An introduction to cryptography and cryptanalysis

Edward Schaefer
Santa Clara University
eschaefer@scu.edu

I have given history short-shrift in my attempt to get to modern cryptography as quickly as possible. As sources for these lectures I used conversations with DeathAndTaxes (bitcointalk.org), K. Dyer, T. Elgamal, B. Kaliski, H.W. Lenstra, P. Makowski, Jr., M. Manulis, K. McCurley, A. Odlyzko, C. Pomerance, M. Robshaw, and Y.L. Yin as well as the publications listed in the bibliography. I am very grateful to each person listed above. Any mistakes in this document are mine. Please notify me of any that you find at the above e-mail address.

Table of contents

Introduction

Cryptography is used to hide information. It is not only use by spies but for phone, fax and e-mail communication, bank transactions, bank account security, PINs, passwords and credit card transactions on the web. It is also used for a variety of other information security issues including electronic signatures, which are used to prove who sent a message.

1 Vocabulary

A plaintext message, or simply a plaintext, is a message to be communicated. A disguised version of a plaintext message is a ciphertext message or simply a ciphertext. The process of creating a ciphertext from a plaintext is called encryption. The process of turning a ciphertext back into a plaintext is called decryption. The verbs encipher and decipher are synonymous with the verbs encrypt and decrypt. In England, cryptology is the study of encryption and decryption and cryptography is the application of them. In the U.S., the terms are synonymous, and the latter term is used more commonly.

In non-technical English, the term encode is often used as a synonym for encrypt. We will not use it that way. To encode a plaintext changes the plaintext into a series of bits (usually) or numbers (traditionally). A bit is simply a 0 or a 1. There is nothing secret about encoding. A simple encoding of the alphabet would be A $\rightarrow$ 0, ..., Z $\rightarrow$ 25. Using this, we could encode the message HELLO as 7 4 11 11 14. The most common method of encoding a message nowadays is to replace it by its ASCII equivalent, which is an 8 bit representation for each symbol. See Appendix A for ASCII encoding. Decoding turns bits or numbers back into plaintext.

A stream cipher operates on a message symbol-by-symbol, or nowadays bit-by-bit. A block cipher operates on blocks of symbols. A digraph is a pair of letters and a trigraph is a triple of letters. These are blocks that were used historically in cryptography. The Advanced Encryption Standard (AES) operates on 128 bit strings. So when AES is used to encrypt a text message, it encrypts blocks of $128/8 = 16$ symbols.

A transposition cipher rearranges the letters, symbols or bits in a plaintext. A substitution cipher replaces letters, symbols or bits in a plaintext with others without changing the order. A product cipher alternates transposition and substitution. The concept of stream versus block cipher really only applies to substitution and product ciphers, not transposition ciphers.

An algorithm is a series of steps performed by a computer (nowadays) or a person (traditionally) to perform some task. A cryptosystem consists of an enciphering algorithm and a deciphering algorithm. The word cipher is synonymous with cryptosystem. A symmetric key cryptosystem requires a secret shared key. We will see examples of keys later on. Two users must agree on a key ahead of time. In a public key cryptosystem, each user has an encrypting key which is published and a decrypting key which is not.

Cryptanalysis is the process by which the enemy tries to turn CT into PT. It can also mean the study of this.

Cryptosystems come in 3 kinds:
1. Those that have been broken (most).

2. Those that have not yet been analyzed (because they are new and not yet widely used).
3. Those that have been analyzed but not broken. (RSA, Discrete log cryptosystems, Triple-DES, AES).

3 most common ways for the enemy to turn ciphertext into plaintext:
1. Steal/purchase/bribe to get key
2. Exploit sloppy implementation/protocol problems (hacking). Examples: someone used spouse's name as key; someone sent key along with message
3. Cryptanalysis
Alice is the sender of an encrypted message. Bob is the recipient. Eve is the eavesdropper who tries to read the encrypted message.

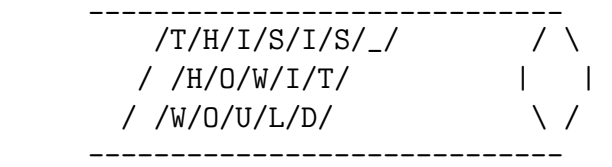
2 Concepts

1. Encryption and decryption should be easy for the proper users, Alice and Bob. Decryption should be hard for Eve.

Computers are much better at handling discrete objects. Number theory is an excellent source of discrete (i.e. finite) problems with easy and hard aspects.
2. Security and practicality of a successful cryptosystem are almost always tradeoffs. Practicality issues: time, storage, co-presence.
3. Must assume that the enemy will find out about the nature of a cryptosystem and will only be missing a key.

3 History

400 BC Spartan scytale cipher (sounds like *Italy*). Example of transposition cipher. Letters were written on a long thin strip of leather wrapped around a cylinder. The diameter of the cylinder was the key.



Julius Caesar's substitution cipher. Shift all letters three to the right. In our alphabet that would send $A \to D, B \to E, \ldots, Z \to C$.

