When Good Randomness Goes Bad:

Virtual Machine Reset Vulnerabilities and Hedging Deployed Cryptography

Thomas Ristenpart Scott Yilek

UCSDCSE
Computer Science and Engineering

Today's talk in one slide

Virtual machine snapshot technology:

run a VM twice from same snapshot

software reuses cryptographic randomness

expose TLS sessions or steal TLS server secret key

Exploiting a reset vulnerability:
software unaware of resets, crypto fragile

Hedged deployed cryptography:

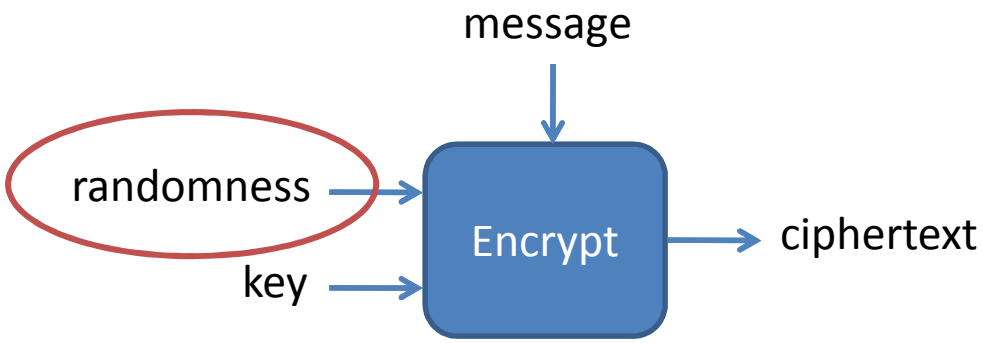
routine crypto operations fragile given predictable or reused randomness

improve security via graceful degradation of provable security

framework to "patch" crypto to achieve hedging

Cryptographic operations require suitable randomness

String of bits that are:
- Uniformly distributed
- Freshly sampled for each message
- Private



Security of operation relies on having good randomness

How is randomness generated in systems?

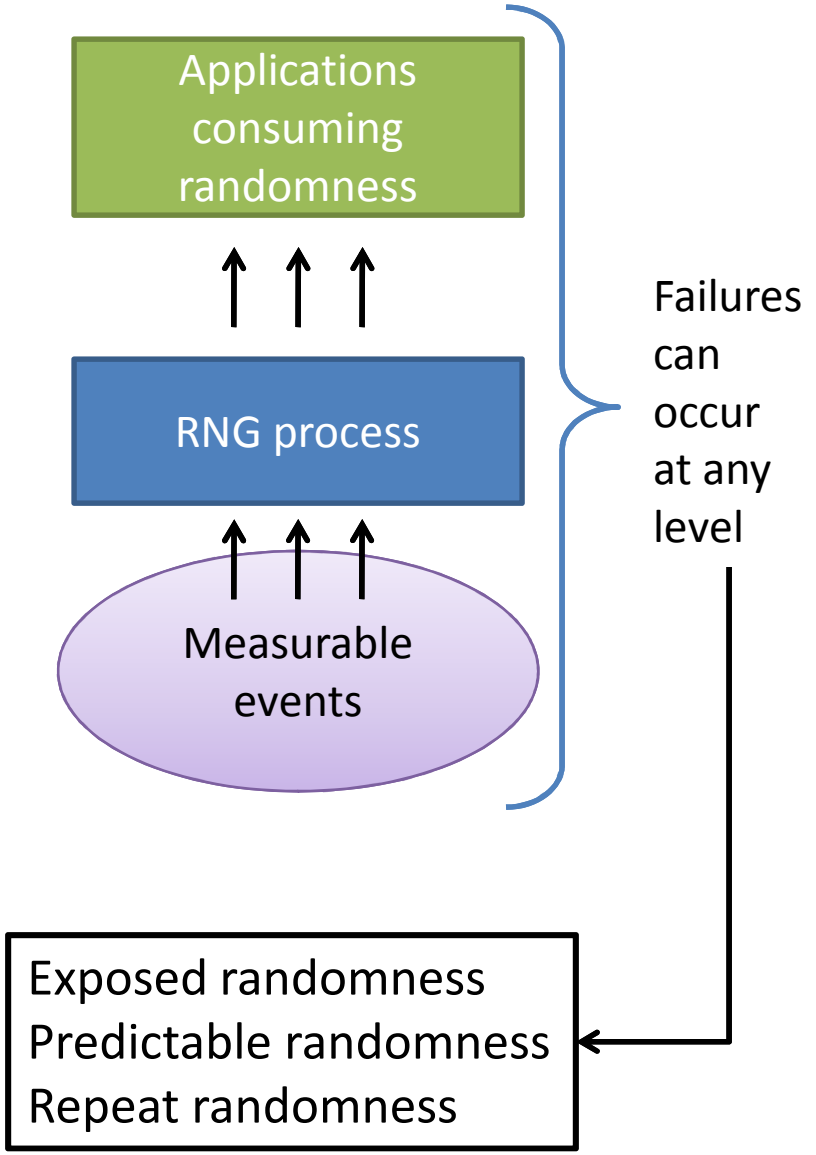
Cryptographic Random Number Generators (RNGs)

Measure a variety of events
- User input timings (keyboard, mouse)
- Network and OS interrupts
- File system reads
- …

An RNG takes measurements and produces bits that are (hopefully) uniform

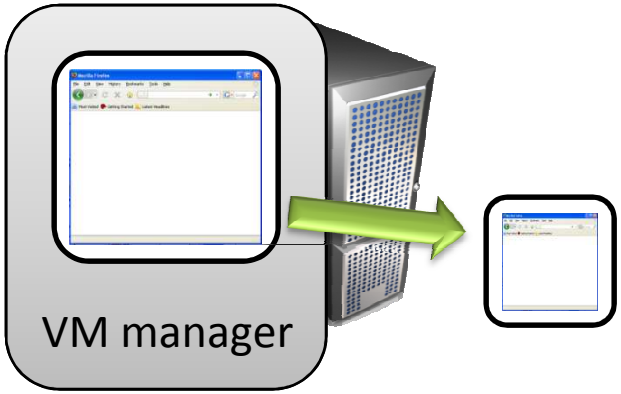
Long literature showcasing RNG failures

[Wagner, Goldberg 1996]
[Gutterman, Malkhi 2006]
[Gutterman, Pinkas, Reinman 2006]
[Dorrendorf, Gutterman, Pinkas 2007]
[Woolley et al. 2007]
[Bello 2008]
[Mueller 2008]
[Abeni et al. 2008]
[Yilek et al. 2009]



