Protection of cryptographic key material in modern computers

André Nordbø

Gjøvik University College 12HMISA

December 19, 2012

Abstract

This paper will look at methods for protecting keys and related key material being actively used in modern computer systems with focus on moving keys from memory to other locations like CPU registers, trusted platform modules and other external hardware.

****

1 Introduction

This term paper is inspired by the paper "The persistence of memory: Forensic identification and extraction of cryptographic keys" [Maartmann-Moea et al., 2009] presented in a forensics class where the authors show how cryptographic key material, mostly from block ciphers like AES, Serpent and TwoFish can be carved from an image of the RAM or paging file. In the following pages the focus will be to describe protection or mitigation mechanisms related to protecting cryptographic keys and the confidentiality of the key determines the trust we can assign to how these tasks succeed.

Encryption keys are used for many purposes:

- privacy and secrecy concerns like protecting personal, corporate and government information stored and in communication.
- verifying identities (signing with public key encryption (PKI)).
- intellectual property protection like Digital Rights Management (DRM)[1]

[1] Privacy and DRM are actually more or less the same thing, trying to protect who is given access to something that in itself is not a secret, as discussed in episode 387 of the "security now" podcast at http://twit.tv/show/security-now/378

It's often seen that one of the reasons why security is easily breached on consumer devices is because of the need for availability. The mentioned tasks can be solved by applying cryptographic functions requiring cryptographic keys. Keys can be symmetric in that the same key is present at the sender and receiving end and these algorithms are used for bulk encryption because they are fast. Asymmetric algorithms have different keys for encipher and decipher operations making them suitable for source verification and electronic signing as one of the keys can be kept private while the other one can be publicly available. Asymmetric ciphers are typically much slower[2] and primarily used to exchange symmetric keys and to sign data. The signing part can be used to build chains of trust, as seen by the trusted root certificates found in modern operating systems. The goal of this paper is not to give a detailed overview of cryptology but these differences in how keys are used is important. Cryptographic keys can be used directly or indirectly protected by easy to remember passwords in order for humans to use them, hence protection of passwords is as important as protecting the keys.

2 Background

The tool "Interrogate" developed by the authors of [Maartmann-Moea et al., 2009] was developed with a forensically perspective as it is intended to work on a off-line copy of memory. It has different search methods depending on the cryptographic algorithm being used, the key size and the cryptographic application targeted. One example of how a key can be found is the implementation related detail that the key is stored in memory together with the corresponding key schedule for AES in TrueCrypt[3]. Right in front of it to be precise. Since the key schedule is fully deterministic given the key, this method requires walking through the memory with a key length sliding window, calculating a key schedule and comparing it with the following block of bits. If a match is found then a very likely AES key[4] is found. In case of possible bit decay in the image a small hamming distance[5] can be tolerated.

This search methods works on both AES and Serpent, but for the TwoFish algorithm a weakness related to the entropy of the substitution boxes had to be used to find keys. This was because the key itself was not used as part of keying material [Maartmann-Moea et al., 2009]. It is obvious that such a search tool would have to be tuned to match specific cryptographic applications illustrated by the fact that inserting some random bits between the key and key schedule in memory, or storing the key after the key schedule would result in no keys found for this implementation. Other obscurity solutions could include splitting

[2]Elliptic curve based cryptology seems to be able to perform much better than RSA prime based algorithms so this might change [Gibson and Laporte, 2012]

[3]TrueCrypt is a disk encryption software for Windows, Mac and Linux `http://www.truecrypt.org/docs/`

[4]My group presented this paper, tried the mentioned python tool and retrieved keys from TrueCrypt running in a different virtual machine and operating system

[5]The number of bits that must be changed to transfer one pattern to another pattern

the key and schedule data in smaller chunks and interleave them. Perhaps in a random order for each installation, inspired by the way TwoFish generates a lot of key material bigger than normal paging sizes of 4KiB resulting in splitting across several pages [Maartmann-Moea et al., 2009]. The paper mentions using tools to rearrange the "random" pages in a memory dump into the virtual address space for each process by finding the page directory base using tools like PTfinder and Volatility.