1910's British Playfair cipher (Boer War, WWI). One of the earliest to operate on digraphs. Also a substitution cipher. Key PALMERSTON

P	A	L	M	E
R	S	T	O	N
B	C	D	F	G
H	IJ	K	Q	U
V	W	X	Y	Z

To encrypt SF, make a box with those two letter as corners, the other two corners are the ciphertext OC. The order is determined by the fact that S and O are in the same row as are F and C. If two plaintext letters are in the same row then replace each letter by the letter to its right. So SO becomes TN and BG becomes CB. If two letters are in the same column then replace each letter by the letter below it. So IS becomes WC and SJ becomes CW. Double letters are separated by X's so The plaintext BALLOON would become BA LX LO ON before being encrypted. There are no J's in the ciphertext, only I's.

The Germany Army's ADFGVX cipher used during World War I. One of the earliest product ciphers.

There was a fixed table.

$$\begin{array}{c|cccccc} & A & D & F & G & V & X \\ \hline A & K & Z & W & R & 1 & F \\ D & 9 & B & 6 & C & L & 5 \\ F & Q & 7 & J & P & G & X \\ G & E & V & Y & 3 & A & N \\ V & 8 & O & D & H & 0 & 2 \\ X & U & 4 & I & S & T & M \end{array}$$

To encrypt, replace the plaintext letter/digit by the pair (row, column). So plaintext PRODUCTCIPHERS becomes FG AG VD VF XA DG XV DG XF FG VG GA AG XG. That's the substitution part. Transposition part follows and depends on a key with no repeated letters. Let's say it is DEUTSCH. Number the letters in the key alphabetically. Put the tentative ciphertext above, row by row under the key.

D	E	U	T	S	C	H
2	3	7	6	5	1	4
F	G	A	G	V	D	V
F	X	A	D	G	X	V
D	G	X	F	F	G	V
G	G	A	A	G	X	G

Write the columns numerically. Ciphertext: DXGX FFDG GXGG VVVG VGFG GDFA AAXA (the spaces would not be used).

In World War II it was shown that alternating substitution and transposition ciphers is a very secure thing to do. ADFGVX is weak since the substitution and transposition each occur once and the substitution is fixed, not key controlled.

In the late 1960's, threats to computer security were considered real problems. There was a need for strong encryption in the private sector. One could now put very complex algorithms on a single chip so one could have secure high-speed encryption. There was also the possibility of high-speed cryptanalysis. So what would be best to use?

The problem was studied intensively between 1968 and 1975. In 1974, the Lucifer cipher was introduced and in 1975, DES (the Data Encryption Standard) was introduced. In 2002, AES was introduced. All are product ciphers. DES uses a 56 bit key with 8 additional bits for parity check. DES operates on blocks of 64 bit plaintexts and gives 64 bit ciphertexts.

It alternates 16 substitutions with 15 transpositions. AES uses a 128 bit key and alternates 10 substitutions with 10 transpositions. Its plaintexts and ciphertexts each have 128 bits. In 1975 came public key cryptography. This enables Alice and Bob to agree on a key safely without ever meeting.

4 Crash course in Number Theory

You will be hit with a lot of number theory here. Don't try to absorb it all at once. I want to get it all down in one place so that we can refer to it later. Don't panic if you don't get it all the first time through.

Let $\mathbf{Z}$ denote the integers $\ldots, -2, -1, 0, 1, 2, \ldots$. The symbol $\in$ means *is an element of*. If $a, b \in \mathbf{Z}$ we say a divides b if $b = na$ for some $n \in \mathbf{Z}$ and write $a|b$. a divides b is just another way of saying b is a multiple of a. So $3|12$ since $12 = 4 \cdot 3$, $3|3$ since $3 = 1 \cdot 3$, $5|-5$ since $-5 = -1 \cdot 5$, $6|0$ since $0 = 0 \cdot 6$. If $x|1$, what is x? (Answer ± 1). Properties:
If $a, b, c \in \mathbf{Z}$ and $a|b$ then $a|bc$. I.e., since $3|12$ then $3|60$.
If $a|b$ and $b|c$ then $a|c$.
If $a|b$ and $a|c$ then $a|b \pm c$.
If $a|b$ and $a \not| c$ (not divide) then $a \not| b \pm c$.

The primes are $2, 3, 5, 7, 11, 13 \ldots$.

The Fundamental Theorem of Arithmetic: Any $n \in \mathbf{Z}$, $n > 1$, can be written uniquely as a product of powers of distinct primes $n = p_1^{\alpha_1} \cdot \ldots \cdot p_r^{\alpha_r}$ where the α_i's are positive integers.

For example $90 = 2^1 \cdot 3^2 \cdot 5^1$.