Our first contribution is revealing a new type of RNG failure in practice

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



Microsoft Virtualization

vmware

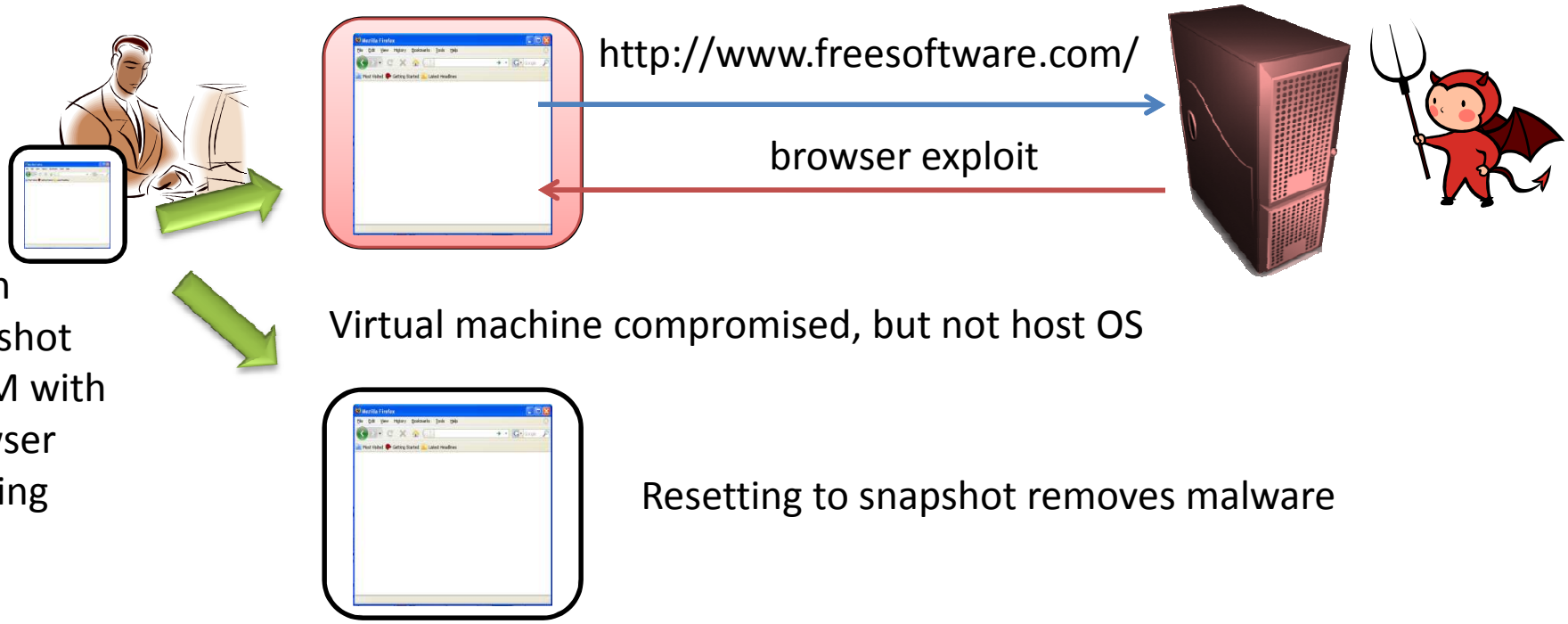
Xen™

VirtualBox

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

Backup Migration Replication Fault or intrusion recovery

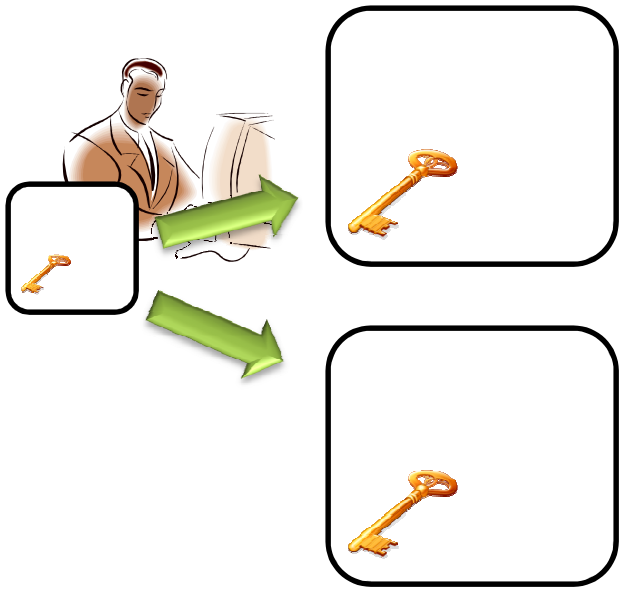
"**Protect Against Adware and Spyware:** Users protect their PCs against adware, spyware and other malware while browsing the Internet with Firefox in a virtual machine."
[http://www.vmware.com/company/news/releases/player.html]



"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]

[Garfinkel, Rosenblum '05] discuss possibility that snapshot use could lead to security issues

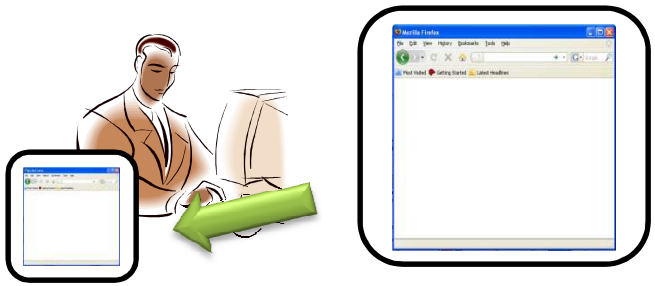
Problems might stem from reuse of security-critical state



Hypothetical example:
reuse of a one-time-only cryptographic key

VM reset vulnerabilities:
multiple uses of a VM snapshot can lead to security violations

We exhibit reset vulnerabilities in TLS clients and servers due to cryptographic randomness reuse

