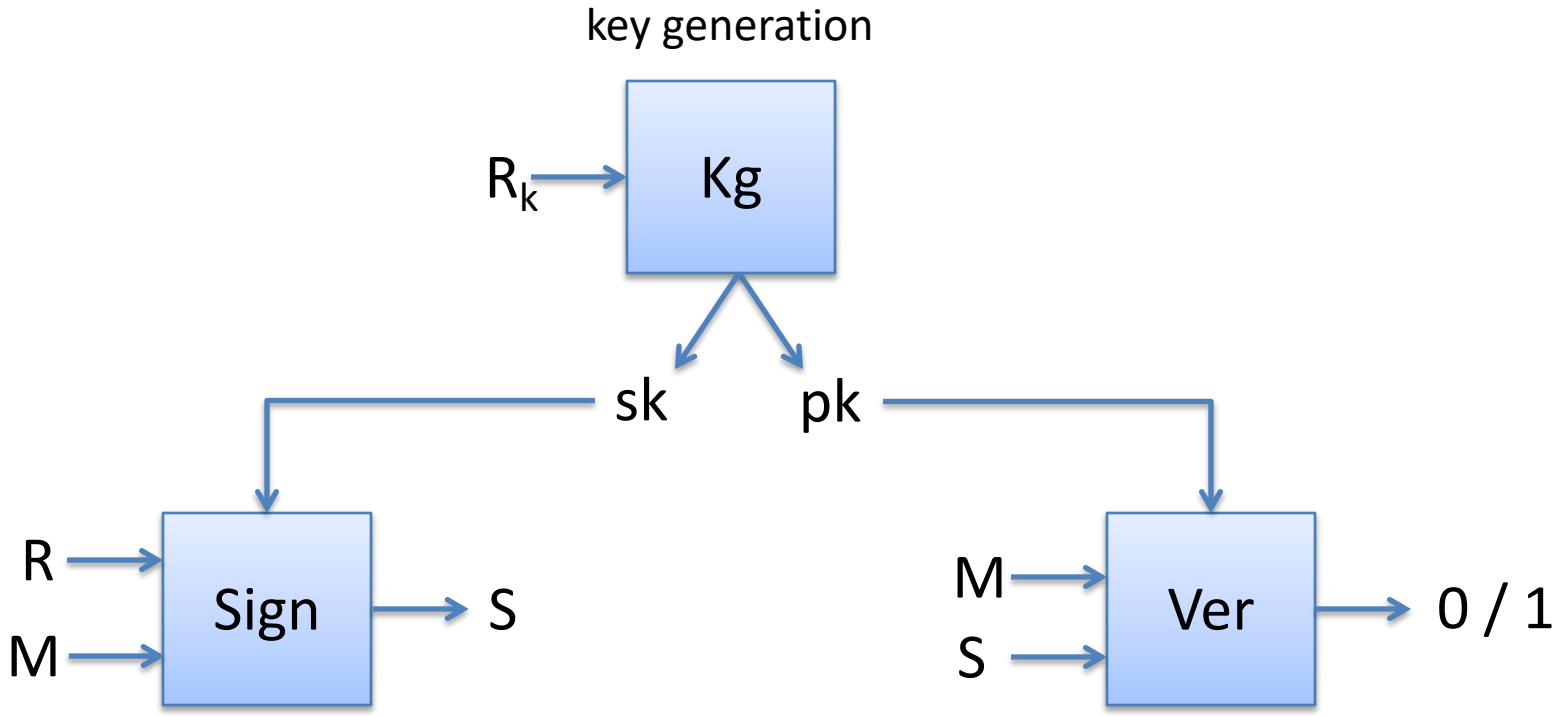
Man-in-the-middle attacks

Suppose authentication vulnerability:
CERT can be forged, Client doesn't check CERT, etc.