The goal of the reviewed paper was to compare different classes of encryption schemes "full disk", "virtual disk" and "session based" with different system states like "live", "screen saver", "hibernation" and "reboot" etc. Except from the most obvious findings, it is to be noted that some applications fail in removing key material from memory after use, applicable for session and "virtual disk" keys on dismount or session end. Another interesting finding is that keys are recoverable even after a reboot because of residual voltage in the memory modules.

The methods described assume the memory is already dumped. Another paper [Rabaiotti and Hargreaves, 2010] describes how a memory dump from an embedded gaming console (Xbox) was performed. The paper describes in detail how a buffer overflow in the parsing of cut scene video names from a save game were not correctly validated resulting in an exploitable buffer overrun. By crafting a special save game, custom code could dump the shared game/kernel memory. The paper also has a list of techniques for dumping memory on desktop computers in the related work section:

- Dedicated PCI card: In the paper [Carrier and Grand, 2004] a specially crafted PCI card is installed in a system prior to RAM extraction. The card is activated by a physical switch, and will then suspend the CPU (if possible) while it uses direct memory access (DMA) to copy the contents of RAM to a storage device directly connected to the PCI card. The proof of concept card "Tribble" had trouble reading parts of RAM, the Upper Memory Area (UMA[6]) reserved for video memory and BIOS.

- Low Pin Count is a bus used to connect low bandwidth devices to the motherboard as a replacement for ISA. The idea is to use this bus to perform DMA reads as an alternative to a dedicated PCI card.

- FireWire-1394: Basically the same kind of attack using DMA access. The presentation [Boileau, 2006] present FireWire as an expansion bus having direct memory access via the bus, able to address by 32-bit addresses.

  > "With Firewire, I can read and write main memory...
  > ...without the OS being involved...
  > ...because that's how it's meant to work." [Boileau, 2006]

[6] A reason why is mentioned here: http://ntsecurity.nu/onmymind/2006/2006-09-02.html, and caused by a write-back to a read only memory mapped region

The presentation also mention something very interesting, a real mode keyboard buffer that contains the last written 16 bytes before the system left real mode, and could possibly contain PGP or BIOS passwords. The only good thing about FireWire is that it's not very popular except from high end desktops and laptops and it's getting replaced by USB and other buses. The question is whether they are more secure.

- Cold boot: The paper [Halderman et al., 2009a] explains how dynamic random access memory (DRAM) retain their content for several seconds after power is interrupted. Combined with cooling the chips to minus 50 degrees Celsius, bit decay less than 1% after 10 minutes were observed, and the effect was more efficient the colder the chips were brought. The test system were limited to models produced between 1999 and 2007. The paper notes that these findings were known at least back in 1978.

- Logic analyzer: It's described as the "ultimate hardware approach" by monitoring the bus between the motherboard and the memory chips in real time, sampling the signal between the motherboard and the memory modules and then calculating the state of the chip at a given time. The problems mentioned it the need for storage capacity to store all this data and time to analyze it. Equipment capable would be very expensive.

The introduction of the TRESOR paper [Müller et al., 2011] also contains known attacks classified as "DMA attacks" and "cold boot attacks". It also mentions that most common encryption tools do not protect against these kinds of attacks.

As long as the keys are available in memory in the system in active use, given enough motivation and time an attack is always possible. This situation has always been a great headache in protecting intellectual property as the untrusted devices has to be able to play back or run the protected content. In the situation of gaming consoles and dedicated movie players a bit more control of the device is possible but it still needs to posses the unlocking mechanism. If the key can not be trusted to the devices memory, can it be located somewhere else?

In the following chapters I will write about how encryption can be run without using the RAM, directly on the CPU, how a trusted platform module can be used and how the iOS uses keys in hardware, before I conclude.