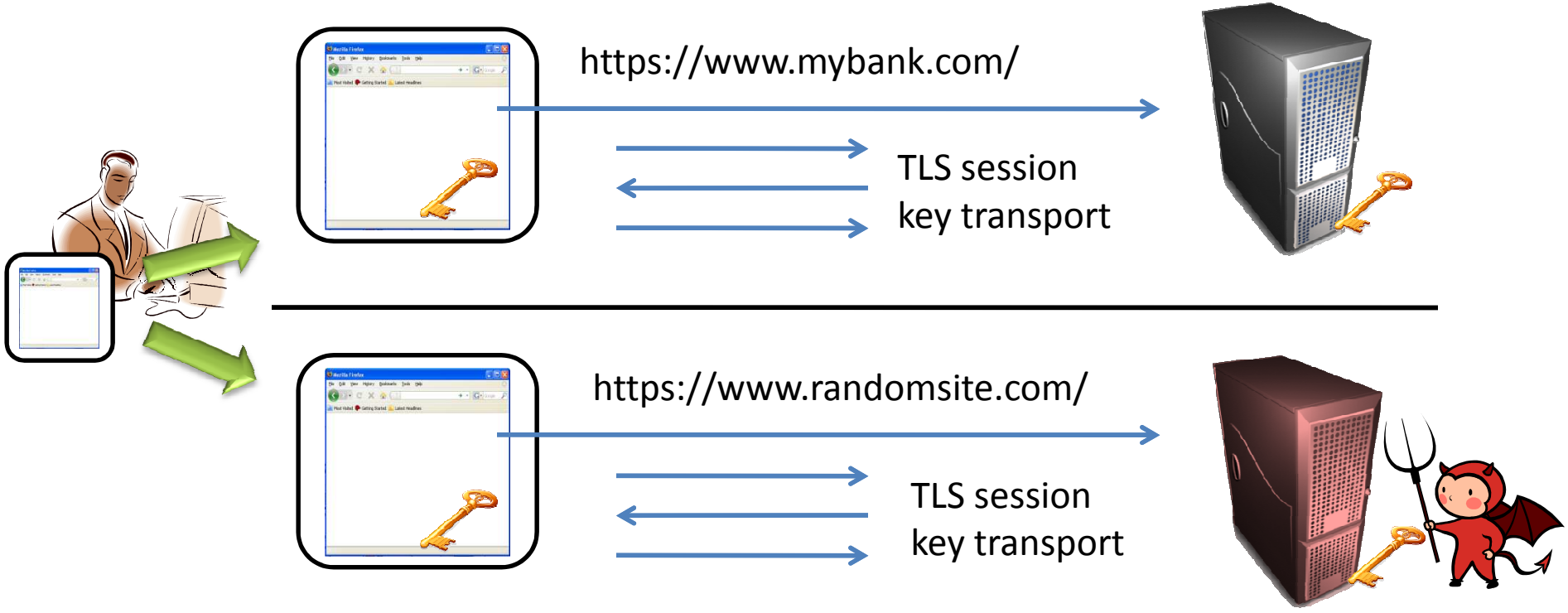


Fresh VM

Load browser

Take snapshot

We show that in some widely-used browser/VM combinations:



Browser's TLS client chooses same premaster secret (PMS)

This could expose TLS sessions

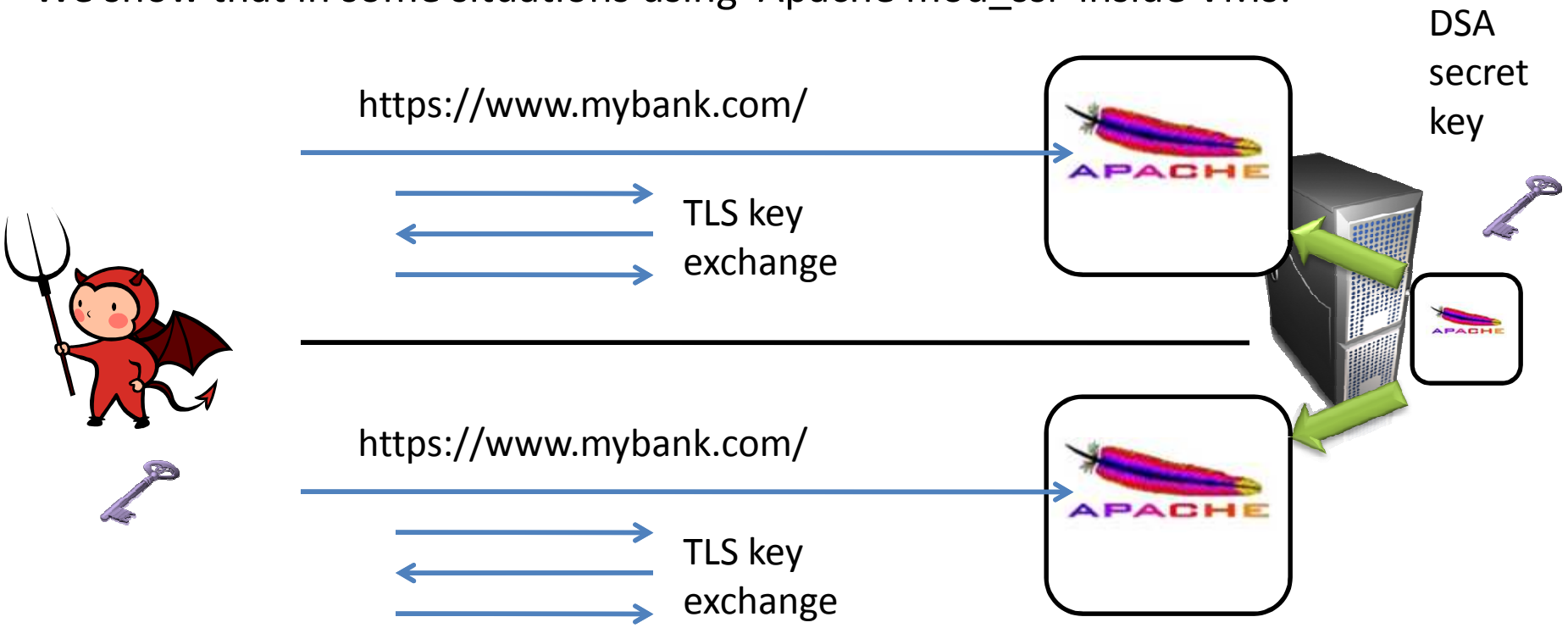
TLS Client	Guest OS	Same PMS to different sites?	Same PMS to same site?	Comments
Firefox 3.5	Windows XP	Always	Always	<100 mouse events
Chrome 3.0	Windows XP	Never	Sometimes	
IE 6.0	Windows XP	Never	Sometimes	
Safari 4.0	Windows XP	Never	Sometimes	
Firefox 3.0	Ubuntu Linux	Always	Always	<100 mouse events
Chrome 4.0	Ubuntu Linux	Always	Always	Must visit a HTTPS site before snapshot