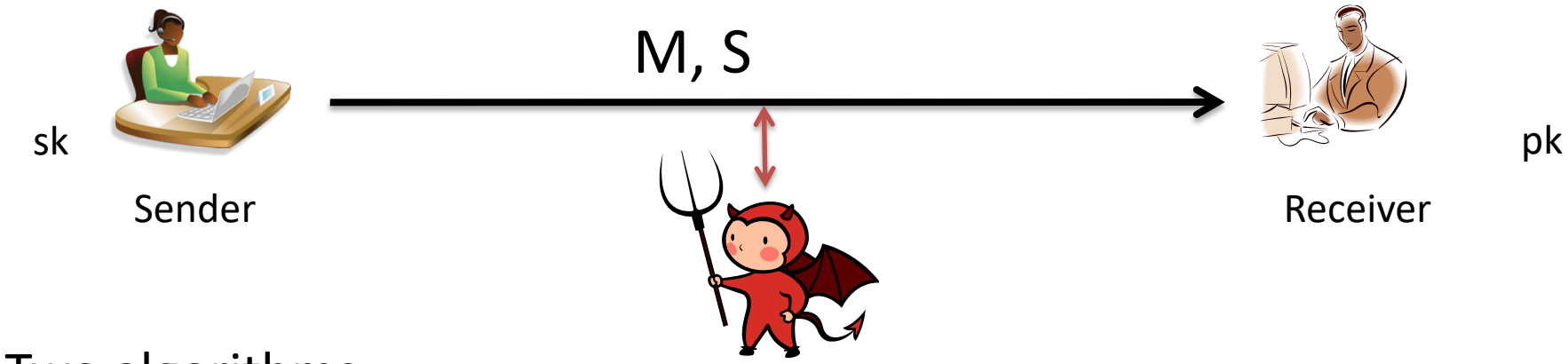
Attacker can choose X*, Y*, so it knows discrete logs
Completes handshake on both sides
Client thinks its talking to Server
All communications decrypted by adversary, re-encrypted and forwarded to server

Digital signatures



Anyone with public key can verify a signature
Only holder of secret key should be able to generate a signature

Digital signatures



Two algorithms:

(1) Key generation outputs (pk,sk)

(2) Sign (sk,M) outputs a signature S (may be randomized)

(3) Ver(pk,M,S) outputs 0/1 (invalid / valid)

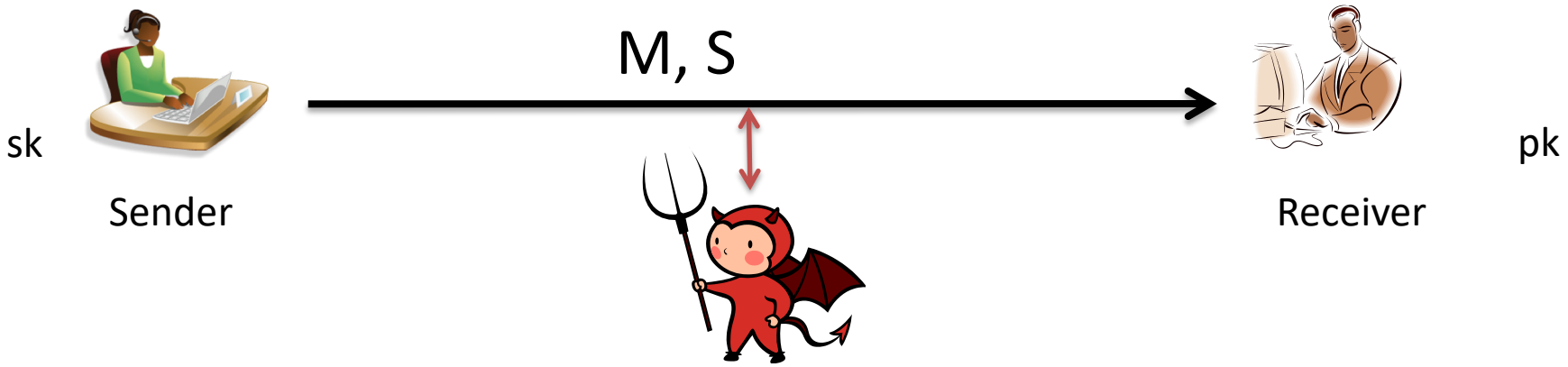
Correctness: Ver(pk,M,Sign(sk,M)) = 1 always

Security: No computationally efficient attacker can forge valid signature for a new message even when attacker gets

$$(M_1 , S_1) , (M_2 , S_2), \dots , (M_q , S_q)$$

for messages of his choosing and reasonably large q.

Digital signatures



"Raw" RSA as a signature scheme:

Key generation gives (N,e) , (N,d) s.t. $(M^e)^d \bmod N = M$

Sign((N,d),M) = $M^d \bmod N$ Verify((N,e),M,S) checks if $S^e \bmod N = M$

Secure?

No!

