Passwords, RNGs, Implementation issues

CS642: Computer Security

Professor Ristenpart

http://www.cs.wisc.edu/~rist/

rist at cs dot wisc dot edu

More topics in crypto

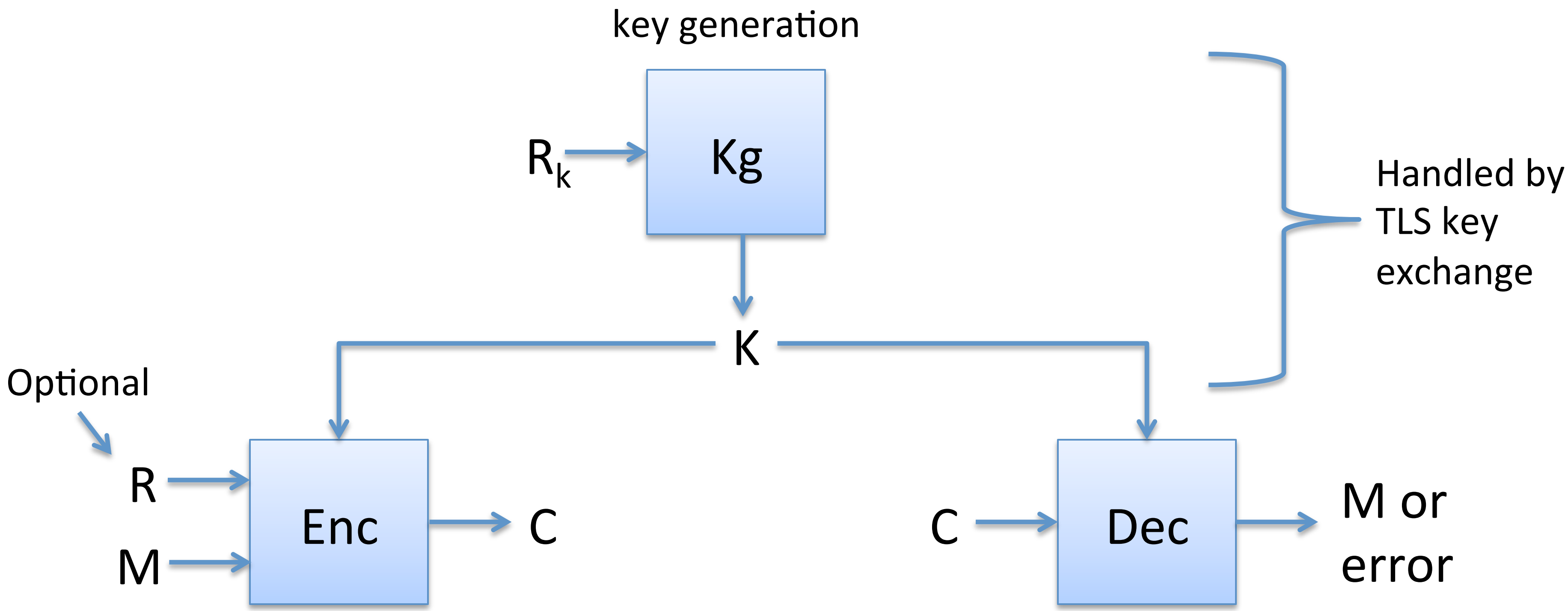
Password-based crypto

Password cracking, WPA

Random number generators (RNGs)

