Today in Cryptography (5830)

Digital signatures
Schnorr signatures, DSA
PKI

Katz-Lindell Chapter 12
Various PKI sources available on web (some references within slides)

Client

TLS handshake for Diffie-Hellman Key Exchange

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Pick random Ns

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

Pick random x

$X = g^x$

CERT = (pk_s , signature over it) ←

p , g , X , σ = Sign(sk_s, Nc||Ns||p||g||X) ←

Check CERT using CA public verification key

Check σ

Pick random y

$Y = g^y$

Y →

PMS = g^{xy}

PMS = g^{xy}

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

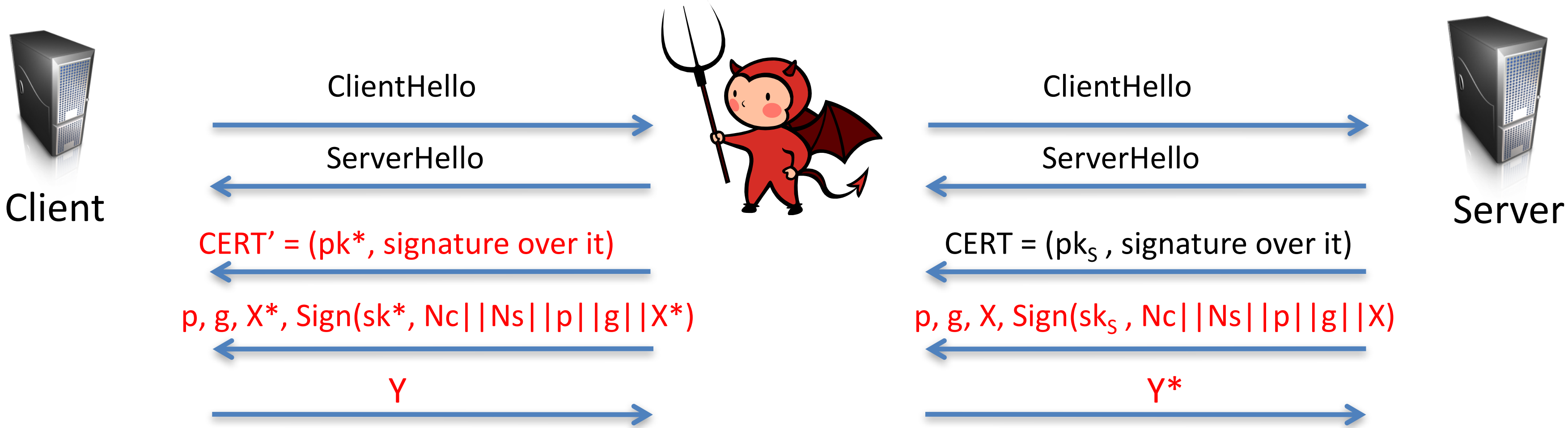
ChangeCipherSpec,
{ Finished, PRF(MS, “Server finished” || H(transcript’)) } ←

Bracket notation means contents encrypted

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

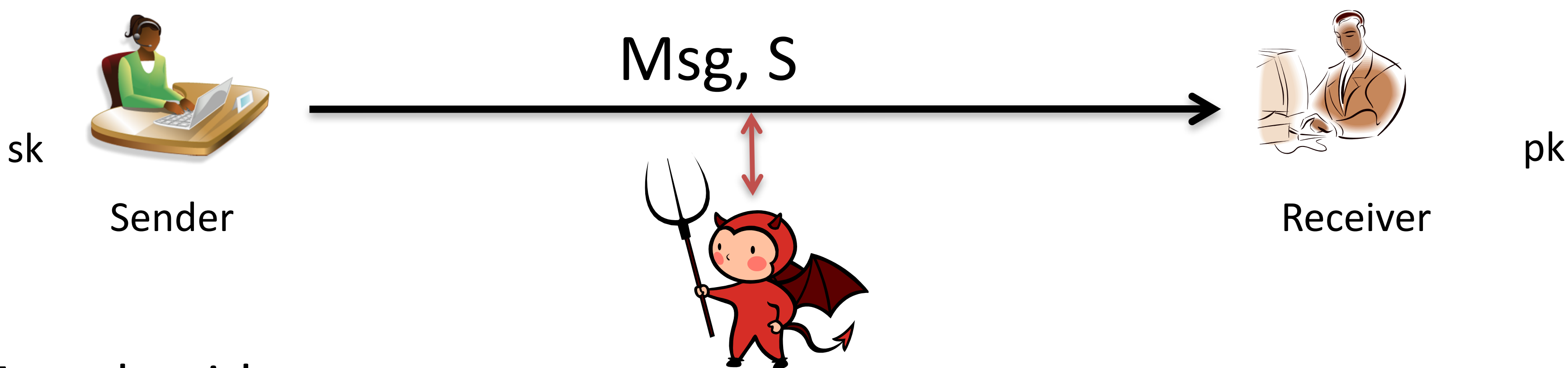
Man-in-the-middle attacks

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



<u>Two algorithms:</u>

(1) Key generation outputs (pk,sk)

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

(3) Verify(pk,Msg,S) outputs 0/1 (invalid / valid)

Correctness: Verify(pk,Msg,Sign(sk,Msg)) = 1 always

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

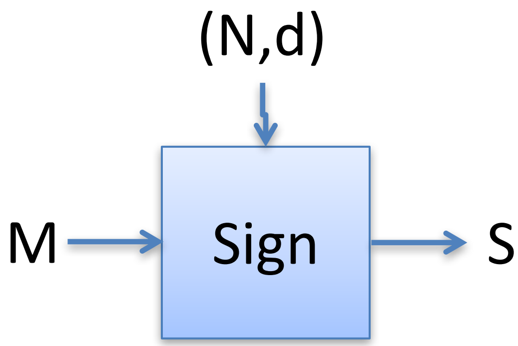
$$(\text{Msg}_1 , S_1) , (\text{Msg}_2 , S_2), \dots , (\text{Msg}_q , S_q)$$

for messages of his choosing and reasonably large q.