Results hold for both VMWare Server 1.0 and VirtualBox 3.0 virtual machine managers (VMMs) both running on Ubuntu Linux

TLS clients are choosing randomness long before connection request

Potential for problems anywhere snapshots are used

We show that in some situations using Apache mod_ssl inside VMs:



Key extraction is possible

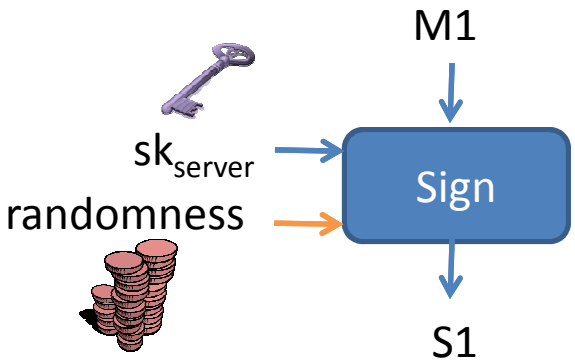
DSA secret key allows impersonating server

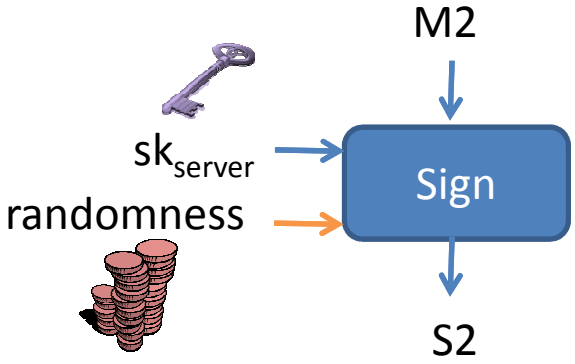
https://www.mybank.com/

TLS key exchange

DSA secret key

A few minutes with pen & paper --or-- just check wikipedia article on DSA:





If adversary has public key, (M1,S1) and (M2,S2) then adversary easily computes sk_{server}

https://www.mybank.com/

TLS key exchange

DSA secret key

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

Randomness used to sign

RNG updated with time, child PID, stack

DSA signing

key exch msg

sk_{server}

Sign

sig

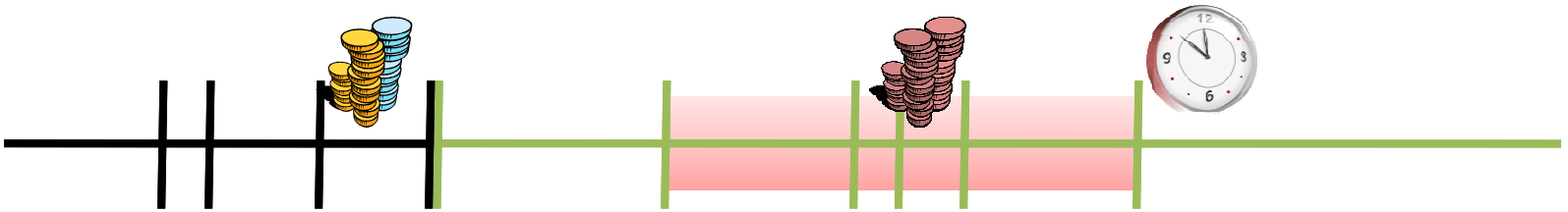
sig sent to client

https://www.mybank.com/

TLS key exchange

DSA secret key

A logical timeline of events



VM clock synch

HTTPS request handled by a child

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

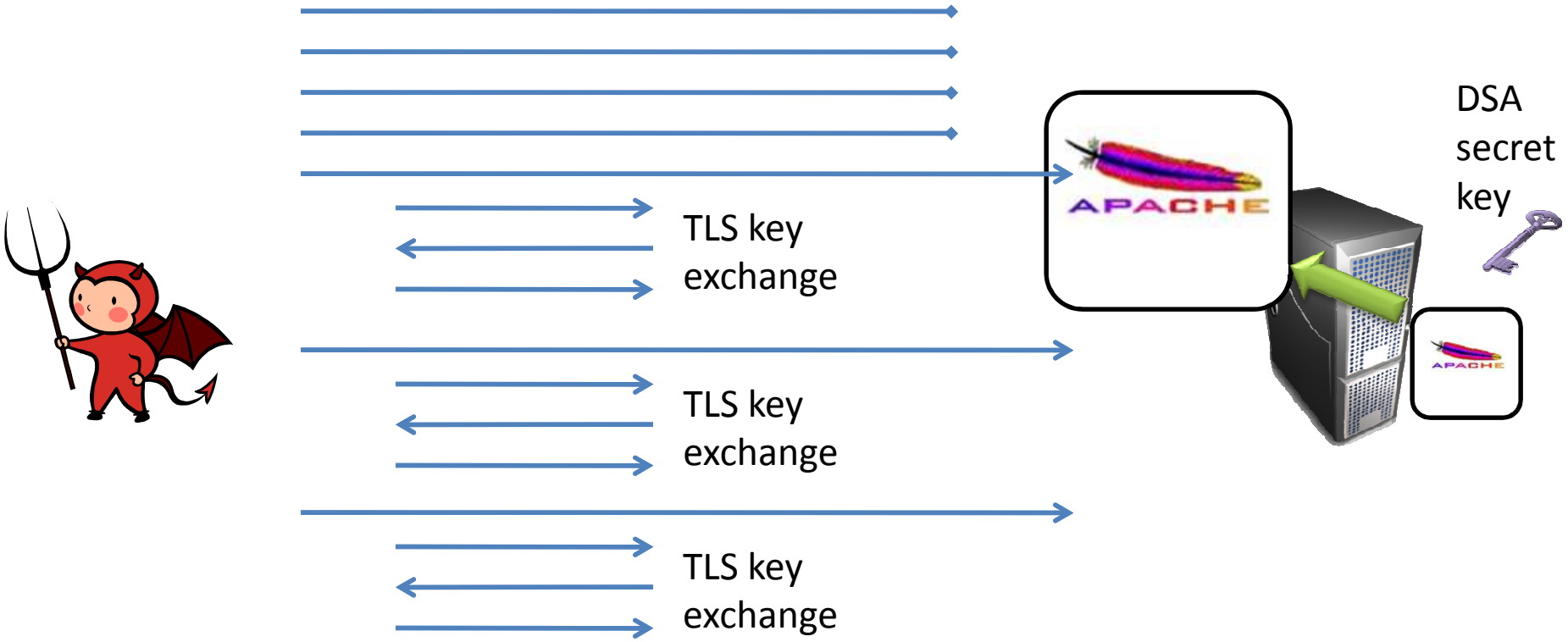
sig

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



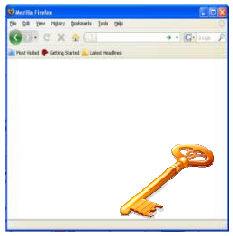
This is one trial.
We performed 5 trials for each VMM without rebooting physical server
We performed 5 trials for each VMM with rebooting physical server
Looked for reuse of randomness across pairs of successful connections

