Error Correction and the Cryptographic Key

Nadia Anne Heninger

A Dissertation
Presented to the Faculty
of Princeton University
in Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance
by the Department of
Computer Science
Adviser: Bernard Chazelle

May 2011

Abstract

We will look at a collection of mathematical problems suggested by side-channel attacks against public key cryptosystems, and how the techniques inspired by this work relate to a variety of different applications.

First, we discuss the cold boot attack, a side-channel attack against disk encryption systems that uses the phenomenon of DRAM remanence to recover encryption keys from a running computer. In the course of the attack, however, there may be errors introduced in the keys that the attacker obtains. It turns out that the structure of the key data in an AES key schedule can allow an attacker to more efficiently recover the private key in the presence of such errors.

We extend this idea to a RSA private keys, and show how the structure of RSA private key data can allow an attacker to recover a key in the presence of random errors from 27% of the bits of the original key.

Most previous work on RSA key recovery used the lattice-based techniques introduced by Coppersmith for finding low-degree roots of polynomials mod numbers of unknown factorization. We show how this approach can be extended from the integers to the ring of polynomials, and give a new proof via lattice basis reduction of Guruswami-Sudan list-decoding of Reed-Solomon codes. These theorems are in fact instances of a general approach, which we extend to give an algorithm to find small solutions to polynomials modulo ideals in number fields and a list-decoding algorithm for multi-point algebraic-geometric codes.

Acknowledgements

I would like to begin by thanking Bernard Chazelle and the Princeton CS department.

I am deeply indebted to Neil Sloane and Henry Cohn for their mathematical mentorship and to everyone at AT&T Labs in Florham Park, Microsoft Research New England, and MIT for the wonderful time I spent in each.

This thesis would not exist without Alex Halderman, whose enthusiasm suddenly and unexpectedly set me on an entirely new trajectory. I am also grateful to Hovav Shacham, Jake Appelbaum, and Ed Felten for collaboration, conversation, and support.

And finally, thank you to my friends at Princeton whose conversation, companionship, and shenanigans have shaped both life and work: Sam Taylor, Vivek Shende, Clay Hambrick, Sharon Goldberg, and Jan Nordbotten.

The work in this thesis was supported by a National Science Foundation Graduate Research Fellowship, an AT&T Labs Fellowship, and internships at Microsoft Research New England.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"You can't hide secrets from the future with math."

MC Frontalot

An encryption algorithm is a mathematical procedure to obscure a message so that it may only be read by a recipient who possesses some secret key. Both cryptography (the making of such codes) and cryptanalysis (the breaking of such codes) have been studied since ancient times. The study of cryptography was revolutionized by the introduction of fast computers and the related study of computational complexity theory. At the same time, the development of cryptography has been intimately entwined with many of the most spectacular technological achievements of the twentieth century. Encryption protects the data on the laptops of CEOs as they travel, the telephone conversations of mobile phone users from eavesdroppers, the credit card numbers of shoppers as they purchase books online, and the email of political dissidents from their governments.

The fact that an encryption algorithm must run on an actual physical device gives an attacker two options: either try to break the underlying algorithm directly, or use some properties of the device performing the calculation to overcome the encryption without breaking the underlying algorithm. This latter is called a side-channel attack.

An attacker who wishes to directly attack a cryptosystem will typically try to take advantage of some property of the algorithm that permits an attack faster than brute-forcing the secret key. For example, in a symmetric cipher one might analyze the diffusion of each bit of the key. In an asymmetric cipher one might try to take advantage of the relationship between the public key and the private key in order to discover the private key from public information. In a cryptographer's paradise, it would be possible to rigorously prove that an adversary must expend a large amount of computational effort to learn this secret key from public information; one way of accomplishing this would be to design a public-key cryptosystem such that learning the secret key corresponds to solving a worst-case instance of an NP-hard problem, and then to prove that P$\neq$NP. At this time, it is not known how to accomplish either of these tasks, and the most widely used public-key algorithms are designed so that breaking the encryption should lead to progress on a well-studied mathematical problem that is hoped to be difficult to solve.

In this thesis, we will examine the interplay between the above ideas. First, we will show how an attacker can use the mathematical structure of real encryption algorithms to efficiently obtain a full break of a cryptosystem even if only partial information about a key is learned during the course of an attack. This structure acts as an error-correcting code for the key, and we will present algorithms for efficient decoding in certain cases. From this point of view, the diffusion properties of a good cipher are similar to the desired properties of a good error-correcting code. The last part of this thesis explores this connection between coding theory and cryptography from a different perspective, and gives a unified look at algorithms for a wide class of key recovery problems and the problem of list-decoding of Reed-Solomon codes. The algorithmic techniques used in the analysis of side-channel attacks can be extended to much broader mathematical principles, and result in efficient algorithms to solve "constructive" problems.

In Chapter 2, we develop a real side-channel attack, the "cold boot" attack. In this attack, an attacker uses the phenomenon of DRAM remanence to read encryption keys from the memory of a running computer. We investigate the properties of DRAM remanence experimentally and show how this attack can be used to break several popular disk encryption programs in practice. This chapter and parts of the next are based on work [49] originally published in the Proceedings of the 17th Usenix Security Symposium in collaboration with Alex Halderman, Seth Schoen, William Clarkson, William Paul, Joseph Calandrino, Ariel Feldman, Jacob Appelbaum, and Edward Felten.

However, information obtained using the cold boot attack may contain errors. In Chapter 3, we introduce a model for these errors and show how to correct AES and RSA private keys with such errors. In the model, key bits decay unidirectionally in random positions. Much of this chapter is based on work [50] originally published in the Proceedings of Crypto 2009 in collaboration with Hovav Shacham.

In Chapter 4, we take a second look at techniques used in the correction of RSA private keys in the case when the attacker learns a large block of consecutive bits of the private key. This technique, introduced by Coppersmith and extended by Howgrave-Graham, uses lattice basis reduction to find small solutions of a polynomial equation modulo a large divisor of a given integer. In this chapter, we show how to extend this technique beyond the integers to polynomials, number fields, and function fields, where they give an improvement to the Guruswami-Sudan algorithm for list-decoding of Reed-Solomon codes, an algorithm for finding small solutions of polynomial equations modulo ideals in number fields, and an algorithm for list decoding of algebraic-geometric codes. This chapter is based on work [27] originally published in the Proceedings of Innovations in Computer Science 2011 in collaboration with Henry Cohn.

RSA is designed to be based on the hardness of factoring, so that any progress on cryptanalyzing RSA would lead to improvements in factoring algorithms. After more than three decades, significant progress has been made on methods for factoring, but we still have neither a polynomial-time algorithm for factoring nor a proof that breaking RSA would lead to an efficient algorithm for factoring. However, the results of Chapter 3 introduce a new tradeoff: the known weaknesses of RSA with respect to certain side-channel attacks translate into powerful results in coding theory.

Chapter 2

Cold boot attacks on encryption keys

> "If a bad guy has unrestricted
> physical access to your computer,
> it's not your computer anymore."
>
> *Microsoft's Ten Immutable*
> *Laws of Security*

Contrary to popular assumption, dynamic RAM (DRAM), the main memory used in most modern computers, commonly retains its contents for several seconds after power is lost, even at room temperature and even when removed from a motherboard. In this chapter, we show how this phenomenon, called *memory remanence*, limits the ability of an operating system to protect cryptographic key material against an attacker with physical access to a machine, and we demonstrate how it can be exploited to defeat several popular on-the-fly disk encryption systems.

Most security practitioners have assumed that a computer's memory is erased almost immediately when it loses power, or that whatever data remains is difficult to retrieve without specialized equipment. We show that these assumptions are incorrect.

Without power, DRAM loses its contents gradually over a period of seconds, and data will persist for minutes or even hours if the chips are kept at low temperatures. Residual data can be recovered using simple, nondestructive techniques that require only momentary physical access to the machine.

We present a suite of attacks that exploit DRAM remanence to recover cryptographic keys held in memory. They pose a particular threat to laptop users who rely on disk encryption products. An adversary who steals a laptop while an encrypted disk is mounted could employ our attacks to access the contents, even if the computer is screen-locked or suspended when it is stolen. Because on-the-fly disk encryption software typically stores the encryption key in RAM while the disk is mounted, an attacker with access to the contents of RAM can learn the key and decrypt the disk.

We demonstrate this risk by defeating five popular disk encryption systems—BitLocker, TrueCrypt, FileVault, LoopAES, and dm-crypt—and we expect many similar products are also vulnerable.

Our attacks come in three variants of increasing resistance to countermeasures. The simplest is to reboot the machine and launch a custom kernel with a small memory footprint that gives the adversary access to the residual memory. A more advanced attack is to briefly cut power to the machine, then restore power and boot a custom kernel; this deprives the operating system of any opportunity to scrub memory before shutting down. An even stronger attack is to cut the power, transplant the DRAM modules to a second PC prepared by the attacker, and use it to extract their state. This attack additionally deprives the original BIOS and PC hardware of any chance to clear the memory on boot.

If the attacker is forced to cut power to the memory for too long, the data will become corrupted. We examine two methods for reducing corruption and for correcting errors in recovered encryption keys. The first is to cool the memory chips prior to cutting power, which dramatically prolongs data retention times. The second is to

apply algorithms we have developed for correcting errors in private and symmetric keys. These techniques can be used alone or in combination.

While our principal focus is disk encryption, any sensitive data present in memory when an attacker gains physical access to the system could be subject to attack. For example, we found that Mac OS X leaves the user's login password in memory, where we were able to recover it. SSL-enabled web servers are vulnerable, since they normally keep in memory private keys needed to establish SSL sessions. DRM systems may also face potential compromise; they sometimes rely on software to prevent users from accessing keys stored in memory, but attacks like the ones we have developed could be used to bypass these controls.

It may be difficult to prevent all the attacks that we describe even with significant changes to the way encryption products are designed and used, but in practice there are a number of safeguards that can provide partial resistance. We suggest a variety of mitigation strategies ranging from methods that average users can employ today to long-term software and hardware changes. However, each remedy has limitations and trade-offs, and we conclude that there is no simple fix for DRAM remanence vulnerabilities.

2.1 Previous work

Though our investigation was, to our knowledge, the first security study to focus on DRAM data remanence and the first to demonstrate how it can be used to conduct practical attacks against real disk encryption systems, we were not the first to suggest that data in DRAM might survive reboots or that this might have security implications. Hints that memory behavior did not fit the widely held mental models can be found in the literature going back more than thirty years.

Among electrical engineering circles, it has known since at least the 1970s that DRAM cell contents survive to some extent even at room temperature and that retention times can be increased by cooling. In a 1978 experiment [66], a DRAM showed no data loss for a full week without refresh when cooled with liquid nitrogen.

The first mention we can find in the computer security literature comes from Anderson [6], who briefly discusses remanence in his 2001 book:

> [A]n attacker can ... exploit ... memory remanence, the fact that many kinds of computer memory retain some trace of data that have been stored there. ... [M]odern RAM chips exhibit a wide variety of memory remanence behaviors, with the worst of them keeping data for several seconds even at room temperature...

Anderson cites Skorobogatov [100], who found significant data retention times with *static* RAMs at room temperature. Our results for modern DRAMs show even longer retention in some cases.

Anderson's main focus is on "burn-in" effects that occur when data is stored in RAM for an extended period. Gutmann [47, 48] also examines "burn-in," which he attributes to physical changes that occur in semiconductor memories when the same value is stored in a cell for a long time. Accordingly, Gutmann suggests that keys should not be stored in one memory location for longer than several minutes. Our findings concern a different phenomenon: the remanence effects we have studied occur in modern DRAMs even when data is stored only momentarily. These effects do not result from the kind of physical changes that Gutmann described, but rather from the capacitance of DRAM cells.

We owe the suggestion that modern DRAM contents can survive cold boot to Pettersson [86], who seems to have obtained it from Chow, Pfaff, Garfinkel, and Rosenblum [25]. Pettersson suggested that remanence across cold boot could be used to acquire forensic memory images and obtain cryptographic keys, although he did

not experiment with the possibility. Chow et al. discovered this property in the course of an experiment on data lifetime in running systems. While they did not exploit the property, they remark on the negative security implications of relying on a reboot to clear memory.

In a recent presentation, MacIver [70] stated that Microsoft considered memory remanence attacks in designing its BitLocker disk encryption system. He acknowledged that BitLocker is vulnerable to having keys extracted by cold-booting a machine when it is used in "basic mode" (where the encrypted disk is mounted automatically without requiring a user to enter any secrets), but he asserted that BitLocker is not vulnerable in "advanced modes" (where a user must provide key material to access the volume). He also discussed cooling memory with dry ice to extend the retention time. MacIver apparently has not published on this subject.

Other methods for obtaining memory images from live systems include using privileged software running under the host operating system [107], or using DMA transfer on an external bus [38], such as PCI [23], mini-PCI, Firewire [14, 34, 35], or PC Card. Unlike these techniques, our attacks do not require access to a privileged account on the target system, they do not require specialized hardware, and they are resistant to operating system countermeasures. Sophisticated tools have been developed for analyzing memory images, regardless of the acquisition method [111].

The intelligence community may well have been aware of the attacks we describe here, but we were unable to find any publications acknowledging this. A 1991 NSA report entitled "A Guide to Understanding Data Remanence in Automated Information Systems" (the "Forest Green Book") makes no mention of remanence in RAM, discussing only remanence on other storage media such as tapes and disks [81].

	Memory Type	Chip Maker	Memory Density	Make/Model	Year
A	SDRAM	Infineon	128Mb	Dell Dimension 4100	1999
B	DDR	Samsung	512Mb	Toshiba Portégé	2001
C	DDR	Micron	256Mb	Dell Inspiron 5100	2003
D	DDR2	Infineon	512Mb	IBM T43p	2006
E	DDR2	Elpida	512Mb	IBM x60	2007
F	DDR2	Samsung	512Mb	Lenovo 3000 N100	2007

Table 2.1: **Test systems** We experimented with six test systems (designated A–F) that encompass a range of recent DRAM architectures and circuit densities.

2.2 DRAM remanence

A DRAM cell is essentially a capacitor that encodes a single bit when it is charged or discharged [95, 48]. Over time, charge leaks out, and eventually the cell will lose its state, or, more precisely, it will decay to its *ground state*, either zero or one depending on how the cell is wired. To forestall this decay, each cell must be *refreshed*, meaning that the capacitor must be recharged to hold its value—this is what makes DRAM "dynamic." Manufacturers specify a maximum *refresh interval*—the time allowed before a cell is recharged—that is typically on the order of a few milliseconds. These times are chosen conservatively to ensure extremely high reliability for normal computer operations where even infrequent bit errors can cause problems, but, in practice, a failure to refresh any individual DRAM cell within this time has only a tiny probability of actually destroying the cell's contents.

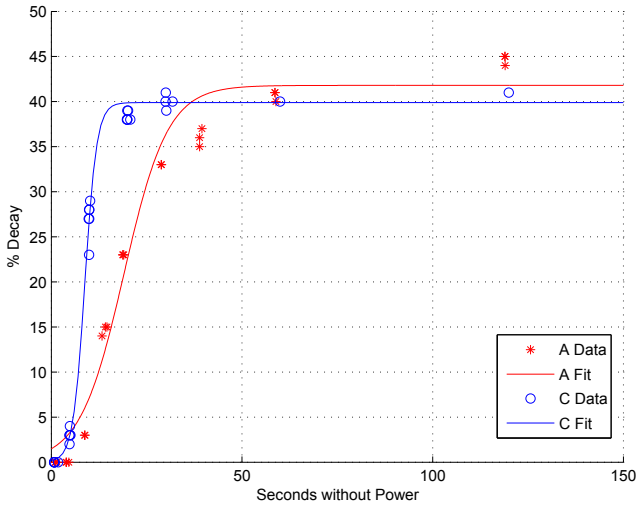
We conducted a series of experiments to characterize DRAM remanence effects and better understand the security properties of modern memories. We performed trials using PC systems with different memory technologies, as shown in Table 2.1. These systems included models from several manufacturers and were manufactured between 1999 and 2007.

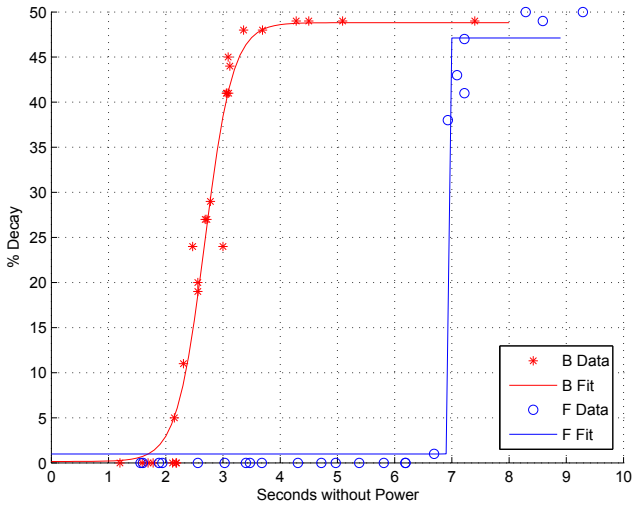
In each experiment, we filled representative memory regions with a pseudorandom test pattern, and read back the data after suspending refreshes for varying periods of time by cutting power to the machine. We measured the error rate for each sample as the number of bit errors (the Hamming distance from the pattern we had written) divided by the total number of bits. Fully decayed memory would have an error rate of approximately 50%, since half the bits would match by chance.

Decay at operating temperature Our first tests measured the decay rate of each machine's memory under normal operating temperature, which ranged from 25.5 °C to 44.1 °C. We found that the decay curves from different machines had similar shapes, with an initial period of slow decay, followed by an intermediate period of rapid decay, and then a final period of slow decay, as shown in Figure 2.1.

The dimensions of the decay curves varied considerably between machines, with the fastest exhibiting complete data loss in approximately 2.5 seconds and the slowest taking over a minute. Newer machines tended to exhibit a shorter time to total decay, possibly because newer chips have higher density circuits with smaller cells that hold less charge, but even the shortest times were long enough to enable some of our attacks. While some attacks will become more difficult if this trend continues, manufacturers may attempt to *increase* retention times to improve reliability or lower power consumption.

Decay at reduced temperature Colder temperatures are known to increase data retention times [66, 6, 114, 48, 101, 100]. We performed another series of tests to measure these effects. On machines A–D, we loaded a test pattern into memory, and, with the computer running, cooled the memory module to approximately −50 °C. We then cut power to the machine and maintained this temperature until power and refresh were restored. We achieved these temperatures using commonly available "canned air" duster products (see Section 2.3), which we discharged, with the can





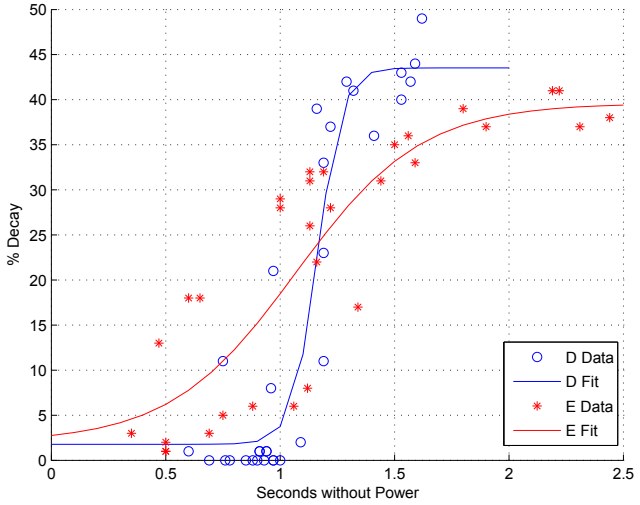


Figure 2.1: **DRAM decay curves** We measured DRAM decay curves for the six test systems. These reflect the average number of bits in a pseudorandom test pattern that changed value after a given interval without power. Data and fits are shown here for machines A and C *(top)*, B and F *(middle)*, and D and E *(bottom)*. All memories were running at normal operating temperature—i.e., without any special cooling. Note that graphs are at different scales.

	Seconds Without Power	Average Error Rates (%)	
		(No Cooling)	(−50 °C)
A	60	41	[no errors]
	300	50	0.000095
B	360	50	[no errors]
	600	50	0.000036
C	120	41	0.00105
	360	42	0.00144
D	40	50	0.025
	80	50	0.18

Table 2.2: **Cooling prolongs remanence** We measured DRAM error rates for systems A–D after different intervals without power, first at normal operating temperatures (no cooling) and then at a reduced temperature of −50 °C. Decay occurred much more slowly under the colder conditions.

inverted, directly onto the chips. As expected, we observed significantly slower rates of decay under these reduced temperatures (see Table 2.2). On all of our test systems, the decay was slow enough that an attacker who cut power for 1 minute would recover at least 99.9% of bits correctly.

As an extreme test of memory cooling, we performed another experiment using liquid nitrogen as an additional cooling agent. We first cooled the memory module of machine A to −50 °C using the "canned air" product. We then cut power to the machine, and quickly removed the DRAM module and placed it in a canister of liquid nitrogen. We kept the memory module submerged in the liquid nitrogen for 60 minutes, then returned it to the machine. We measured only 14,000 bit errors within a 1 MB test region (0.17% decay). This suggests that, even in modern memory modules, data may be recoverable for hours or days with sufficient cooling.

Decay patterns and predictability We observed that the DRAMs we studied tended to decay in highly nonuniform patterns. While these patterns varied from chip

to chip, they were very predictable in most of the systems we tested. Figure 2.2 shows the decay in one memory region from machine A after progressively longer intervals without power.

There seem to be several components to the decay patterns. The most prominent is a gradual decay to the "ground state" as charge leaks out of the memory cells. In the DRAM shown in Figure 2.2, blocks of cells alternate between a ground state of 0 and a ground state of 1, resulting in the series of horizontal bars. Other DRAM models and other regions within this DRAM exhibited different ground states, depending on how the cells are wired.

We observed a small number of cells that deviated from the "ground state" pattern, possibly due to manufacturing variation. In experiments with 20 or 40 runs, a few "retrograde" cells (typically $\sim 0.05\%$ of memory cells, but larger in a few devices) always decayed to the opposite value of the one predicted by the surrounding ground state pattern. An even smaller number of cells decayed in different directions across runs, with varying probabilities.

Apart from their eventual states, the *order* in which different cells decayed also appeared to be highly predictable. At a fixed temperature, each cell seems to decay after a consistent length of time without power. The relative order in which the cells decayed was largely fixed, even as the decay times were changed by varying the temperature. This may also be a result of manufacturing variations, which result in some cells leaking charge faster than others.

To visualize this effect, we captured degraded memory images, including those shown in Figure 2.2, after cutting power for intervals ranging from 1 second to 5 minutes, in 1 second increments. We combined the results into a video. Each test interval began with the original image freshly loaded into memory. We might have expected to see a large amount of variation between frames, but instead, most bits appear stable from frame to frame, switching values only once, after the cell's decay

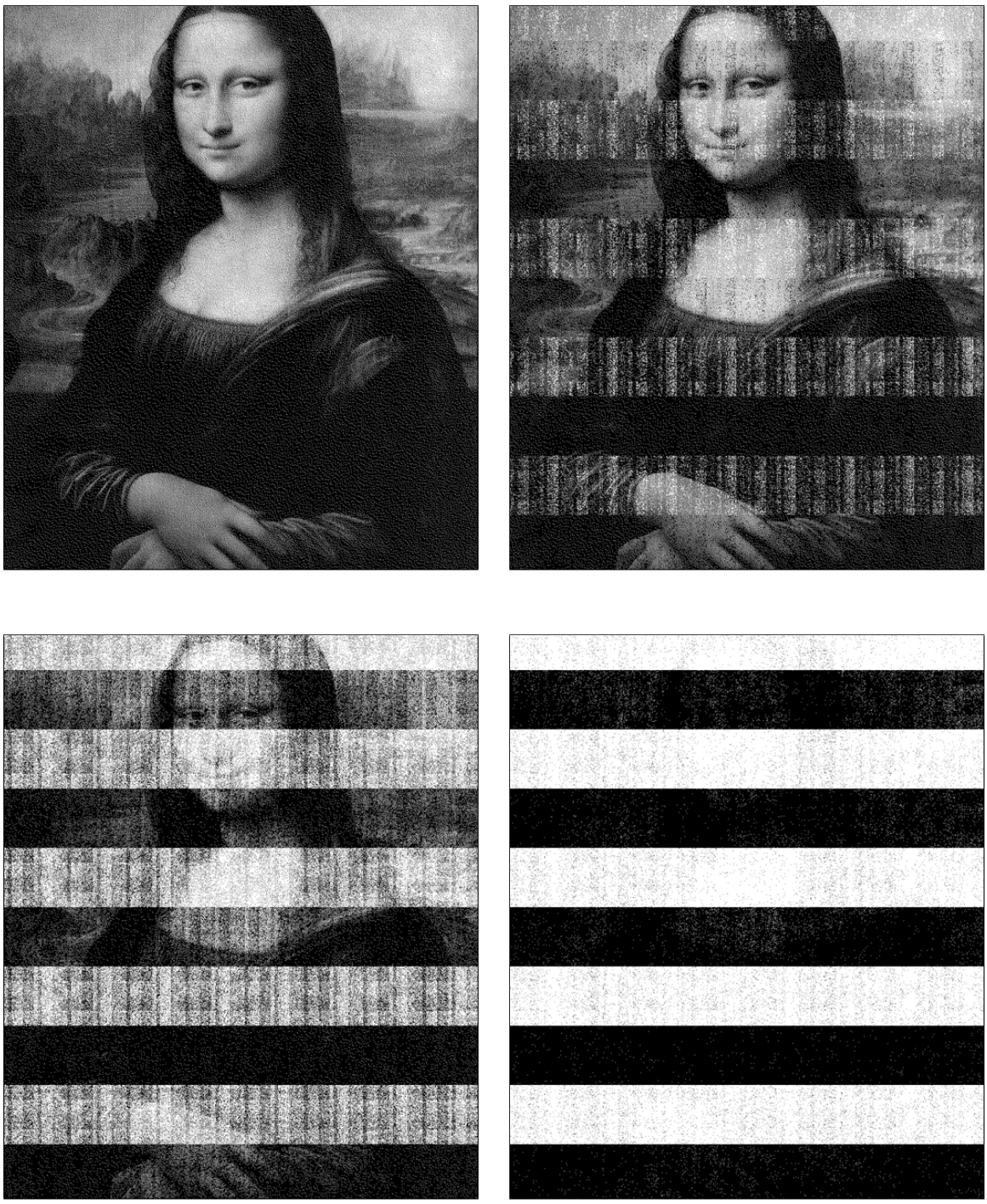


Figure 2.2: **Visualizing remanence and decay** We loaded a bitmap image into memory on test machine A, then cut power for varying intervals. After 5 seconds *(top left)*, the image is nearly indistinguishable from the original; it gradually becomes more degraded, as shown after 30 seconds, 60 seconds, and 5 minutes. Even after this longest trial, traces of the original remain. Note patterns due to ground states (horizontal bands) and physical variations in the chip (fainter vertical bands).

interval. The video also shows that the decay intervals themselves follow higher order patterns, likely related to the physical geometry of the DRAM.

BIOS footprints and memory wiping Even if memory contents remain intact while power is off, the system BIOS may overwrite portions of memory when the machine boots. In the systems we tested, the BIOS overwrote only relatively small fractions of memory with its own code and data, typically a few megabytes concentrated around the bottom of the address space.

On many machines, the BIOS can perform a destructive memory check during its Power-On Self Test (POST). Most of the machines we examined allowed this test to be disabled or bypassed (sometimes by enabling an option called "Quick Boot").

On other machines, mainly high-end desktops and servers that support ECC memory, we found that the BIOS cleared memory contents without any override option. ECC memory must be set to a known state to avoid spurious errors if memory is read without being initialized [10], and we believe many ECC-capable systems perform this wiping operation whether or not ECC memory is installed.

ECC DRAMs are not immune to retention effects, and an attacker could transfer them to a non-ECC machine that does not wipe its memory on boot. Indeed, ECC memory could turn out to *help* the attacker by making DRAM more resistant to bit errors.