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)

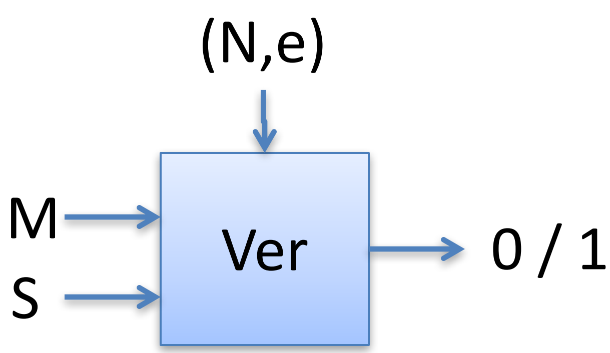


<u>Sign((N,d), M)</u>

X = 00 || H(1||M) || … || H(k||M)

S = X^d mod N

Return S



<u>Ver((N,e), M, S)</u>

X = S^e mod N

X' = 00 || H(1||M) || … || H(k||M)

If X = X' then

    Return 1

Return 0

Probabilistic Signature Scheme (PSS) provides stronger security bounds and also deployed now, see PKCS#1 v2

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 $G = \mathbf{Z}_p^* = \{1,2,3,\ldots, p-1\}$
Let g be generator of sub-group of order q:

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

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 use $h = 2$

Schnorr signatures

p,q,g specified

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

<u>Sign(x, M)</u>

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

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

Return (R,z)

<u>Ver(X, M, (R,z))</u>

c = H(M || R)

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

Return 0

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

Schnorr signatures

p,q,g specified

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

<u>Sign(x, M)</u>

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

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

Return (c,z)

<u>Ver(X, M, (c,z))</u>

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

c' = H(M || R)

If c' = c then Return 1

Return 0

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

Security of Schnorr signatures

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 Schnorr

Repeat randomness failure:

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

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)

DSA (digital signature algorithm)

p,q,g specified

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

<u>Sign(x, M)</u>

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

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

Return (R,z)

<u>Ver(X, M, (R,z))</u>

w = $z^{-1} \bmod q$

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

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

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

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)

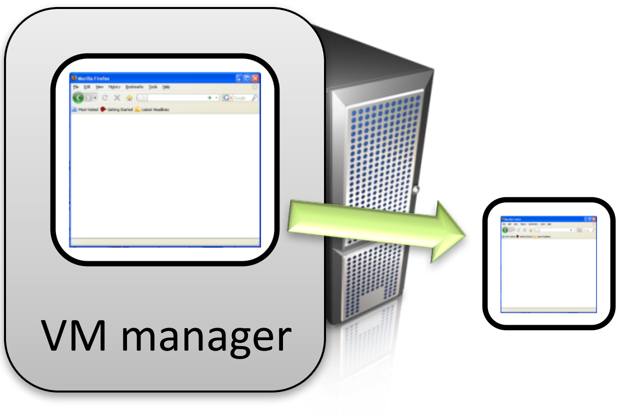
Hackers Describe PS3 Security As Epic Fail, Gain Unrestricted Access

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

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

Another example randomness reuse: Virtual machines reset vulnerabilities

Virtual machine (VM) encapsulates entire guest operating system and (virtualized) hardware resources



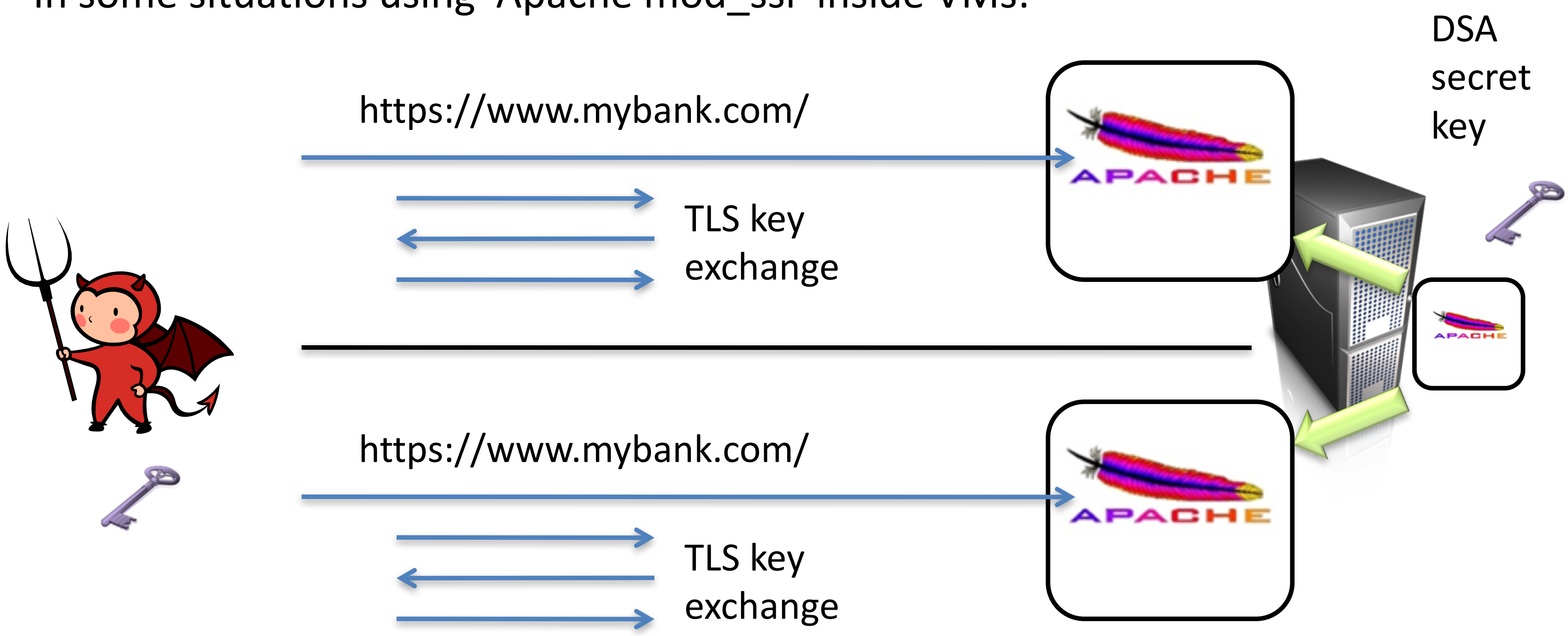
VM snapshots save entire state (memory, persistant storage, etc.) of a VM