Experimenting with DSA key extraction

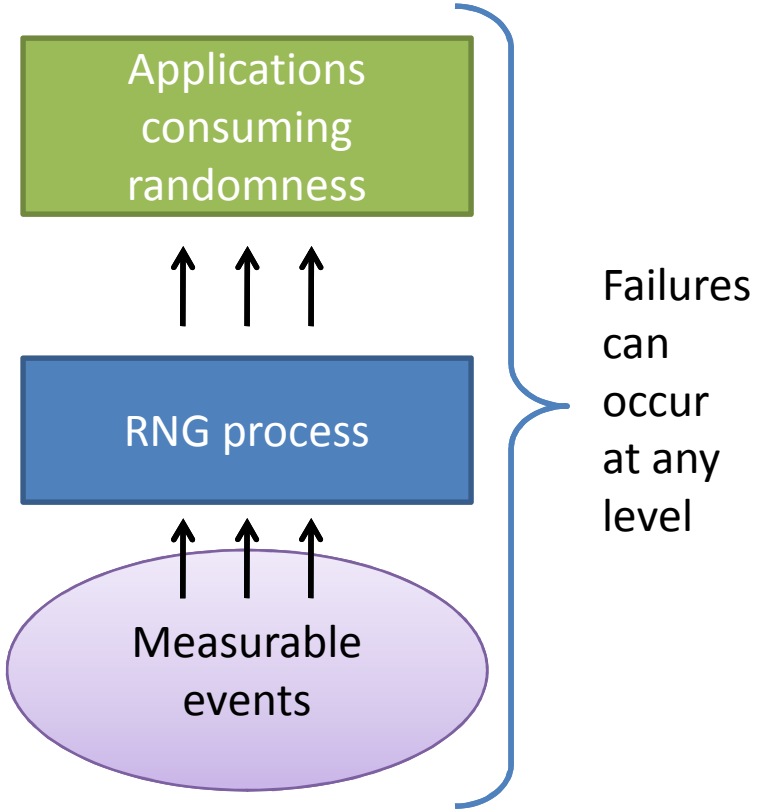
VMM	Time sync?	Always reboot physical machine?	# pairs w/ repeat sesion IDs	# pairs w/ DSA key extractable
VirtualBox	Yes	No	10/10	10/10
VirtualBox	Yes	Yes	10/10	10/10
VMWare	Yes	No	0/10	0/10
VMWare	Yes	Yes	4/10	3/10
VMWare	No	No	6/10	6/10
VMWare	No	Yes	3/10	1/10

Snapshot reuse leads to repeated cryptographic randomness

1) Applications cache to-be-used randomness



2) Insufficient differential entropy in RNG source events across resets



Fixes?

Don’t cache to-be-used randomness

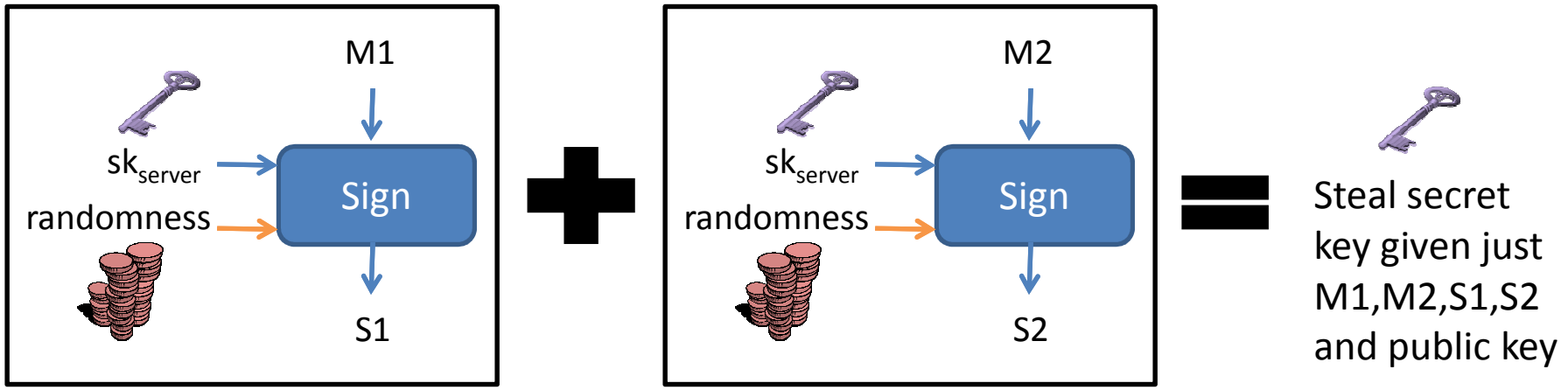
Good RNG should be invoked immediately before consumption

This might be harder than it looks!
(e.g., state of /dev/random gets rolled back as well)

Probably should use hardware-based RNG

RNG design and use needs careful engineering in light of virtualization

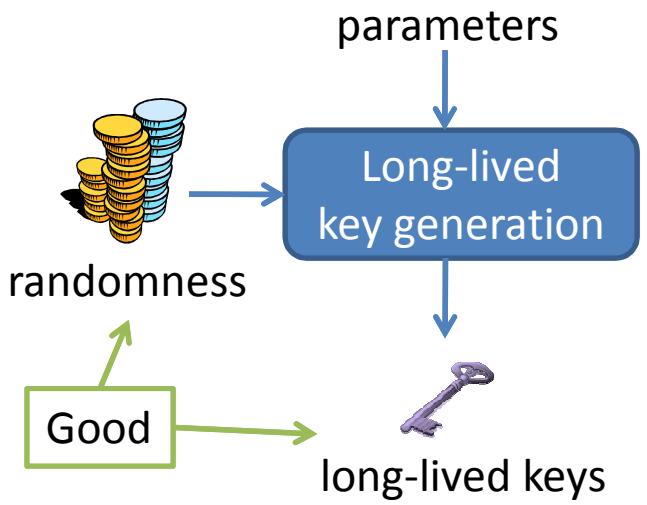
Our attacks beg another question



Why is the crypto **so fragile** in face of bad randomness?