Given $a, b \in \mathbf{Z}_{\geq 0}$ (the non-negative integers), not both 0, the greatest common divisor of a and b is the largest integer d dividing both a and b. It is denoted $\gcd(a, b)$ or just (a, b). As examples: $\gcd(12, 18) = 6$, $\gcd(12, 19) = 1$. You were familiar with this concept as a child. To get the fraction 12/18 into lowest terms, cancel the 6's. The fraction 12/19 is already in lowest terms.

If you have the factorization of a and b written out, then take the product of the primes to the minimum of the two exponents, for each prime, to get the gcd. $2520 = 2^3 \cdot 3^2 \cdot 5^1 \cdot 7^1$ and $2700 = 2^2 \cdot 3^3 \cdot 5^2 \cdot 7^0$ so $\gcd(2520, 2700) = 2^2 \cdot 3^2 \cdot 5^1 \cdot 7^0 = 180$. Note $2520/180 = 14$, $2700/180 = 15$ and $\gcd(14, 15) = 1$. We say that two numbers with gcd equal to 1 are relatively prime.

Factoring is slow with large numbers. The Euclidean algorithm for gcd'ing is very fast with large numbers. Find $\gcd(329, 119)$. Recall long division. When dividing 119 into 329 you get 2 with remainder of 91. In general dividing y into x you get $x = qy + r$ where $0 \leq r < y$. At each step, previous divisor and remainder become the new dividend and divisor.

$$\begin{aligned} 329 &= 2 \cdot \underline{119} + \underline{91} \\ 119 &= 1 \cdot \underline{91} + \underline{28} \\ 91 &= 3 \cdot \underline{28} + \underline{7} \\ 28 &= 4 \cdot \underline{7} + \underline{0} \end{aligned}$$

The number above the 0 is the gcd. So $\gcd(329, 119) = 7$.

We can always write $\gcd(a,b) = na + mb$ for some $n, m \in \mathbf{Z}$. At each step, replace the smaller underlined number.

$$\begin{array}{rcll} 7 & = & \underline{91} - 3 \cdot \underline{28} & \text{replace smaller} \\ & = & \underline{91} - 3(\underline{119} - 1 \cdot \underline{91}) & \text{simplify} \\ & = & 4 \cdot \underline{91} - 3 \cdot \underline{119} & \text{replace smaller} \\ & = & 4 \cdot (\underline{329} - 2 \cdot \underline{119}) - 3 \cdot \underline{119} & \text{simplify} \\ 7 & = & 4 \cdot \underline{329} - 11 \cdot \underline{119} & \end{array}$$

So we have $7 = 4 \cdot 329 - 11 \cdot 119$ where $n = 4$ and $m = -11$.

Modulo. There are two kinds, that used by number theorists and that used by computer scientists.

Number theorist's: $a \equiv b (\mathrm{mod} m)$ if $m|a-b$. In words: a and b differ by a multiple of m. So $7 \equiv 2(\mathrm{mod}5)$, since $5|5$, $2 \equiv 7(\mathrm{mod}5)$ since $5|-5$, $12 \equiv 7(\mathrm{mod}5)$ since $5|5$, $12 \equiv 2(\mathrm{mod}5)$ since $5|10$, $7 \equiv 7(\mathrm{mod}5)$ since $5|0$, $-3 \equiv 7(\mathrm{mod}5)$ since $5|-10$. Below, the integers with the same symbols underneath them are all congruent (or equivalent) mod 5.

-4	-3	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$\cap$	$\star$	$\vee$	$\oplus$	$\dagger$	$\cap$	$\star$	$\vee$	$\oplus$	$\dagger$	$\cap$	$\star$	$\vee$	$\oplus$	$\dagger$	$\cap$	$\star$	$\vee$	$\oplus$

In general working mod m breaks the integers into m subsets. Each subset contains exactly one representative in the range $[0, m-1]$. The set of subsets is denoted $\mathbf{Z}/m\mathbf{Z}$ or $\mathbf{Z}_m$. We see that $\mathbf{Z}/m\mathbf{Z}$ has m elements. So the number $0, \ldots, m-1$ are representatives of the m elements of $\mathbf{Z}/m\mathbf{Z}$.

Computer scientist's: $b\mathrm{mod}m = r$ is the remainder you get $0 \leq r < m$ when dividing m into b. So $12\mathrm{mod}5 = 2$ and $7\mathrm{mod}5 = 2$. (Note the mathematician's is a notation that says $m|a-b$. The computer scientist's can be thought of as a function of two variables b and m giving the output r.)

Here are some examples of mod you are familiar with. Clock arithmetic is mod 12. If it's 3 hours after 11 then it's 2 o'clock because $11 + 3 = 14\mathrm{mod}\,12 = 2$. Even numbers are those numbers that are $\equiv 0(\mathrm{mod}\,2)$. Odd numbers are those that are $\equiv 1(\mathrm{mod}\,2)$.