Backup   Migration   Replication   Fault or intrusion recovery

Volume snapshots (as used in EC2) only save persistant storage. OS must be booted

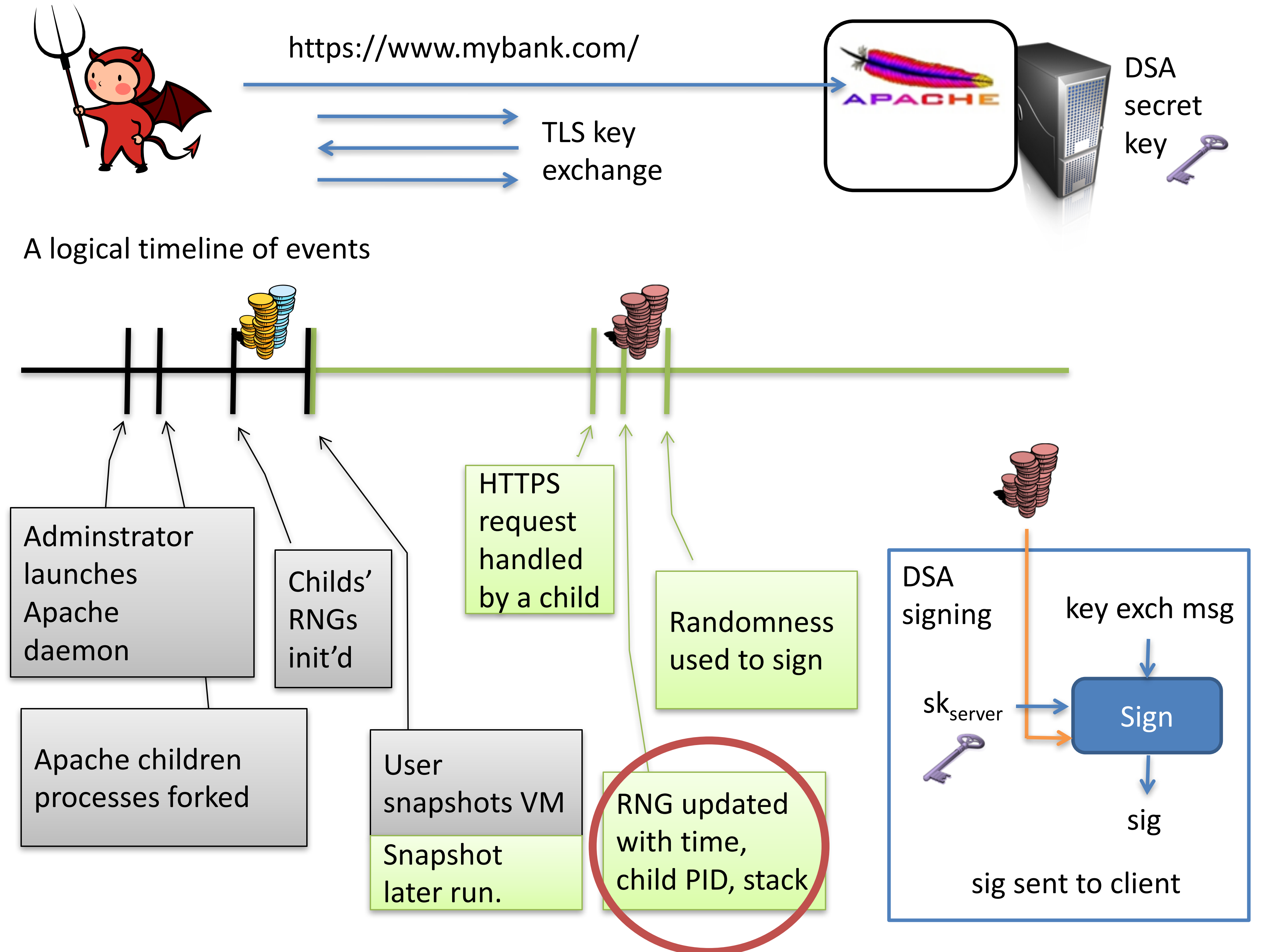
Another example randomness reuse: Virtual machines reset vulnerabilities

In some situations using Apache mod_ssl inside VMs:

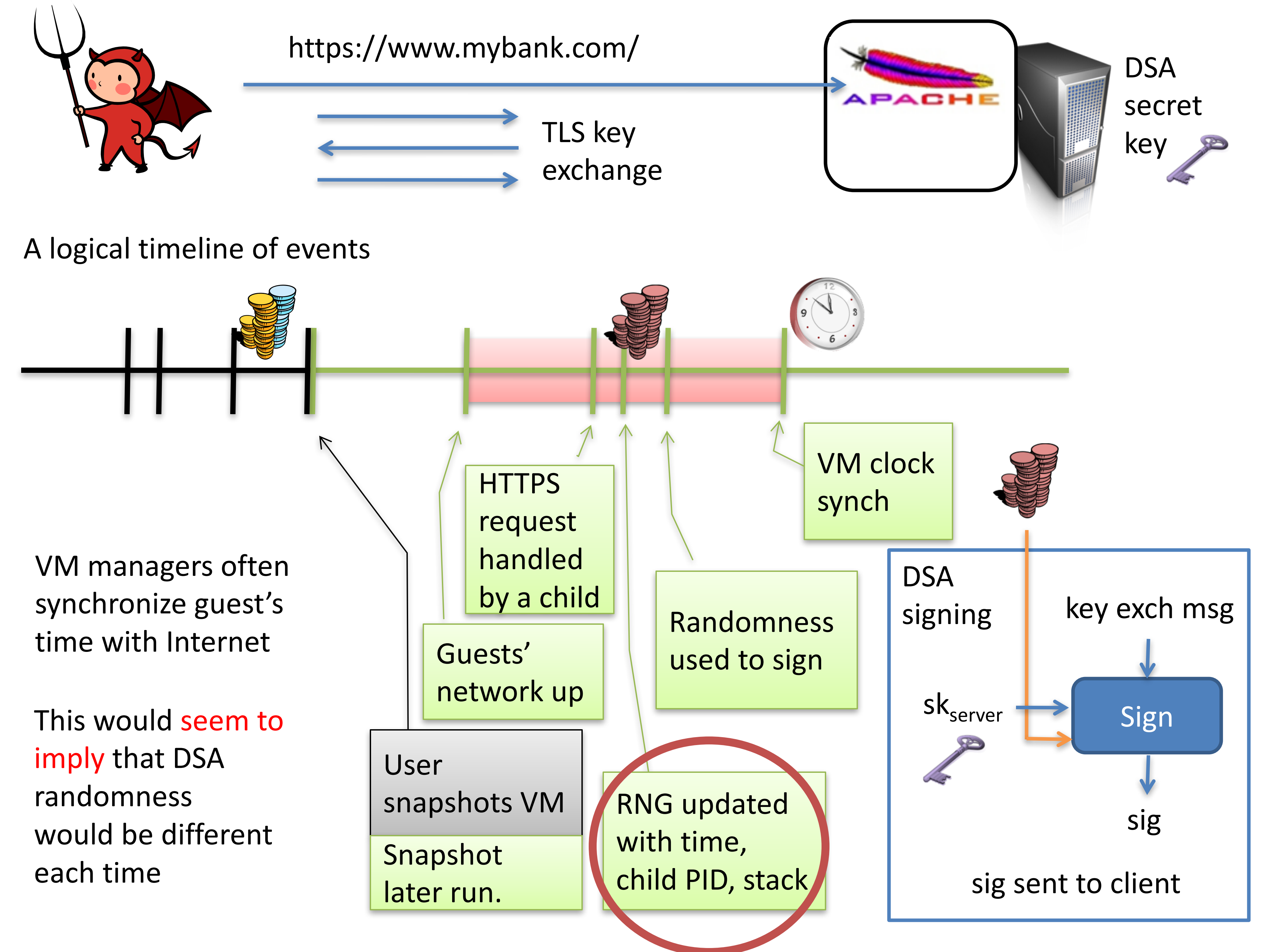


Key extraction is possible

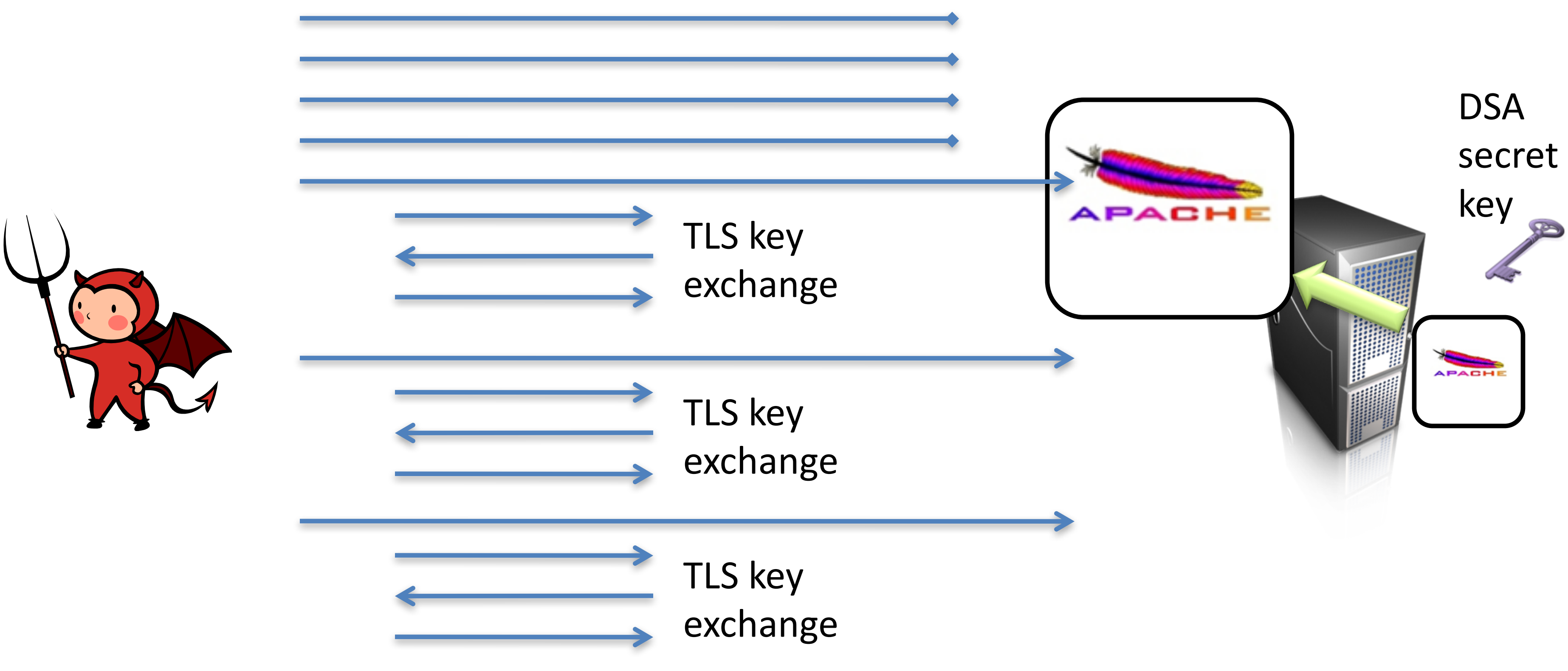
DSA secret key allows impersonating server