The attacks don't abuse long-lived key generation!

Attacks abuse routine operations: signing, key exchange



But actually this is endemic within cryptography:

Algorithm	Reused randomness	Predictable randomness
TLS key transport	Session compromise	Session compromise
DSA signatures	Secret key revelation	Secret key revelation
OAEP public-key encryption	Distinguishing attacks	Message recovery (e = 3)
Identification protocols	Secret key revelation	Secret key revelation
Fiat-Shamir signatures	Secret key revelation	Secret key revelation
CTR mode encryption	Partial message recovery	Partial message recovery
CBC mode	Distinguishing attacks	Partial message recovery
RSA PKCS #1	Distinguishing	Partial message

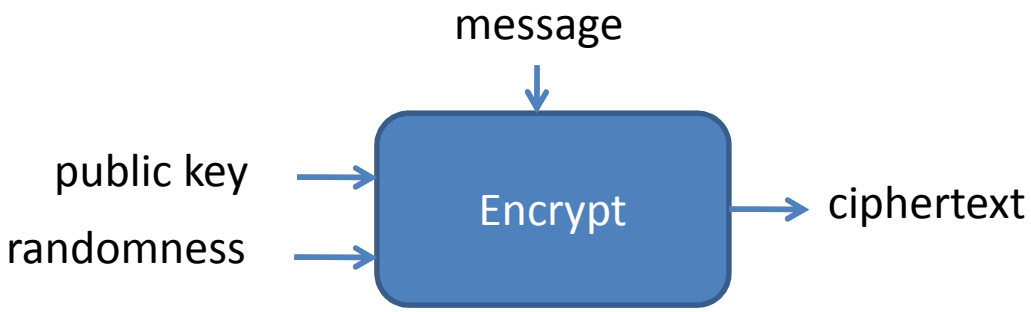
Cryptographers generally assume good randomness when designing primitives (and assessing their security)

For many security goals, it is well known that (provably) one needs good randomness

Hedged cryptography

Expand security models to include randomness failures for routine operations

Build crypto to be as secure as possible for varying qualities of randomness



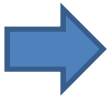
Randomness	Hedged public-key encryption
Good	Semantic security (nothing leaked about message)
Repeated	Nothing leaked but message equality
Predictable	Nothing leaked but message equality (as long as message is unpredictable)

Prove that our crypto operations give gracefully degrading security

In some cases, there is no significant degradation (DSA)

Hedged cryptography

Symmetric encryption with various types of poor randomness	[Rogaway 2004] [Rogaway, Shrimpton 2006] [Kamara, Katz 2008]
Public-key encryption (predictable/exposed randomness)	[Bellare, Brakerski, Naor, Ristenpart, Segev, Shacham, Yilek 2009]
Public-key encryption (reused randomness)	[Yilek 2009]

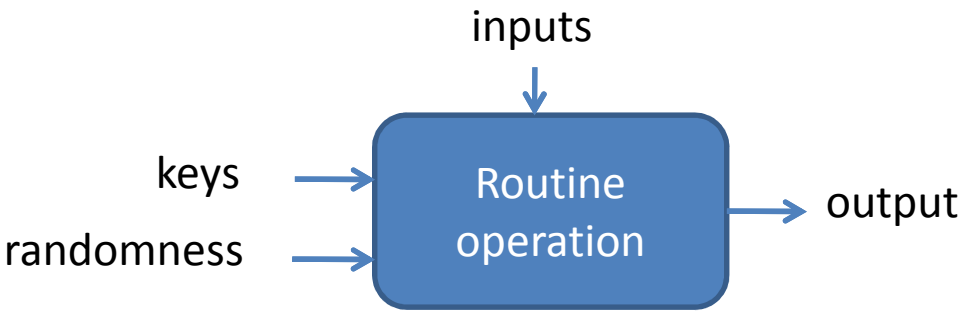


<u>Our contributions to theory of hedging:</u>

- Simple framework for hedging of arbitrary crypto operations
- New digital signature secure notion
- Analyses for various primitives (signatures, symmetric encryption, public-key encryption)

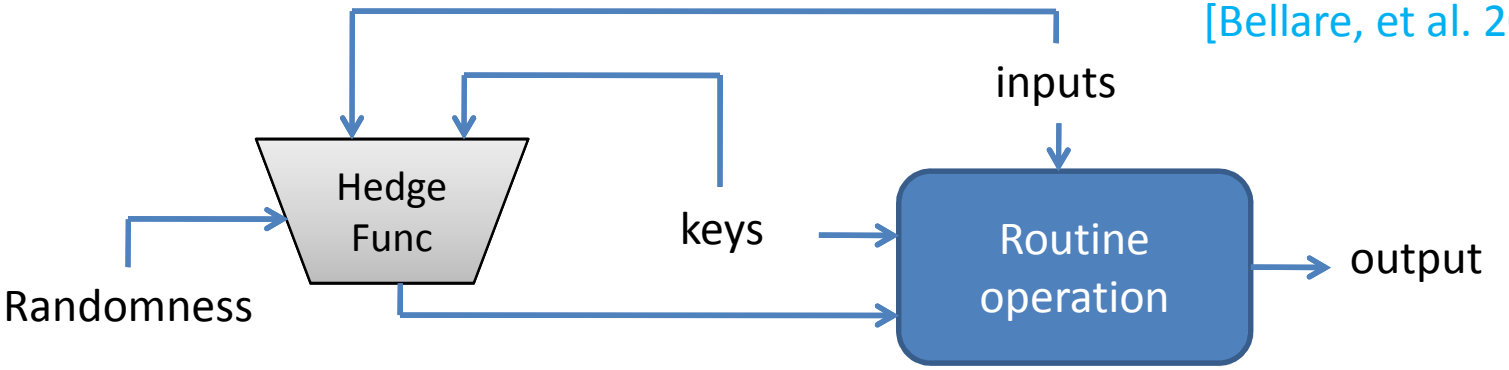
Our framework for hedging is simple

Integrates approaches from
[Bellare, et al. 2009] [Yilek 2009]