2.3 Tools and attacks

Extracting residual memory contents requires no special equipment. When the system is powered on, the memory controller immediately starts refreshing the DRAM, reading and rewriting each bit value; at this point, the values are fixed, decay halts, and programs running on the system can read any residual data using normal memory-access instructions.

One challenge is that booting the system will necessarily overwrite some portions of memory. While we observed in our tests that the BIOS typically overwrote only a small fraction of memory, loading a full operating system would be very destructive. One solution is to use tiny special-purpose programs that, when booted from either a warm or cold reset state, copy the memory contents to some external medium with minimal disruption to the original state.

Our memory-imaging tools make use of several different attack vectors to boot a system and extract the contents of its memory. For simplicity, each saves memory images to the medium from which it was booted.

PXE network boot Most modern PCs support network booting via Intel's Preboot Execution Environment (PXE) [55], which provides rudimentary startup and network services. We implemented a tiny (9 KB) standalone application that can be booted via PXE and whose only function is streaming the contents of system RAM via a UDP-based protocol. Since PXE provides a universal API for accessing the underlying network hardware, the same binary image will work unmodified on any PC system with PXE support. In a typical attack setup, a laptop connected to the target machine via an Ethernet crossover cable runs DHCP and TFTP servers as well as a simple client application for receiving the memory data. We have extracted memory images at rates up to 300 Mb/s (around 30 seconds for a 1 GB RAM) with gigabit Ethernet cards.

USB drives Alternatively, most PCs can boot from an external USB device such as a USB hard drive or flash device. We implemented a small (10 KB) plug-in for the SYSLINUX bootloader [7] that can be booted from an external USB device or a regular hard disk. It saves the contents of system RAM into a designated data partition on this device. We succeeded in dumping 1 GB of RAM to a flash drive in approximately 4 minutes.

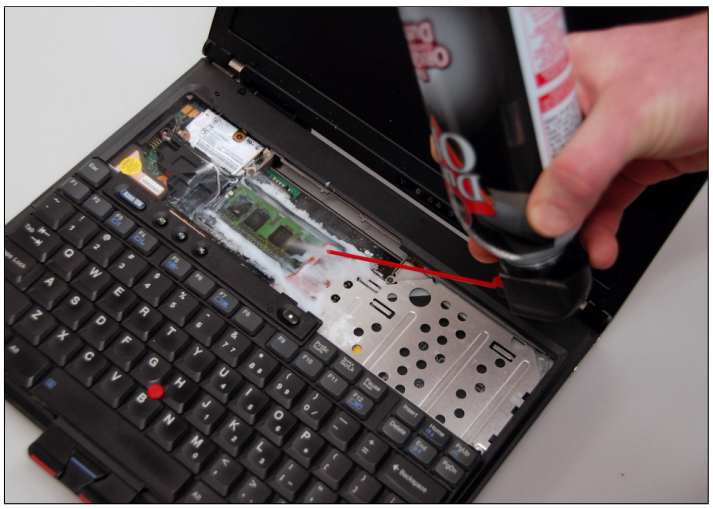


Figure 2.3: **Advanced cold-boot attack** An advanced cold-boot attack involves reducing the temperature of the memory chips while the computer is still running, then physically moving them to another machine that the attacker has configured to read them without overwriting any data. Before powering off the computer, the attacker can spray the chips with "canned air," holding the container in an inverted position so that it discharges cold liquid refrigerant instead of gas *(top)*. This cools the chips to around −50 °C *(middle)*. At this temperature, the data will persist for several minutes after power loss with minimal error, even if the memory modules are removed from the computer *(bottom)*.

EFI boot Some recent computers, including all Intel-based Macintosh computers, implement the Extensible Firmware Interface (EFI) instead of a PC BIOS. We have also implemented a memory dumper as an EFI netboot application. We have achieved memory extraction speeds up to 136 Mb/s, and we expect it will be possible to increase this throughput with further optimizations.

iPods We have installed memory imaging tools on an Apple iPod (which behaves like a USB disk) so that it can be used to covertly capture memory dumps without impacting its functionality as a music player. This provides a plausible way to conceal the attack in the wild.

Imaging attacks

An attacker could use tools like these in a number of ways, depending on his level of access to the system and the countermeasures employed by hardware and software. The simplest attack is to reboot the machine and configure the BIOS to boot the memory extraction tool. A warm boot, invoked with the operating system's restart procedure, will normally ensure that refresh is not interrupted and the memory has no chance to decay, though software will have an opportunity to wipe sensitive data. A cold boot, initiated using the system's restart switch or by briefly removing power, may result in a small amount of decay, depending on the memory's retention time, but denies software any chance to scrub memory before shutting down.

Even if an attacker cannot force a target system to boot memory extraction tools, or if the target employs countermeasures that erase memory contents during boot, an attacker with sufficient physical access can transfer the memory modules to a computer he controls and use it to extract their contents. Cooling the memory before powering it off slows the decay sufficiently to allow it to be transplanted with minimal data loss. Widely-available "canned air" dusters, usually containing a compressed

fluorohydrocarbon refrigerant, can easily be used for this purpose. When the can is discharged in an inverted position, as shown in Figure 2.3, it dispenses its contents in liquid form instead of as a gas. The rapid drop in pressure inside the can lowers the temperature of the discharge, and the subsequent evaporation of the refrigerant causes a further chilling. By spraying the contents directly onto memory chips, we can cool their surfaces to $-50\,^{\circ}\mathrm{C}$ and below. If the DRAM is cooled to this temperature before power is cut, and kept cold, we can achieve nearly lossless data recovery even after the chip is out of the computer for several minutes.

2.4 Identifying keys in memory

After extracting the memory from a running system, an attacker needs some way to locate the cryptographic keys. This is like finding a needle in a haystack, since the keys might occupy only tens of bytes out of gigabytes of data. Simple approaches, such as attempting decryption using every block of memory as the key, are intractable if the memory contains even a small amount of decay.

We have developed fully automatic techniques for locating encryption keys in memory images, even in the presence of errors. The most commonly used symmetric ciphers typically expand the secret key into a *key schedule* consisting of a sequence of several round keys. We target this key schedule instead of the key itself, searching for blocks of memory that satisfy the properties of a valid key schedule. Using these methods we have been able to recover keys from closed-source encryption programs without having to disassemble them and reconstruct their key data structures, and we have even recovered partial key schedules that had been overwritten by another program when the memory was reallocated.

Although previous approaches to key recovery do not require a scheduled key to be present in memory, they have other practical drawbacks that limit their usefulness

for our purposes. Shamir and van Someren [97] propose visual and statistical tests of randomness which can quickly identify regions of memory that might contain key material, but their methods are prone to false positives that complicate testing on decayed memory images. Even perfect copies of memory often contain large blocks of random-looking data that might pass these tests (e.g., compressed files). Pettersson [86] suggests a plausibility test for locating a particular program data structure that contains key material based on the range of likely values for each field. This approach requires the operator to manually derive search heuristics for each encryption application, and it is not very robust to memory errors.

2.4.1 Identifying AES keys

For 128-bit keys, the AES key schedule consists of 11 round keys, each made up of four 32-bit words. Figure 3.1 illustrates the key schedule generation algorithm. The first round key is equal to the key itself. To generate the first word of the next round key, the last word of the previous round key is run through a sequence of operations known as the key schedule core, which rotates the bytes left, runs each byte through an S-box to map it to a new value, and XORs the output with a round-dependent value. The second word is generated by XORing the first word of the new round key with the second word of the previous round key, the third word is the XOR of the second word of the new round key with the third word of the previous round key, and the fourth word is the XOR of the third word of the new round key with the fourth word of the previous round key. This procedure is repeated nine more times to generate all 11 round keys.

The key schedule algorithm for 256-bit AES is very similar.

In order to identify scheduled AES keys in a memory image, we propose the following algorithm:

1. Iterate through each byte of memory. Treat the following block of 176 or 240 bytes of memory as an AES key schedule.

2. For each word in the potential key schedule, calculate the Hamming distance from that word to the key schedule word that should have been generated from the surrounding words.

3. If the total number of bits violating the constraints on a correct AES key schedule is sufficiently small, output the key.

We implemented this algorithm for 128- and 256-bit AES keys in an application called `keyfind`. The program takes a memory image as input and outputs a list of likely keys. It assumes that key schedules are contained in contiguous regions of memory and in the byte order used in the AES specification [1]; this can be adjusted to target particular cipher implementations. A threshold parameter controls how many bit errors will be tolerated in candidate key schedules. We apply a quick test of entropy to reduce false positives.

We expect that this approach can be applied to many other ciphers. For example, to identify DES keys based on their key schedule, calculate the distance from each potential subkey to the permutation of the key.

2.4.2 Identifying tweak keys

A similar method works to identify keys for tweakable encryption modes [67], which are commonly used in disk encryption systems.

LRW LRW augments a block cipher E (and key K_1) by computing a "tweak" X for each data block and encrypting the block using the formula $E_{K_1}(P \oplus X) \oplus X$. A tweak key K_2 is used to compute the tweak, $X = K_2 \otimes I$, where I is the logical block identifier. The operations $\oplus$ and $\otimes$ are performed in the finite field $GF(2^{128})$.

In order to speed tweak computations, implementations commonly precompute multiplication tables of the values $K_2 x^i \mod P$, where x is the primitive element and P is an irreducible polynomial over $GF(2^{128})$ [58]. In practice, $Qx \mod P$ is computed by shifting the bits of Q left by one and possibly XORing with P.

We can search for keys by searching for the structure of such a multiplication table in memory.

XEX and XTS For XEX [90] and XTS [54] modes, the tweak for block j in sector I is $X = E_{K_2}(I) \otimes x^j$, where I is encrypted with AES and x is the primitive element of $GF(2^{128})$. Assuming the key schedule for K_2 is kept in memory, we can use the AES key finding techniques to find this tweak key.

2.4.3 Identifying RSA keys

Methods proposed for identifying RSA private keys range from the purely algebraic (Shamir and van Someren suggest, for example, multiplying adjacent key-sized blocks of memory [97]) to the *ad hoc* (searching for the RSA Object Identifiers found in ASN.1 key objects [88]). The former ignores the widespread use of standard key formats, and the latter seems insufficiently robust.

The most widely used format for an RSA private key is specified in PKCS #1 [93] as an ASN.1 object of type RSAPrivateKey with the following fields: version, modulus n, publicExponent e, privateExponent d, prime1 p, prime2 q, exponent1 $d \bmod (p-1)$, exponent2 $d \bmod (q-1)$, coefficient $q^{-1} \bmod p$, and optional other information. This object, packaged in DER encoding, is the standard format for storage and interchange of private keys.

This format suggests two techniques we might use for identifying RSA keys in memory: we could search for known *contents* of the fields, or we could look for memory

that matches the *structure* of the DER encoding. We tested both of these approaches on a computer running Apache 2.2.3 with `mod_ssl`.

One value in the key object that an attacker is likely to know is the public modulus. (In the case of a web server, the attacker can obtain this and the rest of the public key by querying the server.) We tried searching for the modulus in memory and found several matches, all of them instances of the server's public or private key.

We also tested a key finding method described by Ptacek [88] and others: searching for the RSA Object Identifiers that should mark ASN.1 key objects. This technique yielded only false positives on our test system.

Finally, we experimented with a new method, searching for identifying features of the DER-encoding itself. We looked for the sequence identifier (0x30) followed a few bytes later by the DER encoding of the RSA version number and then by the beginning of the DER encoding of the next field (02 01 00 02). This method found several copies of the server's private key, and no false positives. To locate keys in decayed memory images, we can adapt this technique to search for sequences of bytes with low Hamming distance to these markers and check that the subsequent bytes satisfy some heuristic entropy bound.

2.5 Attacking disk encryption software

Encrypting hard drives is an increasingly common countermeasure against data theft, and many users assume that disk encryption products will protect sensitive data even if an attacker has physical access to the machine. A California law adopted in 2002 [21] requires disclosure of possible compromises of personal information, but offers a safe harbor whenever data was "encrypted." Though the law does not include any specific technical standards, many observers have recommended the use of full-disk or file system encryption to obtain the benefit of this safe harbor. (At least 38 other states

have enacted data breach notification legislation [82].) Our results below suggest that disk encryption, while valuable, is not necessarily a sufficient defense.

We have applied some of the tools developed in this chapter to attack popular on-the-fly disk encryption systems. The most time-consuming parts of these tests were generally developing system-specific attacks and setting up the encrypted disks. Actually imaging memory and locating keys took only a few minutes and were almost fully automated by our tools. We expect that most disk encryption systems are vulnerable to such attacks.

BitLocker BitLocker, which is included with some versions of Windows Vista and Windows 7, operates as a filter driver that resides between the file system and the disk driver, encrypting and decrypting individual sectors on demand. The keys used to encrypt the disk reside in RAM, in scheduled form, for as long as the disk is mounted.

In a paper released by Microsoft, Ferguson [40] describes BitLocker in enough detail for us both to discover the roles of the various keys and to program an independent implementation of the BitLocker encryption algorithm without reverse engineering any software. BitLocker uses the same pair of AES keys to encrypt every sector on the disk: a sector pad key and a CBC encryption key. These keys are, in turn, indirectly encrypted by the disk's master key. To encrypt a sector, the plaintext is first XORed with a pad generated by encrypting the byte offset of the sector under the sector pad key. Next, the data is fed through two diffuser functions, which use a Microsoft-developed algorithm called Elephant. The purpose of these un-keyed functions is solely to increase the probability that modifications to any bits of the ciphertext will cause unpredictable modifications to the entire plaintext sector. Finally, the data is encrypted using AES in CBC mode using the CBC encryption key. The initialization vector is computed by encrypting the byte offset of the sector under the CBC encryption key.

We have created a fully-automated demonstration attack called BitUnlocker. It consists of an external USB hard disk containing Linux, a custom SYSLINUX-based bootloader, and a FUSD [39] filter driver that allows BitLocker volumes to be mounted under Linux. To use BitUnlocker, one first cuts the power to a running Windows Vista system, connects the USB disk, and then reboots the system off of the external drive. BitUnlocker then automatically dumps the memory image to the external disk, runs `keyfind` on the image to determine candidate keys, tries all combinations of the candidates (for the sector pad key and the CBC encryption key), and, if the correct keys are found, mounts the BitLocker encrypted volume. Once the encrypted volume has been mounted, one can browse it like any other volume in Linux. On a modern laptop with 2 GB of RAM, we found that this entire process took approximately 25 minutes.

BitLocker differs from other disk encryption products in the way that it protects the keys when the disk is not mounted. In its default "basic mode," BitLocker protects the disk's master key solely with the Trusted Platform Module (TPM) found on many modern PCs. This configuration, which may be quite widely used [40], is particularly vulnerable to our attack, because the disk encryption keys can be extracted with our attacks even if the computer is powered off for a long time. When the machine boots, the keys will be automatically loaded into RAM from the TPM before the login screen, without the user having to enter any secrets.

It appears that Microsoft is aware of this problem [70] and recommends configuring BitLocker in "advanced mode," where it protects the disk key using the TPM along with a password or a key on a removable USB device. However, even with these measures, BitLocker is vulnerable if an attacker gets to the system while the screen is locked or the computer is asleep (though not if it is hibernating or powered off).

FileVault Apple's FileVault disk encryption software has been examined and reverse-engineered in some detail [112]. In Mac OS X 10.4, FileVault uses 128-bit AES in CBC mode. A user-supplied password decrypts a header that contains both the AES key and a second key k_2 used to compute IVs. The IV for a disk block with logical index I is computed as HMAC-SHA1$_{k_2}(I)$.

We used our EFI memory imaging program to extract a memory image from an Intel-based Macintosh system with a FileVault volume mounted. Our `keyfind` program automatically identified the FileVault AES key, which did not contain any bit errors in our tests.

With the recovered AES key but not the IV key, we can decrypt 4080 bytes of each 4096 byte disk block (all except the first AES block). The IV key is present in memory. Assuming no bits in the IV key decay, an attacker can identify it by testing all 160-bit substrings of memory to see whether they create a plausible plaintext when XORed with the decryption of the first part of the disk block. The AES and IV keys together allow full decryption of the volume using programs like `vilefault` [113].

In the process of testing FileVault, we discovered that Mac OS X 10.4 and 10.5 keep multiple copies of the user's login password in memory, where they are vulnerable to imaging attacks. Login passwords are often used to protect the default keychain, which may protect passphrases for FileVault disk images.

TrueCrypt TrueCrypt is a popular open-source disk encryption product for the Windows, Mac OS, and Linux platforms. It supports a variety of ciphers, including AES, Serpent, and Twofish. In version 4, all ciphers used LRW mode; in version 5, they use XTS mode (see Section 2.4.2). TrueCrypt stores a cipher key and a tweak key in the volume header for each disk, which is then encrypted with a separate key derived from a user-entered password.

We tested TrueCrypt versions 4.3a and 5.0a running on a Linux system. We mounted a volume encrypted with a 256-bit AES key, then briefly cut power to the system and used our memory imaging tools to record an image of the retained memory data. In both cases, our `keyfind` program was able to identify the 256-bit AES encryption key, which did not contain any bit errors. For TrueCrypt 5.0a, `keyfind` was also able to recover the 256-bit AES XTS tweak key without errors.

To decrypt TrueCrypt 4 disks, we also need the LRW tweak key. We observed that TrueCrypt 4 stores the LRW key in the four words immediately preceding the AES key schedule. In our test memory image, the LRW key did not contain any bit errors. (Had errors occurred, we could have recovered the correct key by applying the techniques we will develop in Section 3.5.)

dm-crypt Linux kernels starting with 2.6 include built-in support for dm-crypt, an on-the-fly disk encryption subsystem. The dm-crypt subsystem handles a variety of ciphers and modes, but defaults to 128-bit AES in CBC mode with non-keyed IVs.

We tested a dm-crypt volume created and mounted using the LUKS (Linux Unified Key Setup) branch of the `cryptsetup` utility and kernel version 2.6.20. The volume used the default AES-CBC format. We briefly powered down the system and captured a memory image with our PXE kernel. Our `keyfind` program identified the correct 128-bit AES key, which did not contain any bit errors. After recovering this key, an attacker could decrypt and mount the dm-crypt volume by modifying the `cryptsetup` program to allow input of the raw key.

Loop-AES Loop-AES is an on-the-fly disk encryption package for Linux systems. In its recommended configuration, it uses a so-called "multi-key-v3" encryption mode, in which each disk block is encrypted with one of 64 encryption keys. By default, it encrypts sectors with AES in CBC mode, using an additional AES key to generate IVs.

We configured an encrypted disk with Loop-AES version 3.2b using 128-bit AES encryption in "multi-key-v3" mode. After imaging the contents of RAM, we applied our `keyfind` program, which revealed the 65 AES keys. An attacker could identify which of these keys correspond to which encrypted disk blocks by performing a series of trial decryptions. Then, the attacker could modify the Linux `losetup` utility to mount the encrypted disk with the recovered keys.

Loop-AES attempts to guard against the long-term memory burn-in effects described by Gutmann [48] and others. For each of the 65 AES keys, it maintains two copies of the key schedule in memory, one normal copy and one with each bit inverted. It periodically swaps these copies, ensuring that every memory cell stores a 0 bit for as much time as it stores a 1 bit. Not only does this fail to prevent the memory remanence attacks that we describe here, but it also makes it easier to identify which keys belong to Loop-AES and to recover the keys in the presence of memory errors. After recovering the regular AES key schedules using a program like `keyfind`, the attacker can search the memory image for the inverted key schedules. Since very few programs maintain both regular and inverted key schedules in this way, those keys are highly likely to belong to Loop-AES. Having two related copies of each key schedule provides additional redundancy that can be used to identify which bit positions are likely to contain errors.

2.6 Countermeasures and their limitations

Memory remanence attacks are difficult to prevent because cryptographic keys in active use must be stored *somewhere*. Potential countermeasures focus on discarding or obscuring encryption keys before an adversary might gain physical access, preventing memory extraction software from executing on the machine, physically protecting the DRAM chips, and making the contents of memory decay more readily.

Suspending a system safely Simply locking the screen of a computer (i.e., keeping the system running but requiring entry of a password before the system will interact with the user) does not protect the contents of memory. Suspending a laptop's state to RAM ("sleeping") is also ineffective, even if the machine enters a screen-locked state on awakening, since an adversary could simply awaken the laptop, power-cycle it, and then extract its memory state. Suspending to disk ("hibernating") may also be ineffective unless an externally held secret is required to decrypt the disk (or the memory image stored on disk) when the system is awakened.

With most disk encryption systems, users can protect themselves by powering off the machine completely when it is not in use, then guarding the machine for a minute or so until the contents of memory have decayed sufficiently. Though effective, this countermeasure is inconvenient, since the user will have to wait through the lengthy boot process before accessing the machine again.

Suspending can be made safe by requiring a password or other external secret to reawaken the machine and encrypting the contents of memory under a key derived from the password. If encrypting all of memory is too expensive [25], the system could encrypt only those pages or regions containing important keys. An attacker might still try to guess the password and check his guesses by attempting decryption (an offline password-guessing attack), so systems should encourage the use of strong passwords and employ password strengthening techniques [19] to make checking guesses slower. Some existing systems, such as Loop-AES, can be configured to suspend safely in this sense, although this is usually not the default behavior [9].

Storing keys differently Our attacks show that using precomputation to speed cryptographic operations can make keys more vulnerable, because redundancy in the precomputed values helps the attacker reconstruct keys in the presence of memory errors. To mitigate this risk, implementations could avoid storing precomputed values,

instead recomputing them as needed and erasing the computed information after use. This improves resistance to memory remanence attacks but can carry a significant performance penalty. (These performance costs are negligible compared to the access time of a hard disk, but disk encryption is often implemented on top of disk caches that are fast enough to make them matter.)

Implementations could transform the key as it is stored in memory in order to make it more difficult to reconstruct in the case of errors. This problem has been considered from a theoretical perspective; Canetti et al. [22] define the notion of an *exposure-resilient function* (ERF) whose input remains secret even if all but some small fraction of the output is revealed. This carries a performance penalty because of the need to reconstruct the key before using it.

Physical defenses It may be possible to physical defend memory chips from being removed from a machine, or to detect attempts to open a machine or remove the chips and respond by erasing memory. In the limit, these countermeasures approach the methods used in secure coprocessors [37] and could add considerable cost to a PC. However, a small amount of memory soldered to a motherboard would provide moderate defense for sensitive keys and could be added at relatively low cost.

Architectural changes Some countermeasures involve changes to the computer's architecture that might make future machines more secure. DRAM systems could be designed to lose their state quickly, though this might be difficult given the need to keep the probability of decay within a DRAM refresh interval vanishingly small. Key-store hardware could be added—perhaps inside the CPU—to store a few keys securely while erasing them on power-up, reset, and shutdown. Some proposed architectures would routinely encrypt the contents of memory for security purposes [65, 60, 36]; these would prevent the attacks we describe as long as the keys are reliably destroyed on reset or power loss.

Encrypting in the disk controller Another approach is to perform encryption in the disk controller rather than in software running on the main CPU and to store the key in the controller's memory instead of the PC's DRAM. In a basic form of this approach, the user supplies a secret to the disk at boot, and the disk controller uses this secret to derive a symmetric key that it uses to encrypt and decrypt the disk contents [96].

For this method to be secure, the disk controller must erase the key from its memory whenever the computer is rebooted. Otherwise, an attacker could reboot into a malicious kernel that simply reads the disk contents. For similar reasons, the key must also be erased if an attacker attempts to transplant the disk to another computer.

While we leave an in-depth study of encryption in the disk controller to future work, we did perform a cursory test of two hard disks with this capability, the Seagate Momentus 5400 FDE.2 and the Hitachi 7K200. We found that they do not appear to defend against the threat of transplantation. We attached both disks to a PC and confirmed that every time we powered on the machine, we had to enter a password via the BIOS in order to decrypt the disks. However, once we had entered the password, we could disconnect the disks' SATA cables from the motherboard (leaving the power cables connected), connect them to another PC, and read the disks' contents on the second PC without having to re-enter the password.

Trusted computing Trusted Computing hardware, in the form of Trusted Platform Modules (TPMs) [105] is now deployed in some personal computers. Though useful against some attacks, most TPMs deployed in PCs today do not prevent the attacks described here.

Such hardware generally does not perform bulk data encryption itself; instead, it monitors the boot process to decide (or help other machines decide) whether it is safe

to store a key in RAM. If a software module wants to safeguard a key, it can arrange that the usable form of that key will not be stored in RAM unless the boot process has gone as expected [70]. However, once the key is stored in RAM, it is subject to our attacks. Today's TPMs can prevent a key from being loaded into memory for use, but they cannot prevent it from being captured once it is in memory.

In some cases using a TPM can make the problem worse. BitLocker, in its default "basic mode," protects the disk keys solely with Trusted Computing hardware. When the machine boots, BitLocker automatically loads the keys into RAM from the TPM without requiring the user to enter any secrets. Unlike other disk encryption systems we studied, this configuration is at risk even if the computer has been shut down for a long time—the attacker needs only power on the machine to have the keys loaded back into memory, where they are vulnerable to our attacks.

2.7 Conclusions

Contrary to common belief, DRAMs hold their values for surprisingly long time intervals without power or refresh. We show that this fact enables attackers to extract cryptographic keys and other sensitive information from memory despite the operating system's efforts to secure memory contents. The attacks we describe are practical—for example, we have used them to defeat several popular disk encryption systems. These results imply that disk encryption on laptops, while beneficial, does not guarantee protection.

The attack described in this chapter is an example of a side-channel attack, an attack that takes advantage of the physical properties of a system running encryption rather than attacking the algorithm itself. Examples of these attacks use timing information, power consumption, or electromagnetic radiation to learn information about an encryption process. Such attacks had been modeled in the theoretical

cryptography literature by Micali and Reyzin [77], who stated as their first informal axiom of a "physically observable" cryptographic system that *"Computation, and only computation, leaks information"*. Subsequently, a large number of works developed "leakage-resilient" cryptographic schemes secure in this model.

However, as noted by Goldwasser in Eurocrypt 2009 [44], the cold boot attack violates this assumption, and thus new ideas are needed to develop cryptographic schemes that are provably secure even against a very weak form of such attacks.

There seems to be no easy remedy for memory remanence attacks. Ultimately, it might become necessary to treat DRAM as untrusted and to avoid storing sensitive data there, but this will not be feasible until architectures are changed to give running software a safe place to keep secrets.

Chapter 3

Reconstructing cryptographic keys

> "For example the paper tries to factor $N = pq$ by writing it as a set of binary equations over the bits of p and q."
>
> *Ten Reasons why a Paper is Rejected from a Crypto Conference*

In the previous chapter, we saw how an attacker can use physical effects to extract cryptographic keys from a running computer and use this information to break encryption programs in practice.

However, as Figure 2.2 shows, the information extracted from memory may contain a large number of errors. In this chapter, we will explore some techniques that an attacker can use to recover full cryptographic keys even in the presence of error.

In the case of a perfectly random symmetric encryption key, it would seem that an attacker with only partial knowledge of a key has few options other than brute force decryptions or finding a weakness in the encryption algorithm itself.

The naïve approach to key error correction, a brute-force search over keys with a low Hamming distance from the decayed key that was retrieved from memory, is

computationally burdensome even with a moderate amount of error. Consider the case of a 128-bit key with 10% error. The size of the search space, that is, the number of keys with Hamming distance 12 from the recovered key, is $\binom{128}{12} > 2^{54}$, and in expectation the attacker must try half of those keys before finding the correct key.

Our algorithms achieve significantly better performance by considering data other than the raw form of the key. In practice, most encryption programs speed up computation by storing data precomputed from the encryption keys—for block ciphers, this is most often a key schedule, with subkeys for each round; for RSA, this includes the public key as well as an extended form of the private key which includes the primes p and q and several other values derived from d. This data contains much more structure than the key itself, and we can use this structure to efficiently reconstruct the original key even in the presence of errors.

We will see how the structure of this data can in many cases be used as a sort of "error-correcting code" for the key itself, allowing an attacker to efficiently reconstruct a key even in the presence of a large amount of error.

For asymmetric encryption keys, the question of whether the public key could be used to reconstruct a private key from partial data was first studied decades ago. Our error models pose a more difficult setting for this problem. We observe that implementations of RSA typically also precompute several redundant values from the private key, and we show how to take advantage of the algebraic relationships between these values to correct errors in RSA private keys even in the presence of bitwise errors.