3 Perform encryption directly on CPU

The paper [Müller et al., 2011] presents the idea of moving the keys from the main volatile memory, the random access memory (RAM) and running it directly on the central processing unit (CPU). As they mention, others have also done similar tings, and they mention "Loop-Amnesia" [Simmons, 2011] using machine specific registers and "Frozen Cache" [Pabel,] trying to control the CPU cache between the registers and the RAM.

The proof of concept software TRESOR is a patch to the Linux kernel implementing full disk encryption with the symmetric cipher AES to use CPU registers only and thus avoid storing keys in the main memory. A summary of important details follows.

As the authors explain, avoiding using the RAM is difficult. They found that the four debug registries dr0 to dr3 on x86 CPUs used for setting break points in hardware were seldom used. In 32 and 64 bit systems they could store respectively 128 and 256 bit keys. AES uses 10-14 rounds each with a derived round key. These round keys are normally precomputed and stored in RAM for performance. Because the authors only got storage space for the master key, they had to calculate these round keys for every block encrypted or decrypted. Speed was a concern, but speed tests performed looked like performance was even better than methods utilizing RAM. They thought it could be because of time penalty of accessing RAM compared to recalculation. They utilize the new hardware accelerated instructions available on both Intel and AMD processors called "AES-NI" to speed up and simplify their implementation since these instructions can encrypt and decrypt one round AES in one CPU cycle.

TRESOR will ask for a password in kernel mode during boot. The key is calculated by hashing several times, and during this process key material will be present in RAM for a short while before the final key is copied into the debug registers and the RAM content is overwritten. The key will remain in each core's registers as long as the system is running. The debug registers are protected with ring 0 access. The AES-NI instructions uses 16 SSE registers xmm0 to xmm15 to store the temporary round keys and the current round result[7]. TRESSOR has to avoid these registers being written to RAM during a context switch and thus have to make decrypting and encrypting of an AES block an atomic operation so that resetting of content in the SSE registers can be performed before any other process is swapped in. Scheduling is avoided by running TRESOR in kernel mode and interrupts has to be disabled while in the atomic state. As mentioned disabling of interrupts can be limited to a single core. The debug registers also had to be protected so that no API call could start a debugger that would overwrite the stored key corrupting the whole system, and this was done by patching user space and other kernel modules.

TRESOR was then implemented as a module in the Linux kernel crypto API that will take care of the different cipher modes as using electronic code book (ECB) mode is known to be unsafe [Huang et al., 2011]. They had to hack around the key management in the crypto API using a dummy key.

According to the speed tests performed by the authors in their table 1, maximum speed without using the module were 47MiB/s on a disk copy and when their TRESOR was used between 15-17MiB/s was achieved. This was slightly faster than RAM based implementations of AES-NI. The benchmarking was run on a Intel Core i7-620M CPU, a dual core at 2.66GHz[8] using only one

[7]Even when 256 bit keys are used, as long as the block size it 128 bit then only 128 bit chunks of the key schedule is used in a given round and thus 15*128 bit are needed of key schedule material (initial + 14 rounds) [Ebermann, 2012].

[8]It has a turbo mode at 3.33GHz

core. Resulting in about 149 cycles per byte. Comparing these numbers with the benchmark of AES available in Truecrypt on a CPU [Intel,] without AES-NI yields 471MiB/s or 24.3 cycles per byte (adjusted for 4 cores). From what I can tell, their disk benchmark should not be limited by their CPU, as the AES-NI should perform much better than the 24.3 cycles per byte without. They also mentions that an 128 block uses about 440 ns to encrypt on their CPU resulting in about 9.1[9] cycles per byte which is closer to what should be expected.