M = 1
Then S = 1 valid

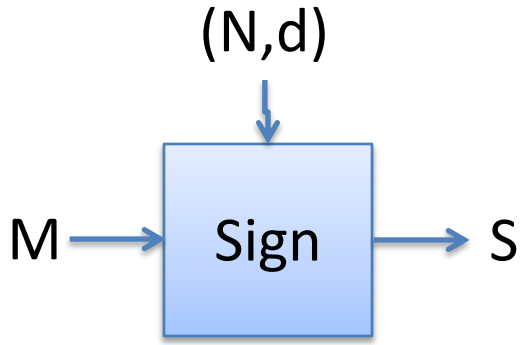
Get $S_1 = M^d \bmod N$
Then S = $(M^d)^2 \bmod N$ is valid signature for M^2

PKCS #1 v1.5 RSA signing

Kg outputs (N,e),(N,d) where $|N|_8 = n$

Want to sign using hash with output length m bytes

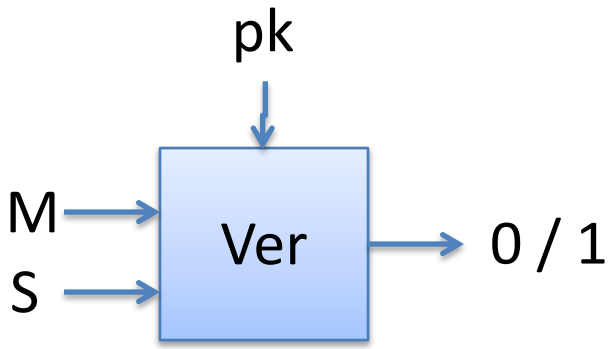
Let p = n – m - 3



Sign((N,d), M)

Y = 00 || 01 || FF^p || 00 || H(M)

Return Y^d mod N



Verify((N,e), M, S)

Y = S^e mod N ; aa||bb||cc||dd||h = Y

If (aa ≠ 00) or (bb ≠ 01) or (cc ≠ FF^p)

or (dd ≠ 00)

Return error

Return H(M) = h

PKCS#1 v1.5 digital signature security

Padding oracle attacks:

- PKCS#1 v1.5 decryption oracle can be used to forge
- PKCS#1 v1.5 signing oracle can be used to forge

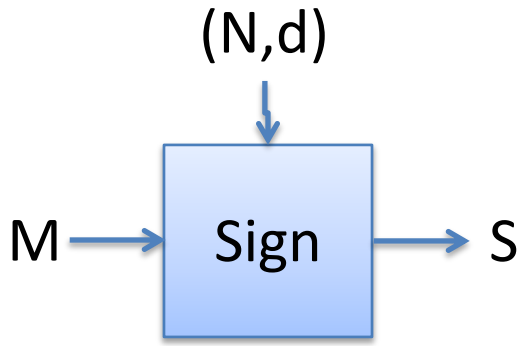
If one uses same RSA key pair for signing and encryption, then decryption and signing oracles offer similar forging / decryption opportunities

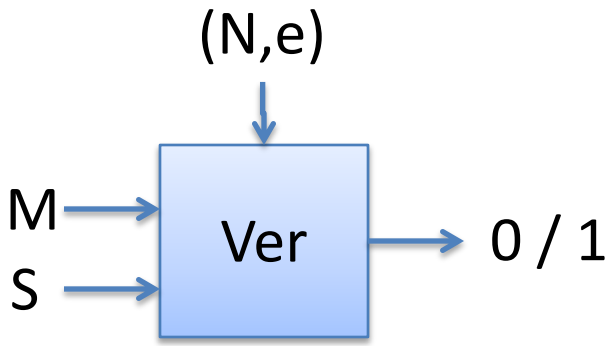
- Never use same keys for signing and encryption

Full Domain Hash RSA

Kg outputs pk = (N,e) , sk = (N,d) where $|N|_8 = n$

H is hash with m-byte output k = ceil((n – 1)/m)





Sign((N,d), M)

X = 00 || H(1||M) || … || H(k||M)

$S = X^d \bmod N$

Return S

Ver((N,e), M, S)

$X = S^e \bmod N$

X' = 00 || H(1||M) || … || H(k||M)

If X = X' then

    Return 1

Return 0

Probabilistic Signature Scheme (PSS) variant of FDH that is widely deployed see PKCS#1 v2

Digital signatures from discrete log

- Schnorr signatures
- Digital Signature Algorithm (designed by DSA)
  - Variant of Schnorr signatures designed by NSA
- ECDSA
  - Elliptic curve variant of DSA

- Reference slides describing included at end of lecture

Certificate Authorities & Public-Key Infrastructure

Installed with client, e.g., browsers maintain list of CA keys

Verify that request emanating from amazon.com

(pk,sk)

VeriSign®

pk

M = (pk',data)
S = Sign(sk,M)