Side channel attacks

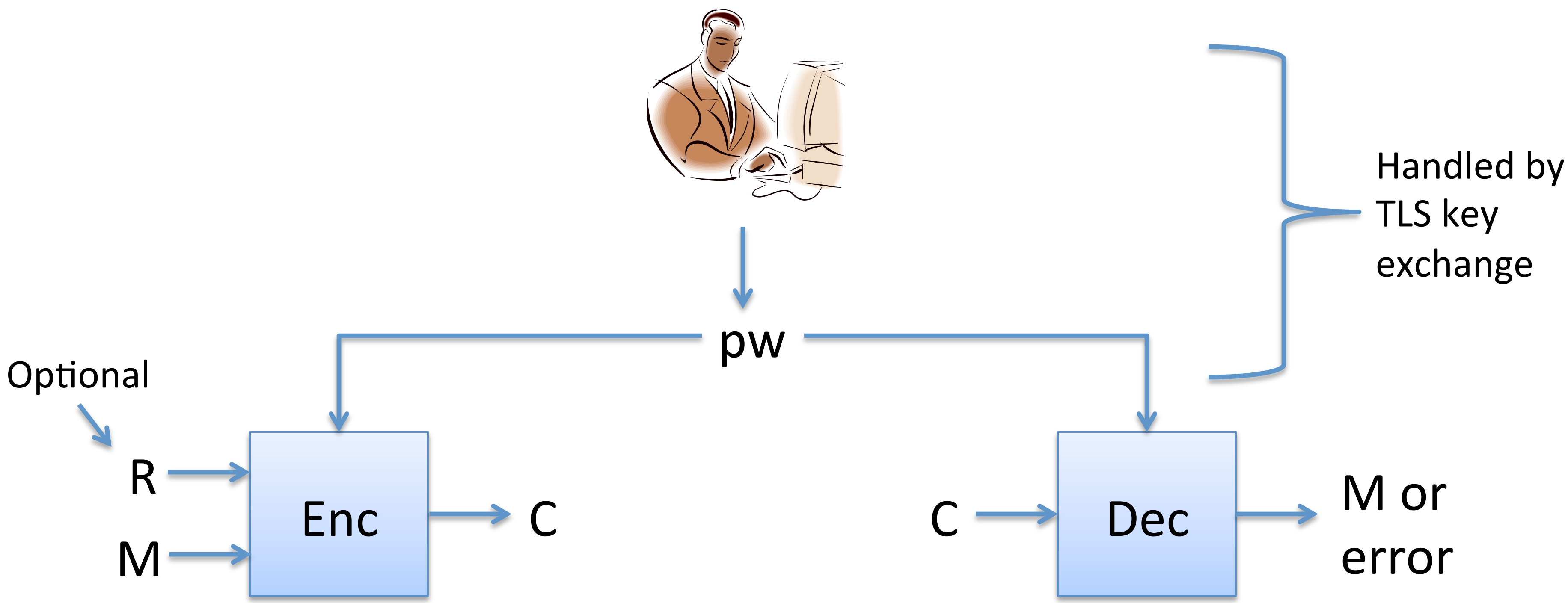
Symmetric encryption



C is a ciphertext

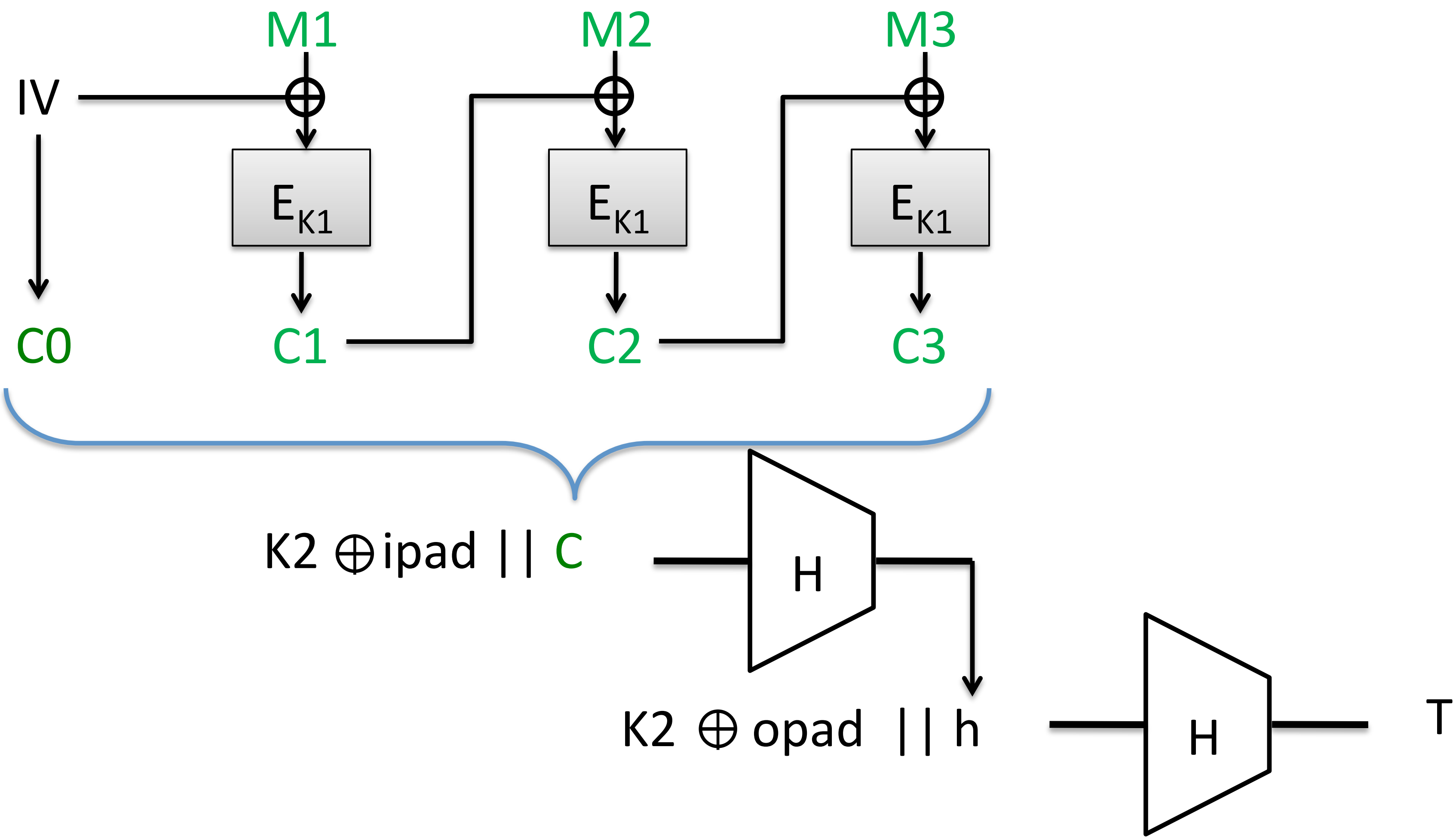
Correctness: D(K , E(K,M,R)) = M with probability 1 over randomness used

Password-based symmetric encryption



C is a ciphertext

Correctness: D(K , E(K,M,R)) = M with probability 1 over randomness used

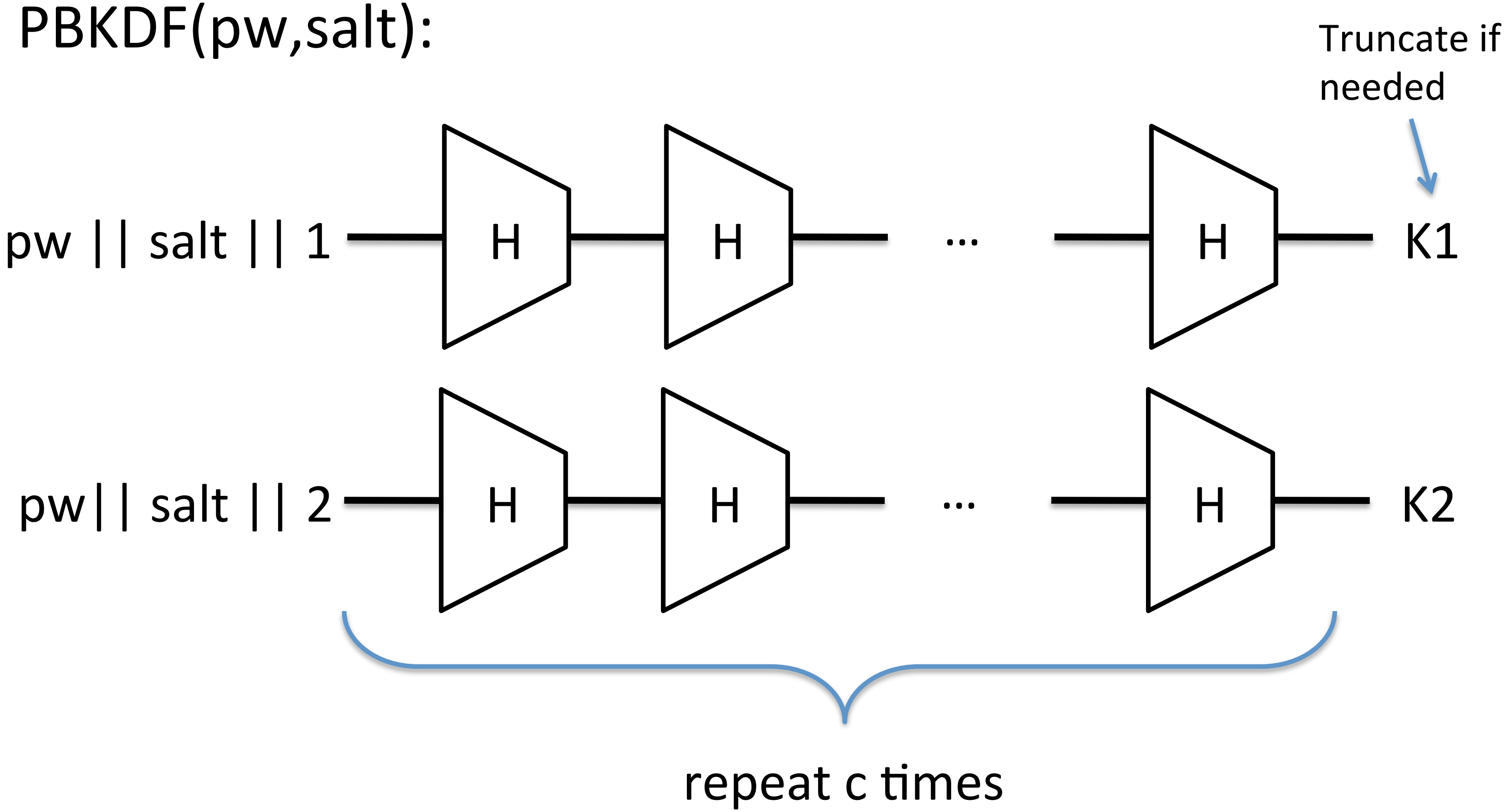


Ciphertext is C,T

How do we use with a pw?