The authors of TRASOR used tools like interrogate [Maartmann-Moea et al., 2009] and AESKeyFinder [Halderman et al., 2009b] to look for keys in memory of TRASOR running inside a virtual machine and they also searched for the known key directly. No substring longer than 3 bytes (24 bit) were found. Just because the authors did not find any leakage, does not mean no leakage is possible, as quoted from the paper:

> "we have no persuasive argument that the key never enters RAM. Admittedly, it is unlikely that a piece of code other than context switching swaps debug registers into RAM, but it can not be ruled out" [Müller et al., 2011]

Other attacks discussed are attacks on the CPU itself. They argue that unless the kernel is compiled with loadable kernel module (LKM) support or kernel virtual memory (KMEM), even root should not be able to get access to the cipher keys in software. In hardware one attack might be rebooting and trying to read back the registers. This works in virtual machines, but the same effect was not possible in real CPU's tested with their tool Cobra "Cold Boot Register Attack". Another attack mentioned in the video of "Frozen Cache" [Pabel,] by a commenter is to connect to the CPU using JTAG to dump state. JTAG ports are not common on x86 CPU's but they exists [coreboot.org,]. The commenter also comment on the x86 architecture summarizing some of the difficulties doing hacks like these:

> "The x86 is an evil architecture, that is full of flaws and it's not worth to do serious stuff with it" [Pabel,]

4 Move encryption away from the CPU

Lets look at how keys can be stored out of reach of the CPU. Methods described so far have focused on disk encryption and hiding keys out of sight. The main reason for doing encryption in software is because it's flexible. New algorithms can be implemented easily, bugs fixed, and dedicated hardware is often expensive. Hardware can be custom build to speed up a certain algorithm, like the AES new instructions, but then again they only speed up AES.

[9] (440E-9 * 2.66E9) / 128[B] given one core in use

4.1 TPM module

One of the main ideas behind a trusted platform module (TPM) is to have a separate piece of hardware storing and performing cryptographic operations in a way that is not accessible for the central processing unit (CPU) and thus software running on the machine. The TPM is internationally standardized by the Trusted Computing Group (TCG) former Trusted Computing Platform Alliance (TCPA), and it was formed in 2003 [TrustedComputingGroup, 2012]. The technology had a rough start being accused of enabling media corporations to get their needed leverage to perform Digital Rights Management (DRM) [Stallman,]. Videos were made to inform people to rise against the "Trusted computing" implementation. [Stephan and Vogel,]. Today TPM chips are found almost everywhere: In desktops, laptops, phones, pads, gaming consoles an so on. They are cheap. In desktop and laptops the module has traditionally been off by default, an "opt-in" solution [Branscombe,] and has to be enabled in the BIOS.

The chip usually contains cryptographic modules for

- Hashing
- Symmetric cryptographic operations
- Asymmetric cryptographic operations
- (true) Random number generator
- Tamper resistant storage of keys
- Input and output circuits

The idea is that the chip has a master private key (Storage Root Key) it will never give away. A hierarchy of keys can be generated, signed and encrypted by the trusted master key and stored on disk unreadable for the rest of the machine. Because these chips are small and cheap, they don't contain much memory and not that much computational power [Wang et al., 2008, ch 3.1]. The TPM chips are designed to be tamper resistant, although they have been compromised as demonstrated in this video [Tarnovsky,] during Black Hat 2010. The clip is actually quite interesting as it describes how an Infineon chip is broken down layer by layer to figure out how to access the unencrypted data buss of the CPU core. As noted during the clip, as technology is getting more advanced these chips get smaller and this task is getting more difficult. Sensors are usually put in place to detect this kind of probing and during the video we see the presenter talk about how he has to avoid them. It was not that hard because they were wide apart. Whats even more interesting to learn is that these chips are in every gaming console and all the accessories like gaming pads. The gaming industry can afford to include such authentication, but attacks on industrial control systems like shown by the Stuxnet attack are wide open to give and receive commands.

In the paper "Password Caching and Verification Using TPM" [Wang et al., 2008] the authors explain how traditional TPM implementations in password managers will store passwords in the TPM, but retrieve them in plain text every time they are used for verification. During this period the keys will be in RAM. The authors goal is to implement a system, PwdCaVe, that will do the verification part in the TPM itself. Because of limited computational abilities of the TPM the Object Independent Authorization Protocol (OIAP) is used. Now a client can mediate a verification request from a remote server to the TPM module.