Properties of mod

1) $a \equiv a(\mathrm{mod}m)$

2) if $a \equiv b(\mathrm{mod}m)$ then $b \equiv a(\mathrm{mod}m)$

3) if $a \equiv b(\mathrm{mod}m)$ and $b \equiv c(\mathrm{mod}m)$ then $a \equiv c(\mathrm{mod}m)$

4) If $a \equiv b(\mathrm{mod}m)$ and $c \equiv d(\mathrm{mod}m)$ then $a \pm c \equiv b \pm d(\mathrm{mod}m)$ and $a \cdot c \equiv b \cdot d(\mathrm{mod}m)$. So you can do these operations in $\mathbf{Z}/m\mathbf{Z}$.

Another way to explain 4) is to say that mod respects $+$, $-$ and $\cdot$.

$$\begin{array}{ccc} 12, 14 & \overset{\mathrm{mod}\,5}{\rightarrow} & 2, 4 \\ +\downarrow & & \downarrow + \\ 26 & \overset{\mathrm{mod}\,5}{\rightarrow} & 1 \end{array}$$

Say $m = 5$, then $\mathbf{Z}/5\mathbf{Z} = \{0, 1, 2, 3, 4\}$. $2 \cdot 3 = 1$ in $\mathbf{Z}/5\mathbf{Z}$ since $2 \cdot 3 = 6 \equiv 1(\mathrm{mod}5)$. $3 + 4 = 2$ in $\mathbf{Z}/5\mathbf{Z}$ since $3 + 4 = 7 \equiv 2(\mathrm{mod}5)$. $0 - 1 = 4$ in $\mathbf{Z}/5\mathbf{Z}$ since $-1 \equiv 4(\mathrm{mod}5)$. Addition table in $\mathbf{Z}/5\mathbf{Z}$.

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \end{array} \\ \left[\begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 & 0 \\ 2 & 3 & 4 & 0 & 1 \\ 3 & 4 & 0 & 1 & 2 \\ 4 & 0 & 1 & 2 & 3 \end{array}\right] \end{array}$$

5) An element x of $\mathbf{Z}/m\mathbf{Z}$ has a multiplicative inverse $(1/x)$ or x^{-1} in $\mathbf{Z}/m\mathbf{Z}$ when $\gcd(x,m) = 1$. The elements of $\mathbf{Z}/m\mathbf{Z}$ with inverses are denoted $\mathbf{Z}/m\mathbf{Z}^*$. Note $1/2 = 2^{-1} \equiv 3(\text{mod}5)$ since $2 \cdot 3 \equiv 1(\text{mod}5)$.

When we work in $\mathbf{Z}/9\mathbf{Z} = \{0, 1, \ldots, 8\}$ we can use $+, -, \cdot$. When we work in $\mathbf{Z}/9\mathbf{Z}^* = \{1, 2, 4, 5, 7, 8\}$ we can use $\cdot, \div$.

Find the inverse of 7 mod 9, i.e. find 7^{-1} in $\mathbf{Z}/9\mathbf{Z}$ (or more properly in $\mathbf{Z}/9\mathbf{Z}^*$). Use the Euclidean algorithm

$$\begin{array}{rcl} 9 & = & 1 \cdot 7 + 2 \\ 7 & = & 3 \cdot 2 + 1 \\ (2 & = & 2 \cdot 1 + 0) \\ \text{so} & & \\ 1 & = & 7 - 3 \cdot 2 \\ 1 & = & 7 - 3(9 - 7) \\ 1 & = & 4 \cdot 7 - 3 \cdot 9 \end{array}$$

Take that equation mod 9 (we can do this because $a \equiv a(\text{mod}\, m)$). We have $1 = 4 \cdot 7 - 3 \cdot 9 \equiv 4 \cdot 7 - 3 \cdot 0 \equiv 4 \cdot 7(\text{mod}9)$. So $1 \equiv 4 \cdot 7(\text{mod}9)$ so $7^{-1} = 1/7 = 4$ in $\mathbf{Z}/9\mathbf{Z}$ or $7^{-1} \equiv 4(\text{mod}9)$ and also $1/4 = 7$ in $\mathbf{Z}/9\mathbf{Z}$.

What's $2/7$ in $\mathbf{Z}/9\mathbf{Z}$? $2/7 = 2 \cdot 1/7 = 2 \cdot 4 = 8 \in \mathbf{Z}/9\mathbf{Z}$. So $2/7 \equiv 8(\text{mod}9)$. Note $2 \equiv 8 \cdot 7(\text{mod}9)$ since $9|(2 - 56 = -54)$.

6 can't have an inverse mod 9. If $6x \equiv 1(\text{mod}9)$ then $9|6x - 1$ so $3|6x - 1$ and $3|6x$ so $3| - 1$ which is not true which is why 6 can't have an inverse mod 9.

6) If $a \equiv b(\text{mod}m)$ and $c \equiv d(\text{mod}\, m)$, $\gcd(c,m) = 1$ (so $\gcd(d,m) = 1$) then $ac^{-1} \equiv bd^{-1}(\text{mod}m)$ or $a/c \equiv b/d(\text{mod}m)$. In other words, division works well as long as you divide by something relatively prime to the modulus m, i.e. invertible. It is like avoiding dividing by 0.