Give me a certificate for pk', please

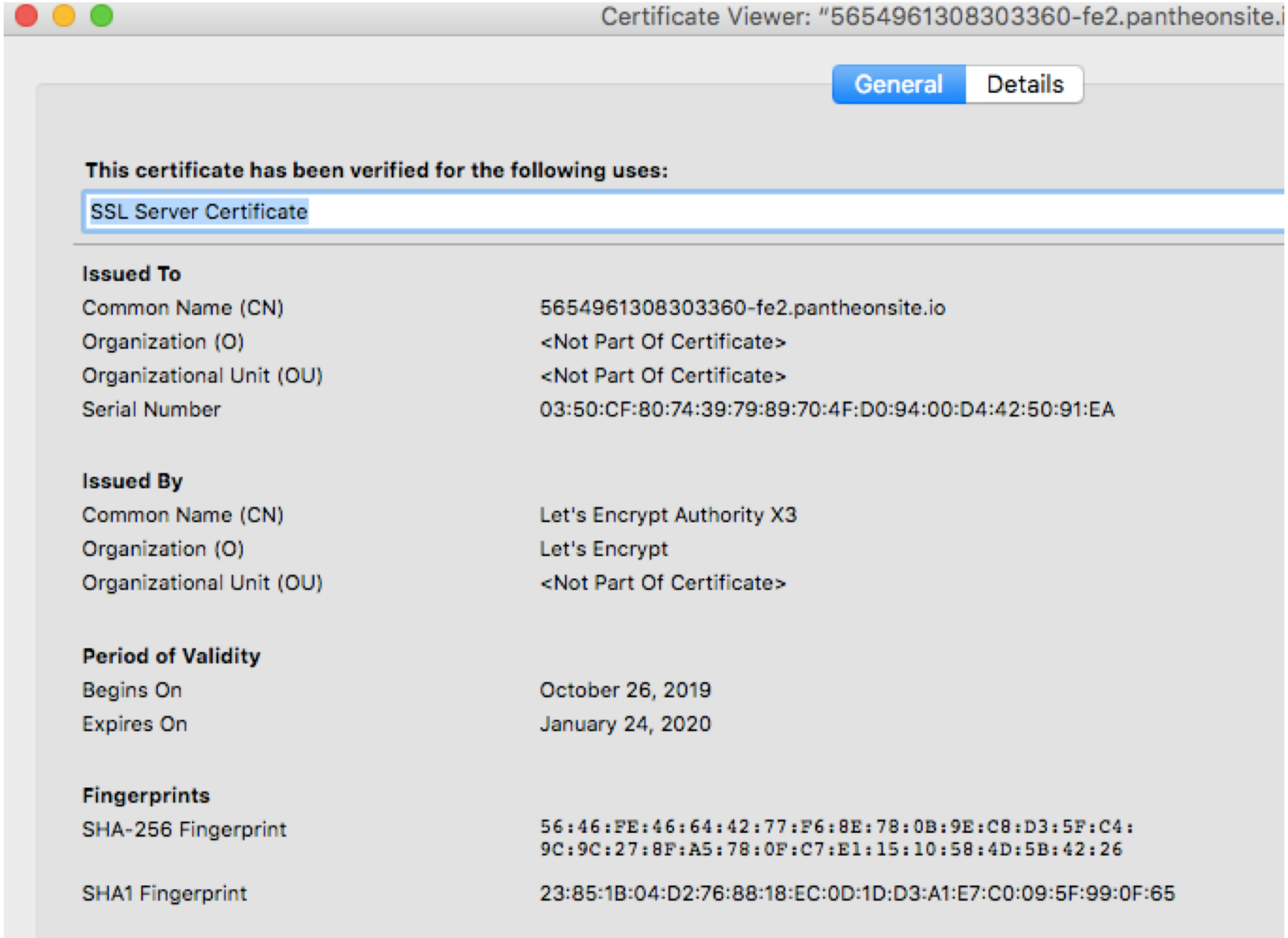
S

http://amazon.com

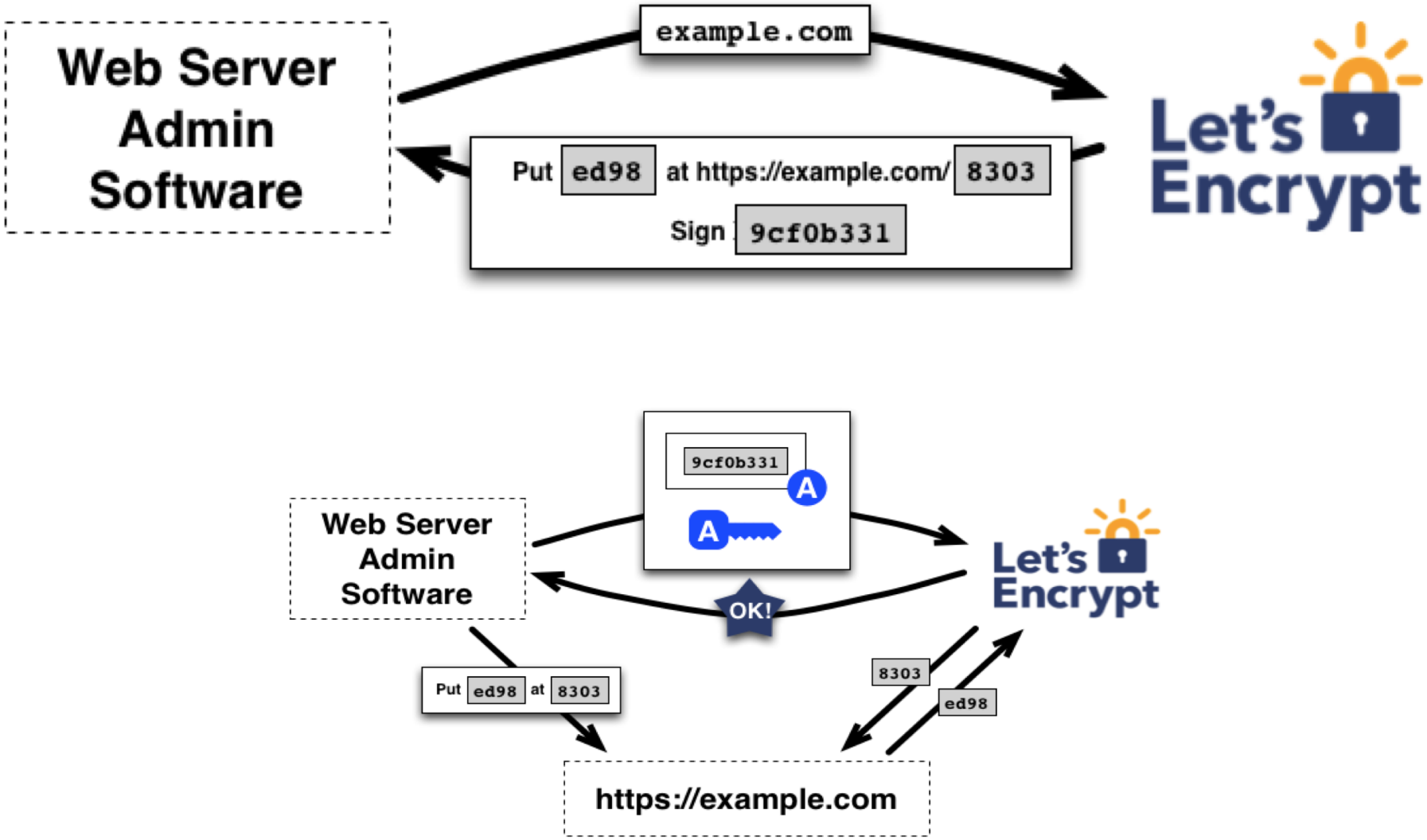
pk', data, S

M = (pk',data)
If Ver(pk,M,S) then
trust pk'

(pk',sk')


Certificate Viewer: "5654961308303360-fe2.pantheonsite.i
General
Details
This certificate has been verified for the following uses:
SSL Server Certificate
Issued To
Common Name (CN)
5654961308303360-fe2.pantheonsite.io
Organization (O)
<Not Part Of Certificate>
Organizational Unit (OU)
<Not Part Of Certificate>
Serial Number
03:50:CF:80:74:39:79:89:70:4F:D0:94:00:D4:42:50:91:EA
Issued By
Common Name (CN)
Let's Encrypt Authority X3
Organization (O)
Let's Encrypt
Organizational Unit (OU)
<Not Part Of Certificate>
Period of Validity
Begins On
October 26, 2019
Expires On
January 24, 2020
Fingerprints
SHA-256 Fingerprint
56:46:FE:46:64:42:77:F6:8E:78:0B:9E:C8:D3:5F:C4:
9C:9C:27:8F:A5:78:0F:C7:E1:15:10:58:4D:5B:42:26
SHA1 Fingerprint
23:85:1B:04:D2:76:88:18:EC:0D:1D:D3:A1:E7:C0:09:5F:99:0F:65

Free CAs


Web Server Admin Software
example.com
Put ed98 at https://example.com/ 8303
Sign 9cf0b331
Let's Encrypt
9cf0b331
A
A
OK!
Web Server Admin Software
Let's Encrypt
Put ed98 at 8303
8303
ed98
https://example.com