Our approach to key reconstruction has the advantage that it is completely self-contained, in that we can recover the key without having to test the decryption of ciphertext. The data derived from the key, and not the decoded plaintext, provides a certificate of the likelihood that we have found the correct key.

The algorithms presented in this chapter can efficiently reconstruct keys when as few as 27% of the bits are known, depending on the type of key.

These results imply an interesting trade-off between efficiency and security. All of the disk encryption systems we studied in the previous chapter (see 2.5) precompute key schedules and keep them in memory for as long as the encrypted disk is mounted. While this practice saves some computation for each disk block that needs to be encrypted or decrypted, we find that it greatly simplifies key recovery attacks.

3.1 Related work

To our knowledge, we were the first to look at the problem of reconstructing symmetric encryption keys in this particular setting, though similar questions arise in the analysis of other side-channel attacks.

For asymmetric keys, there has been a great deal of work on both factoring and reconstructing RSA private keys given incomplete information about the private key.

Maurer [73] shows that integers can be factored in polynomial time given access to an oracle answering ϵn yes/no questions.

In a slightly stricter model, the algorithm has access to a fixed subset of consecutive bits of the integer factors or RSA private keys. Rivest and Shamir [89] first solved the problem for a 2/3-fraction of the least significant bits of a factor using integer programming. This was improved to 1/2 of the least or most significant bits of a factor using lattice-reduction techniques pioneered by Coppersmith [28]; we refer the reader surveys by Boneh [15] and May [75] as well as May's Ph. D. thesis [74] for bibliographies. More recently, Herrmann and May extended these techniques to efficiently factor given at most $\log \log N$ known blocks of bits [51]. We will discuss Coppersmith's techniques in much greater detail in Chapter 4.

The problem of reconstructing RSA private keys in the context of the cold boot attack can be viewed as a further relaxation of the conditions on access to the key bits to a fully random subset. The lattice-reduction techniques used in the works cited above are not directly applicable to our problem because they rely on recovering *consecutive* bits of the key (expressed as small integer solutions to modular equations), whereas the missing bits we seek to find are randomly distributed throughout the degraded keys. It is possible to express our reconstruction problem as a knapsack, and there are lattice techniques for solving knapsack problems (see, e.g., Nguyen and Stern [83]), but we have not managed to improve on our solution by this approach.

Inspired by cold boot attacks, Akavia, Goldwasser, and Vaikuntanathan [4] formally introduced *memory attacks*, a class of side-channel attacks in which the adversary is leaked a (shrinking) function of the secret key. One research direction, pursued by Akavia, Goldwasser, and Vaikuntanathan and, in followup work, Naor and Segev [80], is constructing cryptosystems provably secure against memory attacks.[1] Our work in this chapter can be viewed as the cryptanalytic counterpart to this line of work, to evaluate the security of existing cryptosystems against memory attacks.

3.2 Modeling the decay

To explain our error model, we have found it useful to adopt terminology from coding theory. We may imagine that the expanded key schedule forms a sort of *error correcting code* for the key, and the problem of reconstructing a key from memory may be recast as the problem of finding the closest *code word* (valid key schedule) to the data once it has been passed through a channel that has introduced bit errors.

Our experiments with DRAM remanence in the previous chapter (see Section 2.2) showed that almost all memory bits tend to decay to predictable ground states, with

[1] There has been substantial other recent work on designing cryptosystems secure in related key-leakage models (e.g., [87, 33, 5]); for a survey, see Goldwasser's invited talk at Eurocrypt 2009 [44] and the references therein.

only a tiny fraction flipping in the opposite direction. In describing the algorithms in this chapter, we assume, for simplicity, that all bits decay to the same ground state. (They can be implemented without this requirement, assuming that the ground state of each bit is known.)

If we assume we have no knowledge of the decay patterns other than the ground state, we can model the decay as a *binary asymmetric channel*, in which the probability of a 1 flipping to 0 is some fixed δ_0 and the probability of a 0 flipping to a 1 is some fixed δ_1.

In practice, the probability of decaying to the ground state approaches 1 as time goes on, while the probability of flipping in the opposite direction remains relatively constant and tiny (less than 0.1% in our tests). The ground state decay probability can be approximated from recovered key schedules by counting the fraction of 1s and 0s, assuming that the original key schedule contained roughly equal proportions of each value.

We also observed that bits tended to decay in a predictable order that could be learned over a series of timed decay trials, although the actual order of decay appeared fairly random with respect to location. That is, when comparing two decayed memory images that had been read after the computer was powered off for different lengths of time, a bit that had decayed in the shorter of the trials was very likely to also have decayed in the longer of the trials. An attacker with the time and physical access to run such a series of tests could easily adapt any of the approaches in this section to take this order into account and improve the performance of the error-correction. With unidirectional decay, an attacker who knew the exact decay order of each bit would be able determine the set of bits that could possibly have decayed in the image by locating in the decay order the transition between ground state and not-yet decayed bits, and then restrict her attention to only those bits that could possibly have decayed.

Ideally such tests would be able to replicate the conditions of the memory extraction exactly, but knowledge of the decay order combined with an estimate of the fraction of bit flips is enough to give a very good estimate of an individual decay probability of each bit. This probability can be used in our reconstruction algorithms to prioritize guesses.

For simplicity and generality, we will analyze the algorithms assuming no knowledge of this decay order.

3.3 Reconstructing DES keys

We begin with a relatively simple application of the above ideas: an error-correction technique for DES keys. Before software can encrypt or decrypt data with DES, it must expand the secret key K into a set of *round keys* that are used internally by the cipher. The set of round keys is called the *key schedule*; since it takes time to compute, programs typically cache it in memory as long as K is in use. The DES key schedule consists of 16 round keys, each a permutation of a 48-bit subset of bits from the original 56-bit key. A copy of each bit of the key appears in about 14 of the 16 round keys. For example, bit 1 of the key is replicated in bit 20 of round key 1, bit 10 of round key 2, bit 16 of round key 3, bit 24 of round key 4, bit 40 of round key 5, does not appear in round key 6, bit 26 of round key 7, and so on.

In coding theory terms, we can treat the DES key schedule as a repetition code: the message is a single bit, and the corresponding codeword is a sequence of n copies of this bit.

We begin with a partially decayed DES key schedule. For each bit of the key, we consider the n bits extracted from memory that were originally all identical copies of that key bit. Since we know roughly the probability that each bit decayed $0 \to 1$ or

$1 \to 0$, we can calculate whether the extracted bits were more likely to have resulted from the decay of repetitions of 0 or repetitions of 1.

If δ_0 is the probability of a 1 flipping to a 0, and we assume that each flip occurs uniformly at random, the probability that at least k of the n copies of a bit have flipped is

$$\sum_{i=k}^{n} \binom{n}{k} \delta_0^k (1-\delta_0)^{n-k}.$$

For $\delta_0 = .05$, that is, a 5% probability of a bit flip, the probability that a bit with 14 copies is incorrectly decoded is the probability that at least seven of the bits have flipped; this is less than 2×10^{-6}. We can very conservatively bound the probability of incorrectly decoding any of the 56 bits using a union bound; in this example the probability is about 10^{-4}.

This technique can be trivially extended to correct errors in Triple DES keys. Triple DES applies the same key schedule algorithm to two or three 56-bit key components (depending on the version of Triple DES).

3.4 Reconstructing AES keys

AES is a more modern cipher than DES, and it uses a key schedule with a more complex structure, but nevertheless we can efficiently reconstruct keys.

The algorithm in this section has since been improved (see Tsow [106] who recovers AES 128 key schedules from 30% of bits known), but we present our proof of concept algorithm here for completeness.

Recall that for 128-bit keys, the AES key schedule consists of 11 round keys, each made up of four 32-bit words. The first round key is equal to the key itself. Each subsequent word of the key schedule is generated either by XORing two earlier words, or by performing an operation called the key schedule core (in which the bytes of a

word are rotated and each byte is mapped to a new value) on an earlier word and XORing the result with another earlier word.

Instead of trying to correct an entire key at once, we can examine a smaller set of the bits at a time and then combine the results. This separability is enabled by the high amount of linearity in the key schedule. Consider a "slice" of the first two round keys consisting of byte i from words 1–3 of the first two round keys, and byte $i - 1$ from word 4 of the first round key (see Figure 3.1). This slice is 7 bytes long, but is uniquely determined by the 4 bytes from the first round key.

Our algorithm exploits this fact as follows. For each possible set of 4 key bytes, we generate the relevant 3 bytes of the next round key, and we order these possibilities by the likelihood that these 7 bytes might have decayed to the corresponding bytes extracted from memory. Now we may recombine four slices into a candidate key, in order of decreasing likelihood. For each candidate key, we calculate the key schedule. If the likelihood of this key schedule decaying to the bytes we extracted from memory is sufficiently high, we output the corresponding key.

When the decay is largely unidirectional, this algorithm will almost certainly output a unique guess for the key. This is because a single flipped bit in the key results in a cascade of bit flips through the key schedule, half of which are likely to flip in the "wrong" direction.

Our implementation of this algorithm is able to reconstruct keys with 7% of the bits decayed in a fraction of a second. It succeeds within 30 seconds for about half of keys with 15% of bits decayed.

This idea can be extended to 256-bit keys by dividing the words of the key into two sections—words 1–3 and 8, and words 4–7, for example—then comparing the words of the third and fourth round keys generated by the bytes of these words and combining the result into candidate round keys to check.

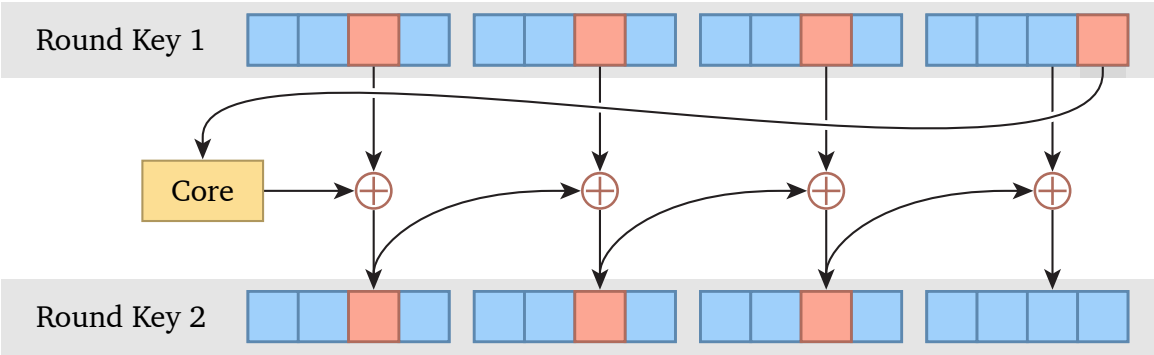


Figure 3.1: **Error correction for AES keys** In the AES-128 key schedule, four bytes from each round key completely determine three bytes of the next round key, as shown here. Our error correction algorithm "slices" the key into four groups of bytes with this property. It computes a list of likely candidate values for each slice, then checks each combination to see if it is a plausible key.

3.5 Reconstructing tweak keys

The same methods can be applied to reconstruct keys for tweakable encryption modes [67], which are commonly used in disk encryption systems.

LRW Recall from Section 2.4.2 that LRW implementations commonly precompute a large multiplication table generated from the tweak key, each entry of which is generated by shifting and possibly XORing with a known value. An entire multiplication table will contain many copies of nearly all of the bits of K_2, allowing us to reconstruct the key in much the same way as the DES key schedule.

As an example, we apply this method to reconstruct the LRW key used by the TrueCrypt 4 disk encryption system. TrueCrypt 4 precomputes a 4048-byte multiplication table consisting of 16 blocks of 16 lines of 4 words of 4 bytes each. Line 0 of block 14 contains the tweak key.

The multiplication table is generated line by line from the LRW key by iteratively applying the shift-and-XOR multiply function to generate four new values, and then XORing all combinations of these four values to create 16 more lines of the table. The shift-and-XOR operation is performed 64 times to generate the table, using the

irreducible polynomial $P = x^{128} + x^7 + x^2 + x + 1$. For any of these 64 values, we can shift right i times to recover $128 - (8 + i)$ of the bits of K_2, and use these recovered values to reconstruct K_2 with high probability.

XEX and XTS For XEX [90] and XTS [54] modes, assuming the tweak key schedule is kept in memory, we can use the AES key reconstruction techniques above to reconstruct the tweak key.

3.6 Reconstructing RSA private keys

In this section, we give an algorithm for the problem of reconstructing RSA private keys given a random δ-fraction of their bits. For RSA keys with small public exponent, our algorithm reconstructs the private key with high probability when $\delta \geq 0.27$. The runtime analysis of our algorithm relies on an assumption (Conjecture 3.6.3) and is thus heuristic; but we have verified experimentally that it succeeds with high probability.

We will first present the algorithm, and then give a combinatorial analysis of the algorithm's runtime behavior for random inputs that shows that it will succeed in expected quadratic time when $\delta \geq .27$. The runtime analysis depends crucially on both a uniformly random distribution of known bits and the assumption that the effect of a bit error during reconstruction is propagated uniformly through subsequent bits of the key.

Instead of a key schedule, the redundancy that our algorithm makes use of is five components of the RSA private key: p, q, d, d_p, and d_q. We can use known bits in d, d_p, and d_q to make progress where bits in p and q are not known. To relate d to the rest of the private key, we make use of techniques due to Boneh, Durfee, and Frankel [17]; to relate d_p and d_q to the rest of the private key, we make new observations about

the structure of RSA keys that may be of independent interest. This is discussed in Section 3.6.1.

If the algorithm has access to fewer components of the RSA private key, the algorithm will still perform well given a sufficiently large fraction of the bits. For example, it can efficiently recover a key given

$\delta = .27$ fraction of the bits of p, q, d, d_p, and d_q.

$\delta = .42$ fraction of the bits of p, q, and d.

$\delta = .57$ fraction of the bits of p and q.

The reconstruction algorithm itself, described in Section 3.6.2, is elementary and does not make use of the lattice basis reduction or integer programming techniques that have been applied to other kinds of RSA key reconstruction problems. At each step, it branches to explore all possible keys, and prunes these possibilities using our understanding of the structure of RSA keys and the partial information we are given about key bits. We give an analysis of the algorithm for random inputs in Section 3.6.3. We obtain a sharp threshold around $2 - 2^{(4/5)} \approx 27\%$ of known key bits. Below this threshold, the expected number of keys examined is exponential in the number of bits of the key, and above this threshold, the expected number of keys examined is close to linear. Note that this threshold applies only to our particular approach. We suspect these results could be improved using more sophisticated methods.

Finally, we have implemented our algorithm and performed extensive experiments using it. The results are described in Section 3.6.7. The algorithm's observed behavior matches our analytically derived bounds and validates the heuristic assumptions made in the analysis.

Small public-exponent RSA. Our algorithm is specialized to the case where *the public exponent e is small.* The small-e case is, for historical reasons, the overwhelm-

ingly common one in deployed RSA applications such as SSL/TLS. For example, until recently Internet Explorer would reject TLS server certificates with an RSA public exponent longer than 32 bits [18, p. 8]. The choice $e = 65537 = 2^{16} + 1$ is especially widespread. Of the certificates observed in the UCSD TLS Corpus [115] (which was obtained by surveying frequently-used TLS servers), 99.5% had $e = 65537$, and all had e at most 32 bits.

3.6.1 RSA private keys

The PKCS#1 standard specifies [94, Sect. A.1.2] that an RSA private key include at least the following information:

- the (n-bit) modulus N and public exponent e;
- the private exponent d;
- the prime factors p and q of N;
- d modulo $p - 1$ and $q - 1$, respectively denoted d_p and d_q; and
- the inverse of q modulo p, denoted q_p^{-1}.

In practice, an RSA key in exactly this format can be recovered from the RAM of a machine running Apache with OpenSSL (see Section 2.4.3). The first items — N and e — make up the public key and are already known to the attacker. A naïve RSA implementation would use d to perform the private-key operation $c \mapsto c^d \bmod N$, but there is a more efficient approach, used by real-world implementations such as OpenSSL, that is enabled by the remaining private-key entries. In this approach, one computes the answer modulo p and q as $(c \bmod p)^{d_p}$ and $(c \bmod q)^{d_q}$, respectively; then combines these two partial answers by means of q_p^{-1} and the Chinese Remainder Theorem (CRT). This approach requires two exponentiations but of smaller numbers, and is approximately four times as fast as the naïve method [76, p. 613].

Recall that the Chinese Remainder Theorem tells us that for p and q relatively prime, there is a unique solution $m \bmod pq$ to the equations

$$m \equiv m_p \pmod{p} \quad \text{and} \quad m \equiv m_q \pmod{q}. \tag{3.6.1}$$

One way of solving for m is to find two values b_p and b_q such that

$$b_p \equiv 1 \pmod{p} \quad \text{and} \quad b_p \equiv 0 \pmod{q}$$

and

$$b_q \equiv 0 \pmod{p} \quad \text{and} \quad b_q \equiv 1 \pmod{q}.$$

Then we can take

$$m = m_p b_p + m_q b_q$$

and it is easy to see that m satisfies 3.6.1.

Given $q_p^{-1} = q^{-1} \bmod p$ we can calculate b_p and b_q as follows. We know that

$$q q_p^{-1} \equiv 1 \bmod p$$

and in particular, there is some a such that

$$q q_p^{-1} + ap = 1$$

Then $q q_p^{-1}$ satisfies the properties we need from b_p, and ap satisfies the properties we need from b_q.

Observe that the information included in PKCS#1 private keys is *highly redundant*. In fact, knowledge of any single one of p, q, d, d_p, d_q, or q_p^{-1} is sufficient to reveal

the factorization of N.[2] It is this redundancy that we will use in reconstructing a corrupted RSA key.

We now derive relations between p, q, d, d_p, and d_q that will be useful in mounting the attack. The first such relation is obvious:

$$N = pq \ . \tag{3.6.2}$$

Next, since d is the inverse of e modulo $\varphi(N) = (p-1)(q-1) = N - p - q + 1$, we have

$$ed \equiv 1 \pmod{\varphi(N)}$$

and, modulo $p-1$ and $q-1$,

$$ed_p \equiv 1 \pmod{p-1} \qquad \text{and} \qquad ed_q \equiv 1 \pmod{q-1} \ .$$

As it happens, it is more convenient for us to write explicitly the terms hidden in the three congruences above, obtaining

$$ed = k(N - p - q + 1) + 1 \tag{3.6.3}$$

$$ed_p = k_p(p-1) + 1 \tag{3.6.4}$$

$$ed_q = k_q(q-1) + 1 \ . \tag{3.6.5}$$

It may appear that we have thereby introduced three new unknowns: k, k_p, and k_q. But in fact for small e we can compute each of these three variables given even a badly-degraded version of d.

[2]This is obvious for p and q and well known for d (cf. [32]); d_p reveals p as $\gcd(a^{ed_p-1} - 1, N)$ with high probability for random a provided $d_p \neq d_q$, and similarly for d_q; if d_p and d_q are equal to each other then they are also equal to d. Phong Nguyen observes (in personal communication, August 2009) that knowledge of q_p^{-1} also reveals the factorization of N, as follows. Let A be q_p^{-1}. Then we have $Aq = 1 + Bp$ for some B, and, multiplying again by q, we have $Aq^2 = q + BN$. Written another way, this means $Aq^2 - q = 0 \bmod N$, from which q can be recovered via Coppersmith's method.

Computing k. The following argument, due to Boneh, Durfee, and Frankel [17], shows that k must be in the range $0 < k < e$. We know $d < \varphi(N)$. Assume $e \leq k$; then $ed < k\varphi(N) + 1$, which contradicts (3.6.3). The case $k = 0$ is also impossible, as can be seen by reducing (3.6.3) modulo e. This shows that we can enumerate all possible values of k, having assumed that e is small.

For each such choice k', define

$$\tilde{d}(k') \stackrel{\text{def}}{=} \left\lfloor \frac{k'(N+1)+1}{e} \right\rfloor \ .$$

As Boneh, Durfee, and Frankel observe, when k' equals k, this gives an excellent approximation for d:

$$0 \leq \tilde{d}(k) - d \leq k(p+q)/e < p+q \ .$$

In particular, when p and q are balanced, we have $p + q < 3\sqrt{N}$, which means that $\tilde{d}(k)$ agrees with d on their $\lfloor n/2 \rfloor - 2$ most significant bits. (Our analysis applies also in the less common case when p and q are unbalanced, but we omit the details.) This means that small-public-exponent RSA leaks half the bits of the private exponent in one of the candidate values $\tilde{d}(1), \ldots, \tilde{d}(e-1)$.

The same fact allows us to go in the other direction, using information about d to determine k, as again noted by Boneh, Durfee, and Frankel. We are given $\tilde{d}$, a corrupted version of d. We enumerate $\tilde{d}(1), \ldots, \tilde{d}(e-1)$ and check which of these agrees, in its more significant half, with the known bits of $\tilde{d}$. Provided that $\delta n/2 \gg \lg e$, there will be just one value of k' for which $\tilde{d}(k')$ matches; that value is k. Even for 1024-bit N and 32-bit e, there is, with overwhelming probability, enough information to compute k for any δ we consider in this paper. This observation has two implications:

1. we learn the correct k used in (3.6.3); and

2. we correct the more significant half of the bits of $\tilde{d}$, by copying from $\tilde{d}(k)$.

Computing k_p and k_q. Once we have determined k, we can compute k_p and k_q. First, observe that by an analysis like that above, we can show that $0 < k_p, k_q < e$. This, of course, means that $k_p = (k_p \bmod e)$ and $k_q = (k_q \bmod e)$; when we solve for k_p and k_q modulo e, this will reveal the actual values used in (3.6.4) and (3.6.5). Now, reducing equations (3.6.2)–(3.6.5) modulo e, we obtain the following congruences:

$$N \equiv pq \tag{3.6.6}$$

$$0 \equiv k(N - p - q + 1) + 1 \tag{3.6.7}$$

$$0 \equiv k_p(p - 1) + 1 \tag{3.6.8}$$

$$0 \equiv k_q(q - 1) + 1 \ . \tag{3.6.9}$$

These are four congruences in four unknowns: p, q, k_p, and k_q; we solve them as follows. From (3.6.8) and (3.6.9) we write $(p-1) \equiv -1/k_p$ and $(q-1) \equiv -1/k_q$; we substitute these into the equation obtained from using (3.6.6) to reexpress $\varphi(N)$ in (3.6.7): $0 \equiv k(N-p-q+1)+1 \equiv k(p-1)(q-1)+1 \equiv k(-1/k_p)(-1/k_q)+1 \equiv k/(k_pk_q)+1$, or

$$k + k_pk_q \equiv 0 \ . \tag{3.6.10}$$

Next, we return to (3.6.7), substituting in (3.6.8), (3.6.9), and (3.6.10):

$$\begin{aligned} 0 &\equiv k(N - p - q + 1) + 1 \\ &\equiv k(N - 1) - k(p - 1 + q - 1) + 1 \\ &\equiv k(N - 1) - (-k_pk_q)(-1/k_p - 1/k_q) + 1 \\ &\equiv k(N - 1) - (k_q + k_p) + 1 \ ; \end{aligned}$$

we solve for k_p by substituting $k_q = -k/k_p$, obtaining

$$0 \equiv k(N-1) - (k_p - k/k_p) + 1 \ ,$$

or, multiplying both sides by k_p and rearranging,

$$k_p^2 - \big[k(N-1)+1\big]k_p - k \equiv 0 \ . \tag{3.6.11}$$

This congruence is easy to solve modulo e and, in the common case where e is prime, has two solutions, just as it would over $\mathbb{C}$. One of the two solutions is the correct value of k_p; and it is easy to see, by symmetry, that the other must be the correct value of k_q. We need therefore try just two possible assignments to k_p and k_q in reconstructing the RSA key. When e has m distinct prime factors, there may be up to 2^m roots [17].

If information about d_p and d_q only is available, k is not known and there are e possible choices for the pair (k_p, k_q). The analysis for this case was given by Percival [85].

Note that we also learn the values of p and q modulo e. If we then use the procedure outlined below to decode the r least significant bits of p (up to a list of possibilities), we will know $p \bmod e2^r$; we can then factor N, provided $r + \lg e > n/4$, by applying Boneh, Durfee, and Frankel's Corollary 2.2 ([17]; a generalization of Coppersmith's attack on RSA with known low-order bits [28, Theorem 5] that removes the restriction that the partial knowledge of p must be modulo a power of 2).

3.6.2 The reconstruction algorithm

Once we have the above relationships between key data, the remainder of the attack consists of enumerating all possible partial keys and pruning those that do not satisfy these constraints. More precisely, given bits 1 through $i-1$ of a potential key, generate

all combinations of values for bit i of p, q, d, d_p, d_q, and keep a candidate combination if it satisfies (3.6.2), (3.6.3), (3.6.4), and (3.6.5) mod 2^i.

The remainder of this section details how to generate and prune these partial solutions.

Our algorithm is quite similar to the algorithm given by Percival [85] for reconstructing an RSA private key given partial information about d_p and d_q, which he obtained using a timing side-channel attack. Our contribution is mainly in relating five components of the private key to each other (Section 3.6.1), and in providing an analysis of the algorithm's runtime behavior (Section 3.6.3).

In what follows, we assume that we know the values of k_p and k_q. When equation (3.6.11) has two distinct solutions, we must run the algorithm twice, once for each of the possible assignments to k_p and k_q.

Let $p\,[i]$ denote the ith bit of p, where the least significant bit is bit 0, and similarly index the bits of q, d, d_p and d_q. Let $\tau(x)$ denote the exponent of the largest power of 2 that divides x.

As p and q are large primes, we know they are odd, so we can correct $p\,[0] = q\,[0] = 1$. It follows that $2 \mid p-1$, so $2^{1+\tau(k_p)} \mid k_p(p-1)$. Thus, reducing (3.6.4) modulo $2^{1+\tau(k_p)}$, we have

$$ed_p \equiv 1 \pmod{2^{1+\tau(k_p)}} \ .$$

Since we know e, this allows us immediately to correct the $1+\tau(k_p)$ least significant bits of d_p. Similar arguments using (3.6.5) and (3.6.3) allow us to correct the $1+\tau(k_q)$ and $2+\tau(k)$ bits of d_q and d, respectively.

What is more, we can easily see that, having fixed bits $< i$ of p, a change in $p\,[i]$ affects d_p not in bit i but in bit $i+\tau(k_p)$; and, similarly, a change in $q\,[i]$ affects $d_q\left[i+\tau(k_q)\right]$, and a change in $p\,[i]$ or $q\,[i]$ affects $d\left[i+\tau(k)\right]$. When any of k, k_p, or k_q is odd, this is just the trivial statement that changing bit i of the right-hand side

of an equation changes bit i of the left-hand side. Powers of 2 in k_p shift left the bit affected by $p\,[i]$, and similarly for the other variables.

Having recovered the least-significant bits of each of our five variables, we now attempt to recover the remaining bits. For each bit index i, we consider a slice of bits:

$$p\,[i] \qquad q\,[i] \qquad d\left[i+\tau(k)\right] \qquad d_p\left[i+\tau(k_p)\right] \qquad d_q\left[i+\tau(k_q)\right] \quad .$$

For each possible solution up to bit slice $i-1$, generate all possible solutions up to bit slice i that agree with that solution at all but the ith position. If we do this for all possible solutions up to bit slice $i-1$, we will have enumerated all possible solutions up to bit slice i. Above, we already described how to obtain the only possible solution up to $i=0$; this is the solution we use to start the algorithm. The factorization of N will be revealed in one or more of the possible solutions once we have reached $i=\lfloor n/2 \rfloor$.[3]

All that remains is how to lift a possible solution (p', q', d', d'_p, d'_q) for slice $i-1$ to possible solutions for slice i. Naïvely it might seem that there are $2^5 = 32$ such possibilities, but in fact there are at most 2 and, for large enough δ, almost always fewer.