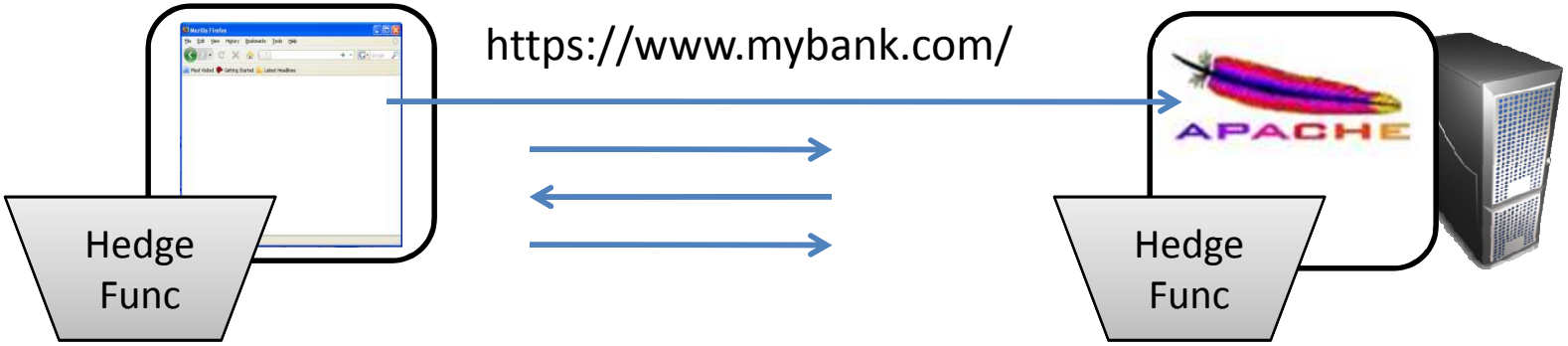


Our framework for hedging is simple

Integrates approaches from
[Bellare, et al. 2009] [Yilek 2009]

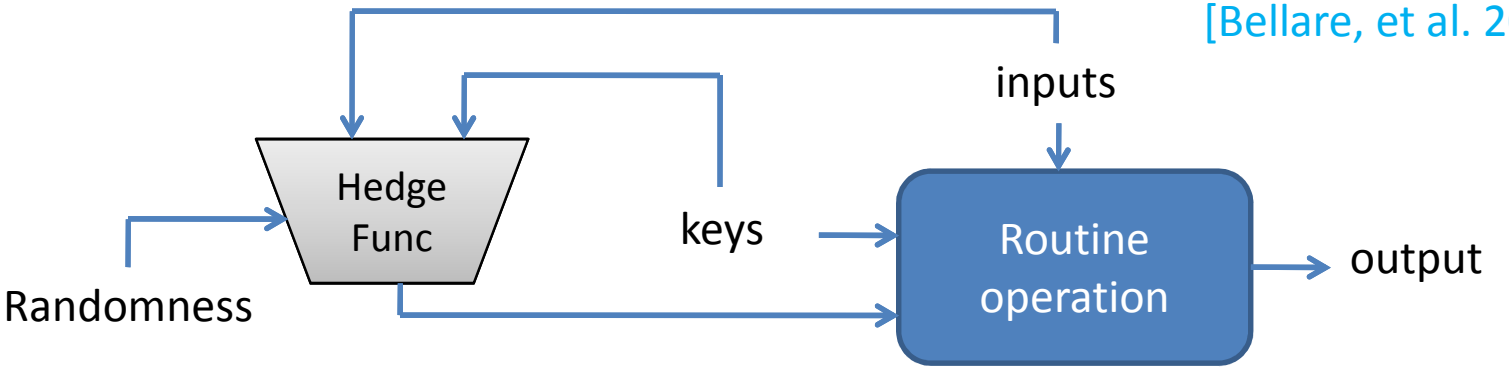


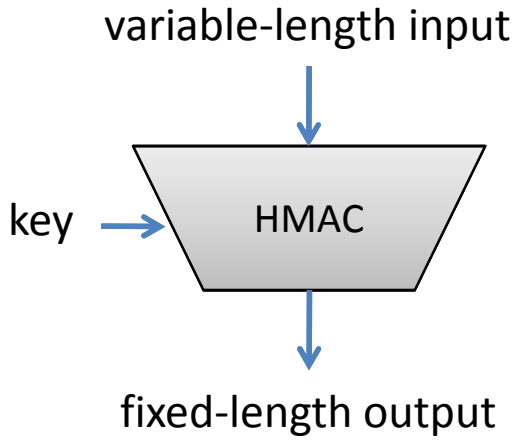
Hedging does not impact functionality, allowing immediate deployability



Our framework for hedging is simple

Integrates approaches from
[Bellare, et al. 2009] [Yilek 2009]





Our suggestion is to instantiate **Hedge Func** using HMAC.

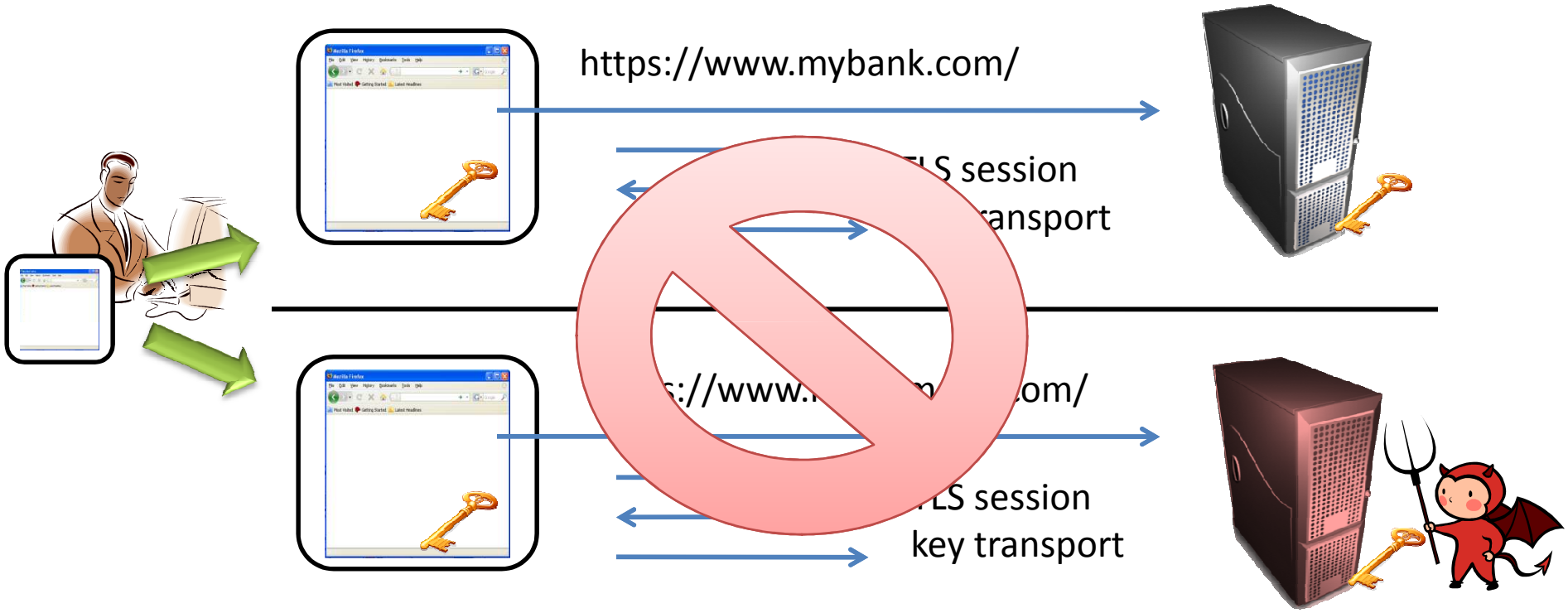
HMAC is built using a cryptographic hash function, e.g. SHA-256 or SHA-512