7) Solving $ax \equiv b(\text{mod}m)$ with a, b, m given. If $\gcd(a,m) = 1$ then the solutions are all numbers $x \equiv a^{-1}b(\text{mod}m)$. If $\gcd(a,m) = g$ then there are solutions when $g|b$. Then the equation is equivalent to $ax/g \equiv b/g(\text{mod}m/g)$. Now $\gcd(a/g, m/g) = 1$ so $x \equiv (a/g)^{-1}(b/g)(\text{mod}m/g)$ are the solutions. If $g \not| b$ then there are no solutions.

Solve $7x \equiv 6(\text{mod}11)$. $\gcd(7,11) = 1$. So $x \equiv 7^{-1} \cdot 6(\text{mod}11)$. Find $7^{-1}(\text{mod}11)$: $11 = 1 \cdot 7 + 4, 7 = 1 \cdot 4 + 3, 4 = 1 \cdot 3 + 1$. So $1 = 4 - 1(3) = 4 - 1(7 - 1 \cdot 4) = 2 \cdot 4 - 1 \cdot 7 = 2(11 - 1 \cdot 7) - 1 \cdot 7 = 2 \cdot 11 - 3 \cdot 7$. Thus $1 \equiv -3 \cdot 7(\text{mod}11)$ and $1 \equiv 8 \cdot 7(\text{mod}\, 11)$. So $7^{-1} \equiv 8(\text{mod}\, 11)$. So $x \equiv 6 \cdot 8 \equiv 4(\text{mod}11)$.

Solve $6x \equiv 8(\text{mod}10)$. $\gcd(6,10) = 2$ and $2|8$ so there are solutions. This is the same as $3x \equiv 4(\text{mod}5)$ so $x \equiv 4 \cdot 3^{-1}(\text{mod}5)$. We've seen $3^{-1} \equiv 2(\text{mod}5)$ so $x \equiv 4 \cdot 2 \equiv 3(\text{mod}5)$. Another way to write that is $x = 3 + 5n$ where $n \in \mathbf{Z}$. Best for cryptography is $x \equiv 3$ or $8(\text{mod}10)$.

Solve $6x \equiv 7(\text{mod}10)$. Can't since $\gcd(6,10) = 2$ and $2 \not| 7$.

Let's do some cute practice with modular inversion. A computer will always use the Euclidean algorithm. But cute tricks will help us understand mod better. Example: Find the inverses of all elements of $\mathbf{Z}/17\mathbf{Z}^*$. The integers that are 1 mod 17 are those of the form $17n + 1$. We can factor a few of those. The first few positive integers that are $17n + 1$ bigger than 1 are 18, 35, 52. Note $18 = 2 \cdot 9$ so $2 \cdot 9 \equiv 1(\text{mod}\,17)$ and $2^{-1} \equiv 9(\text{mod}\,17)$ and $9^{-1} \equiv 2(\text{mod}\,17)$. We also have $18 = 3 \cdot 6$, so 3 and 6 are inverses mod 17. We have $35 = 5 \cdot 7$ so 5 and 7 are inverses. We have $52 = 4 \cdot 13$. Going back, we have $18 = 2 \cdot 9 \equiv (-2)(-9) \equiv 15 \cdot 8$ and $18 = 3 \cdot 6 = (-3)(-6) \equiv 14 \cdot 11$. Similarly we have $35 = 5 \cdot 7 = (-5)(-7) \equiv 12 \cdot 10$. Note that $16 \equiv -1$ and $1 = (-1)(-1) \equiv 16 \cdot 16$. So now we have the inverse of all elements of $\mathbf{Z}/17\mathbf{Z}^*$.

Practice using mod: Show $x^3 - x - 1$ is never a perfect square if $x \in \mathbf{Z}$. Solution: All numbers are $\equiv 0, 1, \text{or}\, 2(\text{mod}3)$. So all squares are $\equiv 0^2, 1^2, \text{or}\, 2^2(\text{mod}3) \equiv 0, 1, 1(\text{mod}3)$. But $x^3 - x - 1 \equiv 0^3 - 0 - 1 \equiv 2, 1^3 - 1 - 1 \equiv 2, \text{or} 2^3 - 2 - 1 \equiv 2(\text{mod}3)$.