First, observe that we have four constraints on the five variables: equations (3.6.2), (3.6.3), (3.6.4), and (3.6.5). By plugging in the values up to slice $i-1$, we obtain from each of these a constraint on slice i, namely values $c_1, \ldots, c_4$ such that the following

[3]In fact, as we discussed in Section 3.6.1 above, information sufficient to factor N will be revealed much earlier, at $i = \lceil n/4 - \lg e \rceil$.

congruences hold modulo 2:

$$\begin{aligned} p[i] + q[i] &\equiv c_1 \pmod 2 \\ d\left[i + \tau(k)\right] + p[i] + q[i] &\equiv c_2 \pmod 2 \\ d_p\left[i + \tau(k_p)\right] + p[i] &\equiv c_3 \pmod 2 \\ d_q\left[i + \tau(k_q)\right] + q[i] &\equiv c_4 \pmod 2 \ . \end{aligned} \tag{3.6.12}$$

For example, if N and $p'q'$ agree at bit i, $c_1 = 0$; if not, $c_1 = 1$. Four linearly independent constraints on five unknowns means that there are exactly two possible choices for bit slice i satisfying these four constraints. (Expressions for the c_is are given in (3.6.14).)

Next, it may happen that we know the correct value of one or more of the bits in the slice, through our partial knowledge of the private key. These known bits might agree with neither, one, or both of the possibilities derived from the constraints above. If neither possible extension of a solution up to $i-1$ agrees with the known bits, that solution is pruned. If δ is sufficiently large, the number of possibilities at each i will be kept small.

3.6.3 Algorithm runtime analysis

The main result of this section is summarized in the following informal theorem.

Theorem 3.6.1. *Given the values of a $\delta = .27$ fraction of the bits of p, q, d, $d \bmod p$, and $d \bmod q$, the algorithm will correctly recover an n-bit RSA key in expected $O(n^2)$ time with probability $1 - \frac{1}{n^2}$.*

The running time of the algorithm is determined by the number of partial keys examined. To bound the total number of keys seen by the program, we will first understand how the structure of the constraints on the RSA key data determines

the number of partial solutions generated at each step of the algorithm. Then we will use this understanding to calculate some of the distribution of the number of solutions generated at each step over the randomness of p and q and the missing bits. Finally we characterize the global behavior of the program and provide a bound on the probability that the total number of branches examined over the entire run of the program is too large.

Lifting solutions mod 2^i. The process of generating bit i of a partial solution given bits 0 through $i-1$ can be seen as lifting a solution to the constraint equations mod 2^i to a solution mod 2^{i+1}. Hensel's lemma characterizes the conditions when this is possible.

Lemma 3.6.2 (Multivariate Hensel's Lemma)**.** *A root* $\mathbf{r} = (r_1, r_2, \ldots, r_n)$ *of the polynomial* $f(x_1, x_2, \ldots, x_n) \bmod \pi^i$ *can be lifted to a root* $\mathbf{r} + \mathbf{b} \bmod \pi^{i+1}$ *if* $\mathbf{b} = (b_1\pi^i, b_2\pi^i, \ldots, b_n\pi^i)$, $0 \le b_j \le \pi - 1$ *is a solution to the equation*

$$f(\mathbf{r} + \mathbf{b}) = f(\mathbf{r}) + \sum_j b_j \pi^i f_{x_j}(\mathbf{r}) \equiv 0 \pmod{\pi^{i+1}} \ .$$

(Here, f_{x_j} is the partial derivative of f with respect to x_j.)

We can rewrite the lemma using the notation of Section 3. Write $\mathbf{r}$ in base $\pi = 2$ and assume the i first bits are known. Then the lemma tells us that the next bit of $\mathbf{r}$, $\mathbf{r}[i] = (r_1[i], r_2[i], \ldots)$, must satisfy

$$f(\mathbf{r})[i] + \sum_j f_{x_j}(\mathbf{r}) r_j[i] \equiv 0 \pmod{2} \ . \tag{3.6.13}$$

In our case, the constraint polynomials generated in Section 3.6.1, equations (3.6.2)–(3.6.5) form four simultaneous equations in five variables. Given a partial solution (p', q', d', d'_p, d'_q) up to slice i of the bits, we apply the condition in equation (3.6.13) above to each polynomial and reduce modulo 2 to obtain the following conditions on

bit i:

$$\begin{aligned}
p\,[i] + q\,[i] &\equiv (n - p'q')\,[i] && \pmod 2 \\
d\left[i + \tau(k)\right] + p\,[i] + q\,[i] &\equiv \left(k(N+1) + 1 - k(p'+q') - ed'\right)\left[i + \tau(k)\right] && \pmod 2 \\
d_p\left[i + \tau(k_p)\right] + p\,[i] &\equiv \left(k_p(p'-1) + 1 - ed'_p\right)\left[i + \tau(k_p)\right] && \pmod 2 \\
d_q\left[i + \tau(k_q)\right] + q\,[i] &\equiv \left(k_q(q'-1) + 1 - ed'_q\right)\left[i + \tau(k_q)\right] && \pmod 2
\end{aligned} \tag{3.6.14}$$

These are precisely (3.6.12).

3.6.4 Local branching behavior

Without additional knowledge of the keys, the system of equations in (3.6.14) is underconstrained, and each partial satisfying assignment can be lifted to two partial satisfying assignments for slice i. If bit $i-1$ of a variable x is known, the corresponding $x\,[i-1]$ is fixed to the value of this bit, and the new partial satisfying assignments correspond to solutions of (3.6.14) with these bit values fixed. There can be zero, one, or two new solutions at bit i generated from a single solution at bit $i-1$, depending on the known values.

Now that we have a framework for characterizing the partial solutions generated at step i from a partial solution generated at step $i-1$, we will assume that a random fraction δ of the bits of the key values are known, and estimate the expectation and variance of the number of these solutions that will be generated.

In order to understand the number of solutions to the equation, we would like to understand the behavior of the c_i when the partial solution may not be equal to the real solution. Let $\Delta x = x - x'$, then substituting $x' = x - \Delta x$ into (3.6.14) we see

that any solution to (3.6.12) corresponds to a solution to

$$\begin{aligned}
\Delta p\,[i] + \Delta q\,[i] &\equiv (q\Delta p + p\Delta q + \Delta p\Delta q)\,[i] && \pmod 2 \\
\Delta d\left[i + \tau(k)\right] + \Delta p\,[i] + \Delta q\,[i] &\equiv (e\Delta d + k\Delta p + k\Delta q))\left[i + \tau(k)\right] && \pmod 2 \\
\Delta d_p\left[i + \tau(k_p)\right] + \Delta p\,[i] &\equiv (e\Delta d_p - k_p\Delta p)\left[i + \tau(k_p)\right] && \pmod 2 \\
\Delta d_q\left[i + \tau(k_q)\right] + \Delta q\,[i] &\equiv (e\Delta d_q - k_q\Delta q)\left[i + \tau(k_q)\right] && \pmod 2
\end{aligned}$$

and $\Delta x\,[i]$ is restricted to 0 if bit i of x is fixed.

Incorrect solutions generated from a correct solution. When the partial satisfying assignment is correct, all of the Δx will be equal to 0. If all of the $\Delta x\,[i]$ are unconstrained or if only $\Delta d\,[i + \tau(k)]$ is set to 0, there will be two possible solutions (of which we know one is "good" and the other is "bad"), otherwise there will be a single good solution. Let Z_g be a random variable denoting the number of bad solutions at bit $i + 1$ generated from a single good solution at bit i. Since each $\Delta x\,[i]$ is set to 0 independently with probability δ, the expected number of bad solutions generated from a good solution is equal to

$$\mathrm{E}\, Z_g = \delta(1-\delta)^4 + (1-\delta)^5$$

and

$$\mathrm{E}\, Z_g^2 = \mathrm{E}\, Z_g \ .$$

Both these expressions are dependent only on δ.

Incorrect solutions generated from an incorrect solution. When the partial satisfying assignment is incorrect, at least one of the Δx is nonzero. The expected

number of new incorrect satisfying assignments generated from an incorrect satisfying assignment is dependent both on δ and on the behavior of the b_j.

We conjecture the following is close to being true:

Conjecture 3.6.3. *For random p and q and for Δx not all zero and satisfying*

$$\begin{aligned} q\Delta p + p\Delta q - \Delta p\Delta q &= 0 \pmod{2^i} \\ e\Delta d + k\Delta p + k\Delta q &= 0 \pmod{2^{i+\tau(k)}} \\ e\Delta d_p - k_p\Delta p &= 0 \pmod{2^{i+\tau(k_p)}} \\ e\Delta d_q - k_q\Delta q &= 0 \pmod{2^{i+\tau(k_q)}} \ , \end{aligned}$$

the value of the right-hand side of each equation in 3.6.14 at bit $i+1$ is satisfied independently with probability near $1/2$.

We tested this empirically; each value of the vector (b_1, b_2, b_3, b_4) occurs with probability approximately $1/16$. (The error is approximately 5% for $\delta = 0.25$ and $n = 1024$, and approximately 2% for $\delta = 0.25$ and $n = 4096$.)

Let W_b be a random variable denoting the number of bad solutions at bit $i+1$ generated from a single bad solution at bit i. Assuming Conjecture 3.6.3,

$$\mathrm{E}\, W_b = \frac{(2-\delta)^5}{16}$$

and

$$\mathrm{E}\, W_b^2 = \mathrm{E}\, W_b + \delta(1-\delta)^4 + 2(1-\delta)^5 \ .$$

Note that the expectation is over the randomness of p and q and the positions of the unknown bits of the key.

When partial knowledge of some of the values (p, q, d, d_p, d_q) is totally unavailable, we can obtain a similar expression.

3.6.5 Global branching behavior at each step of the program

Now that we have characterized the effect that the constraints have on the branching behavior of the program, we can abstract away all details of RSA entirely and examine the general branching process of the algorithm. We are able to characterize the behavior of the algorithm, and show that if the expected number of branches from any partial solution to the program is less than one, then the total number of branches examined at any step of the program is expected to be constant. All of the following analysis assumes Conjecture 3.6.3.

Let X_i be a random variable denoting the number of bad assignments at step i, and recall that Z_g and W_b are random variables denoting the number of bad solutions at bit $i+1$ generated from a single good or bad solution at bit i.

Theorem 3.6.4.

$$\mathrm{E}\, X_i = \frac{\mathrm{E}\, Z_g}{1 - \mathrm{E}\, W_b}(1 - (\mathrm{E}\, W_b)^i)$$

This expression can be calculated in a number of ways; we demonstrate how to do so using generating functions in the following section.

Computing the expectation and variance

In this section, we derive expressions for the expectation and variance of the number of incorrect keys generated at each step of the program. Let X_i be a random variable denoting the number of bad assignments at step i. We will calculate the expectation $\mathrm{E}\, X_i$ and variance $\mathrm{Var}\, X_i$. (We know that the number of good assignments is always equal to one.)

To calculate these values, we will use probability generating functions. For more information on this approach, see e.g., [57, Ch. 8]. A probability generating function $F(s) = \sum \Pr[X = k] s^k$ represents the distribution of the discrete random variable X.

$F(s)$ satisfies the following identities:

$$F(1) = 1$$

$$\mathrm{E}\, X = F'(1)$$

$$\mathrm{Var}\, X = F''(1) + F'(1) - F'(1)^2 \ .$$

Let $G_i(s)$ be the probability generating function for the X_i, $z(s)$ the probability generating function for the Z_g (the number of bad assignments generated from a correct assignment) and $w(s)$ the probability generating function for the W_b (the number of bad assignments generated from a bad assignment).

From Section 3.6.3, we know that

$$z'(1) = \mathrm{E}\, Z_g$$

$$z''(1) = \mathrm{E}\, Z_g^2 - \mathrm{E}\, Z_g$$

$$w'(1) = \mathrm{E}\, W_b$$

and

$$w''(1) = \mathrm{E}\, W_b^2 - \mathrm{E}\, W_b$$

Expectation of X_i. We will calculate $\mathrm{E}\, X_i = G_i'(1)$.

$G_i(s)$ satisfies the recurrence

$$G_{i+1}(s) = G_i(w(s))z(s) \ , \tag{3.6.15}$$

that is, that the number of bad solutions at each step is equal to the number of bad solutions lifted from bad solutions plus the number of bad solutions produced from good solutions. (Recall that a generating function for the sum of two independent

random variables is given by the convolution of their generating functions.) We also have that

$$G_0(s) = 1 \ ,$$

because initially there are no bad solutions.

Differentiating (3.6.15) gives

$$G_i'(s) = (G_{i-1}(w(s))w'(s)z(s) + G_{i-1}(w(s))z'(s) \ . \tag{3.6.16}$$

Set $s = 1$ and use the fact that $G_i(1) = w(1) = z(1) = 1$ to obtain

$$G_i'(1) = w'(1)G_{i-1}'(1) + z'(1) \ .$$

Solving the recurrence yields

$$G_i'(1) = \frac{z'(1)}{1 - w'(1)}(1 - (w'(1))^i) \ . \tag{3.6.17}$$

If $w'(1) < 1$, then $w'(1)^i$ tends to 0 as i increases and

$$EX_i = G_i'(1) < \frac{z'(1)}{1 - w'(1)} \tag{3.6.18}$$

for all i. The expected number of bad solutions at any step of the process will be bounded by a value dependent only on δ and not on i.

Variance of X_i. To compute the variance $\operatorname{Var} X_i = G_i''(1) + G_i'(1) - (G_i'(1))^2$, we differentiate (3.6.16) again to obtain

$$\begin{aligned} G_i''(s) = {} & G_{i-1}''(w(s))w'(s)w'(s)z(s) + G_{i-1}'(w(s))w''(s)z(s) \\ & + 2G_{i-1}'(w(s))w'(s)z'(s) + G_{i-1}(w(s))z''(s) \ . \end{aligned} \tag{3.6.19}$$

Evaluating at $s = 1$ gives

$$G_i''(1) = G_{i-1}''(1)w'(1)^2 + G_{i-1}'(1)w''(1) + 2G_{i-1}'(1)w'(1)z'(1) + z''(1) \ .$$

Substitute in (3.6.17) to get

$$\begin{aligned} G_i''(1) = G_{i-1}''(1)w'(1)^2 &+ \frac{z'(1)}{1-w'(1)}(1-(w'(1))^i)w''(1) \\ &+ 2\frac{z'(1)}{1-w'(1)}(1-(w'(1))^i)w'(1)z'(1) + z''(1) \ . \end{aligned} \tag{3.6.20}$$

The general solution to this recurrence is

$$G_i''(1) = c_1 + c_2 w'(1)^i + c_3 w'(1)^{2i} \tag{3.6.21}$$

with

$$\begin{aligned} c_1 &= \frac{1}{1-w'(1)^2}\left(\frac{z'(1)}{1-w'(1)}(w''(1)+2w'(1)z'(1)) + z''(1)\right) \\ c_2 &= -\frac{1}{1-w'(1)}(w''(1)+2w'(1)z'(1)) \\ c_3 &= -c_1 - c_2 \ . \end{aligned}$$

Finishing the analysis

When $\mathrm{E}\, W_b < 1$, we can bound $\mathrm{E}\, X_i$ from above.

$$\mathrm{E}\, X_i \leq \frac{\mathrm{E}\, Z_g}{1-\mathrm{E}\, W_b}$$

In the previous section, we calculated expressions for $\mathrm{E}\, Z_g$ and $\mathrm{E}\, W_b$ dependent only on δ, thus when $\mathrm{E}\, W_b < 1$, $\mathrm{E}\, X_i$ can be bounded above by a constant dependent on δ and not on i.

We can evaluate this expression numerically using the values for the expected number of bad solutions discovered in the last section.

In the case with four equations and five unknowns (that is, we have partial knowledge of p, q, d, d_p, and d_q), $\mathrm{E}\, W_b < 1$ at $\delta > 2 - 2^{\frac{4}{5}}$. For $\delta = .2589$, $\mathrm{E}\, X_i < 93247$; for $\delta = .26$, $\mathrm{E}\, X_i < 95$; and for $\delta = .27$ $\mathrm{E}\, X_i < 9$.

In a similar fashion we can obtain the following complicated expression for the variance $\mathrm{Var}\, X_i = \mathrm{E}\, X^2 - (\mathrm{E}\, X)^2$.

Theorem 3.6.5.

$$\mathrm{Var}\, X_i = \alpha_1 + \alpha_2(\mathrm{E}\, W_b)^i + \alpha_3(\mathrm{E}\, W_b)^{2i} \tag{3.6.22}$$

with

$$\begin{aligned}
\alpha_1 &= \frac{\mathrm{E}\, Z_g \,\mathrm{Var}\, W_b + (1 - \mathrm{E}\, W_b)\,\mathrm{Var}\, Z_g}{(1 - (\mathrm{E}\, W_b)^2)(1 - \mathrm{E}\, W_b)} \\
\alpha_2 &= \frac{\mathrm{E}\, W_b^2 + \mathrm{E}\, W_b - 2\,\mathrm{E}\, W_b\,\mathrm{E}\, Z_g - \mathrm{E}\, Z_g}{1 - \mathrm{E}\, W_b} + 2\left(\frac{\mathrm{E}\, Z_g}{1 - \mathrm{E}\, W_b}\right)^2 \\
\alpha_3 &= -\alpha_1 - \alpha_2 \ .
\end{aligned}$$

Again evaluating numerically for five unknowns and four equations, at $\delta = .26$ $\mathrm{Var}\, X_i < 7937$, at $\delta = .27$ $\mathrm{Var}\, X_i < 80$, and at $\delta = .28$ $\mathrm{Var}\, X_i < 23$.

Bounding the total number of keys examined

Now that we have some information about the distribution of the number of partial keys examined at each step, we would like to understand the distribution of the total number of keys examined over an entire run of the program.

We know the expected total number of keys examined for an n-bit key is

$$\mathrm{E}\left[\sum_{i=0}^{n} X_i\right] \leq \frac{\mathrm{E}\, Z_g}{1 - \mathrm{E}\, W_b} n \ .$$

We will bound how far the total sum is likely to be from this expectation. First, we apply the following bound on the variance of a sum of random variables:

Lemma 3.6.6.

$$\operatorname{Var} \sum_{i=1}^{n} X_i \leq n^2 \max_i \operatorname{Var} X_i$$

The proof writes the variance of the sum in terms of covariance, and applies the Cauchy-Schwarz inequality and $\sqrt{ab} \leq \frac{a+b}{2}$.

Apply Chebyshev's inequality to bound the likelihood that $\sum X_i$ is too large:

$$\Pr(|\textstyle\sum_i X_i - \mathrm{E} \sum_i X_i| \geq n\alpha) \leq \frac{1}{(n\alpha)^2} \operatorname{Var} \textstyle\sum_i X_i \ .$$

Apply the above lemma to obtain

$$\Pr(|\textstyle\sum_i X_i - \mathrm{E} \sum_i X_i| \geq n\alpha) \leq \frac{1}{\alpha^2} \max_i \operatorname{Var} X_i \ .$$

When $\delta = .27$, setting $\alpha > 9n$ gives that, for an n-bit key, the algorithm will examine more than $9n^2 + 71n$ potential keys with probability less than $\frac{1}{n^2}$.

3.6.6 Missing key fields

The same results apply when we have partial knowledge of fewer key fields.

- If the algorithm has partial knowledge of d, p, and q but no information on d_p and d_q, we know that

$$\begin{aligned}
\mathrm{E}\, Z_g &= \delta(1-\delta)^2 + (1-\delta)^3 \\
\mathrm{E}\, Z_g^2 &= \mathrm{E}\, Z_g \\
\mathrm{E}\, W_b &= \frac{(2-\delta)^3}{4} \\
\mathrm{E}\, W_b^2 &= \mathrm{E}\, W_b + \delta(1-\delta)^2 + 2(1-\delta)^3 \ ,
\end{aligned}$$

	$n = 512$	768	1024	1536	2048	3072	4096	6144	8192
$\delta = 0.27$	0	0	0	0	0	0	0	0	1
0.26	0	0	0	0	1	5	3	4	8
0.25	0	0	3	6	8	10	17	35	37
0.24	4	5	7	27	50	93	121	201	274

Table 3.1: **Algorithm runs reaching panic width** We ran our algorithm 10,000 times at each value and recorded the number of failed runs in which width exceeded 1,000,000.

so $\mathrm{E}\, W_b < 1$ when $\delta > 2 - 2^{\frac{3}{4}} \approx .4126$. Then for $\delta = .42$ the probability that the algorithm examines more than $22n^2 + 24n$ keys is less than $\frac{1}{n^2}$.

- If the algorithm has partial knowledge of p and q but no information on the other values,

$$\begin{aligned} \mathrm{E}\, Z_g &= (1-\delta)^2 \\ \mathrm{E}\, Z_g^2 &= \mathrm{E}\, Z_g \\ \mathrm{E}\, W_b &= \frac{(2-\delta)^2}{2} \\ \mathrm{E}\, W_b^2 &= \mathrm{E}\, W_b + 2(1-\delta)^2 \ . \end{aligned}$$

Then $\mathrm{E}\, W_b < 1$ when $\delta > 2 - 2^{\frac{1}{2}} \approx .5859$. When $\delta = .59$ the probability that the algorithm examines more than $29n^2 + 29n$ keys is less than $\frac{1}{n^2}$.

3.6.7 Implementation and performance

We have developed an implementation of our key reconstruction algorithm in approximately 850 lines of C++, using NTL version 5.4.2 and GMP version 4.2.2. Our tests were run, in 64-bit mode, on an Intel Core 2 Duo processor at 2.4 GHz with 4 MB of L2 cache and 4 GB of DDR2 SDRAM at 667 MHz on an 800 MHz bus.

We ran experiments for key sizes between 512 bits and 8192 bits, and for δ values between 0.40 and 0.24. The public exponent is always set to 65537. In each experiment,

a key of the appropriate size is randomly censored so that exactly a δ fraction of the bits of the private key components considered together is available to be used for reconstruction. To reduce the time spent on key generation, we reused keys: We generated 100 keys for each key size. For every δ and keysize, we ran 100 experiments with each one of the pregenerated keys, for a total of 10,000 experimental runs. In all, we conducted over 1.1 million runs.

For each run, we recorded the *length* and *width*. The length is the total number of keys considered in the run of the algorithm, at all bit indices; the width is the maximum number of keys considered at any single bit index. These correspond essentially to $\sum_{i=1}^{n/2} X_i$ and $\max_i X_i$, in the notation of Section 3.6.3, but can be somewhat larger because we run the algorithm twice in parallel to account for both possible matchings of solutions of (3.6.11) to k_p and k_q. To avoid thrashing, we killed runs as soon as the width for some index i exceeded 1,000,000.

When the panic width was not exceeded, the algorithm always ran to completion and correctly recovered the factorization of the modulus.

Of the 900,000 runs of our algorithm with $\delta \geq 0.27$, only a single run ($n = 8192$, $\delta = 0.27$) exceeded the panic width. Applying a Chebyshev bound in this case (with $\mathrm{E}\, X_i = 9$ and $\mathrm{Var}\, X_i = 80$) suggests that a width of 1,000,000 should happen with extremely low probability.

Even below $\delta = 0.27$, our algorithm almost always finished within the allotted time. Table 3.1 shows the number of runs (out of 10,000) in which the panic width was exceeded for various parameter settings. Even for $n = 8192$ and $\delta = 0.24$, our algorithm recovered the factorization of the modulus in more than 97% of all runs. And in many of the overly long runs, the number of bits recovered before the panic width was exceeded suffices to allow recovering the rest using the lattice methods considered in Section 3.6.1; this is true of 144 of the 274 very long runs at $n = 8192$ and $\delta = 0.24$, for example.

[tb]

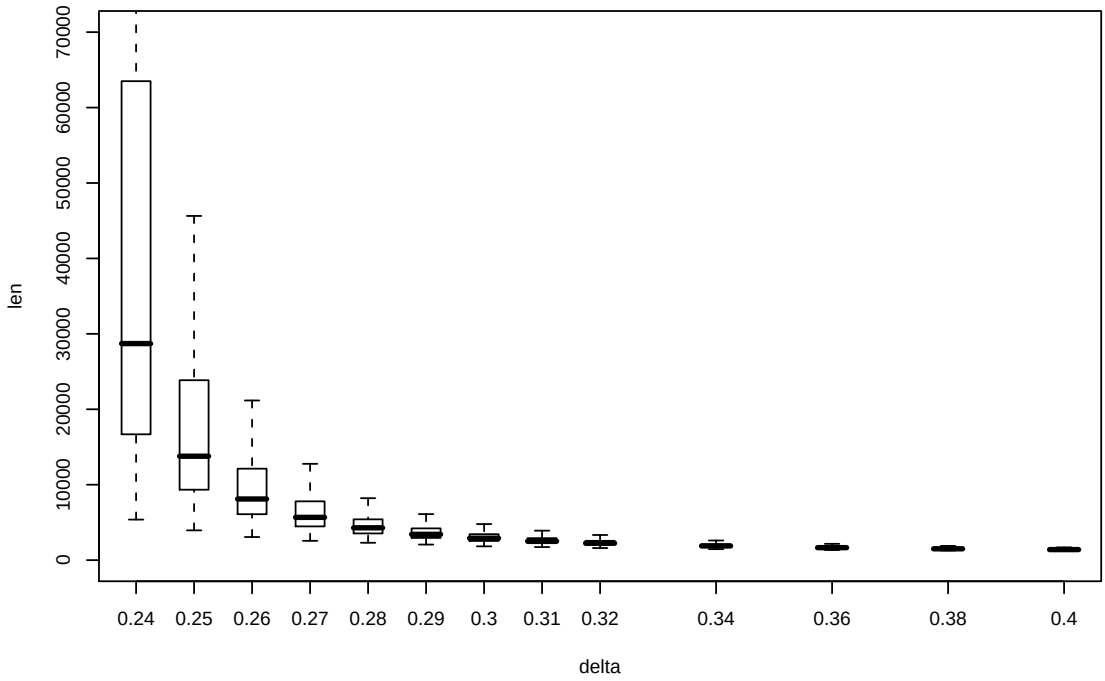


Figure 3.2: **Total keys examined** We generated an RSA key of length 2048 at random, erased a δ-fraction of the bits and random, then used our algorithm to recover the key. The box-and-whisker plot shows the minimum and maximum observed (horizontal lines), the 25th and 75th quartile (upper and lower limits of the box) and median observations for 10,000 runs at each value of δ. A sharp threshold is visible around $\delta = .26$.

As expected, search runtime was essentially linear in the total number of keys examined. For $n = 1024$, for example, examining a single key took approximately 5 μsec; for $n = 6144$, approximately 8 μsec. The setup time varied depending on whether k was closer to 0 or to e, but never exceeded 210 msec, even for $n = 8192$.