We can generate longer outputs by repeated applications of HMAC

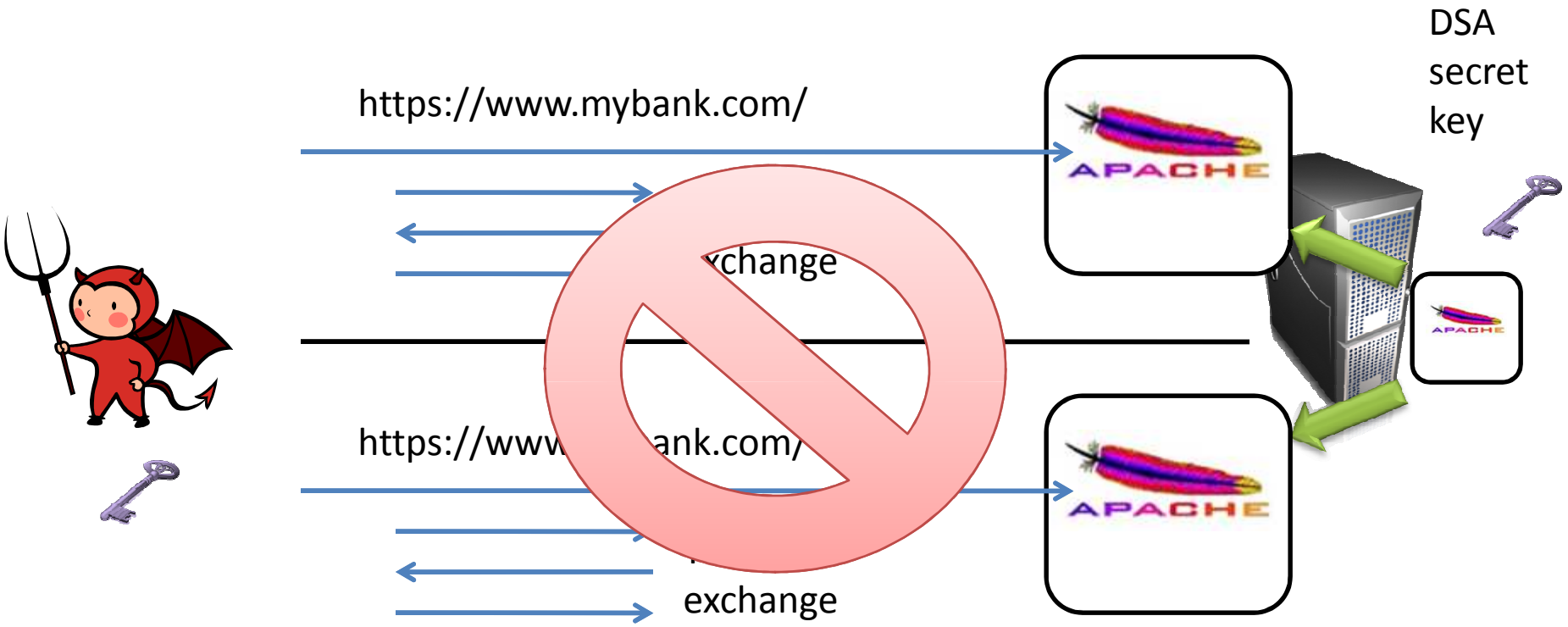
It is keyed, and when key is good it is a secure PRF

When key is "bad", still behaves like a secure hash function

If hedging had already been implemented…



If hedging had already been implemented…



So while hedging doesn't replace need for good RNGs,
it provides significant defense-in-depth should RNGs fail

Hedging is simple to implement

We hedged crypto operations within OpenSSL v.0.9.8k

… and it's efficient

Operation	Plain time (µs) Median (Min,Mean,Max,Std. Dev)	Hedged time (µs) Median (Min,Mean,Max,Std. Dev)
AES128-SHA	6941 (6875,6989,8380,231)	6968 (6890,7310,11334,920)
DHE-RSA-AES128-SHA	52030 (51756,52120,63388,470)	52828 (51150,52618,62841,735)
DHE-DSS-AES128-SHA	50907 (50567,50959,64224,471)	51067 (50011,51010,62020,673)

Results of timing 1,000 TLS handshakes
1,024-bit server keys used

(Client: Dual Pentium 4 3.20 GHz)
(Server: Pentium 4 2.0 GHz)
(Connected via LAN)

VM reset vulnerabilities

(bad)

More and more software will soon be running in VMs

VM snapshots mean programs are unknowingly reset, existing software was not designed for this

More security problems lurking?

Need to carefully design & use RNGs for VM setting

Hedged cryptography

(good)

Provably graceful degradation of security

It can be fast and simple

The catch: more complex analyses of crypto provable security

Make up as much as possible for system failures with better cryptography

Today's talk in one slide

Virtual machine snapshot technology:

run a VM twice from same snapshot

software reuses cryptographic randomness

expose TLS sessions or steal TLS server secret key

Exploiting a reset vulnerability:
software unaware of resets, crypto fragile

Hedged deployed cryptography:

routine crypto operations fragile given predictable or reused randomness

improve security via graceful degradation of provable security

framework to "patch" crypto to achieve hedging