Password-based Key Deriviation (PBKDF)

PBKDF(pw,salt):



PBKDF + Symmetric encryption = PW-based encryption

<u>Enc(pw,M,R):</u>
salt || R' = R
K = PBKDF(pw,salt)
C = Enc'(K,M,R')
Return (salt,C)

<u>Dec(pw,C):</u>
salt || C' = C
K = PBKDF(pw,salt)
M = Enc'(K,C')
Return M

Here Enc' is a normal symmetric encryption scheme (CBC-HMAC)

What can go wrong?

Password Popularity – Top 20

Rank	Password	Number of Users with Password (absolute)
1	123456	290731
2	12345	79078
3	123456789	76790
4	Password	61958
5	iloveyou	51622
6	princess	35231
7	rockyou	22588
8	1234567	21726
9	12345678	20553
10	abc123	17542

Rank	Password	Number of Users with Password (absolute)
11	Nicole	17168
12	Daniel	16409
13	babygirl	16094
14	monkey	15294
15	Jessica	15162
16	Lovely	14950
17	michael	14898
18	Ashley	14329
19	654321	13984
20	Qwerty	13856

From an Imperva study of released RockMe.com password database

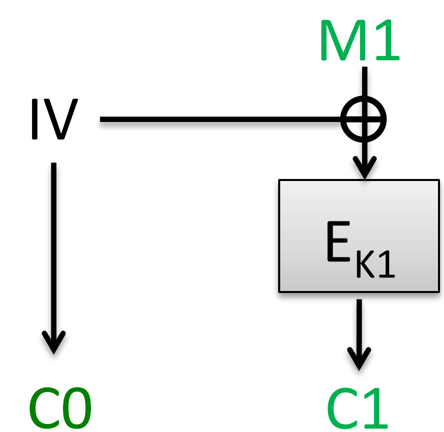
Brute-force attacks

- Given known plaintext, ciphertext pair:
  - M and C = Enc(pw,M)
- Enumerate a dictionary D of possible passwords

R is salt||IV in CBC-based modes

Both are public:
C = salt||IV||C1||…

```
BruteForce1(M,C):
foreach pw* in D do
    C* = Enc(pw*,M,R)
    If C* = C then
        Return pw*
```



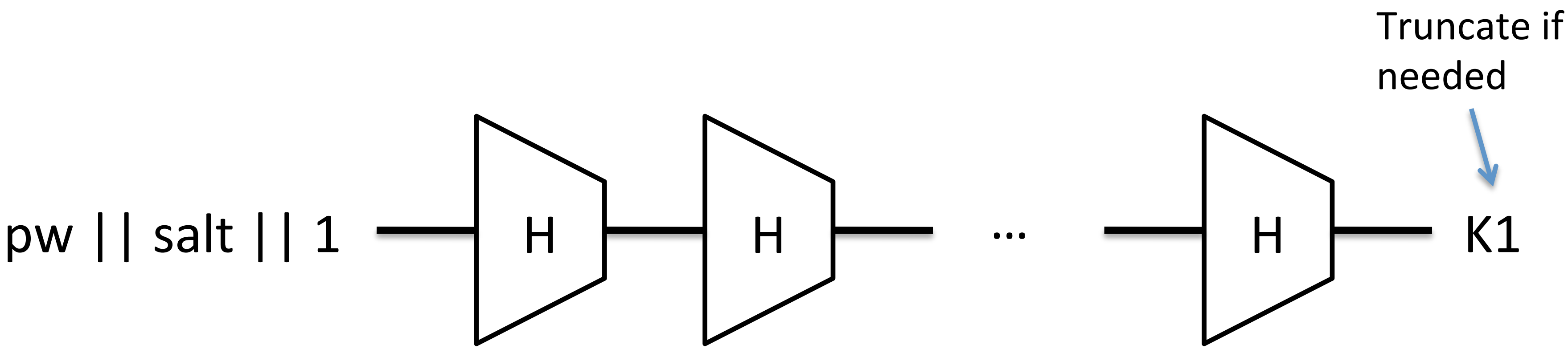
Brute-force attacks

- Given known plaintext, ciphertext pair:
  - M and C = Enc(pw,M)
- Enumerate a dictionary D of possible passwords

```
BruteForce1(M,C):
foreach pw* in D do
     C* = Enc(pw*,M,R)
     If C* = C then
          Return pw*
```

```
BruteForce2(C):
foreach pw* in D do
     M* = Dec(pw*,C)
     If M* looks right then
          Return (pw*,M*)
```

PBKDF design attempts to slow down brute-force attacks



Iterating c times should slow down attacks by factor of c

Salts:

- Different derived keys, even if same password
- Slows down attacks against multiple users
- Prevents precomputation attacks, if salts chosen correctly

```
rist@seclab-laptop1:~/work/teaching/642-fall-2011/slides$ openssl speed sha1
Doing sha1 for 3s on 16 size blocks: 4109047 sha1's in 3.00s
Doing sha1 for 3s on 64 size blocks: 3108267 sha1's in 2.99s
Doing sha1 for 3s on 256 size blocks: 1755265 sha1's in 3.00s
Doing sha1 for 3s on 1024 size blocks: 636540 sha1's in 3.00s
Doing sha1 for 3s on 8192 size blocks: 93850 sha1's in 3.00s
OpenSSL 1.0.0d 8 Feb 2011
```

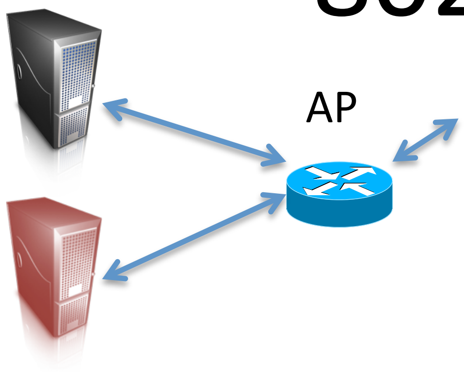
```
rist@seclab-laptop1:~/work/teaching/642-fall-2011/slides$ openssl speed aes-128-
cbc
Doing aes-128 cbc for 3s on 16 size blocks: 27022606 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 64 size blocks: 6828856 aes-128 cbc's in 2.99s
Doing aes-128 cbc for 3s on 256 size blocks: 1653364 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 1024 size blocks: 438909 aes-128 cbc's in 2.99s
Doing aes-128 cbc for 3s on 8192 size blocks: 54108 aes-128 cbc's in 3.00s
OpenSSL 1.0.0d 8 Feb 2011
```

