Recovering cryptographic keys with the cold boot attack

Nadia Heninger

Princeton University

February 15, 2010
Joint work with...

“Lest We Remember: Cold Boot Attacks on Encryption Keys”
with J. Alex Halderman, Seth D. Schoen, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Rick Astley, Jacob Appelbaum, and Edward W. Felten

henceforth known as [HSHCPCFAF 08]

“Reconstructing RSA Private Keys from Random Key Bits”
with Hovav Shacham
Part 1: DRAM Remanence
The Persistence of Memory: Why?

DRAM is an array of tiny capacitors.

To write a bit, the capacitor is charged.

When power is on, the state is refreshed every 10 µs.

Without power, they discharge to a ground state.

But this process takes seconds to minutes.
The Persistence of Memory

5s.  30s.  1m.  5m.
Capturing Residual Data

Residual data can be captured easily, with no special equipment.

Complication
Booting a full OS overwrites large areas of RAM.

Solution
Boot a small low-level program to dump contents of memory.

Implementations

- PXE Dump (9 KB)
- EFI Dump (10 KB)
- USB Dump (22 KB)
Follow-up work

“BootJacker: Compromising Computers using Forced Restarts”
[Chan et al. 08]

Uses memory remanence across reboots to completely revive an existing system, including:

- SSH connections
- SSL sessions
- VPN sessions
- encrypted hard disks

For a good time, press Alt-SysRq-B
Slowing Decay by Cooling

-50°C 0.0001 % to 0.2% decay after 1-5 min.
Even cooler

Liquid Nitrogen -196°C

< 0.1% decay after 1 hour

(not necessary in practice)
Countermeasures: Encrypt Memory During Sleep

When entering screen-lock/hibernate/sleep:

- Encrypt RAM with user’s password

When awakened:

- Require user’s password to decrypt RAM
Countermeasures: Encrypted Memory

Memory encrypted with key

- Randomly chosen at boot

On cache read/write

- Data encrypted/decrypted when written/read

CPU reset clears key
Countermeasures: TPM?

Current TPMs may help attacker

Windows BitLocker in Basic Mode

- On boot, OS loads key from TPM into RAM
  (No password)
- Vulnerable to cold-boot attack
  (Even if completely off)

Future TPMs may help

- Need bulk encryption
Thoughts

In reality, you cannot expect decay.

From a theoretical perspective, hardware countermeasures move the vulnerability around.

Can software encryption be saved?
Part 2: Finding Cryptographic Keys
Looking for cryptographic keys

“Playing ’hide and seek’ with stored keys”
[Shamir, van Someren 99]
The reality of the entropy approach
Finding Keys: Use the structure of the key data.

AES implementations typically precompute a sequence of round keys from the single 128 or 256-bit key.

Generation of the 128-bit AES key schedule.
Identifying AES keys in memory

Every encryption program we looked at computed and stored the key schedules in exactly the same way.

c3284c9f ed58820e c1e923df 78a30623
e59446f1 08ccc4ff c925e720 b186e103
9e5c020b 9690c6f4 5fb521d4 ee33c0d7
9074c1b5 06e40741 59512695 b762e642
bcdd6b33 ba396c72 e3684ae7 540aaaca5
baf0db2e 00c460c0 e3ac2a27 b7a68682
a95428d6 a9904816 4a3c6231 fd9ae4b3
c40090ff 6d90d8e9 27acbad8 da365e6b
bb579527 d6c74dce f16bf716 2b5da97d
44a6d9ef 92619421 630a6337 4857ca4a
92f482ad 0095168c 639f75bb 2bc8bff1
Identifying AES keys in memory

To identify an AES key schedule in memory, scan for a block of memory that has the properties of a key schedule.

c3284c9f  ed58820e  c1e923df  78a30623
e59446f1  08ccc4ff  c925e720  b186e103
9e5c020b  9690c6f4  5fb521d4  ee33c0d7
9074c1b5  06e40741  59512695  b762e642
bcd6bd33  ba396c72  e3684ae7  540aaaca5
baf00cb2  00c460c0  e3ac2a27  b7a68682
a95428d6  a9904816  4a3c6231  fd9ae4b3
c4090ff  6d90d8e9  27acbad8  da365e6b
bb579527  d6c74dce  f16bf716  2b5da97d
44a6d9ef  92619421  630a6337  4857ca4a
92f482ad  0095168c  639f75bb  2bc8bff1
Identifying AES keys in memory