https://www.mybank.com/
APACHE
DSA
secret
key
TLS key
exchange
A logical timeline of events
Adminstrator
launches
Apache
daemon
Childs'
RNGs
init'd
Apache children
processes forked
User
snapshots VM
Snapshot
later run.
HTTPS
request
handled
by a child
RNG updated
with time,
child PID, stack
Randomness
used to sign
DSA
signing
key exch msg
sk_server
Sign
sig
sig sent to client


https://www.mybank.com/
APACHE
TLS key exchange
DSA secret key
A logical timeline of events
HTTPS request handled by a child
VM clock synch
Guests' network up
Randomness used to sign
User snapshots VM
Snapshot later run.
RNG updated with time, child PID, stack
DSA signing
key exch msg
sk$_{server}$
Sign
sig
sig sent to client
VM managers often synchronize guest's time with Internet
This would seem to imply that DSA randomness would be different each time

Experimenting with DSA key extraction



This is one trial.
We performed 5 trials for each VMM without rebooting physical server
We performed 5 trials for each VMM with rebooting physical server
Looked for reuse of randomness across pairs of successful connections

Experimenting with DSA key extraction

VMM	Time sync?	Always reboot physical machine?	# pairs w/ repeat sesion IDs	# pairs w/ DSA key extractable
VirtualBox	Yes	No	10/10	10/10
VirtualBox	Yes	Yes	10/10	10/10
VMWare	Yes	No	0/10	0/10
VMWare	Yes	Yes	4/10	3/10
VMWare	No	No	6/10	6/10
VMWare	No	Yes	3/10	1/10

Digital signature schemes

	Problems?	Proofs?	Uses
RSA PKCS#1 v1.5	Bleichanbacher attacks		TLS, Certificates, XML
RSA PSS (PKCS#1 v2)		Security reduces to hardness of inverting RSA	
Schnorr	Randomness fragility	Security reduces to discrete log problem	
DSA	Randomness fragility	No security reduction	Bitcoin (ECC version), TLS, SSH, elsewhere

Measurement studies of TLS ecosystem

- Use internet-wide scanning to measure ecosystem

  Fewer than 2^{32} IP addresses: ~3,706,452,992
  Exclude private addresses (e.g., 192.168.0.0/16), multicast addresses, ...

- Zmap is state-of-art tool: can scan one port of entire Internet in ~45 minutes
- Extend to perform TLS handshakes with servers that are open on port 443

TLS handshake for Diffie-Hellman Key Exchange


Client


Server

Client		Server
Pick random Nc	ClientHello, MaxVer, Nc, Ciphers/CompMethods →	
Check CERT using CA public verification key	← ServerHello, Ver, Ns, SessionID, Cipher/CompMethod	Pick random Ns
	← CERT = (pk_s , signature over it)	Pick random x $X = g^x$
Check σ	← p , g , X , σ = Sign(sk_s, Nc||Ns||p||g||X)	
Pick random y $Y = g^y$	Y →	PMS = g^{xy}
PMS = g^{xy}	ChangeCipherSpec, { Finished, PRF(MS, “Client finished” || H(transcript)) } →	
	← ChangeCipherSpec, { Finished, PRF(MS, “Server finished” || H(transcript’)) }	