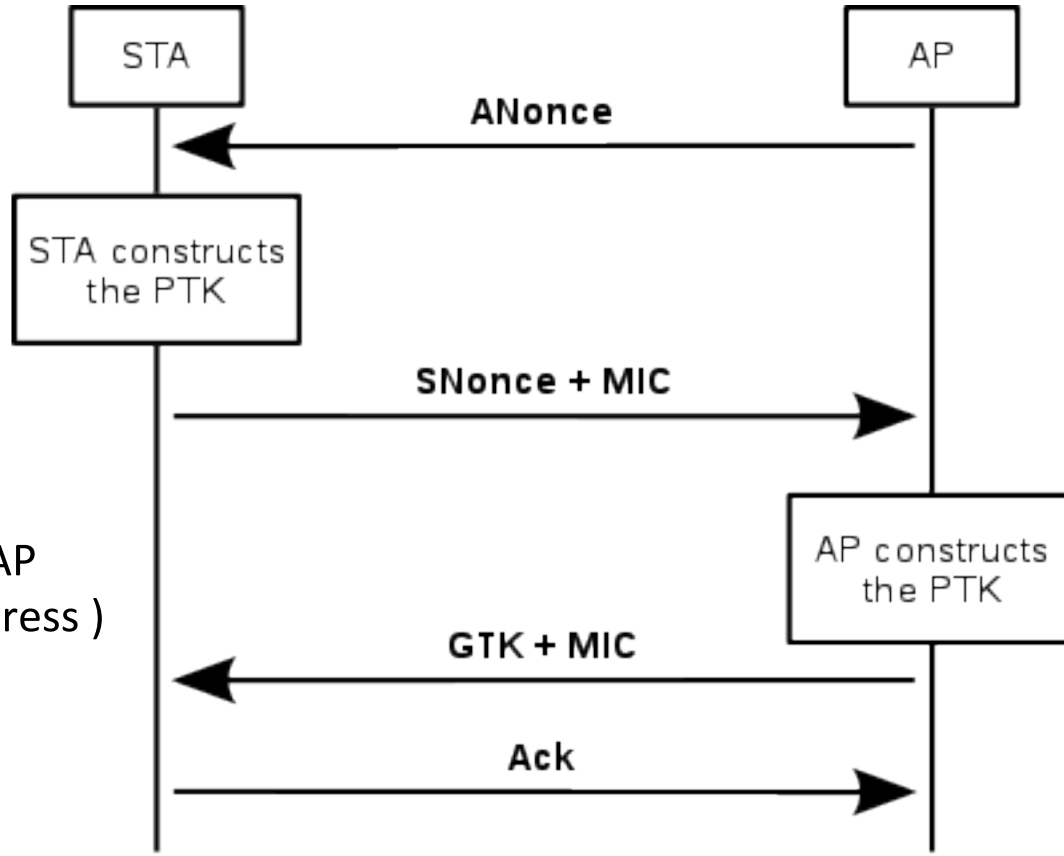
Say c = 4096. Generous back of envelope* suggests that in 1 second, can test 252 passwords and a naïve brute-force:

6 numerical digits	10^6 = 1,000,000	~ 3968 seconds
6 lower case alphanumeric digits	36^6 = 2,176,782,336	~ 99 days
8 alphanumeric + 10 special symbols	72^8 = 722,204,136,308,736	~ 33million days

* I did the arithmetic…

802.11 WPA passwords





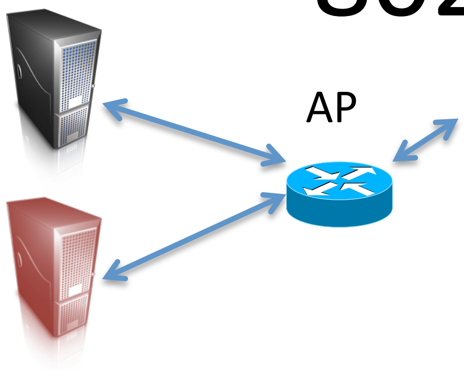
PMK = PBKDF(pw, ssid||ssidlength)
with c = 4096

PTK = H(PMK || ANonce || SNonce || AP MAC address || STA MAC address)

MIC = HMAC-MD5(PTK, 2nd message)

So after sniffing one handshake by another party, we can mount offline brute force attack

802.11 WPA passwords



PMK = PBKDF(pw, ssid||ssidlength)
with c = 4096

PTK = H(PMK || ANonce || SNonce || AP MAC address || STA MAC address)

MIC = HMAC-MD5(PTK, 2nd message)

<u>BruteForce2(MIC,ANonce,SNonce,2nd message):</u>

foreach pw* in D do
    PMK* = PBKDF(pw*,ssid||ssidlength)
    PTK* = H(PMK* ||ANonce || …)
    MIC* = HMAC-MD5(PTK*, 2nd message)
    If MIC* = MIC then
        Return pw*

We can also use precomputation for common SSID's

PMK = F(pw,ssid)

- PMK = PBKDF(pw, ssid||ssidlength)
  with c = 4096

MIC = G(PMK,data)

- PTK =
  H(PMK || ANonce || SNonce || AP MAC address || STA MAC address)
- MIC = HMAC-MD5(PTK, 2nd message)

<u>Online(D,SsidList):</u>

```
foreach pw* in D do
    foreach ssid* in Ssidlist do
        PMK* = F(pw*,ssid*)
        T[PMK*] = pw*
        Add PMK* to P[ssid*]
Return P,T
```

<u>Online(P,T,MIC,ANonce, …):</u>

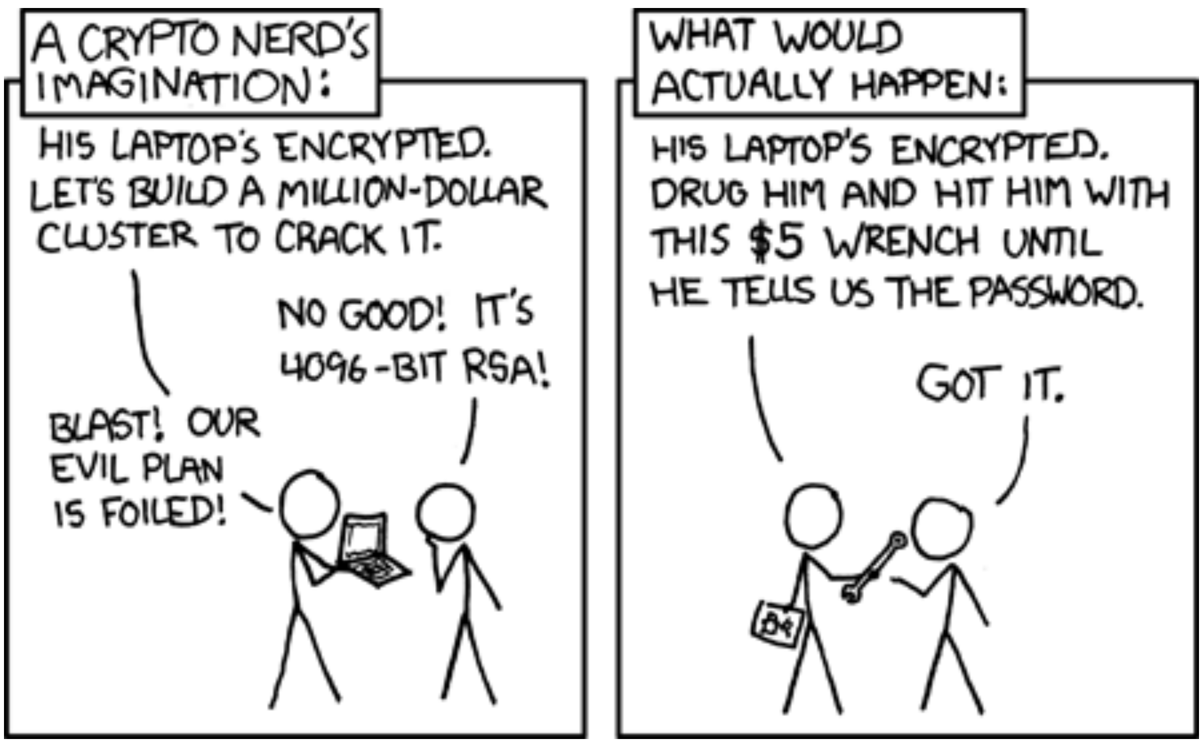
```
foreach PMK* in P[ssid] do
    MIC* = G(PMK*,data)
    If MIC* = MIC then
        Return T[PMK*]
```