Another issue with the TPM has to do with mobility. As the authors of the paper "A Portable TPM Scheme for General-purpose Trusted Computing Based on EFI" [Han et al., 2009] argue, the existing solution is inefficient when it comes to migration of keys from one platform to another. They propose a portable TPM module using USB connections, thus being removable.

Microsoft has released a paper "Towards a Verified Reference Implementation of a Trusted Platform Module" [Mukhamedov et al., 2009] discussing the increasing set of commands available in the latest 1.2 version of the TPM specification. More than 90 is already implemented and their concern is ambiguity in implementation as thy mention several known vulnerabilities described in the literature. They then implemented OIAP and OSAP in F# and did a formal verification on the code using FS2PV.

[Anderson et al., 2006]

4.2 iOS protection

The most popular smart phones and tablets are based on Apples iOS and Android [Miller, 2011]. Where Android is a general operating system to be used on different hardware from many companies, Apple has the advantage of designing both the iOS operating system and the hardware it runs on. In a related work our group [Abalenkovs et al., 2012] performed in digital forensics on mobile forensics, we discovered that implementation of security features were more dominant in iOS devices, and the newer devices has encryption keys protected by hardware, similar to a TPM module. The book [Zdziarski, 2012] talks about what we know of it.

There are two encryption keys in a dedicated hardware module not accessible by the devices CPU: a key shared by all devices of that model and a key that is unique for every phone. Because these keys cannot be extracted from the device[10], any brute force of user PIN or password must be performed on the device, and the algorithms are designed so that brute force takes a long time for each try. There are many derived keys used in the iOS operating system, and files can be given different sets of protection: Files can be set to use per file individual seeds that combined with a master keys unlocks them, other files will only be encrypted by a master key while some files are not encrypted at

[10] An attack based on the physical probing [Tarnovsky,] might work but is very time consuming and requires deep knowledge about the chip in use

all. The reason must be because of convenience, as users want to be able to receive SMS, listen to music, take photos, having alarms go off and even have files backed up to the cloud when the device is "locked". Locking the phone, removes some master keys from the device and thus rendering everything it protects unavailable. To get access to master key(s) again, the chosen PIN is combined with the unique hardware key. The problem however is that setting protection class is up to the developers of "apps" often not used, and like already mentioned enabling cloud synchronization disables per file encryption for files that must be synchronized while the device is locked. The shared hardware key is used to enforce how Apple restricts unsigned applications to run.

A jailbroken iOS device simply disables the checks being performed, as they require root access to the device by exploiting a vulnerability, either in the boot sequence[11] or within applications pre-installed by Apple running with high privileges, typically the Safari browser. The paper [Gómez-Miralles and Arnedo-Moreno, 2012] describes methodology for how such an attack can be performed.

4.3 Dedicated external hardware

If the goal of encryption is to secure communication between trusted hosts, a solution like the Thales TCE621 [Thales,] can be used. It is approved up to NATO CTS level using the NICE algorithm and it can also use the AES encryption algorithm only allowed up to NATO secret. It's usually put as a gateway to allow two or more internal local area networks (LAN) to communicate over untrusted communication links. Keys can be loaded locally and remotely, it's shielded against releasing any revealing electromagnetic radiation and it even got a removable card rendering the unit inoperable when removed. Because the unit only encrypts and decrypts IP packages one could argue attacking it from the outside and physically is very difficult. Still one have to trust all the nodes on the inside to behave. Of course this unit is not for sale to anybody and probably expensive so it's not a general solution to the problem but it illustrates a different way to handle the problem.