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

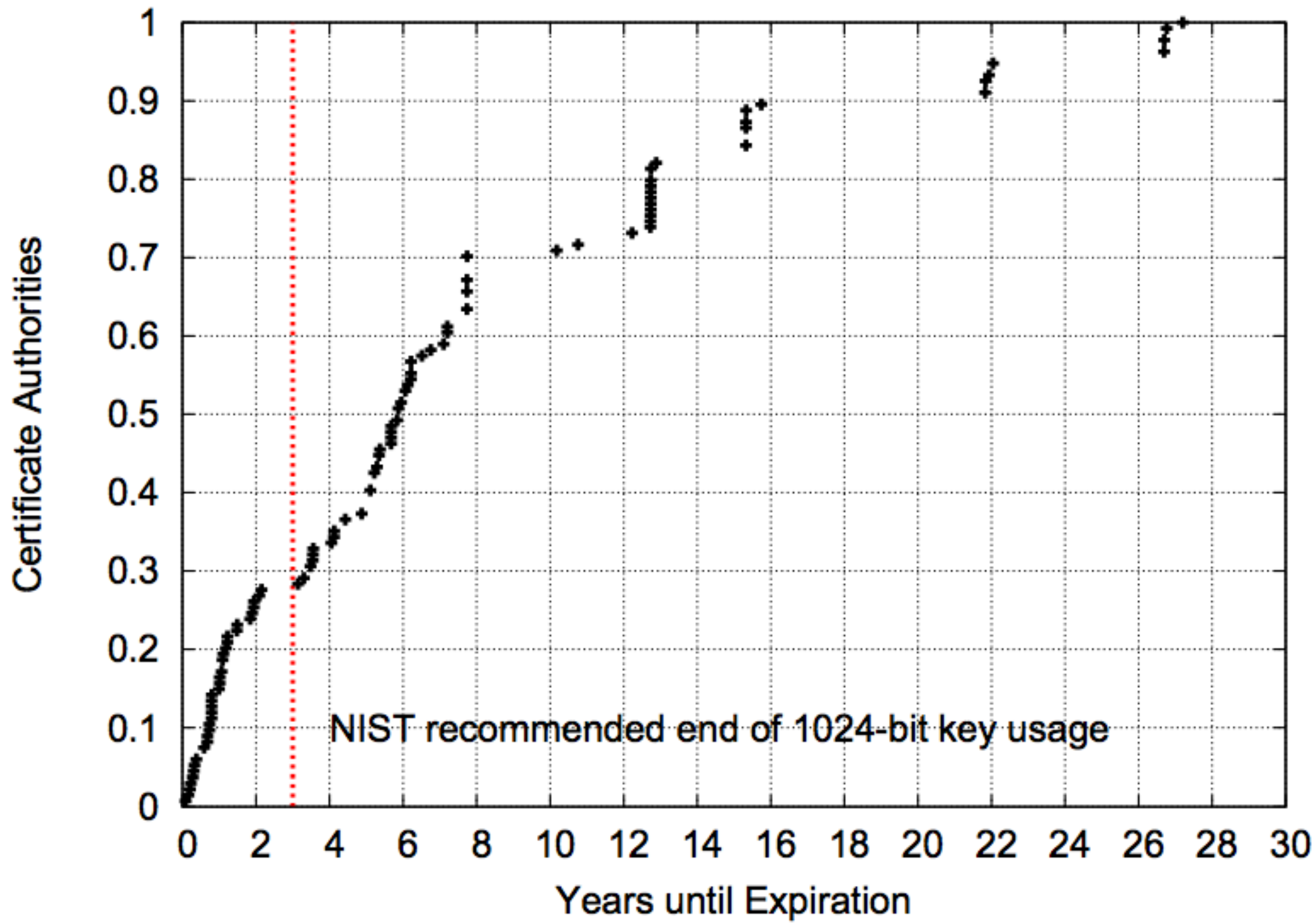
Some scans

- 2011 scan (Heninger et al.)
  - 12,828,613 certificate chains recovered
  - 5,656,519 distinct RSA public keys
  - 6,241 distinct DSA public keys
  - Found many *factorable* RSA keys, DSA keys signing with repeat randomness
- 2013 scan (Durumeric et al.)
  - Almost all CA's using RSA keys for signing certs
  - ~8 million unique certificates across ~30 million hosts

2013 scan: breakdown by CA

Parent Company	Signed Leaf Certificates	
Symantec	1,184,723	(34.23%)
GoDaddy.com	1,008,226	(29.13%)
Comodo	422,066	(12.19%)
GlobalSign	170, 006	(4.90%)
DigiCert Inc	145,232	(4.19%)
StartCom Ltd.	88,729	(2.56%)
Entrust, Inc.	76,990	(2.22%)
Network Solutions	62,667	(1.81%)
TERENA	42,310	(1.22%)
Verizon Business	32,127	(0.92%)

2013 scan: 1024-bit RSA CA keys



Turktrust incident

- Turktrust is a CA used by Turkish government
- December, 2012 detected an erroneous CERT for *.google.com signed by Turktrust
  - Gives whoever owned private key ability to MITM most Google websites

Summary

- Schnorr and DSA allow compact signatures (256 bits), but are fragile without hedging
  - Based on difficulty of computing discrete log
- Certificate authority system is big and complicated
  - Lots of security problems arise

Certificate Authorities and Public-key Infrastructure

(pk,sk)