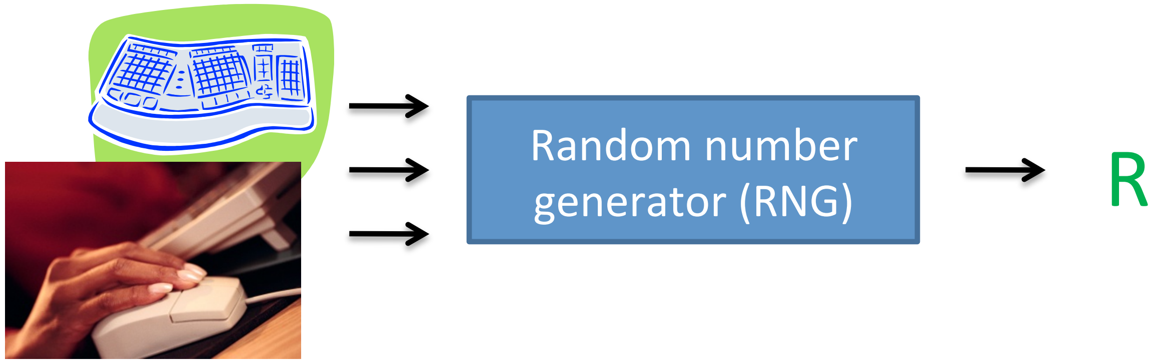
Time-space trade-off

Password recap

- Short passwords can be cracked easily (JohnTheRipper, aircrack)
- Salting and iteration help
  - Salts must be sufficiently large and unpredictable

From xkcd.com



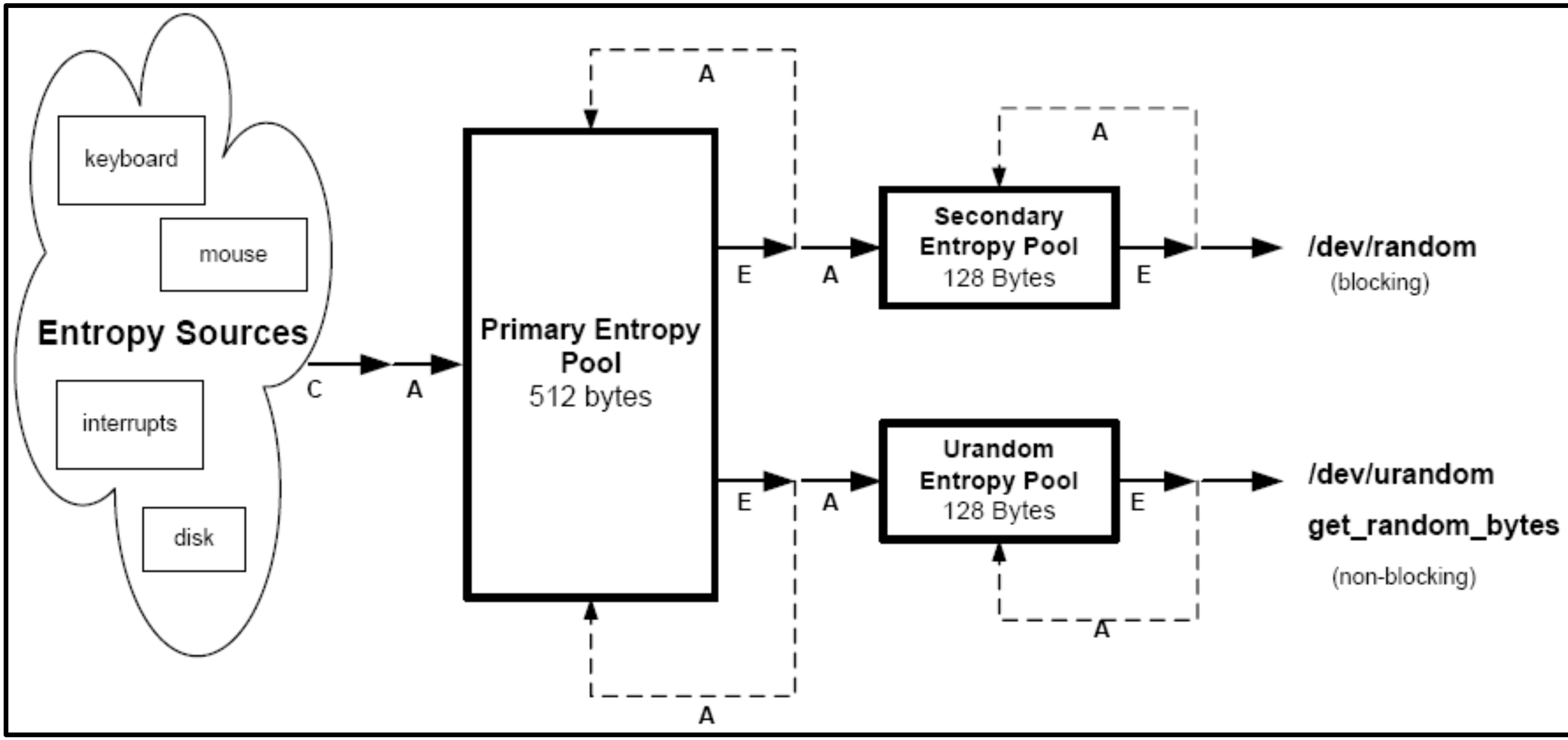


- Random number generation
- Measure events on system, harvest entropy (unpredictability from them)
  - keyboard presses and timing
  - file/network interrupts
  - mouse movements
- Hash entropy down to “extract” (hopefully) uniform bit strings

Linux /dev/random

Linux random number generator (2500 lines of undocumented code)