In Figure 3.3, we show the runtime behavior of the algorithm for the parameters $n = 2048$ and $\delta = 0.27$. The y axis shows the fraction of the 10,000 runs in which the total number of keys examined by the algorithm (i.e., the length of the run) exceeded the length given in the x axis. For example, 1442 runs had a length in excess of 10,000 and 237 had a length in excess of 25,000. Using a boxplot, we can examine the behavior of the algorithm for different values of δ. The plot in Figure 3.2 gives the behavior for $n = 2048$. The bar for $\delta = 0.27$ summarizes the data presented in Figure 3.3. (In our boxplot, generated using R's `boxplot` function, the central bar

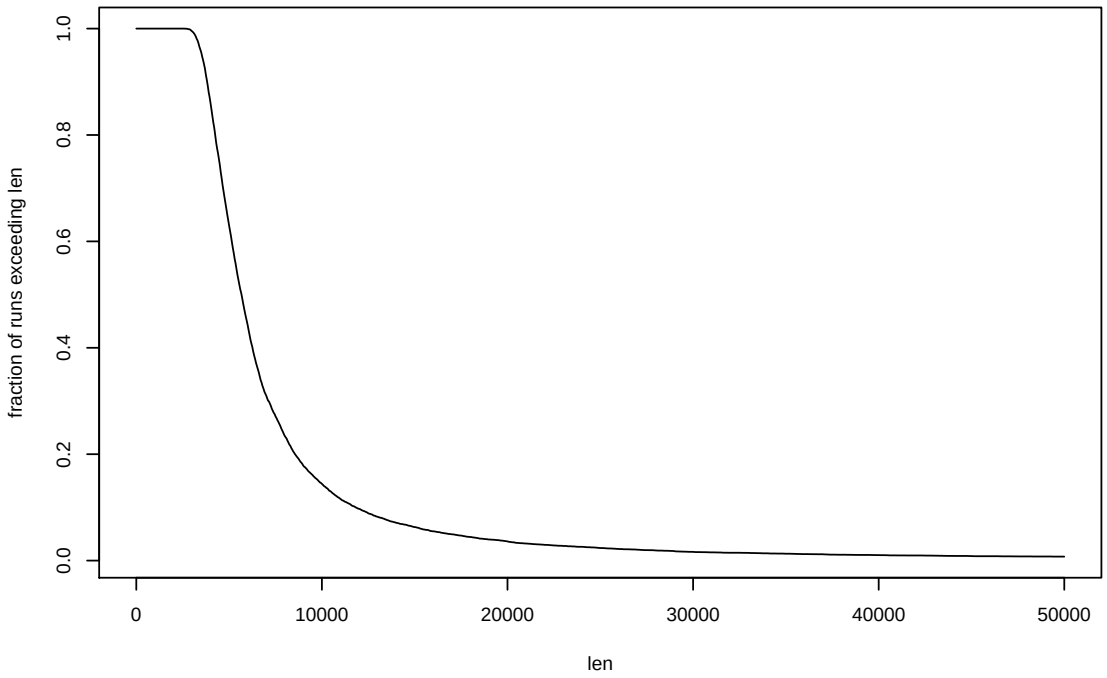


Figure 3.3: **Total number of keys examined** For 2048-bit RSA keys with 27% of bits known, this plot shows the fraction of our 10,000 runs in which the total number of keys examined over the entire run of the algorithm exceeded the value in the x-axis. There is a sharp threshold below which nearly all the runs are able to complete.

corresponds to the median, the hinges to the first and third quartiles, and the whisker extents depend on the interquartile range.)

Figure 3.4 and Figure 3.5 show the total number of keys examined by the algorithm as a function of n, the number of bits of the modulus, and holding δ constant at, respectively, 0.27 and 0.24. The length is largely linear in n for $\delta = 0.27$ but grows more quickly than linearly for $\delta = 0.24$.

3.7 Conclusions

The results in this chapter provide an elementary stepping stone to the more sophisticated methods we will discuss in the next chapter. In particular, they illustrate how behavior of the physical circuits in a DRAM chip introduces combinatorial constraints when analyzing key error-correction problems. In the case of RSA, these are combina-

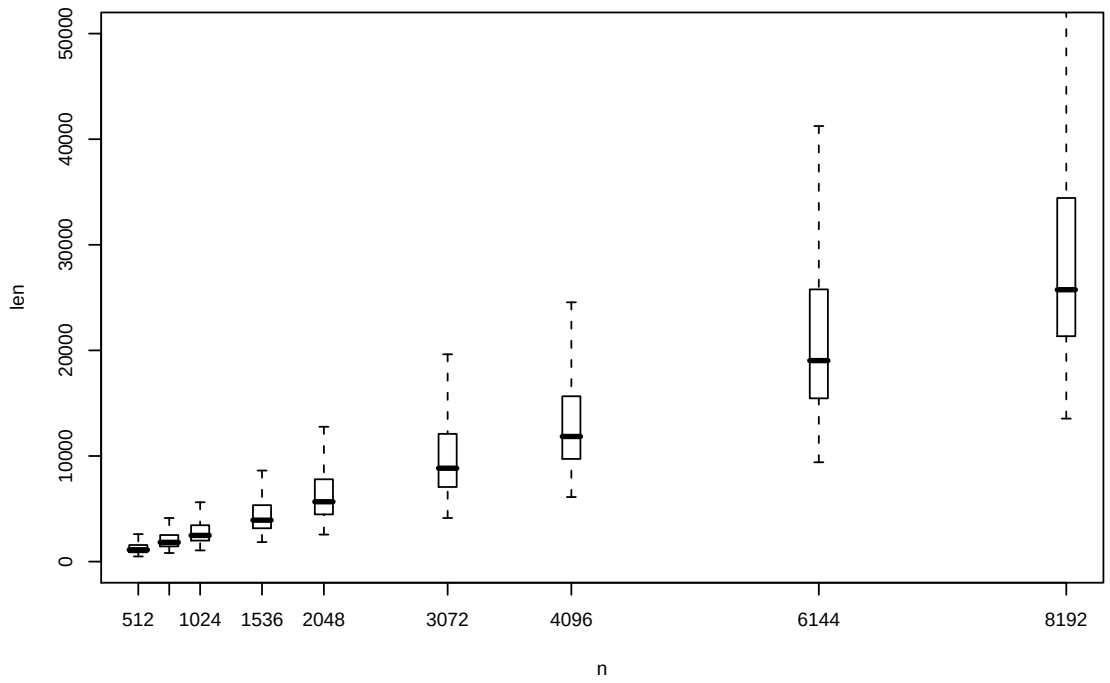


Figure 3.4: **Keys examined as n varies** This box and whisker plot shows the distribution of total number of keys examined over the run of an algorithm when $\delta = 27\%$ of key bits are known, as the key length varies. The median can be seen to grow approximately linearly with the key length, while the maximum value observed grows more quickly.

torial constraints on a number-theoretic problem. We use tools from p-adic analysis in order to think about the bit-wise information that we are given about solutions to a set of equations, but this is far from having a complete understanding of the bit-wise behavior of even simple multiplication.

Such an understanding would have far-reaching results. Here is an example related to me by Mikkel Thorup. Take, for example, the universal family of hash functions $x \mapsto ax + b \bmod p$. It is known that a random choice of constants results in a universal family of hash functions, that is, the probability of a collision is the same as if the hash values were randomly assigned. [24] However, applications sometimes demand stronger properties from hash functions; for example, that they behave like error-correcting codes. Miltersen [78] gives bounds for the probability that a random element of this family gives an error-correcting code with some minimum distance, but it is not known

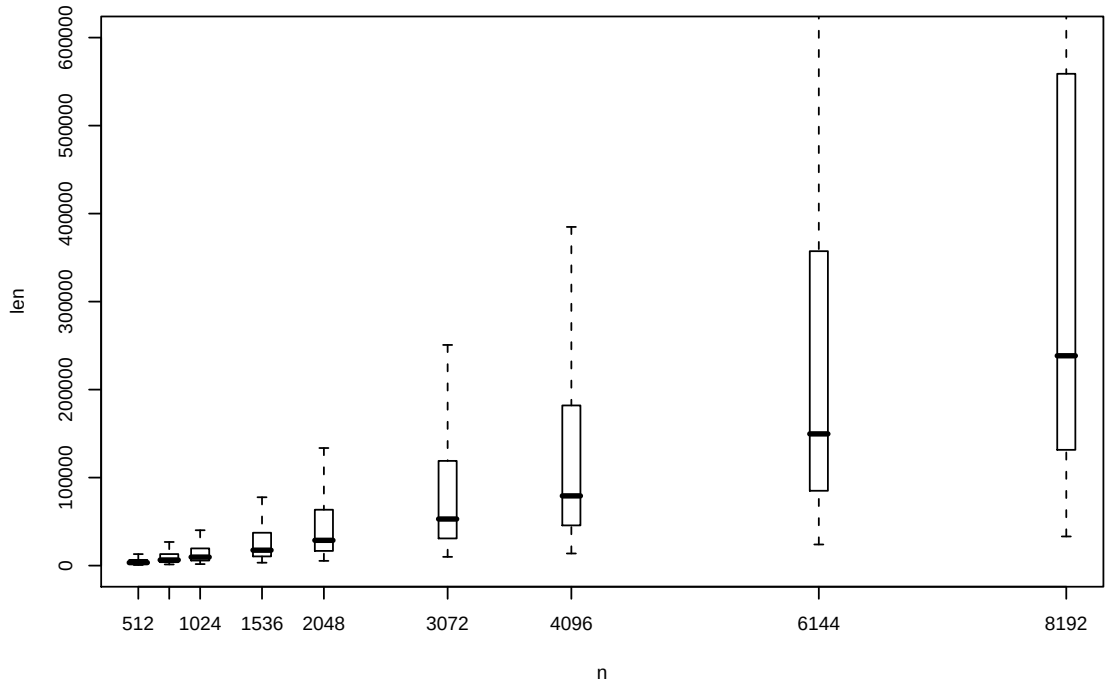


Figure 3.5: **Keys examined as n varies** This box and whisker plot shows the distribution of the total number of keys examined when $\delta = 24\%$. Compared to Figure 3.4, the range of the distribution grows much more quickly.

how to find a concrete set of parameters that works for all inputs outside of brute-force search.

The unusual perspective given by these kinds of key recovery problems can have broad applications outside of cryptography in addition to providing insight to the ciphers themselves.

Chapter 4

Extending lattice-based methods

> "Ideals are like stars; you will not succeed in touching them with your hands. But like the seafaring man on the desert of waters, you choose them as your guides, and following them you will reach your destiny."
>
> Carl Schurz

In the previous chapter, we presented an algorithm for recovering errors in RSA private keys whose bits have been subjected to random deletions.

Prior to our own work, the problem of RSA key recovery had been well studied under a different set of assumptions. In this model, the attacker has access to a large number of contiguous bits of the private key, for example the most significant bits of one of the factors of the modulus N. How many bits does the attacker need in order to efficiently recover a factor? This problem was posed by Rivest and Shamir in 1985 [89], and Coppersmith gave the best known bounds in 1996 [28].

Coppersmith's algorithm is a celebrated technique for finding small solutions to polynomial equations modulo integers, and it has several other important applications in cryptanalysis.

In this chapter, we will show how the ideas of Coppersmith's theorem can be extended to a more general framework encompassing the original number-theoretic problem, list decoding of Reed-Solomon and algebraic-geometric codes, and the problem of finding solutions to polynomial equations modulo ideals in rings of algebraic integers. These seemingly different problems are all perfectly analogous when viewed from the perspective of algebraic number theory.

Coppersmith's algorithm provides a key example of the power of lattice basis reduction. In order to extend the method beyond the integers, we illuminate the analogous structures for polynomial rings, number fields, and function fields. Ideals over number fields have a natural embedding into a lattice, and thus we can find a short vector simply by applying the LLL algorithm to this canonical embedding. In contrast to integer lattices, it turns out that lattice basis reduction is much easier over a lattice of polynomials, and in fact a shortest vector can always be found in polynomial time. Recasting the list decoding problem in this framework allows us to take advantage of very efficient reduction algorithms and thus achieve the fastest known list decoding algorithm for Reed-Solomon codes.

To extend this approach to function fields, we must overcome certain technical difficulties. In addition, we prove a much more general result about finding short vectors under arbitrary non-Archimedean norms, which may have further applications beyond list decoding of algebraic-geometric codes. As an illustration of the generality of our approach, we give the first list decoding algorithm that works for all algebraic-geometric codes, not just those defined using a single-point divisor.

4.0.1 Analogies in number theory

The connections we have described are not isolated phenomena. Many theorems in number theory and algebraic geometry have parallel versions for the integers and for polynomial rings, or more generally for number fields and function fields, and

translating statements or techniques between these settings can lead to valuable insights.

One particular advantage of this sort of arbitrage is that proving results for polynomial rings is usually easier. For example, the prime number theorem for $\mathbb{Z}$ is a deep theorem, but the analogue for the polynomial ring $\mathbb{F}_q[z]$ over a finite field is much simpler. It says that asymptotically a $1/n$ fraction of the q^n monic polynomials of degree n are irreducible, and in fact the error term is on the order of $q^{n/2}$ (see Lemma 14.38 in [109]). Proving a similarly strong version of the prime number theorem for $\mathbb{Z}$ would amount to proving the Riemann hypothesis. Similarly, the ABC conjecture for $\mathbb{Z}$ is a profound unsolved problem, while for polynomials rings it has an elementary proof [72].

Thus, polynomial rings are worlds in which many of the fondest dreams of mathematicians have come true. If a result cannot be proved in such a setting, then it is probably not even worth trying to prove it in $\mathbb{Z}$. If it can be proved for polynomial rings, then the techniques may not apply to the integers, but they often provide inspiration for how a proof might work if technical obstacles can be overcome.

Similarly, in computer science many computational problems that appear to be hard for integers are tractable for polynomials. For example, factoring polynomials can be done in polynomial time for many fields, while for the integers the problem seems to be hard. The polynomial analogue of the shortest vector problem for lattices can be solved exactly in polynomial time [108], while for integer lattices the problem is NP-hard [2]. This difference in the difficulty of lattice problems is at the root of the poor running time in Theorem 4.1.3 below for number fields of high degree.

4.1 Overview

We set up our framework with a brief review of Coppersmith's theorem, and then state our theorems on polynomial rings, number fields, and function fields. Each of these results will be discussed in more detail and proved in a later section.

4.1.1 Coppersmith's theorem

The following extension of Coppersmith's theorem [28] was developed by Howgrave-Graham [52] and May [74].

Theorem 4.1.1 ([28, 52, 74])**.** *Let $f(x)$ be a monic polynomial of degree d with coefficients modulo an integer $N > 1$, and suppose $0 < \beta \leq 1$. In time polynomial in $\log N$ and d, one can find all integers w such that*

$$|w| \leq N^{\beta^2/d}$$

and

$$\gcd(f(w), N) \geq N^{\beta}.$$

Note that when $\beta = 1$, this amounts to finding all sufficiently small solutions of $f(w) \equiv 0 \pmod{N}$, and the general theorem amounts to solving $f(w) \equiv 0 \pmod{B}$, where B is a large factor of N.

Here is a more detailed version of the example given above [28, 52]. Imagine that an adversary has obtained through a side-channel attack some knowledge about one of the prime factors p of an RSA modulus $N = pq$, for example some of its most significant bits. We denote this known quantity by r. Then we may write $p = r + w$, where the bound on w depends on how many bits of p are known. Suppose more than half of the bits have leaked, i.e., $0 \leq w \leq N^{1/4-o(1)}$ (we assume, as is typical, that p

and q are both $N^{1/2+o(1)}$). Now let $f(x) = x + r$ and $\beta = 1/2 + o(1)$. Theorem 4.1.1 tells us that we can in polynomial time learn w, and hence p, thereby factoring N.

Further applications of this theorem in cryptography include other partial key recovery attacks against RSA [17, 13], attacks on stereotyped messages and improper padding [28], and the proof of security for the RSA-OAEP+ padding scheme [99]. See [75] for many other applications.

It is remarkable that Theorem 4.1.1 allows us to solve polynomial equations modulo N without knowing the factorization of N, and this fact is critical for the cryptanalytic applications. However, even if one already has the factorization, Theorem 4.1.1 remains nontrivial if N has many prime factors.

To solve an equation modulo a composite number, one generally solves the equation modulo each prime power factor of the modulus and uses the Chinese remainder theorem to construct solutions for the original modulus. (Recall that modulo a prime, such equations can be solved in polynomial time, and we can use Hensel's lemma to lift the solutions to prime power moduli.) The number of possible solutions can be exponential in the number of prime factors, in which case it is infeasible to enumerate all of the roots and then select those that are within the desired range. In fact, the problem of determining whether there is a root in an arbitrary given interval is NP-complete [71]. Of course, if N has only two prime factors, then there can be only d^2 solutions modulo N, but our methods are incapable of distinguishing between numbers with two or many prime factors.

It is not even obvious that the number of roots modulo N of size at most $N^{1/d}$ is polynomially bounded. From this perspective, the exponent $1/d$ is optimal without further assumptions, because $f(x) = x^d$ will have exponentially many roots modulo $N = k^d$ of absolute value at most $N^{1/d+\varepsilon}$ (specifically, the $2N^{\varepsilon}$ such multiples of k). Theorem 4.1.1 can be seen as a constructive bound on the number of solutions. See [29] for further discussion of this argument and [59] for non-constructive bounds.

4.1.2 A polynomial analogue

To introduce our analogies, we will begin with the simplest and most familiar case: polynomials.

There is an important analogy in number theory between the ring $\mathbb{Z}$ of integers and the ring $F[z]$ of univariate polynomials over a field F. To formulate the analogue of Coppersmith's theorem, one just needs to recognize that the degree of a polynomial is the appropriate measure of its size. Thus, the polynomial version of Coppersmith's theorem should involve finding low-degree solutions of polynomial equations over $F[z]$ modulo a polynomial $p(z)$. That is, given a polynomial $f(x) = \sum_{i=0}^{d} f_i(z)x^i$ with coefficients $f_i(z) \in F[z]$, we seek low-degree polynomials $w(z) \in F[z]$ such that $f(w(z)) \equiv 0 \pmod{p(z)}$.

In the following theorem, we assume that we can efficiently represent and manipulate elements of F, and that we can find roots in $F[z]$ of polynomials over $F[z]$. For example, that holds if we can factor bivariate polynomials over F in polynomial time. This assumption holds for many fields, including $\mathbb{Q}$ and even number fields [62] as well as all finite fields [110] (with a randomized algorithm in the latter case).

Theorem 4.1.2. *Let $f(x)$ be a monic polynomial in x of degree d over $F[z]$ with coefficients modulo $p(z)$, where $\deg_z p(z) = n > 0$. In polynomial time, for $0 < \beta \leq 1$, one can find all $w(z) \in F[z]$ such that*

$$\deg_z w(z) < \beta^2 n/d$$

and

$$\deg_z \gcd(f(w(z)), p(z)) \geq \beta n.$$

In the case when $p(z)$ factors completely into linear factors, this theorem is equivalent to the influential Guruswami-Sudan theorem on list decoding of Reed-Solomon codes [46]. See Section 4.4.2 for the details of the equivalence. The above

statement of Theorem 4.1.2 appears to be new, as does the extension to higher-degree irreducible factors.

It has long been recognized that the Coppersmith and Guruswami-Sudan theorems are in some way analogous, although we are unaware of any previous, comparably explicit statement of the analogy. Boneh used Coppersmith's theorem in work on Chinese remainder theorem codes inspired by the Guruswami-Sudan theorem [16], and in a brief aside in the middle of [11], Bernstein noted that the Guruswami-Sudan theorem is the polynomial analogue of a related theorem of Coppersmith, Howgrave-Graham, and Nagaraj [30]. See also [45] for a general ideal-theoretic setting for coding theory, and [103] for a survey of relationships between list decoding and number-theoretic codes.

4.1.3 Number fields

A number field is a finite extension of the field $\mathbb{Q}$ of rational numbers. Thus it is natural to investigate how a statement over the rationals, the simplest number field, extends to more general number fields. We extend our analogy by adapting Coppersmith's theorem to the number field case.

Every number field K is of the form

$$K = \mathbb{Q}(\alpha) = \{a_0 + a_1\alpha + \cdots + a_{n-1}\alpha^{n-1} : a_0, \ldots, a_{n-1} \in \mathbb{Q}\},$$

where α is an algebraic number of degree n (i.e., a root of an irreducible polynomial of degree n over $\mathbb{Q}$). The degree of K is defined to be n. Within K, there is a ring $\mathcal{O}_K$ called the ring of algebraic integers in K. It plays the same role within the field K as the ring $\mathbb{Z}$ of integers plays within $\mathbb{Q}$. Sometimes $\mathcal{O}_K$ is of the form $\mathbb{Z}[\alpha]$, but sometimes it is more subtle.

Recall that an *ideal* in a ring is a non-empty subset closed under addition and under multiplication by arbitrary elements of the ring. (Intuitively, it is a subset modulo which one can reduce elements of the ring.) For example, the multiples of any fixed element form an ideal, called a *principal ideal*. In $\mathbb{Z}$ every ideal is of that form, but that is not usually true in $\mathcal{O}_K$.

In $\mathcal{O}_K$, we study the solutions of polynomial equations modulo ideals, the analogue of such equations modulo integers in $\mathbb{Z}$. To measure the size of a nonzero ideal I in $\mathcal{O}_K$, we will use its norm $N(I) = |\mathcal{O}_K/I|$, i.e., the size of the quotient ring.

A final conceptual issue that makes this case more subtle is that a number field of degree n has n absolute values $|\cdot|_i$ corresponding to its n embeddings into $\mathbb{C}$ (as we will explain in Section 4.5), and to obtain the theorem it is necessary to bound them all simultaneously.

The number field analogue of Coppersmith's theorem is as follows:

Theorem 4.1.3. *Let K be a number field of degree n with ring of integers $\mathcal{O}_K$, $f(x) \in \mathcal{O}_K[x]$ a monic polynomial of degree d, and $I \subsetneq \mathcal{O}_K$ an ideal in $\mathcal{O}_K$. Assume that we are given $\mathcal{O}_K$ and I explicitly by integral bases. For $0 < \beta \leq 1$ and $\lambda_1, \ldots, \lambda_n > 0$, in time polynomial in the input length and exponential in n^2 we can find all $w \in \mathcal{O}_K$ with $|w|_i < \lambda_i$ such that*

$$N(\gcd(f(w)\mathcal{O}_K, I)) > N(I)^\beta,$$

provided that

$$\prod_i \lambda_i < N(I)^{\beta^2/d}.$$

Furthermore, in polynomial time we can find all such w provided that

$$\prod_i \lambda_i < (2 + o(1))^{-n^2/2} N(I)^{\beta^2/d}.$$

Equivalently, we can find small solutions of equations $f(x) \equiv 0 \pmod{J}$, where the ideal J is a large divisor of I. Using improved lattice basis reduction algorithms [3] we can achieve slightly subexponential behavior in n^2. Note also that $\gcd(f(w)\mathcal{O}_K, I)$ is the largest ideal that contains both the principal ideal $f(w)\mathcal{O}_K$ and I; in other words, it is their sum $f(w)\mathcal{O}_K + I$.

When n is fixed, our algorithm runs in polynomial time, but the dependence on n is exponential. That appears to be unavoidable using our techniques, but it is not a serious drawback. Many number-theoretic algorithms behave poorly for high-degree number fields, and most computations are therefore done in low-degree cases. Even for a fixed number field K, Theorem 4.1.3 remains of interest.

Several problems over number fields have been proposed as the basis for cryptosystems; see, for example, [20] for a survey of problems over quadratic number fields. More recently, Peikert and Rosen [84] and Lyubashevsky, Peikert, Regev [69] developed lattice-based cryptographic schemes using lattices representing the canonical embeddings of ideals in number fields. As a special case, Theorem 4.1.3 can be used to solve certain cases of the bounded-distance decoding problem for such lattices, and improving our approximation factor from $(2 + o(1))^{-n^2/2}$ to $2^{-n}\sqrt{|\Delta_K|}$, where Δ_K is the discriminant of K, would solve the problem in general; see Section 4.5.3 for more details.

In addition, number fields have many applications to purely classical problems, the most prominent example being the number field sieve factoring algorithm. All sieve algorithms require generating smooth numbers, and in this context Boneh [16] showed how to use Coppersmith's theorem to find smooth integer solutions of polynomials in short intervals. Using Theorem 4.1.3 analogously, one can do the same over number fields. Nicholas Coxon has independently proven some results along these lines that are to appear in his PhD thesis.

We prove Theorem 4.1.3 in Section 4.5.

4.1.4 Function fields

Algebraic number theorists have developed a more sophisticated version of the analogy between the integers and polynomial rings. A global field is a finite extension of either the field $\mathbb{Q}$ of rational numbers (called a number field, as we have seen) or the field of rational functions on an algebraic curve over a finite field, called function fields (of curves, as opposed to higher-dimensional varieties). The parallels between number fields and function fields are truly astonishing, and this analogy has played a crucial role in the development of number theory over the last century.

We now complete the analogy by extending Coppersmith's theorem to the function field case. See Section 4.6 for a more thorough review of the setting and notation.

Theorem 4.1.4. *Let $\mathcal{X}$ be a smooth, projective, absolutely irreducible algebraic curve over $\mathbb{F}_q$, and let K be its function field over $\mathbb{F}_q$. Let D be a divisor on $\mathcal{X}$ whose support* $\mathrm{supp}(D)$ *is contained in the $\mathbb{F}_q$-rational points $\mathcal{X}(\mathbb{F}_q)$, let S be a subset of $\mathcal{X}(\mathbb{F}_q)$ that properly contains* $\mathrm{supp}(D)$*, let $\mathcal{O}_S$ denote the subring of K consisting of functions with poles only in S, and let $\mathcal{L}(D)$ be the Riemann-Roch space*

$$\mathcal{L}(D) = \{0\} \cup \{f \in K^* : (f) + D \succeq 0\}.$$

Let $f(x) \in \mathcal{O}_S[x]$ be a monic polynomial of degree d, and let I be a proper ideal in $\mathcal{O}_S$.

Then in probabilistic polynomial time, we can find all $w \in \mathcal{L}(D)$ such that

$$N(\gcd(f(w)\mathcal{O}_S, I)) \geq N(I)^{\beta},$$

provided that

$$q^{\deg(D)} < N(I)^{\beta^2/d}.$$

In the case when S contains only a single point, the function field version of Coppersmith's theorem is equivalent to the Guruswami-Sudan theorem on list-decoding

of algebraic-geometric codes, as we will outline in Section 4.6. The Guruswami-Sudan theorem and the earlier Shokrollahi-Wasserman theorem in [98] are specialized to that case, which covers many but not all algebraic-geometric codes. Our theorem extends list decoding to the full range of such codes.

We assume that we can efficiently compute bases of Riemann-Roch spaces for divisors in $\mathcal{X}$. That can be done in many important cases (for example, for a smooth plane curve, or even one with ordinary multiple points [53]), and it is a reasonable assumption because even the encoding problem for algebraic-geometric codes requires a basis of a Riemann-Roch space. Note also that although our algorithm is probabilistic, it is guaranteed to give the correct solution in expected polynomial time; in other words, it is a "Las Vegas" algorithm.

We prove Theorem 4.1.4 in Section 4.6.

4.2 Preliminaries

One of the main steps in Coppersmith's theorem uses lattice basis reduction to find a short vector in a lattice. In this section, we will review preliminaries on integral lattices, and introduce the analogues that we will use in our generalizations.

4.2.1 Integer lattices

Recall that a *lattice* in $\mathbb{R}^m$ is a discrete subgroup of rank m. Equivalently, it is the set of integer linear combinations of a basis of $\mathbb{R}^m$.

The *determinant* $\det(L)$ of a lattice L is the absolute value of the determinant of any basis matrix; it is not difficult to show that it is independent of the choice of basis. One way to see why is that the determinant is the volume of the quotient $\mathbb{R}^m/L$, or equivalently the volume of a fundamental parallelotope.

One of the fundamental problems in lattice theory is finding short vectors in lattices, with respect to the ℓ_p norm

$$|v|_p = \left(\sum_{i=1}^{m} |v_i|^p \right)^{1/p}.$$

Most often we use the ℓ_2 norm, which is of course the usual Euclidean distance. The LLL lattice basis reduction algorithm [63] can be used to find a short vector in a lattice.

Theorem 4.2.1 ([63]). *Given a basis of a lattice L in $\mathbb{Q}^m$, a nonzero vector $v \in L$ satisfying*

$$|v|_2 \leq 2^{(m-1)/4} \det(L)^{1/m}$$

can be found in polynomial time.

Note that the LLL algorithm's input is a rational lattice, and the rationality plays an important role in the running time analysis. In the proof of Theorem 4.1.3, we must apply it to a lattice whose basis vectors are not in $\mathbb{Q}^m$; however, for our purposes using a close rational approximation suffices.

4.2.2 Polynomial lattices

A lattice is a *module* over the ring $\mathbb{Z}$ of integers. In other words, not only is it an abelian group under addition, but we can also multiply lattice vectors by integers and thus take arbitrary integer combinations of them. More generally, a module for a ring R is an abelian group in which we can multiply by elements of R (in a way that satisfies the associative and distributive laws). In other words, an R-module is exactly like an R-vector space, except that R is not required to be a field, as it is in the definition of a vector space.

The module R^m with componentwise scalar multiplication is called a *free* R-module of rank m. Every lattice is a free $\mathbb{Z}$-module, and free R-modules will be the analogous structure for the ring R.

For example, if R is the polynomial ring $F[z]$ over a field F, then we define a *polynomial lattice* to be a free module over $F[z]$ of finite rank. A polynomial lattice will usually be generated by a basis of vectors whose coefficients are polynomials in z. Vectors in our polynomial lattice will be linear combinations of the basis vectors (where the coefficients are also polynomials in z).