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

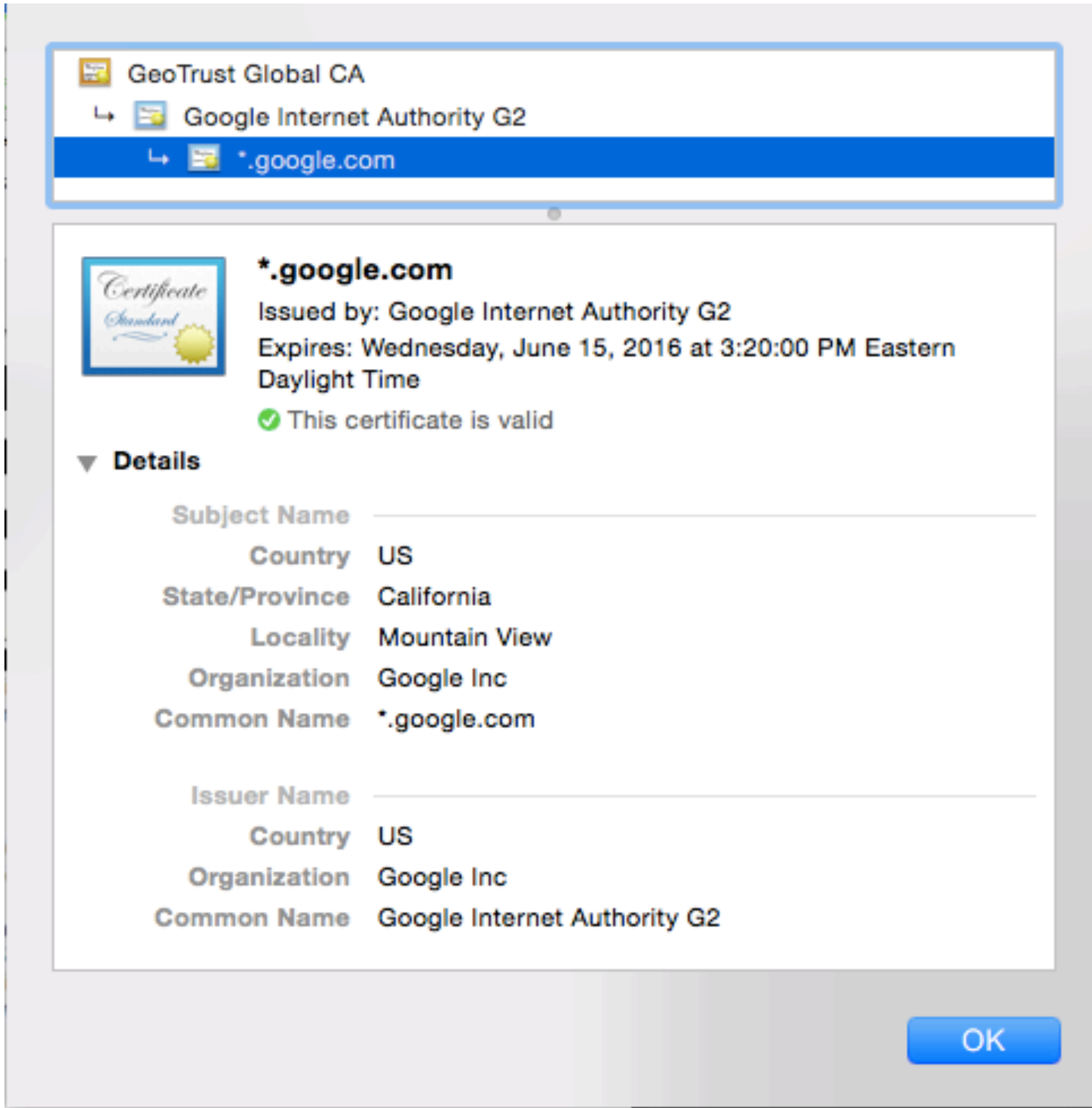
trust pk'

(pk',sk')

This prevents man-in-the-middle (MitM) attacks

```
Certificate:
   Data:
       Version: 1 (0x0)
       Serial Number: 7829 (0x1e95)
       Signature Algorithm: md5WithRSAEncryption
       Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
               OU=Certification Services Division,
               CN=Thawte Server CA/emailAddress=server-certs@thawte.com
       Validity
           Not Before: Jul  9 16:04:02 1998 GMT
           Not After : Jul  9 16:04:02 1999 GMT
       Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
                OU=FreeSoft, CN=www.freesoft.org/emailAddress=baccala@freesoft.org
       Subject Public Key Info:
           Public Key Algorithm: rsaEncryption
           RSA Public Key: (1024 bit)
               Modulus (1024 bit):
                   00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
                   33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
                   66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
                   70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
                   16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
                   c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
                   8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
                   d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
                   e8:35:1c:9e:27:52:7e:41:8f
               Exponent: 65537 (0x10001)
   Signature Algorithm: md5WithRSAEncryption
       93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
       92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
       ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
       d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
       0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
       5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
       8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
       68:9f
```

Certificate chains



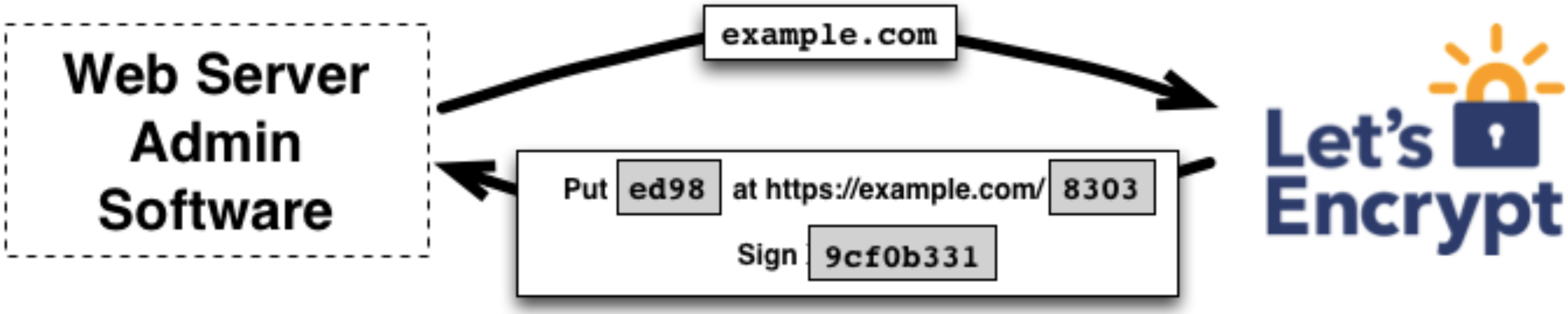
Identity checks?