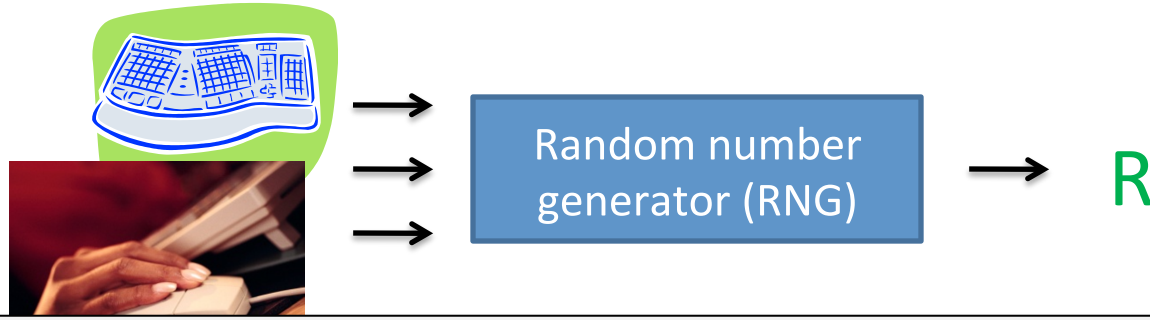
Diagram from [Gutterman, Pinkas, Reinman 2006]



Applications like TLS take randomness from /dev/random

They then maintain an internal pool of random bits

(at least) two points of failure




```
MD_Update(&m,buf,j);
....
MD_Update(&m,buf,j);   /* purify complains */
```

These lines of code commented out from OpenSSL random number generator code (md_rand.c) to **address complaints by security tools Purify and Valgrind**

Only the PID was used as input to RNG.

It took a ~2 years for the bug to be (publicly) discovered!

Ra

[W
[G
[G
[D
[W
[B
[M
[Abe
[Yilek et al. 2009]

Debian OpenSSL bug lead to small set of possible R

Debian Bug Leaves Private SSL/SSH Keys Guessable

Posted by **timothy** on Tuesday May 13 2008, @12:01PM
from the security-is-a-process dept.

SecurityBob writes

> "Debian package maintainers tend to very often modify the source code of the package they are maintaining so that it better fits into the distribution itself. However, most of the time, their changes are not sent back to upstream for validation, which might cause some tension between upstream developers and Debian packagers. Today, a critical security advisory has been released: a Debian packager modified the source code of OpenSSL back in 2006 so as to remove the seeding of OpenSSL random number generator, which in turns makes cryptographic key material generated on a Debian system guessable. The solution? Upgrade OpenSSL and re-generate all your SSH and SSL keys. This problem not only affects Debian, but also all its derivatives, such as Ubuntu."

Reader RichiH also points to Debian's announcement and Ubuntu's announcement.

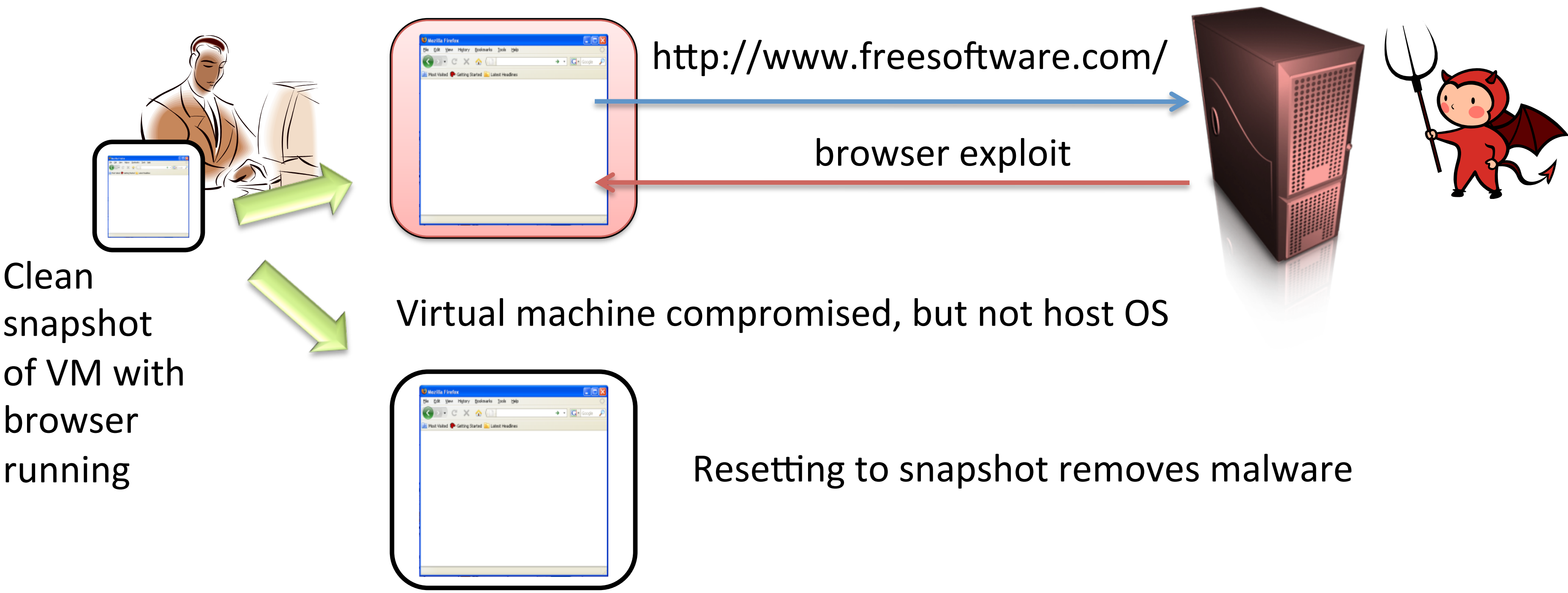
Virtual machines and secure browsing

"**Protect Against Adware and Spyware:** Users protect their PCs against adware, spyware and other malware while browsing the Internet with Firefox in a virtual machine."
[http://www.vmware.com/company/news/releases/player.html]

"Your dad can do his [private] surfing on the virtual machine and can even set it to reset itself whenever the virtual computer is restarted, so there's no need to worry about leaving tracks. … I recommend VMware because you can download a free version of VMware Server for home use. "
[Rescorla, http://www.thestranger.com/seattle/SavageLove?oid=490850]