As we will see later, an appropriate definition of the length (i.e., degree) of such a lattice vector is the maximum degree of its coordinates:

$$\deg_z(v_1(z), v_2(z), \ldots, v_m(z)) = \max_i \deg_z v_i(z). \tag{4.2.1}$$

This defines a non-Archimedean norm. In fact, for lattices with a norm defined as above, it is possible to find the exact shortest vector in polynomial time (see, for example, [108]).

Lattices of polynomials have been well studied because of their applications to the study of linear systems [56]. There are several notions of basis reduction for such lattices. A basis is *column-reduced* (or, as appropriate, *row-reduced*) if the degree of the determinant of the lattice (i.e., of a basis matrix) is equal to the sum of the degrees of its basis vectors. Such bases always contain a minimal vector for the lattice, and m-dimensional column reduction can be carried out in $m^{\omega+o(1)}D$ field operations [43], where ω is the exponent of matrix multiplication and D is the greatest degree occurring in the original basis of the lattice.

In particular, for an m-dimensional lattice L with the norm (4.2.1), the above algorithms are guaranteed to find a nonzero vector v for which

$$\deg v \leq \frac{1}{m} \deg \det L, \tag{4.2.2}$$

where $\det L$ denotes the determinant of a lattice basis.

4.2.3 Finding short vectors under general non-Archimedean norms

The above algorithms are specialized to norms defined by (4.2.1), but there are other non-Archimedean norms, and we will need to use them in the proof of Theorem 4.1.4 in the function field setting. In fact, for all non-Archimedean norms, one can find a vector satisfying the equivalent of (4.2.2) in a lattice by solving a system of linear equations. Solving this system may be less efficient than a specialized algorithm, but it allows us to give a general approach that will work in polynomial time for any norm.

Let $R = F[z]$ be a polynomial ring over a field F, and for $r \in R$ define

$$|r| = C^{\deg_z(r)}$$

for some arbitrary constant $C > 1$; we take $|0| = 0$ as a special case. Note that $|z| = c$, and thus we can write $|r| = |z|^{\deg_z(r)}$.

Suppose we have any norm $|\cdot|$ on R^m that satisfies the following three properties:

1. For all $v \in R^m$, $|v| \geq 0$, and $|v| = 0$ if and only if $v = 0$.
2. For all $v, w \in R^m$, $|v + w| \leq \max(|v|, |w|)$.
3. For all $v \in R^m$ and $r \in R$, $|rv| = |r||v|$.

Note that taking

$$|(v_1(z), v_2(z), \ldots, v_m(z)| = C^{\max_i \deg_z v_i(z)}$$

defines such a norm, but the extra generality will prove useful in Section 4.6.

Let $M \subseteq R^m$ be a submodule of rank m (so the quotient F-vector space R^m/M is finite-dimensional), and let $d = \dim_F(R^m/M)$.

Lemma 4.2.2. *For any R-basis $b_1, \ldots, b_m$ of R^m, there exists a nonzero vector $v \in M$ such that*

$$|v| \leq \sqrt[m]{|b_1| \ldots |b_m|}\, |z|^{d/m}.$$

Proof. We will construct a nonzero vector satisfying $|v| \leq q^c$ for some constant c to be determined, and then we will optimize the choice of c. Let $|b_i| = |z|^{n_i}$, and consider the space of polynomials

$$V = \left\{ \sum_i r_i b_i : r_i \in R \text{ and } \deg_z r_i \leq c - n_i \right\}.$$

Every $v \in V$ satisfies $|v| \leq |z|^c$, and V is an F-vector space. To compute its dimension, note that r_i is determined by $\lfloor c - n_i \rfloor + 1 > c - n_i$ coefficients. Because $b_1, \ldots, b_m$ is an R-basis, $\dim_F V > mc - \sum_i n_i$.

If we take $c = \left(d + \sum_i n_i\right)/m$, then $\dim_F V > d$. Thus, there exists a nonzero element v of V that maps to zero in the d-dimensional quotient space R^m/M and hence lies in M. It satisfies

$$|v| \leq q^c = \sqrt[m]{|b_1| \ldots |b_m|}\, |z|^{d/m},$$

as desired. □

Lemma 4.2.3. *Under the hypothesis of Lemma 4.2.2, a vector satisfying*

$$|v| \leq \sqrt[m]{|b_1| \ldots |b_m|}\, |z|^{d/m}$$

can be found in polynomial time (given an R-basis of M).

Proof. In the notation of the proof of Lemma 4.2.2, we will show that we can find small coefficients $r_1, \ldots, r_m \in R$ (not all zero) such that $\sum_i r_i b_i$ is in M. Suppose $w_1, \ldots, w_m$ is an R-basis of M. Then the elements of M are those that can be written

as $\sum s_i w_i$ with $s_i \in R$. Given a polynomial bound for the degrees of $s_1, \ldots, s_m$, we could determine the coefficients r_i and s_i by solving linear equations over F for their coefficients. To specify these equations, we write $w_1, \ldots, w_m$ as R-linear combinations of $b_1, \ldots, b_m$. Define the matrix W over R by $w_j = \sum_i W_{ij} b_i$ for each j. Then

$$\sum_i r_i b_i = \sum_j s_j w_j$$

amounts to $r = Ws$, where s and r are the column vectors with entries s_i and r_i, respectively.

Thus, s determines r in a simple way, and all we need is to choose $s_1, \ldots, s_m$ so that the relationship $r = Ws$ implies $\deg_z r_i \leq c - n_i$, with c and n_i defined as in the proof of Lemma 4.2.2. It is not difficult to bound the degrees of the polynomials s_i as follows. Let $\widetilde{W}$ be the adjoint matrix of W (so $W\widetilde{W} = \det(W)I$). Then

$$\widetilde{W} r = \det(W) s.$$

It follows that for each i,

$$\deg_z \det(W) + \deg_z s_i \leq \max_j \big(\deg_z \widetilde{W}_{ij} + \deg_z r_j \big).$$

However, the entries $\widetilde{W}_{ij}$ of $\widetilde{W}$ have degree bounded by $m-1$ times the maximum degree of an entry of W (because they are given by determinants of $(m-1) \times (m-1)$ submatrices of W). Thus, $\deg_z s_i$ is polynomially bounded, and we can locate a suitable vector v by solving a system of polynomially many linear equations over F. □

Note that for a rank m submodule M of R^m, the degree of the determinant of a basis matrix B for M is the dimension of the quotient R^m/M. Thus, in Lemma 4.2.2, if $|b_1| = \cdots = |b_m| = 1$, then the norm of a minimal vector is bounded by $|\det(B)|^{1/m}$.

The exponential approximation factor that occurs in LLL lattice basis reduction does not occur here.

4.3 Coppersmith's theorem

We now review how Coppersmith's method works over the integers, as this provides a template for the techniques we will apply later. We will follow the exposition of May [75].

Let $f(x)$ be a monic univariate polynomial of degree d, and N an integer of potentially unknown factorization. We wish to find all small integers w such that $\gcd(f(w), N)$ is large.

To do so, we will choose some positive integer k (to be determined later) and look at integer combinations of the polynomials $x^j f(x)^i N^{k-i}$. If B divides both N and $f(w)$, then B^k will divide $w^j f(w)^i N^{k-i}$ and thus also any linear combination of such polynomials.

Let

$$Q(x) = \sum_{i,j} a_{i,j} x^j f(x)^i N^{k-i} = \sum_i q_i x^i,$$

for some coefficients $a_{i,j}$ and q_i to be determined. We will choose Q so that the small solutions to our original congruence become actual solutions of $Q(x) = 0$ in the integers. This will allow us to find w by factoring $Q(x)$ over the rationals. The construction of Q tells us that

$$Q(w) \equiv 0 \pmod{B^k}. \tag{4.3.1}$$

If in addition we have a lower bound N^β on the size of B, and we can show that

$$|Q(w)| < N^{\beta k} \leq B^k, \tag{4.3.2}$$

then $Q(w) = 0$ and we may find w by factoring Q. In fact, this observation tells us that we can find *all* such w in this way. A similar observation will appear in all of our proofs.

In the case of the integers, we introduce the bound $|w| < X$ on our roots, and the triangle inequality tells us that

$$|Q(w)| \leq \sum_i |q_i| X^i. \tag{4.3.3}$$

To finish the theorem, we will show that if X is sufficiently small, then we can choose Q so that its coefficients q_i satisfy

$$\sum_i |q_i| X^i < N^{\beta k}. \tag{4.3.4}$$

We are now ready to prove Coppersmith's theorem for the integers.

Proof of Theorem 4.1.1. Having outlined the general technique above, it remains to be shown that we can construct a polynomial $Q(x)$ whose coefficients satisfy the bound in (4.3.4).

The polynomial $Q(x)$ will be a linear combination of the polynomials

$$x^j f(x)^i N^{k-i} \quad \text{for} \quad 0 \leq i < k \text{ and } 0 \leq j < d$$

and

$$x^j f(x)^k \quad \text{for} \quad 0 \leq j < t.$$

The right-hand side of (4.3.3) is the ℓ_1 norm of the vector of coefficients of the polynomial $Q(xX)$, which in turn will be a linear combination of the polynomials $(xX)^j f(xX)^i N^{k-i}$. Finding our desired $Q(x)$ is thus equivalent to finding a suitably

short vector in the lattice L spanned by the coefficient vectors of the polynomials $(xX)^j f(xX)^i N^{k-i}$.

To compute the determinant of this lattice, we can order the basis vectors by the degrees of the polynomials they represent to obtain an upper triangular matrix whose determinant is the product of the terms on the diagonal:

$$\det(L) = \prod_{0 \le i < dk+t} X^i \prod_{0 \le j \le k} N^{dj} = X^{(dk+t-1)(dk+t)/2} N^{dk(k+1)/2}.$$

Set $m = dk + t$. We can use the LLL algorithm [63] to find a vector v whose ℓ_2 norm is bounded by

$$|v|_2 \le 2^{(m-1)/4} \det(L)^{1/m}.$$

By Cauchy-Schwarz, $|v|_1 \le \sqrt{m}\,|v|_2$, and hence whenever $|w| < X$,

$$|Q(w)| \le \sqrt{m} 2^{(m-1)/4} \det(L)^{1/m}.$$

We assume $m \ge 7$, and use the weaker bound

$$|Q(w)| \le 2^{(m-1)/2} \det(L)^{1/m}.$$

To prove inequality (4.3.2), we must show that

$$|Q(w)| \le 2^{(m-1)/2} \left(X^{m(m-1)/2} N^{dk(k+1)/2}\right)^{1/m} < N^{\beta k}.$$

This inequality is equivalent to

$$(2X)^{(m-1)/(2k)} N^{d(k+1)/(2m)} < N^{\beta}. \tag{4.3.5}$$

Applying Lemma 4.3.1 below with $\ell = \log 2X$ and $n = \log N$, we obtain parameters k and t such that (4.3.5) holds for

$$2X < N^{\beta^2/d-\varepsilon}.$$

To eliminate ε from the statement of the theorem, take $\varepsilon < \frac{1}{\log_2 N}$. Then our bound becomes $X \leq \frac{1}{4}N^{\beta^2/d}$. We can divide the interval $[-N^{\beta^2/d}, N^{\beta^2/d}]$ into four intervals of width $2X$ and solve the problem for each interval by finding solutions for the polynomials $f(x-3X)$, $f(x-X)$, $f(x+X)$, and $f(x+3X)$. Thus, we achieve a bound of $X \leq N^{\beta^2/d}$, as desired. □

We end with a brief lemma that will tell us how to optimize our parameters in equation (4.3.5).

Lemma 4.3.1. *The inequality* $\ell\frac{m-1}{2k} + nd\frac{k+1}{2m} < n\beta$ *is satisfied for* $\ell < n\left(\frac{\beta^2}{d} - \varepsilon\right)$, *any* $m \geq \left\lceil \frac{2\beta}{\varepsilon} \right\rceil$, *and* $k = \left\lfloor \frac{\beta m}{d} - 1 \right\rfloor$.

As intuition, note that if we set the two terms $\ell\frac{m-1}{2k}$ and $nd\frac{k+1}{2m}$ roughly equal to $\frac{n\beta}{2}$, then we have $\ell m^2 \approx ndk^2 \approx n\beta mk$ and hence $\ell \approx n\beta^2/d$. The proof amounts to making this precise.

Proof. It suffices to show that these values of m and k satisfy $n\left(\frac{\beta^2}{d} - \varepsilon\right)\frac{m-1}{2k} < \frac{n\beta}{2}$ and $nd\frac{k+1}{2m} \leq \frac{n\beta}{2}$.

The first inequality is equivalent to $\frac{k}{m-1} > \frac{\beta}{d} - \frac{\varepsilon}{\beta}$. Similarly, the second is equivalent to $\frac{k+1}{m} \leq \frac{\beta}{d}$. If we set $k = \left\lfloor \frac{\beta m}{d} - 1 \right\rfloor$, then $\frac{k+1}{m} \leq \frac{\beta}{d}$, so the second inequality is satisfied. If in addition we take $m \geq \frac{2\beta}{\varepsilon}$, then $\frac{\varepsilon m}{\beta} \geq 2$ and hence $k > \frac{\beta m}{d} - 2 \geq \frac{\beta m}{d} - \frac{\varepsilon m}{\beta}$. It follows that $k\frac{m}{m-1} > \frac{\beta m}{d} - \frac{\varepsilon m}{\beta}$, which is equivalent to the first inequality. □

It is also worth noting that improving the approximation factor for the length of the short lattice vector that we find will only improve the constants and running time

of the theorem, but will not provide an asymptotic improvement to the bound $N^{\beta^2/d}$ on $|w|$.

4.4 Polynomials and Reed-Solomon list decoding

In this section, we prove Theorem 4.1.2 using an approach analogous to that of the previous section. Guruswami and Sudan's technique for list decoding of Reed-Solomon codes [46] is similar in that it involves constructing a bivariate polynomial that vanishes to high order at particular points. To construct such a polynomial, they write each vanishing condition as a set of linear equations on the coefficients of the polynomial under construction. The linear equations can be solved to obtain the desired polynomial, and the polynomial factored to obtain its roots.

Similarly, the polynomials used in Coppersmith's method are constructed in order to vanish to high order, the condition ensured by equation (4.3.1). The conceptual difference is that this condition follows from the form of the lattice basis, rather than being imposed as linear constraints. With the right definition of lattice basis reduction in the polynomial setting, we can emulate the proof from the integer case.

We regard $f(x)$ as a polynomial in x with coefficients that are polynomials in the variable z. To prove Theorem 4.1.2, we would like to construct a polynomial $Q(x)$ over $F[z]$ from the polynomials $x^j f(x)^i p(z)^{k-i}$. If $b(z)$ divides both $p(z)$ and $f(w(z))$, then $b(z)^k$ divides $w(z)^j f(w(z))^i p(z)^{k-i}$ and thus also any linear combination of such polynomials.

Instead of an integer combination of these polynomials, we will allow coefficients that are polynomials in z. Let

$$Q(x) = \sum_{i,j} a_{i,j}(z) x^j f(x)^i p(z)^{k-i} = \sum_i q_i(z) x^i.$$

If we have an upper bound ℓ on the degree of our root $w(z)$, then the degree of $Q(w(z))$ will be

$$\deg_z Q(w(z)) \leq \max_i \, (\deg_z q_i(z) + \ell i).$$

If similarly we have a lower bound $n\beta$ on the degree of $b(z)$, then if we know that both

$$Q(w(z)) \equiv 0 \pmod{b(z)^k}$$

and

$$\deg_z Q(w(z)) < n\beta k \leq k \deg_z b(z), \tag{4.4.1}$$

then we may conclude that

$$Q(w(z)) = 0.$$

4.4.1 Proof of Theorem 4.1.2

We will show how finding a short vector in a lattice of polynomials will allow us to construct a polynomial $Q(x)$ satisfying (4.4.1).

Let ℓ be the upper bound on the degree of the roots $w(z)$ we would like to find. Using the same idea to bound the length of the vector as in the integer case, we will form a lattice of the coefficient vectors of

$$(z^\ell x)^j f(z^\ell x)^i p(z)^{k-i} \quad \text{for} \quad 0 \leq j < d \text{ and } 0 \leq i < k$$

and

$$(z^\ell x)^j f(z^\ell x)^k \quad \text{for} \quad 0 \leq j < t.$$

As always, we view them as polynomials in powers of x with coefficients that are polynomials in z. Let M be the $F[z]$-module spanned by the coefficient vectors of these polynomials, with the degree of a vector defined by (4.2.1).

The matrix of coefficient vectors of the basis is upper triangular, so its determinant is the product of the diagonal entries. Set $m = kd + t$. Hence

$$\begin{aligned}\deg \det M &= \ell \sum_{i=0}^{m-1} i + nd \sum_{i=0}^{k} i \\ &= \ell \frac{m(m-1)}{2} + nd \frac{k(k+1)}{2}.\end{aligned}$$

Since the dimension of our lattice is m, by Theorem 4.2.3 we can find a vector of degree at most

$$\frac{1}{m}\left(\ell \frac{m(m-1)}{2} + nd \frac{k(k+1)}{2}\right).$$

To prove (4.4.1), we would like this bound to be less than βkn. By Lemma 4.3.1, we can achieve any $\ell \leq n\left(\frac{\beta^2}{d} - \varepsilon\right)$. If we set $\varepsilon < \frac{1}{n^2 d}$ then this becomes $\ell < \frac{\beta^2 n}{d}$, as desired, because β can be taken to have denominator n.

Note that we cannot achieve degree equal to $\beta^2 n/d$ (as opposed to strict inequality): for the equation $x^d \equiv 0 \pmod{p(z)^d}$, there are infinitely many solutions $x = c\,p(z)$ if F is infinite.

4.4.2 Reed-Solomon list decoding and noisy polynomial interpolation

A Reed-Solomon code is determined by evaluating a polynomial $w(z) \in \mathbb{F}_q[z]$ of degree at most ℓ at a collection of distinct points $(x_1, \ldots, x_n)$ to obtain a codeword $(w(x_1), \ldots, w(x_n))$. In the Reed-Solomon decoding problem, we are provided with $(y_1, \ldots, y_n)$, where at most e values have changed, and we want to recover $w(z)$ by finding a polynomial of degree at most ℓ that fits at least $n - e$ points (x_i, y_i). Guruswami and Sudan [46] showed how to correct up to $e = n - \sqrt{n\ell}$ errors by providing a list of all possible decodings.

In the noisy polynomial interpolation problem, at each x_i a set $\{y_{i1}, \ldots, y_{id}\}$ of values is specified, and the goal is to find a low-degree polynomial passing through a point from each set. This problem has been proposed as a cryptographic primitive, for example by Naor and Pinkas [79], and studied by Bleichenbacher and Nguyen [12].

We can use Theorem 4.1.2 to solve both problems, and in particular recover the exact decoding rates of Guruswami-Sudan. The input to our problem is a collection of points

$$\{(x_i, y_{ij}) : 1 \leq i \leq n, 1 \leq j \leq d\}.$$

We set $p(z) = \prod_i (z - x_i)$, and we define a monic polynomial $f(x)$ of degree d in x by

$$f(x) = \sum_{i=1}^{n} \prod_{j=1}^{d} (x - y_{ij}) \prod_{\substack{k=1 \\ k \neq i}}^{n} \frac{z - x_k}{x_i - x_k}.$$

We have constructed $f(x)$ by interpolation so that $f(x) \equiv \prod_j (x - y_{ij}) \pmod{(z - x_i)}$. Thus, $f(y_{ij}) = 0$ whenever $z = x_i$.

To correct e errors, we seek a polynomial $w(z)$ of degree at most ℓ such that for at least $n - e$ values of i, there exists a j such that $w(x_i) = y_{ij}$. In other words, $f(w(z))$ must be divisible by at least $n - e$ factors $z - x_i$, which is equivalent to

$$\deg_z \gcd(f(w(z)), p(z)) \geq n - e.$$

Theorem 4.1.2 tells us that we can solve this problem in polynomial time if $\ell < n(1 - e/n)^2/d$ (since $\beta = 1 - e/n$ in the notation of the theorem). That is equivalent to the Guruswami-Sudan bound $e < n - \sqrt{n\ell d}$.

4.4.3 Running time

The Guruswami-Sudan algorithm consists of two parts: constructing the polynomial $Q(x)$, and finding the roots of $Q(x)$ in $\mathbb{F}_q[z]$. In this paper, we do not address the

second part, but we improve the running time of the first part, which has been the bottleneck in the algorithm.

The time to construct Q is dominated by the lattice basis reduction step, which depends on m the dimension of the lattice and the maximum degree D of a coefficient polynomial.

Lemma 4.3.1 tells us that we have $m = O(\beta/\varepsilon)$, where ε has been defined so that $\ell = n(\beta^2/d - \varepsilon)$, and we can assume $D < nk$, since we can reduce the coefficients of $Q(x)$ modulo $p(z)^k$, which has degree nk. The parameter k is set to $O(\beta m/d)$.

Emulating the analysis from [46], when $(\beta n)^2 = (1+\delta)\ell n$, we have $\delta = \varepsilon n/\ell$,

$$m = O((1+\delta)/(\delta\beta)),$$

and

$$D = O(n(1+\delta)/(d\delta)).$$

Using the fastest row reduction algorithm (see Section 4.2.2), the running time is

$$O\big(Dm^{\omega+o(1)}\big) = O\big(n/(\delta^{\omega+1+o(1)})\big).$$

In the worst case, we set $\varepsilon = 1/(n^2 d)$, which gives $m = O(\beta n^2 d)$, $k = O((\beta n)^2)$, and $D = O(n^3\beta^2)$, so the total running time is $O\big(n^{2\omega+3+o(1)}d\big)$ field operations. With cubic-time matrix multiplication we achieve $O(n^9 d)$, and with fast matrix multiplication [31] we achieve $O\big(n^{7.752+o(1)}d\big)$.

The original Guruswami-Sudan approach [46] requires roughly $O(n^3\delta^{-6})$ field operations, or $O(n^{15})$ in the worst case. (The second part of their algorithm runs in time $O(n^{12})$, although there have been improvements since then [92].) The fastest previous algorithm proposed for this problem [104] apparently runs in worst case

time $\widetilde{O}(n^8)$ when $d = 1$, although its running time analysis is only heuristic (see the footnote on page 13 of [104]).

4.5 Number fields

4.5.1 Background on number fields

See [64] for a beautiful introduction to computational algebraic number theory, or [26] for a more comprehensive treatment.

Recall that number fields are finite extensions of the field $\mathbb{Q}$ of rational numbers. Each number field K is generated by some algebraic number α, and the elements of the number field are polynomials in α with rational coefficients. If the minimal polynomial $p(x)$ of α (the lowest-degree polynomial over $\mathbb{Q}$, not identically zero, for which α is a root) has degree n, then every element of $K = \mathbb{Q}(\alpha)$ will be a polynomial in α of degree at most $n - 1$. In other words,

$$\mathbb{Q}(\alpha) = \{a_0 + a_1\alpha + \cdots + a_{n-1}\alpha^{n-1} : a_0, \ldots, a_{n-1} \in \mathbb{Q}\}.$$

The degree of K is defined to be n. It is the dimension of K as a $\mathbb{Q}$-vector space.

The minimal polynomial $p(x)$ must be irreducible over $\mathbb{Q}$, and thus it has n distinct complex roots $\alpha_1, \ldots, \alpha_n$ (one of which is α). Not all of these roots will necessarily be in the field $K = \mathbb{Q}(\alpha)$. For example, the field $\mathbb{Q}(\sqrt[3]{2}\,)$ is contained in $\mathbb{R}$ and thus does not contain either of the complex roots of $x^3 - 2$.

For each i from 1 to n, we can define an embedding σ_i of K into $\mathbb{C}$ by mapping α to α_i and extending by additivity and multiplicativity. All embeddings into $\mathbb{C}$ arise in this way. If p has r_1 real roots and r_2 pairs of complex conjugate (non-real) roots, then there will be r_1 real embeddings and $2r_2$ complex embeddings.

The absolute values on K are defined by

$$|\gamma|_i = |\sigma_i(\gamma)|$$

(where $|\cdot|$ on the right side is the familiar absolute value on $\mathbb{C}$, and $|\cdot|_i$ does not denote the ℓ_i norm). For each i, this valuation has all the usual properties of the absolute value on $\mathbb{Q}$. These absolute values are not necessarily distinct, since they coincide for complex conjugate roots of $p(x)$: if $\alpha_i = \overline{\alpha_j}$, then $|\gamma|_i = |\gamma|_j$ for all γ. Otherwise, the absolute values are all distinct.

The ring of algebraic integers $\mathcal{O}_K$ in K consists of all the elements of K that are roots of monic polynomials over $\mathbb{Z}$. It is the natural analogue of $\mathbb{Z}$ in K (note that $\mathcal{O}_{\mathbb{Q}} = \mathbb{Z}$). In simple cases, $\mathcal{O}_K$ may equal $\mathbb{Z}[\alpha]$, but that is not always true. When $K = \mathbb{Q}(\sqrt{5}\,)$, we have $\mathcal{O}_K = \mathbb{Z}[(1+\sqrt{5}\,)/2]$, and for some number fields the ring of integers cannot even be generated by a single element.

The norm of an element $\gamma \in K$ is defined as the product

$$N(\gamma) = \sigma_1(\gamma) \cdots \sigma_n(\gamma)$$

in $\mathbb{C}$. (In fact, $N(\gamma)$ is rational for $\gamma \in K$, and it is integral for $\gamma \in \mathcal{O}_K$.) If $\gamma \in \mathcal{O}_K$ and $\gamma \neq 0$, then $|N(\gamma)| = |\mathcal{O}_K/\gamma\mathcal{O}_K|$. More generally, for any nonzero ideal I in $\mathcal{O}_K$, we define its norm $N(I)$ to be $|\mathcal{O}_K/I|$. The norm is multiplicative; i.e., $N(IJ) = N(I)N(J)$.

The norm is a natural measure of size for both ideals and individual elements in $\mathcal{O}_K$. It might be tempting to use the norm as our measure of the size of the roots of the polynomial in Theorem 4.1.3. However, that does not work, because $\mathcal{O}_K$ typically has infinitely many units (elements of norm 1). For example, the powers of $(1+\sqrt{5}\,)/2$ are units in $\mathbb{Z}[(1+\sqrt{5}\,)/2]$, which means the equation $x^2 \equiv 0 \pmod 4$ has infinitely many solutions of norm at most $N(4)^{1/2} = N(2) = 4$, namely the numbers $2((1+\sqrt{5}\,)/2)^k$

for $k \in \mathbb{Z}$. Thus, bounding the norm alone is insufficient even to guarantee that there will be only finitely many solutions, but bounding all the absolute values suffices.

The ring $\mathcal{O}_K$ has an integral basis $\omega_1, \ldots, \omega_n$ (i.e., a basis such that every element of $\mathcal{O}_K$ can be expressed uniquely in the form $\sum_i a_i\omega_i$ with $a_i \in \mathbb{Z}$). We assume we are given such a basis, because finding one is computationally difficult (see Theorem 4.4 in [64]). Any reasonably explicit description of $\mathcal{O}_K$ will yield an integral basis. Fortunately, such a description is known for many concrete examples of number fields, such as cyclotomic fields. Furthermore, if we are working with a fixed number field, finding an integral basis for $\mathcal{O}_K$ can be done with only a fixed amount of preprocessing. We also assume that ideals in $\mathcal{O}_K$ are given in terms of integral bases. It is not difficult to convert any other description of an ideal (such as generators over $\mathcal{O}_K$) to an integral basis.

If we do not know the full ring $\mathcal{O}_K$ of integers, we could nevertheless work with an order in K, i.e., a finite-index subring of $\mathcal{O}_K$. Everything we need works just as well for orders, with one exception, namely that the norm is no longer multiplicative for ideals. Fortunately, it remains multiplicative for invertible ideals (see Proposition 4.6.8 in [26]), and Coppersmith's theorem generalizes to invertible ideals. Specifically, we can find small roots of polynomial equations modulo an invertible ideal I, or modulo any invertible ideal B that contains I and satisfies $N(B) \geq N(I)^\beta$.