- CA’s must check that requestor of cert is who they say they are

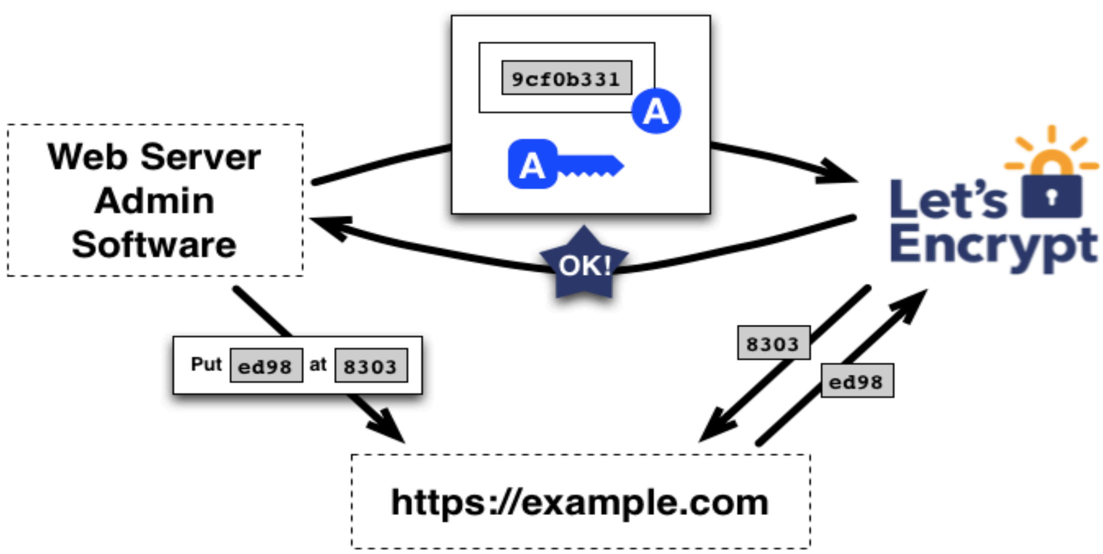
- Domain validated
  - Prove ownership of domain

- Extended validation
  - Establish legal identity of requestor
  - Physical presence of website owner
  - Confirm ownership of domain
  - Etc.

Free CAs



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



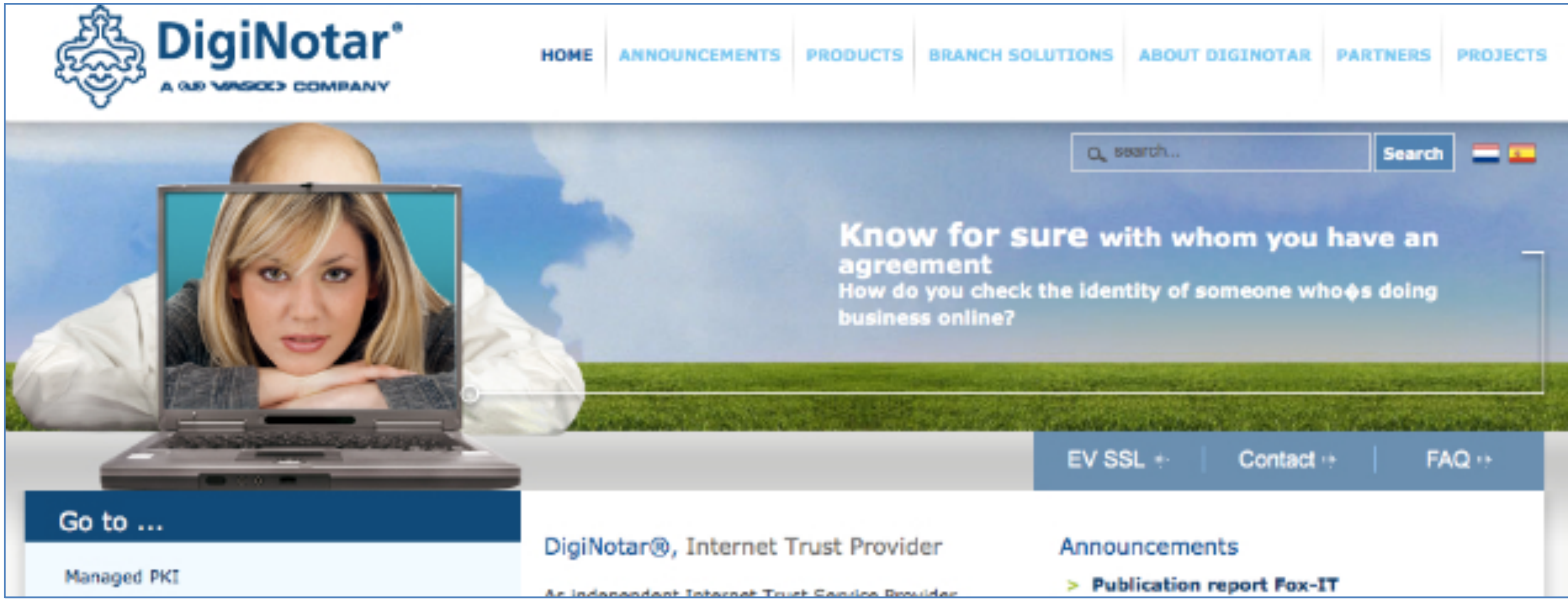
9cf0b331
A
A
Web Server
Admin
Software
OK!
Let's Encrypt
Put ed98 at 8303
8303
ed98
https://example.com

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

The Web PKI Ecosystem

- http://conferences.sigcomm.org/imc/2013/papers/imc257-durumericAemb.pdf
- ~1800 CAs that can sign *any* domain controlled by 683 organizations



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

Certificate/public-key pinning

- 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

- 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

- Web PKI relies on various trust assumptions
  - Can be undermined in many ways
- Digital signature schemes power PKI and verifying identities:
  - unforgeability under chosen message attack
  - RSA based schemes PKCS#1 1.5 and 2.0
  - Schnorr, DSA based on discrete log problem