The Euler phi function: Let $n \in \mathbf{Z}_{>0}$. We have $\mathbf{Z}/n\mathbf{Z}^* = \{a \mid 1 \leq a \leq n, gcd(a,n) = 1\}$. (This is a group under multiplication.) $\mathbf{Z}/12\mathbf{Z}^* = \{1, 5, 7, 11\}$. Let $\phi(n) = |\mathbf{Z}/n\mathbf{Z}^*|$. We have $\phi(12) = 4$. We have $\phi(5) = 4$ and $\phi(6) = 2$. If p is prime then $\phi(p) = p - 1$. What is $\phi(5^3)$? Well $\mathbf{Z}^*_{125} = \mathbf{Z}_{125}$ without multiples of 5. There are $125/5 = 25$ multiples of 5. So $\phi(125) = 125 - 25$. If $r \geq 1$, and p is prime, then $\phi(p^r) = p^r - p^{r-1} = p^{r-1}(p-1)$. If $\gcd(m,n) = 1$ then $\phi(mn) = \phi(m)\phi(n)$. To compute ϕ of a number, break it into prime powers as in this example: $\phi(720) = \phi(2^4)\phi(3^2)\phi(5) = 2^3(2-1)3^1(3-1)(5-1) = 192$. So if $n = \prod p_i^{\alpha_i}$ then $\phi(n) = p_1^{\alpha_1 - 1}(p_1 - 1) \cdots p_r^{\alpha_r - 1}(p_r - 1)$.

Fermat's little theorem. If p is prime and $a \in \mathbf{Z}$ then $a^p \equiv a(\text{mod}p)$. If p does not divide a then $a^{p-1} \equiv 1(\text{mod}p)$.

So it is guaranteed that $4^{11} \equiv 4(\text{mod}11)$ since 11 is prime and $6^{11} \equiv 6(\text{mod}11)$ and $2^{10} \equiv 1(\text{mod}11)$. You can check that they are all true.

If $\gcd(a,m) = 1$ then $a^{\phi(m)} \equiv 1(\text{mod}m)$.

We have $\phi(10) = \phi(5)\phi(2) = 4 \cdot 1 = 4$. $\mathbf{Z}/10\mathbf{Z}^* = \{1, 3, 7, 9\}$. So it is guaranteed that $1^4 \equiv 1(\text{mod}10)$, $3^4 \equiv 1(\text{mod}10)$, $7^4 \equiv 1(\text{mod}10)$ and $9^4 \equiv 1(\text{mod}10)$. You can check that they are all true.

If $\gcd(c,m) = 1$ and $a \equiv b(\text{mod}\phi(m))$ with $a, b \in \mathbf{Z}_{\geq 0}$ then $c^a \equiv c^b(\text{mod}m)$.

Reduce $2^{3005}(\text{mod}21)$. Note $\phi(21) = \phi(7)\phi(3) = 6 \cdot 2 = 12$ and $3005 \equiv 5(\text{mod}12)$ so $2^{3005} \equiv 2^5 \equiv 32 \equiv 11(\text{mod}21)$.

In other words, exponents work mod $\phi(m)$ as long as the bases are relatively prime.

4.1 Calculator algorithms

Reducing a mod m (often the parenthesis are omitted): Reducing 1000 mod 23. On calculator: $1000 \div 23 =$ (you see 43.478...) $-43 =$ (you see .478...) $\times 23 =$ (you see 11). So

$1000 \equiv 11 \bmod 23$. Why does it work? If divide 23 into 1000 you get 43 with remainder 11. So $1000 = 43 \cdot 23 + 11$. $\div 23$ and get $43 + \frac{11}{23}$. -43 and get $\frac{11}{23}$. $\times 23$ and get 11. Note $1000 = 43 \cdot 23 + 11 (\bmod 23)$. So $1000 \equiv 43 \cdot 23 + 11 \equiv 0 + 11 \equiv 11 (\bmod 23)$.

Repeated squares algorithm

Recall, if $(b, m) = 1$ and $x \equiv y (\bmod \phi(m))$ then $b^x \equiv b^y (\bmod m)$. So if computing $b^x (\bmod m)$ with $(b, m) = 1$ and $x \geq \phi(m)$, first reduce x mod $\phi(m)$.

Repeated squares algorithm . This is useful for reducing b^n mod m when $n < \phi(m)$, but n is still large. Reduce 87^{43} mod 103. First write 43 in base 2. This is also called the binary representation of 43. The sloppy/easy way is to write 43 as a sum of different powers of 2 We have $43 = 32 + 8 + 2 + 1$ (keep subtracting off largest possible power of 2). We are missing 16 and 4. So $43 = (101011)_2$ (binary). Recall this means $43 = 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$. A computer uses a program described by the following pseudo-code. Let S be a string
$S = []$
$n = 43$
while $n > 0$
$bit = n \bmod 2$
$S =$concat(bit,S)
$n = (n - bit)/2$ (or $n = n$ div 2)
end while

The output is a vector with the binary representation written backwards. (In class, do the example. Make a table n, bit , S)

Now the repeated squares algorithm for reducing $b^n (\bmod m)$. Write n in its binary representation $(S[k]S[k-1] \ldots S[1]S[0])_2$. Let a be the partial product. At the beginning $a = 1$.

Or as pseudo-code
$a = 1$
if $S[0] = 1$ then $a = b$
for $i = 1$ to k
    $b = b^2 \bmod m$
    if $S[i] = 1$, $a = b \cdot a \bmod m$
end for
print(a)

Now $b^n \bmod m = a$.