Finally, polynomials over number fields can be factored in polynomial time [61].

Modules and canonical embeddings

The analogue of a lattice for $\mathcal{O}_K$ is a finitely generated $\mathcal{O}_K$-submodule of the r-dimensional K-vector space K^r. Recall that an $\mathcal{O}_K$-submodule is a non-empty subset that is closed under addition and under multiplication by any element in $\mathcal{O}_K$.

Unlike the case of $\mathbb{Z}$-lattices, $\mathcal{O}_K$-lattices may not have bases over $\mathcal{O}_K$. However, an $\mathcal{O}_K$-lattice Λ always has a pseudo-basis, i.e., a collection of vectors $v_1, \ldots, v_s \in \Lambda$

and ideals $I_1, \ldots, I_s \subseteq \mathcal{O}_K$ such that

$$\Lambda = I_1 v_1 + \cdots + I_s v_s.$$

The key difference from $\mathbb{Z}$ is that the ideals may not be principal (i.e., they may not simply be the multiples of single elements of $\mathcal{O}_K$).

A natural approach to finding a short vector in an $\mathcal{O}_K$-lattice would be to find an algorithm to reduce a pseudo-basis. Fieker and Pohst [41] developed an $\mathcal{O}_K$-analogue of the LLL lattice basis reduction algorithm, but they were unable to prove that their algorithm runs in polynomial time. More recently, Fieker and Stehlé [42] have given a polynomial-time algorithm to find a reduced pseudo-basis in an $\mathcal{O}_K$-module. Their algorithm runs in two parts. The first is to apply LLL to an embedding of the $\mathcal{O}_K$-module as a $\mathbb{Z}$-lattice to find a full-rank set of short module elements, and the second uses this collection of module elements to reduce the pseudo-basis.

As our application only requires finding a short vector in the module, we do not need the second step of the Fieker-Stehlé algorithm. The remainder of this section describes how to use LLL to find a short vector in an $\mathcal{O}_K$-lattice.

Although $\mathcal{O}_K$-lattices are an algebraic analogue of $\mathbb{Z}$-lattices, their geometry is not as easy to see directly from the definition. It might seem natural simply to use one of the absolute values to define the ℓ_2 norm for vectors, but that breaks the symmetry between them. Instead, it is important to treat each absolute value on an equal footing, and the canonical embedding (defined below) allows us to do so.

We will describe the embedding in several steps. First, we embed $\mathcal{O}_K$ itself as an n-dimensional lattice in $\mathbb{R}^{r_1} \oplus \mathbb{C}^{2r_2}$ by mapping $\gamma \in \mathcal{O}_K$ to $(\sigma_1(\gamma), \ldots, \sigma_n(\gamma))$. An

integral basis $\omega_1, \ldots, \omega_n$ of $\mathcal{O}_K$ is mapped to the rows of the matrix

$$\sigma(\omega) = \begin{pmatrix} \sigma_1(\omega_1) & \sigma_2(\omega_1) & \cdots & \sigma_n(\omega_1) \\ \sigma_1(\omega_2) & \ddots & & \sigma_n(\omega_2) \\ \vdots & & & \vdots \\ \sigma_1(\omega_n) & \sigma_2(\omega_n) & \cdots & \sigma_n(\omega_n) \end{pmatrix},$$

so $\mathcal{O}_K$ is mapped to the $\mathbb{Z}$-linear combinations of the rows.

The discriminant Δ_K of K is defined by

$$\Delta_K = \det \sigma(\omega)^2.$$

It is an integer that measures the size of the ring of integers in K.

The canonical embedding of the principal ideal generated by an element γ is generated by the rows of the matrix product

$$\begin{pmatrix} \sigma_1(\omega_1) & \sigma_2(\omega_1) & \cdots & \sigma_n(\omega_1) \\ \sigma_1(\omega_2) & \ddots & & \sigma_n(\omega_2) \\ \vdots & & & \vdots \\ \sigma_1(\omega_n) & \sigma_2(\omega_n) & \cdots & \sigma_n(\omega_n) \end{pmatrix} \begin{pmatrix} \sigma_1(\gamma) & & & \\ & \sigma_2(\gamma) & & \\ & & \ddots & \\ & & & \sigma_n(\gamma) \end{pmatrix}.$$

More generally, suppose we have an ideal B generated by an integral basis $b_1, \ldots, b_n$. Let M_B be the matrix defined by

$$b_i = \sum_j \left(M_B\right)_{ij} \omega_j.$$

The canonical embedding of B is generated by the rows of

$$\sigma(b) = \begin{pmatrix} \sigma_1(b_1) & \sigma_2(b_1) & \cdots & \sigma_n(b_1) \\ \sigma_1(b_2) & \ddots & & \sigma_n(b_2) \\ \vdots & & & \vdots \\ \sigma_1(b_n) & \sigma_2(b_n) & \cdots & \sigma_n(b_n) \end{pmatrix} = M_B \begin{pmatrix} \sigma_1(\omega_1) & \sigma_2(\omega_1) & \cdots & \sigma_n(\omega_1) \\ \sigma_1(\omega_2) & \ddots & & \sigma_n(\omega_2) \\ \vdots & & & \vdots \\ \sigma_1(\omega_n) & \sigma_2(\omega_n) & \cdots & \sigma_n(\omega_n) \end{pmatrix}.$$

Note that the absolute value of the determinant of $\sigma(b)$ equals $|\det M_B|\sqrt{|\Delta_K|}$, and $|\det M_B| = |\mathcal{O}_K/B| = N(B)$.

Finally, we can easily extend the canonical embedding from $\mathcal{O}_K$ to $\mathcal{O}_K^r$ by embedding each of the r coordinates independently. Given a pseudo-basis $v_1, \ldots, v_r$ with corresponding ideals $I_1, \ldots, I_r$, the canonical embedding of the lattice is generated by the rows of the block matrix whose ij block of size $n \times n$ is equal to

$$M_{I_i}\sigma(\omega) \begin{pmatrix} \sigma_1(v_{ij}) & & & \\ & \sigma_2(v_{ij}) & & \\ & & \ddots & \\ & & & \sigma_n(v_{ij}) \end{pmatrix},$$

where v_{ij} is the j-th component of v_i.

The inner product on $\mathbb{R}^{r_1} \oplus \mathbb{C}^{2r_2}$ is given by the usual dot product on $\mathbb{R}$ and the Hermitian inner product on $\mathbb{C}$ (i.e., $\langle x, y\rangle = x\overline{y}$ for $x, y \in \mathbb{C}$). Thus, it is positive definite.

The canonical embedding's image lies within an n-dimensional real subspace, because the complex embeddings come in conjugate pairs. In fact, we can transform it into a simple real embedding. To do so, consider the r_2 pairs of complex embeddings. For each pair $(\sigma_j(\gamma), \sigma_k(\gamma))$ of complex embeddings that are conjugates of each other, we can map the pair $(\sigma_j(\gamma), \sigma_k(\gamma))$ to $(\sqrt{2}\,\mathrm{Re}(\sigma_j(\gamma)), \sqrt{2}\,\mathrm{Im}(\sigma_j(\gamma)))$. The reason for

the factor of $\sqrt{2}$ is to ensure that the inner product is preserved. Furthermore, the absolute value of the determinant is preserved.

Once we have a real embedding of our $\mathcal{O}_K$-lattice, we can apply the LLL algorithm to find a short vector in the real embedded lattice, which will correspond to a short vector in the original $\mathcal{O}_K$-lattice. Unfortunately, using LLL in the canonical embedding does not preserve the $\mathcal{O}_K$-structure, so it does not produce a reduced pseudo-basis over $\mathcal{O}_K$, but a short vector is sufficient for our purposes here.

4.5.2 Proof of Theorem 4.1.3

The following lemma is the analogue of the statement over the integers that a multiple of n that is strictly less than n in absolute value must be zero.

Lemma 4.5.1. *For a nonzero ideal I in $\mathcal{O}_K$ and an element $\gamma \in I$, if $|N(\gamma)| < N(I)$ then $\gamma = 0$.*

Proof. Consider the principal ideal $\gamma\mathcal{O}_K$ generated by a nonzero element γ of I. The ideal I contains $\gamma\mathcal{O}_K$, and thus $|\mathcal{O}_K/I| \leq |\mathcal{O}_K/\gamma\mathcal{O}_K|$. Because $N(I) = |\mathcal{O}_K/I|$ and $|N(\gamma)| = |\mathcal{O}_K/\gamma\mathcal{O}_K|$, we have $|N(\gamma)| \geq N(I)$, as desired. □

Proof of Theorem 4.1.3. As in the previous proofs, we will construct a polynomial $Q(x)$ in the $\mathcal{O}_K$-module generated by

$$x^j f(x)^i I^{k-i} \quad \text{for} \quad 0 \leq i < k \text{ and } 0 \leq j < d$$

and

$$x^j f(x)^k \quad \text{for} \quad 0 \leq j < t.$$

Note that because of the ideals I^{k-i}, this is really a pseudo-basis rather than a basis.

Let $m = dk + t$. To represent this module, we will write down an $nm \times nm$ matrix whose rows are a $\mathbb{Z}$-basis for a weighted version of the module's canonical embedding.

Finding a short vector in this lattice will correspond to finding a Q that satisfies our bounds.

Our lattice is constructed much as before, except that in place of a single entry for each coefficient of $x^j f(x)^i I^{k-i}$, we will have an $n \times n$ block matrix. Let f_{sij} be the coefficient of x^s in $x^j f(x)^i$. Then we form the ideal $f_{sij} I^{k-i}$, which has an integral basis $b_1, \ldots, b_n$. We incorporate the bounds λ_i on each absolute value into our canonical embedding for the s-th coefficient of $x^j f(x)^i I^{k-i}$ by using

$$\begin{pmatrix} \lambda_1^s \sigma_1(b_1) & \lambda_2^s \sigma_2(b_1) & \cdots & \lambda_n^s \sigma_n(b_1) \\ \lambda_1^s \sigma_1(b_2) & \ddots & & \lambda_n^s \sigma_n(b_2) \\ \vdots & & & \vdots \\ \lambda_1^s \sigma_1(b_n) & \lambda_2^s \sigma_2(b_n) & \cdots & \lambda_n^s \sigma_n(b_n) \end{pmatrix}.$$

This is equal to the product of the matrix with $\lambda_1^s, \ldots, \lambda_n^s$ on the diagonal with the canonical embedding $\sigma(b)$, so the absolute value of the determinant of the block is

$$\lambda_1^s \ldots \lambda_n^s \sqrt{|\Delta_K|}\, |N(f_{sij})|\, N(I)^{k-i}.$$

Now consider a vector v in this lattice and the polynomial $Q(x) = \sum_j q_j x^j$ that it represents. If $|w|_i < \lambda_i$ for all i, then we can bound $|N(Q(w))|$ using the ℓ_1 norm by applying the arithmetic mean-geometric mean inequality. We have

$$|N(Q(w))| = \prod_i \left| \sum_j q_j w^j \right|_i,$$

and hence

$$\begin{aligned}|N(Q(w))|^{1/n} &\leq \frac{1}{n}\sum_i \left|\sum_j q_j w^j\right|_i \\ &\leq \frac{1}{n}\sum_i \sum_j |q_j|_i \lambda_i^j.\end{aligned}$$

Thus,

$$|N(Q(w))| \leq \left(\frac{1}{n}|v|_1\right)^n.$$

As in the integer case, LLL produces a nonzero vector v whose ℓ_1 norm is bounded by

$$\sum_i \sum_j |v_i|_j \leq \sqrt{nm}\, 2^{(nm-1)/4} |\det(M)|^{\frac{1}{nm}}.$$

Note that here, $|v_i|_j$ denotes the j-th number field norm applied to the i-th entry of v.

Now it remains to compute the determinant of our weighted canonical embedding. The lattice basis we produced in our construction is block upper triangular, so the determinant is the product of the blocks on the diagonal. Letting $\prod_i \lambda_i = X$, we get

$$\begin{aligned}|\det M| &= \prod_{0\leq i<m} \left(X^i \sqrt{|\Delta_K|}\right) \prod_{0\leq j\leq k} N(I)^{dj} \\ &= \sqrt{|\Delta_K|}^m X^{m(m-1)/2} N(I)^{dk(k+1)/2}.\end{aligned}$$

Thus, we have

$$|v|_1 < \sqrt{nm}\, 2^{(nm-1)/4} \sqrt{|\Delta_K|}^{\frac{1}{n}} \left(X^{m(m-1)/2} N(I)^{dk(k+1)/2}\right)^{\frac{1}{nm}}.$$

Recall that if $|w|_i < \lambda_i$ for all i, then

$$|N(Q(w))| \leq \frac{1}{n^n}|v|_1^n.$$

We will compute a c so that

$$\left(\frac{1}{n^n}\left(\sqrt{nm}2^{(nm-1)/4}\right)^n\sqrt{|\Delta_K|}\right)^{\frac{2}{m-1}} < c.$$

Then by the same analysis as in the proof of Theorem 4.1.1, we can prove the theorem with a bound of

$$\frac{1}{c}N(I)^{\beta^2/d-\varepsilon}$$

on the product $\prod_i \lambda_i$. A simple asymptotic analysis shows that we can take $c = (2+o(1))^{n^2/2}$ as $m \to \infty$. Thus, we achieve a bound of

$$(2+o(1))^{-n^2/2}N(I)^{\beta^2/d-\varepsilon}.$$

As before, we can take $\varepsilon = 1/\log N(I)$ to achieve in fact $(2+o(1))^{-n^2/2}N(I)^{\beta^2/d}$.

Note that so far, everything runs in polynomial time, with no exponential dependence on n. Unfortunately, removing the factor of $(2+o(1))^{-n^2/2}$ is computationally expensive. We can use the same trick as in Theorem 4.1.1. In the canonical embedding of $\mathcal{O}_K$, the region we would like to cover is a box of dimensions $2\lambda_1 \times \cdots \times 2\lambda_n$ (the factor of 2 comes from including positive and negative signs). The proof so far shows that we can deal with a box that is a factor of $(2+o(1))^{-n/2}$ smaller in each coordinate. We can cover the large box with $(2+o(1))^{n^2/2}$ of the smaller ones and compute the solutions in each smaller box in polynomial time, but the total running time becomes exponential in n^2. □

4.5.3 Solving the closest vector problem in ideal lattices

In [84], Peikert and Rosen proposed using the closest vector problem for ideal lattices as a hard problem for use in constructing lattice-based cryptosystems. In [69], Lyubashevsky, Peikert, and Regev gave hardness reductions for such cryptosystems via the

bounded-distance decoding problem, defined for the ℓ_∞ norm as follows. Given an ideal I in $\mathcal{O}_K$, a distance δ, and an element $y \in K$, find $y+w \in I$ such that $||w||_\infty < \delta$, where $||\cdot||_\infty$ denotes the ℓ_∞ norm on K (i.e., the maximum of the n absolute values).

If $y \in \mathcal{O}_K$, then we can define $f(x) = x + y$ and find the roots w of $f(x) \equiv 0 \pmod{I}$ satisfying

$$||w||_\infty < (2 + o(1))^{-n/2} N(I)^{1/n}.$$

This amounts to taking $d = 1$, $\beta = 1$, and $\lambda_1 = \cdots = \lambda_n = (2 + o(1))^{-n/2} N(I)^{1/n}$. Because we are using the ℓ_∞ norm, the minimal nonzero norm of I is at most $\left(\sqrt{|\Delta_K|} N(I)\right)^{1/n}$. Thus, our algorithm can handle distances δ less than $(2 + o(1))^{-n/2} |\Delta_K|^{-1/(2n)}$ times the minimal norm of I. (Of course, this is somewhat worse than using LLL directly.) Note also that if $y \notin \mathcal{O}_K$, then we can rescale y and I by a positive integer to reduce to the previous case.

If the $(2 + o(1))^{-n^2/2}$ could be improved to $2^{-n}\sqrt{|\Delta_K|}$, then we could solve the bounded-distance decoding problem up to half the minimal distance, by the same argument as above with $\lambda_1 = \cdots = \lambda_n = |\Delta_K|^{1/(2n)} N(I)^{1/n}/2$. This suggests that it will be difficult to remove the multiplicative factor entirely.

4.6 Function Fields

Much as number fields are finite extensions of $\mathbb{Q}$, *function fields* are finite extensions of the field $\mathbb{F}_q(x)$ of rational functions over a finite field $\mathbb{F}_q$. They arise naturally from algebraic curves over $\mathbb{F}_q$, as the field of rational functions on the curve. For example, for a plane curve defined by the polynomial equation $f(x, y) = 0$, the function field will be $\mathbb{F}_q(x, y)/(f(x, y))$ (i.e., rational functions of x and y, where the variables satisfy $f(x, y) = 0$). See [102] and [91] for background on function fields, and [68] for a beautiful account of the analogies between number fields and function fields.

More generally, let $\mathcal{X}$ be an algebraic curve over $\mathbb{F}_q$. Specifically, it must be a smooth, projective curve that remains irreducible over the algebraic closure of $\mathbb{F}_q$. Our function field K will be the field of rational functions on $\mathcal{X}$ defined over $\mathbb{F}_q$. (Note that we are assuming $\mathbb{F}_q$ is the full field of constants in K; in other words, each element of K is either in $\mathbb{F}_q$ or transcendental over $\mathbb{F}_q$.)

Let $\mathcal{X}(\mathbb{F}_q)$ be the set of points on $\mathcal{X}$ with coordinates in $\mathbb{F}_q$. Every point $p \in \mathcal{X}(\mathbb{F}_q)$ gives a valuation v_p on K, which measures the order of vanishing at that point. Poles are treated as zeros of negative order. The corresponding absolute value on K is defined by

$$|f|_p = q^{-v_p(f)}.$$

(Note that this is not the ℓ_p norm on a vector; in this section, the ℓ_p norm will not be used.) In other words, high-order zeros make a function small, while poles make it larger. Not every absolute value on K is of this form—there is a slight generalization that corresponds to points defined over finite extensions of $\mathbb{F}_q$ (more precisely, Galois orbits of such points). For our purposes we can restrict our attention to the absolute values defined above, but in fact all our results generalize naturally to places of degree greater than 1.

In the number field case, the Archimedean absolute values (which come from the complex embeddings) play a special role, although there are infinitely many non-Archimedean absolute values as well, namely the p-adic absolute values measuring divisibility by primes. In the function field case, there are no Archimedean absolute values, and any set of absolute values can play the same role.

Let S be a nonempty subset of $\mathcal{X}(\mathbb{F}_q)$, and let $\mathcal{O}_S$ be the subring of K consisting of all rational functions whose poles are confined to the set S. The ring $\mathcal{O}_S$ is analogous to the ring of algebraic integers in a number field; in this analogy, the condition of having no poles outside S amounts to the condition that an algebraic integer has no

primes in its denominator, because the valuations from points outside S correspond to the p-adic valuations.

For example, if $\mathcal{X}$ is the projective line (i.e., the ordinary line completed with a point at infinity), then K is simply the field $\mathbb{F}_q(z)$ of rational functions in one variable. If we let $S = \{\infty\}$ be the set consisting solely of the point at infinity, then $\mathcal{O}_S$ is the set of rational functions that have poles only at infinity. In other words, it is the polynomial ring $\mathbb{F}_q[z]$. (A polynomial of degree d has a pole of order d at infinity.)

The norm of an element $f \in \mathcal{O}_S$ is defined by

$$N(f) = \prod_{p \in S} |f|_p,$$

and the norm of a nonzero ideal I is defined by $N(I) = |\mathcal{O}_S/I|$. As in the number field case, the norm of the principal ideal $f\mathcal{O}_S$ is $N(f)$.

4.6.1 Background on algebraic-geometric codes

Algebraic-geometric codes are a natural generalization of Reed-Solomon codes. They are of great importance in coding theory, because for certain finite fields they beat the Gilbert-Varshamov bound (which is the performance of a random code, and which aside from algebraic-geometric codes is the best bound known). See Section 8.4 in [102].

To define an algebraic-geometric code on $\mathcal{X}$, we specify for each point in S the maximum allowable order of a pole there (and we allow no poles outside of S). The space of functions satisfying these restrictions is a finite-dimensional $\mathbb{F}_q$-vector space, and we can produce an error-correcting code by looking at the evaluations of these functions at a fixed set of points (disjoint from S).

This is typically described using the language of algebraic geometry. A *divisor* D on $\mathcal{X}$ is a formal $\mathbb{Z}$-linear combination of finitely many points on $\mathcal{X}$; the support of D

is the set of points with nonzero coefficients. (We will restrict our attention to divisors supported at points in $\mathcal{X}(\mathbb{F}_q)$.) The divisor D is called *effective*, denoted $D \succeq 0$, if all its coefficients are nonnegative. For every function $f \in K^*$, the *principal divisor* (f) is the sum of the zeros and poles of f, with their orders as coefficients. (The identically zero function does not define a principal divisor, since it has a zero of infinite order at every point.) The *degree* $\deg(D)$ of D to be the sum of its coefficients, and the degree of a principal divisor is always zero.

Given a divisor D, the Riemann-Roch space $\mathcal{L}(D)$ is defined by

$$\mathcal{L}(D) = \{0\} \cup \{f \in K^* : (f) + D \succeq 0\}.$$

In other words, if the coefficient of p in D is k, then f can have a pole of order at most k at the point p. The space $\mathcal{L}(D)$ is a finite-dimensional $\mathbb{F}_q$-vector space, and the famous Riemann-Roch theorem describes its dimension:

$$\dim_{\mathbb{F}_q} \mathcal{L}(D) = \deg(D) - g + 1 + \dim_{\mathbb{F}_q} \mathcal{L}(W - D),$$

where g is a nonnegative integer called the *genus* of the curve and W is a particular divisor called the *canonical divisor*. It follows that $\dim_{\mathbb{F}_q} \mathcal{L}(D) \geq \deg(D) - g + 1$, and equality holds if $\deg(D) > 2g - 2$.

To translate the definition of an algebraic-geometric code to this language, let D be the divisor with support in S whose coefficients specify the allowed order of a pole at each point, and let $p_1, \ldots, p_n$ be distinct points in $\mathcal{X}(\mathbb{F}_q)$ but not in S. Then the corresponding algebraic-geometric code consists of the codewords $(w(p_1), \ldots, w(p_n))$ for $w \in \mathcal{L}(D)$.

In the case of the projective line, let $S = \{\infty\}$, so $\mathcal{O}_S = \mathbb{F}_q[z]$, and let $D = d\infty$. Then $\mathcal{L}(D)$ is the space of polynomials in $\mathbb{F}_q[z]$ of degree at most d. Thus, this construction yields Reed-Solomon codes as a special case.

Theorem 4.1.4 corresponds to list decoding of algebraic-geometric codes in much the same way as Theorem 4.1.2 does for Reed-Solomon codes. The evaluation points $p_1, \ldots, p_n$ correspond to prime ideals $P_1, \ldots, P_n$ in $\mathcal{O}_S$, where P_i consists of the functions vanishing at p_i, and we can let I be the product $P_1 \ldots P_n$. If the received codeword is $(y_1, \ldots, y_n) \in \mathbb{F}_q^n$, then we define the linear polynomial f so that $f(x) \equiv x - y_i \pmod{P_i}$ for all i. (The Chinese remainder theorem lets us solve this interpolation problem.) Thus, for $w \in \mathcal{O}_S$, $f(w)$ is in the ideal P_i if and only if $w(p_i) = y_i$. We have $N(I) = q^n$, and $\gcd(f(w)\mathcal{O}_S, I)$ is divisible by P_i exactly when $w(p_i) = y_i$. Therefore the inequality

$$N(\gcd(f(w)\mathcal{O}_S, I)) \geq N(I)^\beta$$

simply means that $w(p_i) = y_i$ for at least βn values of i. Thus, Theorem 4.1.4 solves the list decoding problem.

4.6.2 Proof of Theorem 4.1.4

As in the number field case, we would like to deal with lattices over a simpler ring than $\mathcal{O}_S$; there, we used the complex embeddings to construct a $\mathbb{Z}$-module. Here, we will use $\mathbb{F}_q[z]$-modules instead, but there is a key conceptual difference, because there are many embeddings of $\mathbb{F}_q[z]$ into $\mathcal{O}_S$ and we must choose the correct one, while there is only one embedding of $\mathbb{Z}$ into $\mathcal{O}_K$.

The property we would like z to have is that $|z|_p$ should be independent of p, as long as $p \in S$. In that case, the absolute values $|\cdot|_p$ with $p \in S$ will all restrict to the same absolute value on the ring $R = \mathbb{F}_q[z]$, which we will denote $|\cdot|$.

When $|S| = 1$, we can choose any nonconstant element z of $\mathcal{O}_S$. When $|S| > 1$, it is not as trivial, but fortunately there is always such an element:

Lemma 4.6.1. *There exists an integer $a \geq 1$ and an element $z \in \mathcal{O}_S$ such that $v_p(z) = -a$ for all $p \in S$, and we can find such an element in probabilistic polynomial time.*

Proof. Let Δ_a be the divisor

$$\sum_{p \in S} ap$$

with coefficient a for each $p \in S$, and let g be the genus of the curve $\mathcal{X}$. If $a|S| > 2g-2$, then by the Riemann-Roch theorem,

$$\dim_{\mathbb{F}_q} \mathcal{L}(\Delta_a) = a|S| - (g-1).$$

Furthermore, if $a|S| > 2g - 1$, then for each $p \in S$,

$$\dim_{\mathbb{F}_q} \mathcal{L}(\Delta_a - p) = a|S| - g.$$

Thus, if $|S| < q$, then $\mathcal{L}(\Delta_a)$ cannot be contained in the union of $\mathcal{L}(\Delta_a - p)$ over all $p \in S$, and therefore there exists a function with poles of order exactly a at each point in S. If $|S| < q/2$, then it is easy to find such a function by random sampling, since at least half the elements in $\mathcal{L}(\Delta_a)$ will work. (Recall that as mentioned in Section 4.1.4, we assume that we can efficiently compute bases of Riemann-Roch spaces.)

This proof requires $|S| < q$, but the same idea works if we pass to a finite extension $\mathbb{F}_{q^i}$ of $\mathbb{F}_q$, and it can handle $|S| < q^i$. Thus, if we take i large enough, there exists a function defined over $\mathbb{F}_{q^i}$ with poles of equal order a at the points in S (and no poles elsewhere). Now multiplying the i conjugates of this function over $\mathbb{F}_q$ produces such a function over $\mathbb{F}_q$, as desired, with poles of order ai. Taking $q^i > 2|S|$ gives an efficient algorithm as well. □

For the rest of this section, let z be such a function and let $R = \mathbb{F}_q[z]$. Then the ring $\mathcal{O}_S$ is a free R-module of rank $a|S|$ by Theorem 1.4.11 in [102], as is every nonzero ideal in $\mathcal{O}_S$.

As in the previous proofs, we will construct a polynomial $Q(x)$ in the $\mathcal{O}_S$-module $\mathcal{M}$ generated by

$$x^j f(x)^i I^{k-i} \quad \text{for} \quad 0 \leq i < k \text{ and } 0 \leq j < d$$

and

$$x^j f(x)^k \quad \text{for} \quad 0 \leq j < t.$$

Let $m = dk + t$.

The module $\mathcal{M}$ is a submodule of the $\mathcal{O}_S$-module $\mathcal{P}$ of polynomials of degree less than m, which is a free $\mathcal{O}_S$-module of rank m and hence a free R-module of rank $ma|S|$. Thus, as in the setting of Lemmas 4.2.2 and 4.2.3, we are working with an R-module contained in a free R-module.