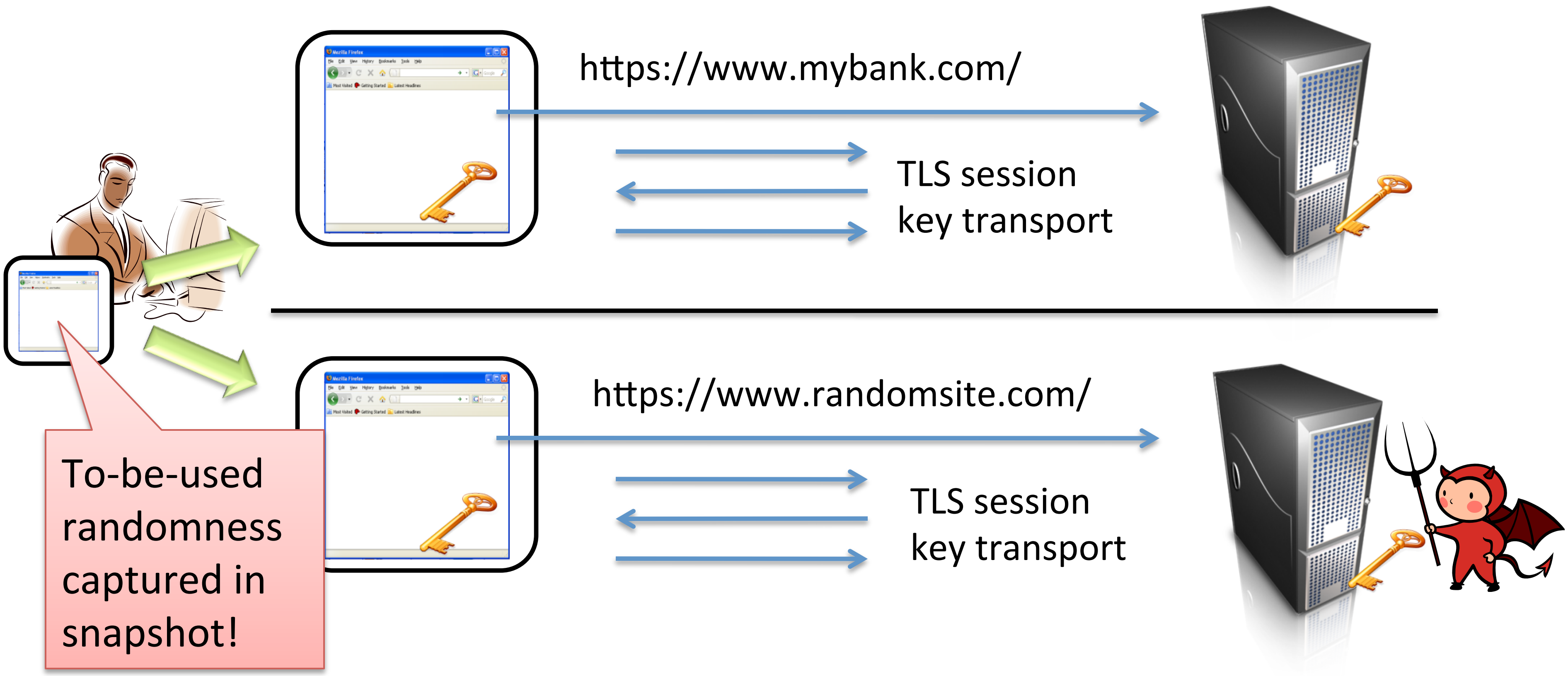
Virtual machines and secure browsing

"**Protect Against Adware and Spyware:** Users protect their PCs against adware, spyware and other malware while browsing the Internet with Firefox in a virtual machine."
[http://www.vmware.com/company/news/releases/player.html]



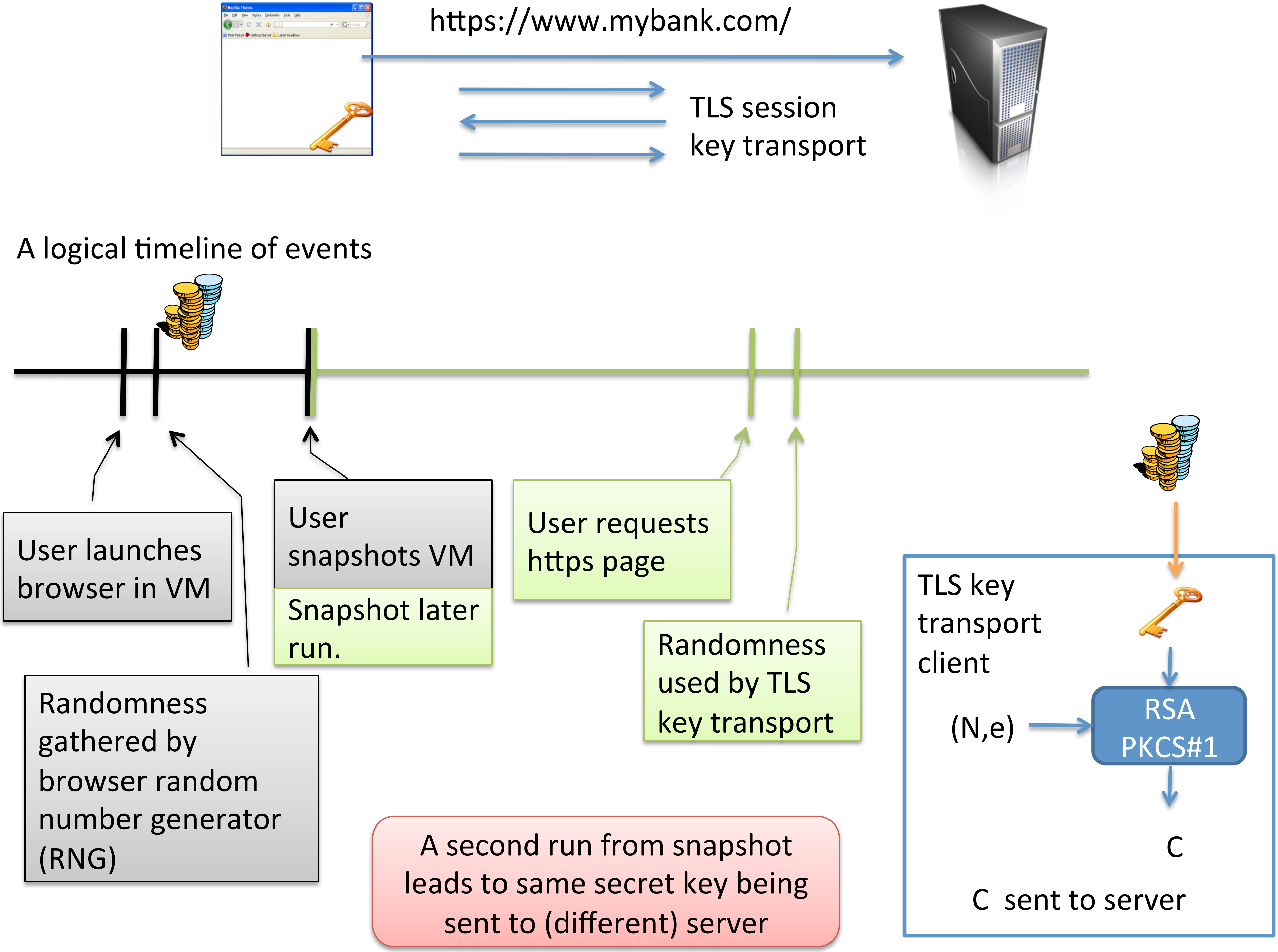
Virtual machine resets lead to RNG failures

[R., Yilek – NDSS '10]



Recent versions of Firefox, Chrome allow session compromise attacks

Apache mod_ssl TLS server: server's secret DSA key can be stolen!