We'll do the above example again with $b = 87$, $n = 43$, $m = 103$. 43 in base 2 is 101011, so $k = 5$, $S[0] = 1$, $S[1] = 1$, $S[2] = 0$, $S[3] = 1$, $S[4] = 0$, $S[5] = 1$ (note backwards).

b	S	a
		1
87	$S[0]=1$	$a=87$
$87^2 \equiv 50$	$S[1]=1$	$a=50\cdot 87\equiv 24$ $(\equiv 87^2\cdot 87^1)$
$50^2 \equiv 28$	$S[2]=0$	$a=24$
$28^2 \equiv 63$	$S[3]=1$	$a\equiv 63\cdot 24\equiv 70$ $(\equiv 87^8\cdot 87^2\cdot 87^1)$
$63^2 \equiv 55$	$S[4]=0$	$a=70$
$55^2 \equiv 38$	$S[5]=1$	$a=38\cdot 70\equiv 85$ $(\equiv 87^{32}\cdot 87^8\cdot 87^2\cdot 87^1)$
		$(\equiv 87^{32+8+2+1}\equiv 87^{43})$

5 Running Time of Algorithms

Encryption and decryption should be fast; cryptanalysis should be slow. To quantify these statements, we need to understand how fast certain cryptographic algorithms run.

Logarithms really shrink very large numbers. As an example, if you took a sheet of paper and then put another on top, and then doubled the pile again (four sheets now) and so on until you've doubled the pile 50 times you would have $2^{50} \approx 10^{15}$ sheets of paper and the stack would reach the sun. On the other hand $\log_2(2^{50}) = 50$. A stack of 50 sheets of paper is 1cm tall.

If x is a real number then $\lfloor x \rfloor$ is the largest integer $\leq x$. So $\lfloor 1.4 \rfloor = 1$ and $\lfloor 1 \rfloor = 1$. Recall how we write integers in base 2. Keep removing the largest power of 2 remaining. Example: $47 \geq 32$. $47-32=15$. $15-8=7$, $7-4=3$, $3-2=1$. So $47 = 32+ \quad +8+4+2+1 = (101111)_2$.

Another algorithm uses the following pseudo-code, assuming the number is represented as 32 bits. Assume entries of v are $v[1], \ldots, v[32]$.

input n
v:=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
i:=0
while $n \neq 0$
    reduction := $n (\mathrm{mod}\, 2)$.
    $v[\mathrm{length}(v) - i]$:=reduction,
    n := (n−reduction)/2.
    $i := i + 1$.

We say 47 is a 6 bit number. The number of base 2 digits of an integer N (often called the length) is its number of bits or $\lfloor \log_2(N) \rfloor + 1$. So it's about $\log_2(N)$. All logarithms differ by a constant multiple; (for example: $\log_2(x) = k\log_{10}(x)$, where $k = \log_2(10)$).)

Running time estimates (really upper bounds) are based on worst/slowest case scenarios where you assume inputs are large. Let me describe a few bit operations. Let's add two n-bit numbers $N + M$. We'll add $219 + 242$ or $11011011 + 11110010$, here $n = 8$

```
 111  1
 11011011
 11110010
---------
111001101
```

We will call what happens in a column a bit operation. It is a fixed set of comparisons and shifts. So this whole thing took $n \approx \log_2(N)$ bit operations. If you add n and m bit numbers together and $n \geq m$ then it still takes n bit operations (since you'll have to 'copy' all of the unaffected digits starting the longer number).

Let's multiply an n-bit number N with an m-bit number M where $n \geq m$. Note that we omit the final addition in the diagram.

```
    10111
     1011
    -----
    10111
   101110
 10111000
```

Two parts: writing rows, then add them up. First part: There are at most m rows appearing below the 1st line, each row has at most $n+m-1$ bits. Just writing the last one down takes $n+m-1$ bit op'ns. So the first part takes at most $m(n+m-1)$ bit op'ns. Second part: There will then be at most $m-1$ add'ns, each of which takes at most $n+m-1$ bit op'ns. So this part takes at most $(m-1)(n+m-1)$ bit op'ns. We have a total of $m(m+n-1)+(m-1)(n+m-1)$ $= (2m-1)(n+m-1)$ bit op'ns. We have $(2m-1)(n+m-1) \leq (2m)(n+m) \leq (2m)(2n) = 4mn$ bit op'ns or $4\log_2(N)\log_2 M$ as a nice upper bound. (We ignore the time to access memory, etc. as this is trivial.) How fast a computer runs varies so the running time is $C \cdot 4\log_2(N)\log_2 M$ where C depends on the computer and how we measure time. Or we could say $C' \cdot \log_2(N)\log_2 M = C'' \cdot \log(N)\log(M)$.

If f and g are positive functions on positive integers (domain $\mathbf{Z}_{>0}$ or $\mathbf{Z}_{>0}^r$ if several variables, range $\mathbf{R}_{>0}$ - the positive real numbers) and there's a constant $c > 0$ such that $f < cg$ for sufficiently large input then we say $f = O(g)$.