4.4 Safe key entry

Even if the machine is able to keep encryption keys hidden while in use, one weak link is how to get the key into the machine in a safe way. Many protection systems rely on keyboard input, and depending on the operating system it is known that implementing a sniffer in user land is possible by subscribing to keyboard events. Windows is using the CTRL ALT DEL combination to put the machine in kernel mode to protect the entry from these loggers. TRESOR

[11] Our group tried an exploit called "limera1n" on an iPad1 to gain a shell, used the tools provided with iphone-dataprotection [Sigwald,], brute forced an easy 4 digit PIN, copied the whole NAND memory and carved for files not encrypted. We found several thousand files. Carving for per file encrypted files would not be possible as the seed is simply removed when the file is deleted. It's also worth mentioning that file table is not encrypted itself, so that file names and modified times are available without decrypting the file system image. It's only the file content that is encrypted.

as mentioned earlier does something similar as it asks for the password before the user land applications are allowed to start. Other methods could include importing keys in an encrypted form from a third party via smart cards or USB dongles.

5 Conclusion

Keeping cryptographic keys in memory can be exploited in several ways, from using buffer overruns to physical cold boot attacks. It has been shown that keys can be kept hidden inside CPU registers with the TRESOR implementation, but it's a kind of a hack and has some drawbacks in terms of compatibility. The trusted platform module is available as a tamper resistant hardware solution to keep keys protected and available in most new computer systems. It does not have much memory or processing power, but that will probably change in time as the demand increases. As seen with the latest products from Apple, phones and pads are equipped with hardware similar with TPM modules keeping device unique keys that have the potential of providing good security if implemented and used correctly. Simply having a TPM module does not give any protection, it's the way it's being used and if it's enabled at all. Lastly key input must be protected, and it's related to the convenience of availability as people want to avoid proving themselves all the time and they want features available even when the device is locked like receiving messages and listening to music.

6 Related work

Other interesting implementations to keep encryption keys secure can be seen in how some software vendors deliver dedicated dongles, often USB, that perform some cryptographic verification or even do some of the calculations on the device while the protected program is in use. Similar external modules can be made to house private keys for web servers to avoid hacking attacks resulting in loss of sensitive private keys.

References

[Abalenkovs et al., 2012] Abalenkovs, D., Bondarenko, P., Pathapati, V. K., Nordbø, A., Piatkivskyi, D., Rekdal, J. E., and Ruthven, P. B. (2012). Mobile forensics: Comparison of extraction and analyzing methods of ios and android. url-https://docs.google.com/open?id=0ByhZsNbhNQqXazd4eURvQnlzRFE.

[Anderson et al., 2006] Anderson, R., Bond, M., Clulow, J., and Skorobogatov, S. (2006). Cryptographic processors-a survey. *Proceedings of the IEEE*, 94(2):357 –369.

[Boileau, 2006] Boileau, A. (2006). Hit by a bus: Physical access attacks with firewire. http://www.security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf (Visited Dec 2012).

[Branscombe,] Branscombe, M. Trusting the trusted platform module. http://www.tomshardware.com/reviews/hardware-based-security-protects-pcs,1771-2.html visited Dec 2012.

[Carrier and Grand, 2004] Carrier, B. D. and Grand, J. (2004). A hardware-based memory acquisition procedure for digital investigations. *Digit. Investig.*, 1(1):50–60.

[coreboot.org,] coreboot.org, W. Jtag/bsdl guide. http://www.coreboot.org/JTAG/BSDL_Guide (Visited Dec 2012).

[Ebermann, 2012] Ebermann, P. (2012). How does the key schedule of rijndael looks for keysizes other than 128 bit? http://crypto.stackexchange.com/a/2496.

[Gibson and Laporte, 2012] Gibson, S. and Laporte, L. (2012). Elliptic curve crypto. http://twit.tv/show/security-now/374 Visited Descender 2012.

[Gómez-Miralles and Arnedo-Moreno, 2012] Gómez-Miralles, L. and Arnedo-Moreno, J. (2012). Versatile ipad forensic acquisition using the apple camera connection kit. *Comput. Math. Appl.*, 63(2):544–553.

[Halderman et al., 2009a] Halderman, J. A., Schoen, S. D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J. A., Feldman, A. J., Appelbaum, J., and Felten, E. W. (2009a). Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98.

[Halderman et al., 2009b] Halderman, J. A., Schoen, S. D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J. A., Feldman, A. J., Appelbaum, J., and Felten, E. W. (2009b). Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM*, 52(5):91–98. Source: https://citp.princeton.edu/research/memory/code/.