To identify an AES key schedule in memory, scan for a block of memory that has the properties of a key schedule.

c3284c9f  ed58820e  c1e923df  78a30623
e59446f1  08ccc4ff  c925e720  b186e103
9e5c020b  9690c6f4  5fb521d4  ee33c0d7
9074c1b5  06e40741  59512695  b762e642
bcd6b33  ba396c72  e3684ae7  540aaca5
bafd0cb2  00c460c0  e3ac2a27  b7a68682
a95428d6  a9904816  4a3c6231  fd9ae4b3
c40090ff  6d90d8e9  27acbda8  da365e6b
bb579527  d6c74dce  f16bf716  2b5da97d
44a6d9ef  92619421  630a6337  4857ca4a
92f482ad  0095168c  639f75bb  2bc8bfff
Identifying AES keys in memory

To identify an AES key schedule in memory, scan for a block of memory that has the properties of a key schedule.

```
c3284c9f  ed58820e  c1e923df  78a30623
e59446f1  08ccc4ff  c925e720  b186e103
9e5c020b  9690c6f4  5fb521d4  ee33c0d7
9074c1b5  06e40741  59512695  b762e642
bcdd6b33  ba396c72  e3684ae7  540aaca5
bafdf0cb2  00c460c0  e3ac2a27  b7a68682
a95428d6  a9904816  4a3c6231  fd9ae4b3
c40090ff  6d90d8e9  27acbad8  da365e6b
bb579527  d6c74dce  f16bf716  2b5da97d
44a6d9ef  92619421  630a6337  4857ca4a
92f482ad  0095168c  639f75bb  2bc8bff1
```
Identifying AES keys in memory

To identify an AES key schedule in memory, scan for a block of memory that has the properties of a key schedule.
How to identify RSA keys in memory?

Try multiplying blocks of memory together?

Try decrypting with every block of memory?
PKCS #1: RSA Cryptography Standard

RSAPublicKey ::= SEQUENCE {
  modulus INTEGER, -- n
  publicExponent INTEGER -- e
}

RSAPrivateKey ::= SEQUENCE {
  version Version,
  modulus INTEGER, -- n
  publicExponent INTEGER, -- e
  privateExponent INTEGER, -- d
  prime1 INTEGER, -- p
  prime2 INTEGER, -- q
  exponent1 INTEGER, -- d mod (p-1)
  exponent2 INTEGER, -- d mod (q-1)
  coefficient INTEGER, -- (inverse of q) mod p
  otherPrimeInfos OtherPrimeInfos OPTIONAL
}
PKCS #1: BER-encoding
(From a computer running Apache with OpenSSL.)
Countermeasures: Avoid Standardization and Precomputation

Don’t store entire key schedules or data structures in memory

- Hurts performance.

XOR key with regions of memory filled with random bits

- Can slow an attacker to some extent.

Tradeoff between security and speed

- Encryption software still needs to efficiently use key.
- Computer still needs to use RAM.
A sad tale of countermeasures: Loop-AES

Loop-AES has elaborate countermeasures against “burn-in”

- Stores 65 different key schedules together with inverted copies
- Key schedules are periodically swapped

In the case of the cold-boot attack, this simplifies finding keys, and makes it easy to identify exactly which keys belong to Loop-AES.
Part 3: Recovering from bit errors
Problem Statement: unidirectional decay

Remove all but a $\delta$-fraction of the bits, chosen at random, from a (private) encryption key.

(Flip a coin at each bit of the key. With probability $\delta$, the attacker gets to see the bit’s value.)

How to efficiently reconstruct the key?
Correcting Errors in Cryptographic Keys: AES

Use the structure of redundant key data to correct errors.

Can retrieve an AES key from 30% of a key schedule in seconds.