https://www.mybank.com/
TLS session
key transport
A logical timeline of events
User launches browser in VM
User snapshots VM
Snapshot later run.
User requests https page
Randomness gathered by browser random number generator (RNG)
Randomness used by TLS key transport
A second run from snapshot leads to same secret key being sent to (different) server
TLS key transport client
(N,e)
RSA PKCS#1
C
C sent to server

RNG recap

- Randomness is often a weak link in crypto implementations
- Building a good RNG is not always easy
- Intel RNG instructions in next generation chips

Side-channel attacks

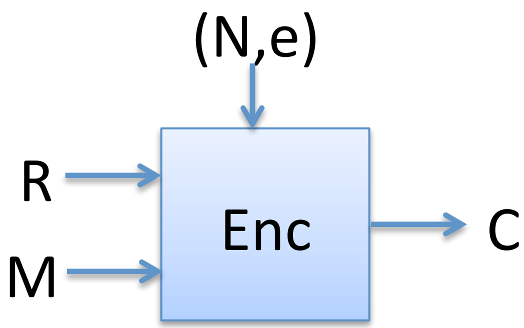
- Implementations might leak information about secret internal state via side-channels:
  - power consumption
  - Electromagnetic emanations (Tempest)
  - timing
  - Shared physical resources (CPU cache)

PKCS #1 RSA encryption

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

Let B = $\{0,1\}^8 / \{00\}$ be set of all bytes except 00

Want to encrypt messages of length $|M|_8 = m$

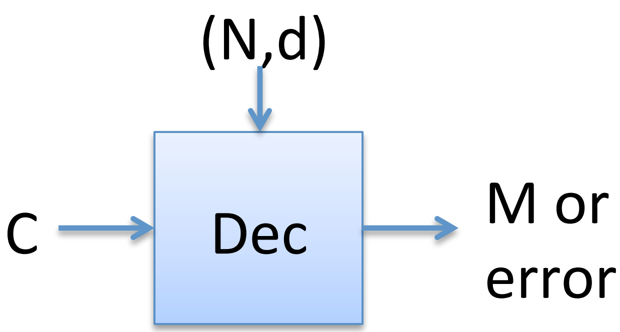


<u>Enc((N,e), M, R)</u>

pad = first n - m - 2 bytes from R that
are in B

X = 00 || 02 || pad || 00 || M

Return X^e mod N



<u>Dec((N,d), C)</u>

X = C^d mod N ; aa||bb||w = X

If (aa ≠ 00) or (bb ≠ 02) or (00 ∉ w)

Return error

pad || 00 || M = w

Return M

Textbook exponentiation

```
ModExp(X,e,N)
X' = X
For i = 2 to d do
    X' = X'*X  mod N
Return X'
```

```
SqrAndMulExp(X,e,N)
b_k,…,b_0 = e
f = 1
For i = k down to 0 do
    f = f^2  mod N
    If b_i = 1 then
        f = f*X mod N
Return f
```

```
SqrAndMulExp(X,e,N)
b_k,…,b_0 = e
f = 1
For i = k down to 0 do
    f = f^2  mod N
    If b_i = 1 then
        f = f*X mod N
Return f
```

$$e = \sum_{b_i \neq 0} 2^i$$

$$X^e = X^{\sum_{b_i \neq 0} 2^i} = \prod_{b_i \neq 0} X^{2^i}$$

$$X^e \bmod N = (\prod_{b_i \neq 0} (X^{2^i} \bmod N)) \bmod N$$

$$X^{11} = x^{1+2+8} = (x)(x^2)(x^8)$$

```
SqrAndMulExp(X,e,N)
b_k,...,b_0 = e
f = 1
For i = k down to 0 do
    f = f^2  mod N
    If b_i = 1 then
        f = f*X mod N
Return f
```

But:
Squaring and multiplying take different amounts of time and power.

From Messerges et al. 1999:

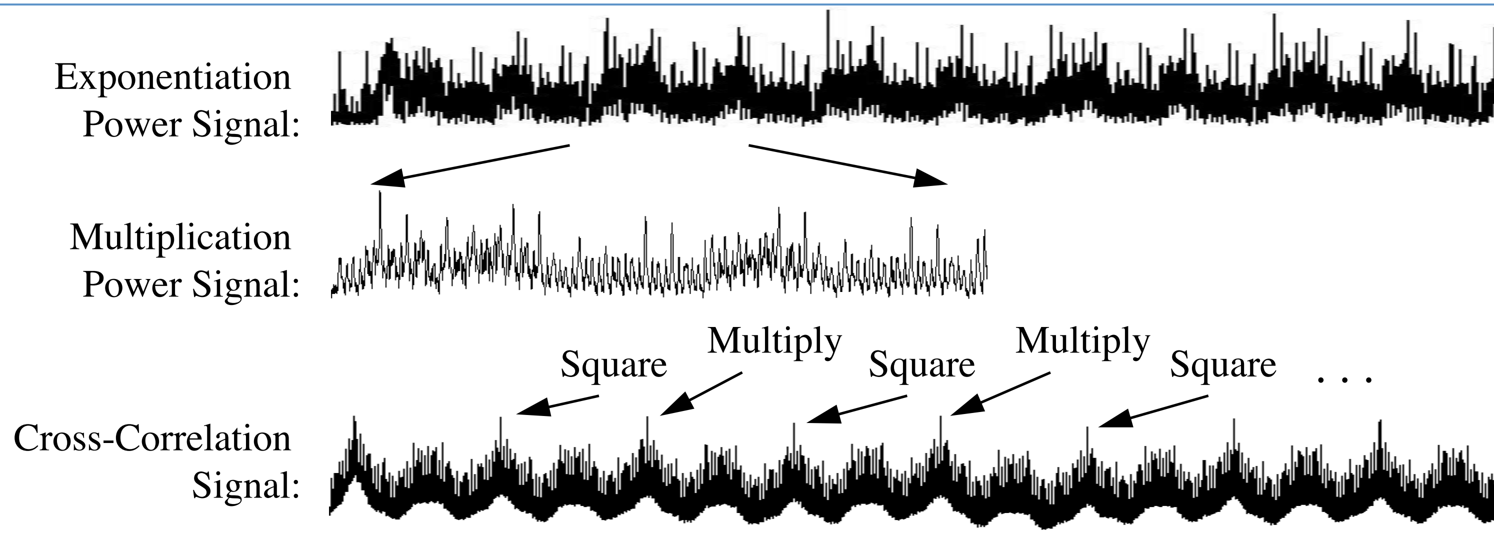


Fig. 2. Cross-Correlation of Multiplication and Exponentiation Power Signals
The above signals were obtained using the power analysis equipment described in Section 4.

```
SqrAndMulExp(X,e,N)
b_k,…,b_0 = e
f = 1
For i = k down to 0 do
     f = f^2  mod N
     If b_i = 1 then
          f = f*X mod N
Return f
```

But:
Squaring and multiplying take different amounts of time and power.

Remote timing attacks against other (Boneh, Brumley 2003)
Chosen ciphertexts + timing = key extraction
~1 million queries (though highly variable)

Lots of other implementation pitfalls

- Hard-coded keys in binaries
- Default passwords
- Developing your own crypto algorithms
- Poor key management (Kerberos, RADIUS)