In-class exercise

Discuss with neighbor for 5 minutes:
What vulnerabilities would undermine server authentication in TLS?
Think about:

- Certificate authority protocols
- TLS handshake

Client implementation vulnerabilities
Theft of server secret signing key
Compromise of web server
CA compromise
CA identity check failure

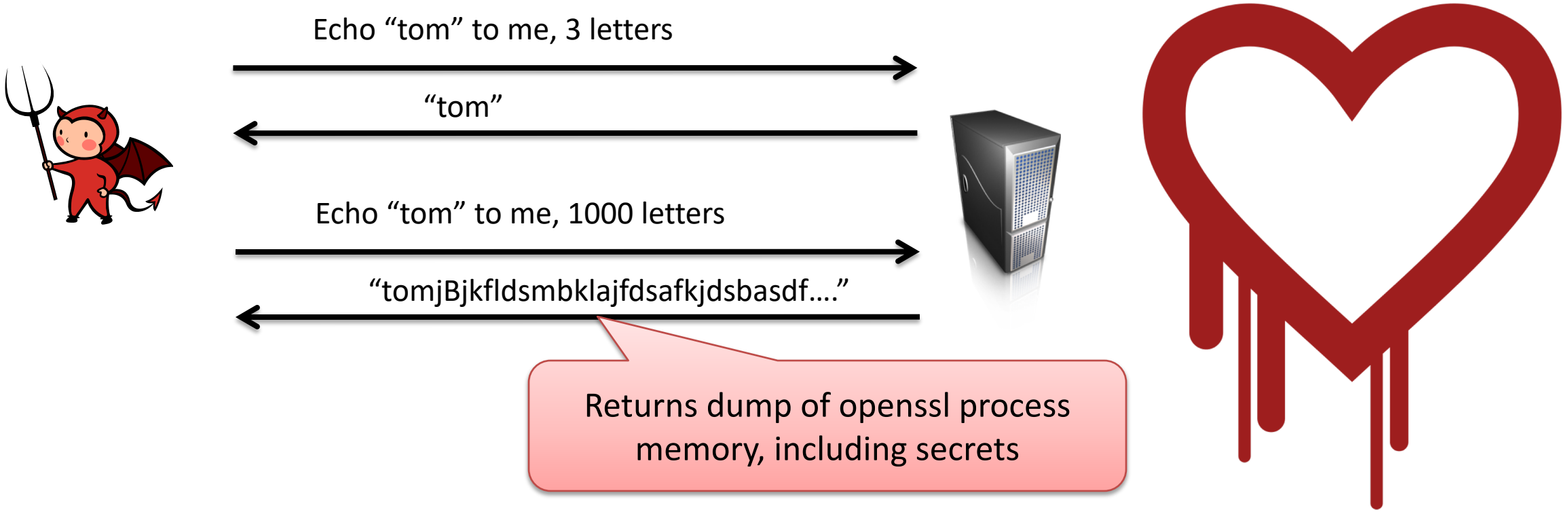
Client side vulnerability:
Apple iOS <7.0.16 signature verification code

```
    if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

        err = sslRawVerify(ctx,
                           ctx->peerPubKey,
                           dataToSign,                          /* plaintext */
                           dataToSignLen,                  /* plaintext length */
                           signature,
                           signatureLen);
        if(err) {
                sslErrorLog("SSLDecodeSignedServerKeyExchange: sslRawVerify "
                    "returned %d\n", (int)err);
                goto fail;
        }

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
```