[HSHCPCFAF 08], “An Improved Recovery Algorithm for Decayed AES Key Schedule Images” [Tsow 09]
Correcting Errors in Cryptographic Keys: RSA

Use the structure of redundant key data to correct errors.

\[ pq = N \]

\[ ed = 1 \pmod{(p - 1)(q - 1)} \]

\[ ed_p = 1 \pmod{p - 1} \]

\[ ed_q = 1 \pmod{q - 1} \]

Can retrieve an RSA key from 27% of key data in seconds.
Step # 1: Relate key values

We can write down the relationships between redundant key information as equations.

\[ pq = N \quad (1) \]

\[ ed = 1 \pmod{(p - 1)(q - 1)} \quad (2) \]

\[ ed_p = 1 \pmod{p - 1} \quad (3) \]

\[ ed_q = 1 \pmod{q - 1} \quad (4) \]
Step # 1: Relate key values over the integers

We can write down the relationships between redundant key information as equations over the integers.

\[ pq = N \]  
\[ ed + k(p + q) = 1 + k(N - 1) \]  
\[ ed_p - g(p - 1) = 1 \]  
\[ ed_q - h(q - 1) = 1 \]  

(upper half of bits of \(d\))

\[ k = \frac{e \tilde{d} - 1}{N + 1} \]  
(trick from [Boneh, Durfee, Frankel 98])

\[ g^2 - [k(N - 1) + 1]g - k \equiv 0 \pmod{e} \]
Step #2: Solve our equations iteratively

Generate a tree of partial solutions, starting at bit 0.

What’s a tree node?
A simultaneous assignment of bits $[0 \ldots i]$ of $p, q, d, d_p, d_q$.

It’s easy to lift a solution mod $2^i$ to all equivalent solutions mod $2^{i+1}$.

How much branching at each level?
32? No, 4 equations for 5 unknowns.
2? No, we can prune a solution when it conflicts with our known bits.
Results for different key redundancy

If the attacker has partial knowledge of... ... then recovery is efficient for...

\[
\begin{align*}
\delta &> 2 - 2^{\frac{4}{5}} \approx 0.2589 \\
\delta &> 2 - 2^{\frac{3}{4}} \approx 0.4126 \\
\delta &> 2 - 2^{\frac{1}{2}} \approx 0.5859 \\
\text{Open problem} &
\end{align*}
\]

fraction of key bits known.
Experimental validation of analysis

Total number of solutions generated vs. fraction of known bits $\delta$

(More than 1 million experiments.)
Problem Statement: bidirectional decay

Provide an error-filled (private) encryption key.

With probability $\delta_i$, the attacker sees the correct value.

Extension of previous algorithm:
Do maximum-likelihood search over tree.
Countermeasures: Cryptography against memory attacks

Model: Adversary chooses lossy function of secret key

“Simultaneous hardcore bits and cryptography against memory attacks.” [Akavia, Goldwasser, Vaikuntanathan]

- Tool: Lattice-based cryptography
- Resilient to leakage of $N(1 - o(1))$ bits.

“Public-key cryptosystems resilient to key leakage.” [Segev, Naor]

- Tool: DDH, hash proof systems
- Resilient to leakage of $N(1 - o(1))$ bits.
Questions

What makes some key reconstruction problems hard and others easy?

For RSA/factoring:

<table>
<thead>
<tr>
<th>Lattice approaches:</th>
<th>Redundancy:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large blocks of contiguous bits, no redundancy.</td>
<td>Non-contiguous bits, redundancy.</td>
</tr>
</tbody>
</table>

[Coppersmith 96], [Boneh, Durfee, Frankel 98], [Blömer and May 03], [Herrmann and May 08]  
[H., Shacham]

What about Diffie-Hellman/discrete log?
Part 4: Putting it all together.
Attacking disk encryption systems

1. Cut the power to the computer.
2. Reboot into a small memory extracting program.
3. Dump the data from RAM to a device of your choosing.
4. Find keys, fix any errors, decrypt hard drive.

Works against:

Microsoft BitLocker
Apple Filevault
TrueCrypt
Loop-AES
dm-crypt

and others...
"BitLocker, meet Bit\textit{Un}Locker."

Demonstration of fully automated attack:
Connect USB drive, reboot, and browse files
For video, paper, and source code, visit:

citp.princeton.edu/memory