[Han et al., 2009] Han, L., Liu, J., Zhang, D., Han, Z., and Wei, X. (2009). A portable tpm scheme for general-purpose trusted computing based on efi. In *Multimedia Information Networking and Security, 2009. MINES '09. International Conference on*, volume 1, pages 140 –143.

[Huang et al., 2011] Huang, C.-W., Tu, Y.-H., Yeh, H.-C., Liu, S.-H., and Chang, C.-J. (2011). Image observation on the modified ecb operations in advanced encryption standard. In *Information Society (i-Society), 2011 International Conference on*, pages 264 –269.

[Intel,] Intel. Core™2 q6600. http://ark.intel.com/products/29765/Intel-Core2-Quad-Processor-Q6600-8M-Cache-2_40-GHz-1066-MHz-FSB.

[Maartmann-Moea et al., 2009] Maartmann-Moea, C., Thorkildsenb, S., and Årnes, A. (2009). The persistence of memory: Forensic identification and extraction of cryptographic keys. *ScienceDirect*. http://dx.doi.org/10.1016/j.diin.2009.06.002.

[Miller, 2011] Miller, C. (2011). Mobile attacks and defense. *Security Privacy, IEEE*, 9(4):68 –70.

[Mukhamedov et al., 2009] Mukhamedov, A., Gordon, A. D., and Ryan, M. (2009). Towards a verified reference implementation of a trusted platform module.

[Müller et al., 2011] Müller, T., Freiling, F. C., and Dewald, A. (2011). Tresor runs encryption securely outside ram. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 17–17, Berkeley, CA, USA. USENIX Association.

[Pabel,] Pabel, J. Frozen cache. http://blog.akkaya.de/jpabel/2010/12/31/After-the-FrozenCache-presentation.

[Rabaiotti and Hargreaves, 2010] Rabaiotti, J. and Hargreaves, C. (2010). Using a software exploit to image ram on an embedded system. *ScienceDirect*. http://dx.doi.org/10.1016/j.diin.2010.01.005.

[Sigwald,] Sigwald, J. Automatic ssh ramdisk creation and loading. http://code.google.com/p/iphone-dataprotection/wiki/README. (Visited December 2012).

[Simmons, 2011] Simmons, P. (2011). Security Through Amnesia: A Software-Based Solution to the Cold Boot Attack on Disk Encryption. *ArXiv e-prints*.

[Stallman,] Stallman, R. Can you trust your computer? http://www.gnu.org/philosophy/can-you-trust.html visited Dec 2012.

[Stephan and Vogel,] Stephan, B. and Vogel, L. Trusted computing, an animated short. http://www.lafkon.net/tc/ visited Dec 2012.

[Tarnovsky,] Tarnovsky, C. Deconstruction a 'secure' processor at black hat 2010. https://media.blackhat.com/bh-dc-10/video/Tarnovsky_Chris/BlackHat-DC-2010-Tarnovsky-DeconstructProcessor-video.m4v visited Dec 2012.

[Thales,] Thales. Tce 621 - high assurance ip encryption for nato. http://www.thales.no/pub/sites/index.php?siteID=23 (Visited Dec 2012).

[TrustedComputingGroup, 2012] TrustedComputingGroup (2012). Trusted computing group (tcg) timeline - february 2011. http://www.trustedcomputinggroup.org/files/resource_files/B8FF1287-1A4B-B294-D0423684DEB619FD/TCG%20Timeline_rev%20Feb%202011.pdf visited Dec 2012.

[Wang et al., 2008] Wang, H., Guo, Y., Zhao, X., and Chen, X. (2008). Keep passwords away from memory: Password caching and verification using tpm. In *Advanced Information Networking and Applications, 2008. AINA 2008. 22nd International Conference on*, pages 755 –762.

[Zdziarski, 2012] Zdziarski, J. (2012). *Hacking and Securing iOS Applications*. O'Reilly Media, first edition.