We want $Q(x)$ to have the property that for $w \in \mathcal{L}(D)$,

$$N(Q(w)) < N(I)^{\beta k}.$$

In fact, we will bound $N(Q(w))$ by

$$N(Q(w)) = \prod_{p \in S} |Q(w)|_p \leq \left(\max_{p \in S} |Q(w)|_p \right)^{|S|},$$

and we will ensure that

$$\left(\max_{p \in S} |Q(w)|_p \right)^{|S|} < N(I)^{\beta k}.$$

Let $q_0, \ldots, q_{m-1}$ denote the coefficients of Q, so

$$Q(x) = \sum_{i=0}^{m-1} q_i x^i.$$

Then

$$|Q(w)|_p \leq \max_i |q_i|_p |w|_p^i.$$

Suppose the divisor D is given by

$$D = \sum_{p \in S} \lambda_p p.$$

Then $|w|_p \leq q^{\lambda_p}$ for $w \in \mathcal{L}(D)$, and thus

$$|Q(w)|_p \leq \max_i |q_i|_p \, q^{i\lambda_p}.$$

To emulate the analysis from Sections 4.3 and 4.4, we would like to find $X \in \mathcal{O}_S$ such that $v_p(X) = -\lambda_p$ for all $p \in S$. However, such an element does not always exist. Instead, we will construct an element with the desired valuations at all but one point in S. This approach is a special case of the strong approximation theorem (Theorem 1.6.5 in [102] or Theorem 6.13 in [91]), but as we need only a weaker conclusion and must consider computational feasibility, we will give a direct proof.

Lemma 4.6.2. *Suppose $q \geq 2|S|$. Then for any point $p_0 \in S$ and each divisor $\sum_{p \in S} \mu_p p$ satisfying $\sum_{p \in S} \mu_p \geq 0$ and $\mu_{p_0} = 0$, there exists an element $X \in \mathcal{O}_S$ such that $v_p(X) = -\mu_p$ for all $p \in S \setminus \{p_0\}$, and $v_{p_0}(X) = -2g$, where g is the genus of $\mathcal{X}$. Furthermore, we can construct such an X in probabilistic polynomial time.*

Proof. Let $\Delta = \sum_{p \in S} \mu_p p + 2g p_0$. Then $\deg(\Delta) \geq 2g$, and it follows from Riemann-Roch that $\dim_{\mathbb{F}_q} \mathcal{L}(\Delta) = \deg(\Delta) - (g-1)$ and that $\dim_{\mathbb{F}_q} \mathcal{L}(\Delta - p) = \dim_{\mathbb{F}_q} \mathcal{L}(\Delta) - 1$ for all $p \in S$. We are looking for an element X in $\mathcal{L}(\Delta)$ but not $\mathcal{L}(\Delta - p)$ for any

$p \in S$. By assumption we can construct these Riemann-Roch spaces, and because $|S| \leq q/2$ at least half the elements of X will have the desired property, so we can find one by random sampling. $\square$

The assumption that $q \geq 2|S|$ will hold in most applications: most algebraic-geometric codes use a small set S, and in fact $|S|$ cannot be much larger than q because $S \subseteq \mathcal{X}(\mathbb{F}_q)$ and $|\mathcal{X}(\mathbb{F}_q)| \leq q + 2g\sqrt{q} + 1$ (see Theorem 5.2.3 in [102]). However, if $|S| > q/2$, then we can simply pass to a finite extension of $\mathbb{F}_q$. Thus, without loss of generality we can assume that $q \geq 2|S|$.

By assumption in Theorem 4.1.4, the support of D is a proper subset of S, so we can let $p_0 \in S$ be a point such that $\lambda_{p_0} = 0$. Because of the limitations of the strong approximation theorem, we require such a point to make the remainder of the proof work. This is not an obstacle to the applicability of the theorem, because algebraic-geometric codes will generally not use every point in $\mathcal{X}(\mathbb{F}_q)$ for poles or evaluation points, and if they do we can pass to a finite extension of $\mathbb{F}_q$ to generate more points. Note also that we can assume $\deg(D) \geq 0$, because otherwise $\mathcal{L}(D)$ is the empty set.

Now, Lemma 4.6.2 lets us construct an element $X \in \mathcal{O}_S$ such that $v_p(X) = -\lambda_p$ for $p \in S \setminus \{p_0\}$. This element has the property that $v_p(X^i) = -i\lambda_p$ for $p \in S \setminus \{p_0\}$. Unfortunately, the valuation at p_0 grows linearly with i as well, and that will damage our bounds. However, we can avoid that problem by applying Lemma 4.6.2 to construct elements X_i so that $v_p(X_i) = -i\lambda_p$ for $p \in S \setminus \{p_0\}$ while maintaining $v_{p_0}(X_i) = -2g$. Of course we set $X_0 = 1$.

In terms of the elements X_i, we have

$$|Q(w)|_p \leq \max_i |q_i X_i|_p$$

for $p \in S \setminus \{p_0\}$. Furthermore, this inequality holds for $p = p_0$ because $v_{p_0}(w) \geq 0 \geq v_{p_0}(X_i)$.

Define the norm of a polynomial $\sum_i c_i x^i \in \mathcal{P}$ (with $c_i \in \mathcal{O}_S$) by

$$\left| \sum_i c_i x^i \right| = \max_i \max_{p \in S} |c_i|_p.$$

Note that this defines a non-Archimedean norm on the free R-module $\mathcal{P}$ satisfying all three properties required in Section 4.2.3 (with the absolute value $|\cdot|$ on R). Here, we crucially use the fact that we have only one absolute value on R; if that were not the case, then property 3 would fail.

Let $T \colon \mathcal{P} \to \mathcal{P}$ be the linear transformation that multiplies the degree i term by X_i. Then

$$\max_{p \in S} |Q(w)|_p \leq \max_{p \in S} \max_i |q_i X_i|_p = |TQ|.$$

Thus, it will suffice to construct a nonzero polynomial $Q \in \mathcal{M}$ such that $|TQ|^{|S|} < N(I)^{\beta k}$.

Now we can apply Lemma 4.2.3. We need to determine two things: the geometric mean C of the norms of an R-basis of $\mathcal{P}$ and the dimension of the quotient $\mathcal{P}/T\mathcal{M}$. Then there exists a nonzero $Q \in \mathcal{M}$ such that

$$|TQ| \leq C|z|^{\dim_{\mathbb{F}_q}(\mathcal{P}/T\mathcal{M})/(a|S|m)} = Cq^{\dim_{\mathbb{F}_q}(\mathcal{P}/T\mathcal{M})/(|S|m)},$$

because these R-modules have rank $a|S|m$ and $|z| = q^a$.

Let $b_1, \ldots, b_{a|S|}$ be any R-basis of $\mathcal{O}_S$, and let

$$C = \left(\prod_{i=1}^{a|S|} \max_{p \in S} |b_i|_p \right)^{\frac{1}{a|S|}}.$$

Then the elements $b_i x^j \in \mathcal{P}$ (with $1 \le i \le a|S|$ and $0 \le j < m$) form an R-basis of $\mathcal{P}$, and the geometric mean of their norms is C because $|b_i x^j|$ is independent of the degree j.

To compute the dimension of $\mathcal{P}/T\mathcal{M}$, note that the generators of $\mathcal{M}$ are triangular (i.e., given by polynomials of each degree). Thus, we merely need to add the dimensions of the quotients of $\mathcal{O}_S$ by the modules of leading coefficients. From the polynomials $X_{di+j} x^j f(x)^i I^{k-i}$, we see that the leading coefficients form the ideal $X_{di+j} I^{k-i}$. Thus,

$$\begin{aligned} q^{\dim_{\mathbb{F}_q} \mathcal{P}/T\mathcal{M}} = |\mathcal{P}/T\mathcal{M}| &= N(I)^{dk(k+1)/2} \prod_{i=0}^{m-1} N(X_i) \\ &= N(I)^{dk(k+1)/2} \left(\prod_{p \in S} q^{\lambda_p m(m-1)/2} \right) \prod_{i=0}^{m-1} |X_i|_{p_0}. \end{aligned}$$

Thus,

$$q^{\dim_{\mathbb{F}_q} \mathcal{P}/T\mathcal{M}} \le N(I)^{dk(k+1)/2} q^{\deg(D) m(m-1)/2} q^{2mg}.$$

Now applying Lemma 4.2.3 shows that we can find a nonzero polynomial $Q \in \mathcal{M}$ such that

$$|TQ|^{|S|} \le C q^{2g} q^{\deg(D)(m-1)/2} N(I)^{dk(k+1)/(2m)}.$$

We want to achieve $|TQ|^{|S|} < N(I)^{\beta k}$. Let $N(I) = q^n$ and

$$\ell = \deg(D) + \frac{2}{m-1} \log_q \left(C q^{2g} \right).$$

Then Lemma 4.3.1 applies, and shows that we can achieve $|TQ|^{|S|} < N(I)^{\beta k}$ whenever $\ell < n \left(\frac{\beta^2}{d} - \varepsilon \right)$, which is equivalent to

$$\left(C q^{2g} \right)^{\frac{2}{m-1}} q^{\deg(D)} < N(I)^{\frac{\beta^2}{d} - \varepsilon}.$$

We can take the denominator of β to be a divisor of n (because $N(I) = q^n$). Thus, $N(I)^{\beta^2/d}$ is an integral power of $q^{1/(nd)}$, as of course is $q^{\deg(D)}$, and to prove the bound in Theorem 4.1.4 it suffices to prove it to within a factor of less than $q^{1/(nd)}$.

Now let $\varepsilon < 1/(2n^2d)$ and $m > 1 + 4nd(2g + \log_q C)$. Then $N(I)^{\varepsilon}$ and $(Cq^{2g})^{\frac{2}{m-1}}$ are both strictly less than $q^{1/(2n^2d)}$. Thus, our algorithm works as long as

$$q^{\deg(D)} < N(I)^{\beta^2/d}.$$

This completes the proof of Theorem 4.1.4.

Bibliography

[1] Advanced Encryption Standard. National Institute of Standards and Technology, FIPS-197, November 2001.

[2] Miklós Ajtai. The shortest vector problem in L_2 is NP-hard for randomized reductions. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 10–19, New York, NY, USA, 1998. ACM.

[3] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 601–610, New York, NY, USA, 2001. ACM.

[4] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In Omer Reingold, editor, *Theory of Cryptography*, volume 5444 of *Lecture Notes in Computer Science*, pages 474–95. Springer-Verlag, March 2009.

[5] Joel Alwen, Yevgeniy Dodis, and Daniel Wichs. Public key cryptography in the bounded retrieval model and security against side-channel attacks. In Shai Halevi, editor, *Proceedings of Crypto 2009*, volume 5677 of *Lecture Notes in Computer Science*. Springer-Verlag, August 2009.

[6] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*, page 282. Wiley, first edition, January 2001.

[7] H. Peter Anvin. SYSLINUX. `http://syslinux.zytor.com/`.

[8] William Arbaugh, David Farber, and Jonathan Smith. A secure and reliable bootstrap architecture. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 65–71, May 1997.

[9] Alon Bar-Lev. Linux, Loop-AES and optional smartcard based disk encryption.

[10] Peter Barry and Gerard Hartnett. *Designing Embedded Networking Applications: Essential Insights for Developers of Intel IXP4XX Network Processor Systems*, page 47. Intel Press, first edition, May 2005.

[11] Daniel J. Bernstein. List decoding for binary Goppa codes. 2008. `http://cr.yp.to/codes/goppalist-20081107.pdf`.

[12] Daniel Bleichenbacher and Phong Q. Nguyen. Noisy polynomial interpolation and noisy Chinese remaindering. In *Proceedings of Eurocrypt 2000*, volume 1802 of *Lecture Notes in Computer Science*, pages 53–69. Springer-Verlag Berlin/Heidelberg, 2000.

[13] Johannes Blömer and Alexander May. New partial key exposure attacks on rsa. In Dan Boneh, editor, *Proceedings of Crypto 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 27–43. Springer Berlin / Heidelberg, 2003.

[14] Adam Boileau. Hit by a bus: Physical access attacks with Firewire. Presentation, Ruxcon, 2006.

[15] Dan Boneh. Twenty years of attacks on the RSA cryptosystem. *Notices of the American Mathematical Society (AMS)*, 46(2):203–13, February 1999.

[16] Dan Boneh. Finding smooth integers in short intervals using CRT decoding. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 265–272. ACM, 2000.

[17] Dan Boneh, Glenn Durfee, and Yair Frankel. An attack on RSA given a small fraction of the private key bits. In Kazuo Ohta and Dingyi Pei, editors, *Proceedings of Asiacrypt 1998*, volume 1514 of *Lecture Notes in Computer Science*, pages 25–34. Springer-Verlag, October 1998.

[18] Dan Boneh and Hovav Shacham. Fast variants of RSA. *RSA Cryptobytes*, 5(1):1–9, Winter/Spring 2002.

[19] Xavier Boyen. Halting password puzzles: Hard-to-break encryption from human-memorable keys. In *Proceedings of the 16th USENIX Security Symposium*, August 2008.

[20] J. Buchmann, T. Takagi, and U. Vollmer. Number field cryptography. In Alf van der Poorten and Andreas Stein, editors, *High Primes and Misdemeanours: Lectures in Honour of the 60th Birthday of Hugh Cowie Williams*, volume 41 of *Fields Institute Communications*. American Mathematical Society, 2004.

[21] California Statutes. Cal. Civ. Code §1798.82, created by S.B. 1386, August 2002.

[22] Ran Canetti, Yevgeniy Dodis, Shai Halevi, Eyal Kushilevitz, and Amit Sahai. Exposure-resilient functions and all-or-nothing transforms. In Bart Preneel, editor, *Proceedings of Eurocrypt 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 453–469. Springer Berlin / Heidelberg, 2000.

[23] Brian D. Carrier and Joe Grand. A hardware-based memory acquisition procedure for digital investigations. *Digital Investigation*, 1:50–60, December 2003.

[24] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143 – 154, 1979.

[25] Jim Chow, Ben Pfaff, Tal Garfinkel, and Mendel Rosenblum. Shredding your garbage: Reducing data lifetime through secure deallocation. In *Proceedings of the 14th USENIX Security Symposium*, pages 331–346, August 2005.

[26] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Number 138 in Graduate Texts in Mathematics. Springer-Verlag, Berlin/Heidelberg, 1993.

[27] Henry Cohn and Nadia Heninger. Ideal forms of Coppersmith's theorem and Guruswami-Sudan list decoding. In *Proceedings of the Second Symposium on Innovations in Computer Science, ICS 2011*. Tsinghua University Press, January 2011.

[28] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–60, December 1997.

[29] Don Coppersmith. Finding small solutions to small degree polynomials. In *CaLC '01: Revised Papers from the International Conference on Cryptography and Lattices*, volume 2146 of *Lecture Notes in Computer Science*, pages 20–31. Springer-Verlag Berlin/Heidelberg, 2001.

[30] Don Coppersmith, Nicholas Howgrave-Graham, and S. V. Nagaraj. Divisors in residue classes, constructively. *Mathematics of Computation*, 77:531–545, 2008.

[31] Don Coppersmith and Shmuel Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9(3):251–280, 1990.

[32] Jean-Sebastien Coron and Alexander May. Deterministic polynomial-time equivalence of computing the RSA secret key and factoring. *Journal of Cryptology*, 20(1):39–50, January 2007.

[33] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In Michael Mitzenmacher, editor, *Proceedings of of the 41st Annual ACM Symposium on the Theory of Computing*. ACM, May 2009.

[34] Maximillian Dornseif. 0wned by an iPod. Presentation, PacSec, 2004.

[35] Maximillian Dornseif. Firewire – all your memory are belong to us. Presentation, CanSecWest/core05, May 2005.

[36] Jeffrey Dwoskin and Ruby B. Lee. Hardware-rooted trust for secure key management and transient trust. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, pages 389–400, October 2007.

[37] Joan G. Dyer, Mark Lindemann, Ronald Perez, Reiner Sailer, Leendert van Doorn, Sean W. Smith, and Steve Weingart. Building the IBM 4758 secure coprocessor. *Computer*, 34:57–66, October 2001.

[38] Knut Eckstein and Maximillian Dornseif. On the meaning of 'physical access' to a computing device: A vulnerability classification of mobile computing devices. Presentation, NATO C3A Workshop on Network-Enabled Warfare, April 2005.

[39] Jeremy Elson and Lewis Girod. Fusd – a Linux framework for user-space devices. `http://www.circlemud.org/~jelson/software/fusd/`.

[40] Niels Ferguson. AES-CBC + Elephant diffuser: A disk encryption algorithm for Windows Vista, August 2006.

[41] C. Fieker and M. E. Pohst. On lattices over number fields. In *Algorithmic Number Theory*, pages 133–139. Springer-Verlag, 1996.

[42] Claus Fieker and Damien Stehlé. Short bases of lattices over number fields. In Guillaume Hanrot, Franois Morain, and Emmanuel Thom, editors, *Algorithmic Number Theory*, volume 6197 of *Lecture Notes in Computer Science*, pages 157–173. Springer Berlin / Heidelberg, 2010.

[43] Pascal Giorgi, Claude-Pierre Jeannerod, and Gilles Villard. On the complexity of polynomial matrix computations. In *Proceedings of the 2003 International Symposium on Symbolic and Algebraic Computation*, pages 135–142, New York, NY, USA, 2003. ACM.

[44] Shafi Goldwasser. Cryptography without (hardly any) secrets ? In Antoine Joux, editor, *Eurocrypt 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 369–370. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-01001-9-21.

[45] Venkatesan Guruswami, Amit Sahai, and Madhu Sudan. "Soft-decision" decoding of Chinese remainder codes. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 159–168, Washington, DC, USA, 2000. IEEE Computer Society.

[46] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 28–39, 1998.

[47] Peter Gutmann. Secure deletion of data from magnetic and solid-state memory. In *Proceedings of the 6th USENIX Security Symposium*, pages 77–90, July 1996.

[48] Peter Gutmann. Data remanence in semiconductor devices. In *Proceedings of the 10th USENIX Security Symposium*, pages 39–54, August 2001.

[49] J. Alex Halderman, Seth Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph Calandrino, Ariel Feldman, Jacob Appelbaum, and Edward Felten. Lest we remember: Cold boot attacks on encryption keys. In Paul Van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium*, pages 45–60. USENIX, July 2008.

[50] Nadia Heninger and Hovav Shacham. Reconstructing RSA private keys from random key bits. In Shai Halevi, editor, *Proceedings of Crypto 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 1–17. Springer-Verlag, August 2009.

[51] Mathias Herrmann and Alexander May. Solving linear equations modulo divisors: On factoring given any bits. In Josef Pieprzyk, editor, *Proceedings of Asiacrypt 2008*, volume 5350 of *Lecture Notes in Computer Science*, pages 406–24. Springer-Verlag, December 2008.

[52] Nicholas Howgrave-Graham. Approximate integer common divisors. In *CaLC '01: Revised Papers from the International Conference on Cryptography and Lattices*, pages 51–66. Springer-Verlag, 2001.

[53] Ming-Deh Huang and Doug Ierardi. Efficient algorithms for the Riemann-Roch problem and for addition in the Jacobian of a curve. *Journal of Symbolic Computation*, 18(6):519–539, 1994.

[54] IEEE 1619 Security in Storage Working Group. IEEE P1619/D19: Draft standard for cryptographic protection of data on block-oriented storage devices, July 2007.

[55] Intel Corporation. Preboot Execution Enviroment (PXE) specification version 2.1, September 1999.

[56] Thomas Kailath. *Linear Systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, 1980.

[57] Samuel Karlin and Howard M. Taylor. *A First Course in Stochastic Processes*. Academic Press, 1975.

[58] Clement Kent. Draft proposal for tweakable narrow-block encryption. `https://siswg.net/docs/LRW-AES-10-19-2004.pdf`, 2004.

[59] S. V. Konyagin and T. Steger. On polynomial congruences. *Mathematical Notes*, 55:596–600, 1994.

[60] Ruby B. Lee, Peter C.S. Kwan, John P. McGregor, Jeffrey Dwoskin, and Zhenghong Wang. Architecture for protecting critical secrets in microprocessors. In *Proceedings of the International Symposium on Computer Architecture*, pages 2–13, 2005.

[61] Arjen K. Lenstra. Factoring polynomials over algebraic number fields. In *Computer Algebra (London, 1983)*, volume 162 of *Lecture Notes in Computer Science*, pages 245–254. Springer, 1983.

[62] Arjen K. Lenstra. Factoring multivariate polynomials over algebraic number fields. *SIAM Journal of Computing*, 16(3):591–598, 1987.

[63] H. W. Lenstra, A. K. Lenstra, and L. Lovász. Factoring polynomials with rational coeficients. *Mathematische Annalen*, 261(4):515–534, 1982.

[64] Hendrik W. Lenstra. Algorithms in algebraic number theory. *Bulletin of the American Mathematical Society*, 26:211–244, 1992.

[65] David Lie, Chandramohan A. Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. In *Symposium on Architectural Support for Programming Languages and Operating Systems*, 2000.

[66] Walter Link and Herbert May. Eigenschaften von MOS-Ein-Transistorspeicherzellen bei tiefen Temperaturen. *Archiv für Elektronik und Übertragungstechnik*, 33:229–235, June 1979.

[67] Moses Liskov, Ronald L. Rivest, and David Wagner. Tweakable block ciphers. In Moti Yung, editor, *Proceedings of Crypto 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 31–46. Springer Berlin / Heidelberg, 2002.

[68] Dino Lorenzini. *An invitation to arithmetic geometry*, volume 9 of *Graduate Studies in Mathematics*. American Mathematical Society, Providence, RI, 1996.

[69] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Proceedings of Eurocrypt 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer Berlin / Heidelberg, 2010.

[70] Douglas MacIver. Penetration testing Windows Vista BitLocker drive encryption. Presentation, Hack In The Box, September 2006.

[71] Kenneth Manders and Leonard Adleman. NP-complete decision problems for quadratic polynomials. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, pages 23–29. ACM, May 1976.

[72] R. C. Mason. *Diophantine Equations over Function Fields*. Number 96 in London Mathematical Society Lecture Note Series. Cambridge University Press, Cambridge, England, 1984.

[73] Ueli Maurer. On the oracle complexity of factoring integers. *Computational Complexity*, 5(3/4):237–47, September 1995.

[74] Alexander May. *New RSA Vulnerabilities Using Lattice Reduction Methods*. PhD thesis, University of Paderborn, October 2003.

[75] Alexander May. Using LLL-reduction for solving RSA and factorization problems: A survey. In Phong Nguyen, editor, *Proceedings of LLL+25*, June 2007.

[76] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[77] Silvio Micali and Leonid Reyzin. Physically observable cryptography. In *Theory of Cryptography*, volume 2951/2004 of *Lecture Notes in Computer Science*, 2004.

[78] Peter Bro Miltersen. Error correcting codes, perfect hashing circuits, and deterministic dynamic dictionaries. In *Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 556–563, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[79] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 245–254. ACM, 1999.

[80] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In Shai Halevi, editor, *Proceedings of Crypto 2009*, volume 5677 of *Lecture Notes in Computer Science*. Springer-Verlag, August 2009.

[81] National Computer Security Center. A guide to understanding data remanence in automated information systems, September 1991.

[82] National Conference of State Legislatures. State security breach notification laws. `http://www.ncsl.org/programs/lis/cip/priv/breachlaws.htm`, January 2008.

[83] Phong Nguyen and Jacques Stern. Adapting density attacks to low-weight knapsacks. In Bimal Roy, editor, *Proceedings of Asiacrypt 2005*, volume 3788 of *Lecture Notes in Computer Science*, pages 41–58. Springer-Verlag, December 2005.

[84] Chris Peikert and Alon Rosen. Lattices that admit logarithmic worst-case to average-case connection factors. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 478–487, New York, NY, USA, 2007. ACM.

[85] Colin Percival. Cache missing for fun and profit. Online: `http://www.daemonology.net/papers/cachemissing.pdf`, May 2005.

[86] Torbjörn Pettersson. Cryptographic key recovery from Linux memory dumps. Presentation, Chaos Communication Camp, August 2007.

[87] Krzysztof Pietrzak. A leakage-resilient mode of operation. In Antoine Joux, editor, *Proceedings of Eurocrypt 2009*, volume 5479 of *Lecture Notes in Computer Science*, pages 462–82. Springer-Verlag, April 2009.

[88] Thomas Ptacek. Recover a private key from process memory. `http://www.matasano.com/log/178/recover-a-private-key-from-process-memory/`.

[89] Ronald Rivest and Adi Shamir. Efficient factoring based on partial information. In Franz Pichler, editor, *Proceedings of Eurocrypt 1985*, volume 219 of *Lecture Notes in Computer Science*, pages 31–4. Springer-Verlag, April 1985.

[90] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Lee, editor, *Proceedings of Asiacrypt 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 55–73. Springer Berlin / Heidelberg, 2004.

[91] Michael Ira Rosen. *Number Theory in Function Fields*. Number 210 in Graduate Texts in Mathematics. Springer-Verlag, New York, 2002.

[92] Ron M. Roth and Gitit Ruckenstein. Efficient decoding of Reed-Solomon codes beyond half the minimum distance. *IEEE Transactions on Information Theory*, 46(1):246–257, 2000.

[93] RSA Laboratories. PKCS #1 v2.1: RSA cryptography standard. `ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf`.

[94] RSA Laboratories. PKCS #1 v2.1: RSA cryptography standard, June 2002. `http://www.rsa.com/rsalabs/node.asp?id=2125`.

[95] Leif Z. Scheick, Steven M. Guertin, and Gary M. Swift. Analysis of radiation effects on individual DRAM cells. *IEEE Transactions on Nuclear Science*, 47:2534–2538, December 2000.

[96] Seagate Corporation. Drivetrust technology: A technical overview. `http://www.seagate.com/docs/pdf/whitepaper/TP564\_DriveTrust\_Oct06.pdf`.

[97] Adi Shamir and Nicko van Someren. Playing hide and seek with stored keys. In Matthew Franklin, editor, *Financial Cryptography*, volume 1648 of *Lecture Notes in Computer Science*, pages 118–124. Springer Berlin / Heidelberg, 1999.

[98] M. Amin Shokrollahi and Hal Wasserman. List decoding of algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45(2):432–437, 1999.

[99] Victor Shoup. OAEP reconsidered. *Journal of Cryptology*, 15:223–249, 2008.

[100] Sergei Skorobogatov. Low-temperature data remanence in static RAM. University of Cambridge Computer Laborary Technical Report No. 536, June 2002.

[101] Sean W. Smith. *Trusted Computing Platforms: Design and Applications*. Springer, first edition, 2005.

[102] Henning Stichtenoth. *Algebraic Function Fields and Codes*. Springer-Verlag, New York, second edition, 2010.

[103] Madhu Sudan. Ideal error-correcting codes: Unifying algebraic and number-theoretic algorithms. In Serdar Boztas and Igor Shparlinski, editors, *AAECC*, volume 2227 of *Lecture Notes in Computer Science*, pages 36–45. Springer, 2001.

[104] Peter V. Trifonov. Efficient interpolation in the Guruswami-Sudan algorithm. *IEEE Transactions on Information Theory*, 56(9):4341–4349, September 2010.

[105] Trusted Computing Group. Trusted Platform Module specification version 1.2. https://www.trustedcomputinggroup.org/specs/TPM/, July 2007.

[106] Alex Tsow. An improved recovery algorithm for decayed aes key schedule images. In Michael Jacobson, Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 215–230. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-05445-7-14.

[107] Timothy Vidas. The acquisition and analysis of random access memory. *Journal of Digital Forensic Practice*, 1:315–323, December 2006.

[108] J. von zur Gathen. Hensel and Newton methods in valuation rings. *Mathematics of Computation*, 42(166):637–661, 1984.

[109] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, Cambridge, England, second edition, 2003.

[110] J. von zur Gathen and E. Kaltofen. Factorization of multivariate polynomials over finite fields. *Mathematics of Computation*, 45(171):251–261, July 1985.

[111] AAron Walters and Nick L. Petroni Jr. FATKit: The forensic analysis toolkit. http://www.4tphi.net/fatkit/.

[112] Ralf-Philip Weinmann and Jacob Appelbaum. Unlocking FileVault. Presentation, 23rd Chaos Communication Congress, December 2006.

[113] Ralf-Philip Weinmann and Jacob Appelbaum. VileFault. http://vilefault.googlecode.com/, January 2008.

[114] Philippe Wyns and Richard L. Anderson. Low-temperature operation of silicon dynamic random-access memories. *IEEE Transactions on Electron Devices*, 36:1423–1428, August 1989.

[115] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In Anja Feldmann and Laurent Mathy, editors, *Proceedings of IMC 2009*, pages 15–27. ACM Press, November 2009.