Web server vulnerability: Heartbleed



Software vulnerability in openssl’s implementation of heartbeat protocol

Buffer overread vulnerability

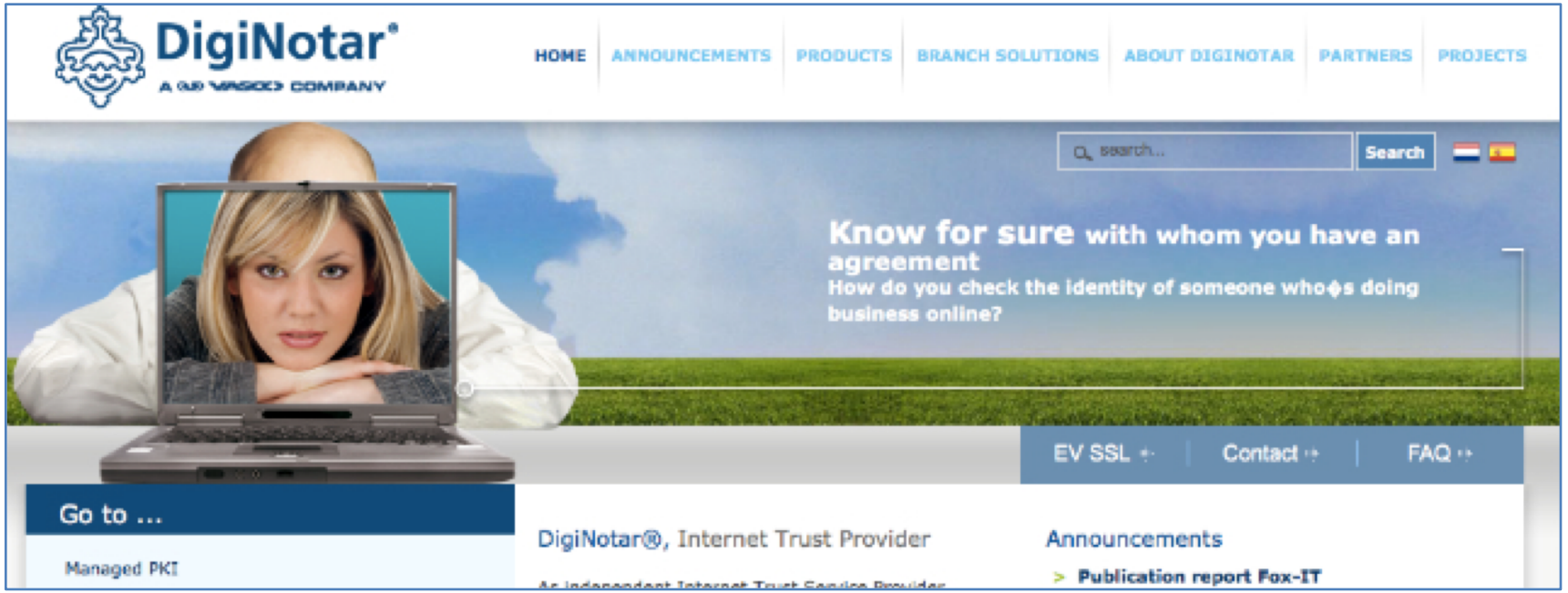
Allowed exfiltrating server signing keys

In code base 2012-2014

PKI ecosystem

- Durumeric et al. Analysis of the HTTPS Certificate Ecosystem, 2013
  - http://conferences.sigcomm.org/imc/2013/papers/imc257-durumericAemb.pdf

- “1,800 entities that are able to issue certificates vouching for the identity of any website”



Today, Microsoft issued a Security Advisory warning that fraudulent digital certificates were issued by the Comodo Certificate Authority. This could allow malicious spoofing of high profile websites, including Google, Yahoo! and Windows Live.

https://nakedsecurity.sophos.com/2011/03/24/fraudulent-certificates-issued-by-comodo-is-it-time-to-rethink-who-we-trust/

https://technet.microsoft.com/library/security/2524375

Revocation

- Certificates must often be revoked
  - Short expirations
  - CRLs (Certificate revocation lists)
  - OCSP (online certificate status protocol)
    - Client queries CA to check on validity of cert
      - privacy concerns, performance / scalability issues
    - Stapling: server periodically gets fresh, time-stamped OCSP signature from CA. Sends to clients

How well does revocation work?

- Liu et al. An End-to-End Measurement of Certificate Revocation in the Web's PKI, 2014
  - http://www.ccs.neu.edu/home/cbw/static/pdf/liu-imc15.pdf

- 8% of in-use certificates revoked

- Many browsers don't bother checking for revocation (accepting revoked certificates)

Certificate/public-key pinning

- Can we prevent MitM due to PKI being undermined?
- Client knows what cert/pk to expect, rejects otherwise
  - Pre-install some keys
  - HPKP (HTTP Public Key Pinning)
    - HTTP header that allows servers to set a hash of public key they will use

```
Public-Key-Pins:
    pin-sha256="d6qzRu9zOECb90Uez27xWltNsj0e1Md7GkYYkVoZWmM=";
    pin-sha256="LPJNul+wow4m6DsqxbninhsWHlwfp0JecwQzYpOLmCQ=";
    max-age=259200
```

https://developers.google.com/web/updates/2015/09/HPKP-reporting-with-chrome-46?hl=en

Certificate transparency

- Can we detect rogue certificates?
- Force CAs to log the certificates they sign in a public tamper-evident register
  - Experimental IETF standard

- Google has been pushing this
  - Chrome requires it for “extra validation” certs
  - DigiCert has implemented

Summary

- Digital signature schemes power PKI and verifying identities:
  - unforgeability under chosen message attack
  - RSA based schemes PKCS#1 1.5, FDH, PSS
  - Discrete-log based schemes include Schnorr, DSA, ECDSA

- Web PKI relies on various trust assumptions
  - Can be undermined in *many* ways
  - Need combination of cryptography plus other mechanisms

Groups for Schnorr and DSA Signatures

Let p be a large prime number

Let q be a prime such that q divides p-1

Example: p = 2q + 1 (so-called safe prime p)

Fix the group G = $\mathbf{Z}_p^*$ = {1,2,3,…, p-1}

Let g be generator of sub-group of order q:

$\{ g^0, g^1, g^2, \ldots, g^{q-1} \}$ subset of G

How to pick g?

$g = h^{(p-1)/q} \bmod p$ for some h and check $g \neq 1 \bmod p$

If so, try repeat with another h. Usually start with h = 2

(Variant of) Schnorr signatures

p,q,g specified

sk = x chosen randomly from $\mathbf{Z}_q$ pk = X = g^x

Sign(x, M)

r <-$ $\mathbf{Z}_q$

R = g^r ; c = H(M || R) ; z = r + cx mod q

Return (R,z)

Ver(X, M, (R,z))

c = H(M || R)

If $g^z = RX^c$ then Return 1

Return 0

Correctness? $g^z = g^{r + cx} = g^r g^{xc} = RX^c$

Security intuition

Assume an adversary that can output forgery (M,(R,z))
Then to be valid:

$$g^z = RX^c \quad \text{implies} \quad z = r + cx$$

for c = H(M || R) .
Assume c is random (H is random oracle)

Imagine we can run adversary twice but force forgery to be on same R, different c .
In second execution, getting (M',(R,z'))

Then success second time around gives:

$$g^{z'} = RX^{c'} \quad \text{implies} \quad z' = r + c'x$$

But now can compute z - z' / (c - c') = x the secret key

Fragility of signatures

Repeat randomness failure:

Sign two messages M ≠ M' and reuse randomness

Sign(x,M) -> (R, z) (R, r + cx mod q)

Sign(x,M') -> (R, z') (R, r + c'x mod q)

Then: $x = (z - z') / (H(M||R) - H(M'||R))$

If r is predictable/leaked, can recover secret from (R,z)

Can improve security by "hedging":

choose r = H(x || M || randomness)

Actual Schnorr signatures

p,q,g specified

sk = x chosen randomly from $\mathbf{Z}_q$ pk = X = g^x

Sign(x, M)

r <-$ $\mathbf{Z}_q$

R = g^r ; c = H(M || R) ; z = r + cx mod q

Return (c,z)

Ver(X, M, (c,z))

R' = $g^s X^{-c}$

c' = H(M || R')

If c' = c then Return 1

Return 0

Correctness? R' = $g^s X^{-c}$ = $g^{r + cx}$ $g^{x/(H(M||R))}$ = g^r

DSA (digital signature algorithm)

p,q,g specified

sk = x chosen randomly from $\mathbf{Z}_q$ pk = X = g^x

Sign(x, M)

r <-$ $\mathbf{Z}_q$; R = $(g^r \bmod p) \bmod q$

z = r^{-1} (H(M) + x R) mod q

Return (R,z)

Ver(X, M, (R,z))

w = z^{-1} mod q

u1 = H(m) * w mod q

u2 = R*w mod q

If R = $(g^{u1} X^{u2} \bmod p) \bmod q$

then Return 1

Else Return 0

Correctness?

$$g^{u1} X^{u2} = g^{H(M)\, w}\, g^{x\, R\, w} = g^{(H(M)+xR)\, w}$$

$$= g^{(H(M)+xR)\,(H(M)+xR)^{-1} r} = g^r$$

Fragility of DSA

Repeat randomness failure:

Sign two messages $M \neq M'$ and reuse random

$$\text{Sign}(x,M) \rightarrow (R,z) = (R, r^{-1}(H(M) + xR) \bmod q)$$
$$\text{Sign}(x,M') \rightarrow (R,z') = (R, r^{-1}(H(M') + xR) \bmod q)$$

Then: Solve for r^{-1}, solve for x

If r is predictable/leaked, can recover secret from (R,z)

Again, can improve security by "hedging":

choose r = H(x || M || randomness)

Hackers Describe PS3 Security As Epic Fail, Gain Unrestricted Access

BY MIKE BENDEL DECEMBER 29, 2010 @ 11:19 AM

http://psx-scene.com/forums/content/sony-s-ps3-security-epic-fail-videos-within-581/