So $f = O(g)$ means f is bounded by a constant multiple of g (usually g is nice).

So the running time of adding N to M where $N \geq M$ is $O(\log(N))$. This is also true for subtraction. For multiplying N and M it's $O(\log(N)\log(M))$. If N and M are about the same size we say the time for computing their product is $O(\log^2(N))$. Note $\log^2(N) = (\log(N))^2 \neq \log(\log(N)) = \text{loglog}(N)$. Writing down N takes time $O(\log(N))$.

There are faster multiplication algorithms that take time $O(\log(N)\text{loglog}(N)\text{logloglog}(N))$.

It turns out that the time to divide N by M and get quotient and remainder is $O(\log(N)\log(M))$. So reducing N mod M same.

Rules:

1. $kO(f(N)) = O(kf(N)) = O(f(N))$.
2. Let $p(N) = a_d N^d + a_{d-1}N^{d-1} + \ldots + a_0$ be a polynomial.

a) Then $p(N) = O(N^d)$. (It is easy to show that $2N^2+5N < 3N^2$ for large N, so $2N^2+5N = O(3N^2) = O(N^2)$.)

b) $O(\log(p(N))) = O(\log(N))$ (since $O(\log(p(N))) = O(\log(N^d)) = O(d\log(N)) = O(\log(N))$.

3. If $h(N) \leq f(N)$ then $O(f(N)) + O(h(N)) = O(f(N))$. Proof: $O(f(N)) + O(h(N)) = O(f(N) + h(N)) = O(2f(N)) = O(f(N))$.
4. $f(N)O(h(N)) = O(f(N))O(h(N)) = O(f(N)h(N))$.

How to do a running time analysis.

A) Count the number of (mega)-steps.

B) Describe the worst/slowest step.

C) Find an upper bound for the running time of the worst step. (Ask: *What is the action?*)

D) Find an upper bound for the running time of the whole algorithm (often by computing *A*) times *C*)).

E) Answer should be of the form $O(\ldots)$.

Review: $F \cdot G$ and $F \div G$ are $O(\log F \log G)$. $F + G$ and $F - G$ are $O(\log(\text{bigger}))$.

Problem 1: Find an upper bound for how long it takes to compute $\gcd(N, M)$ if $N > M$ by the Euclidean algorithm. Solution: gcd'ing is slowest, if the quotients are all 1: Like $\gcd(21, 13)$: The quotients are always 1 if you try to find $\gcd(F_n, F_{n-1})$ where F_n is the nth Fibonacci number. $F_1 = F_2 = 1$, $F_n = F_{n-1} + F_{n-2}$. Note, number of steps is $n - 3$, which rounds up to n. Let $\alpha = (1 + \sqrt{5})/2$. Then $F_n \approx \alpha^n$. So, worst if $N = F_n$, $M = F_{n-1}$. Note $N \approx \alpha^n$ so $n \approx \log_\alpha(N)$. Imp't: Running time upper bound: (number of steps) times (time per step). There are $n = O(\log(N))$ steps. ((Never use n again)). Each step is a division, which takes $O(\log(N)\log(M))$. So $O(\log(N)O(\log(N)\log(M)) \stackrel{\text{rule 4}}{=} O(\log^2(N)\log(M))$ or, rounding up again $O(\log^3(N))$. So if you double the length ($= O(\log(N))$) of your numbers, it will take 8 times as long. Why is this true? Let's say that the time to compute $\gcd(N, M)$ is $k(\log(N))^3$ for some constant k. Assume $M_1, N_1 \approx 2^{500}$. Then the time to compute $\gcd(N_1, M_1)$ is $t_1 = k(\log(2^{500}))^3 = k(500\log(2))^3 = k \cdot 500^3 \cdot (\log(2))^3$. If $M_2, N_2 \approx 2^{1000}$ (so twice the length), then the time to compute $\gcd(N_2, M_2)$ is $t_2 = k(\log(2^{1000}))^3 = k \cdot 1000^3 \cdot (\log(2))^3 = k \cdot 2^3 \cdot 500^3 \cdot (\log(2))^3 = 8t_1$.

If the numbers are sufficiently small, like less than 32 bits in length, then the division takes a constant time depending on the size of the processor.

Problem 2: Find an upper bound for how long it takes to compute $B^{-1} (\mathrm{mod} M)$ with $B \leq M$. Solution: Example: $11^{-1} (\mathrm{mod}\, 26)$.

$$\begin{aligned} 26 &= 2 \cdot 11 + 4 \\ 11 &= 2 \cdot 4 + 3 \\ 4 &= 1 \cdot 3 + 1 \end{aligned}$$

$$\begin{aligned} 1 &= 4 - 1 \cdot 3 \\ &= 4 - 1(11 - 2 \cdot 4) = 3 \cdot 4 - 1 \cdot 11 \\ &= 3(26 - 2 \cdot 11) - 1 \cdot 11 = 3 \cdot 26 - 7 \cdot 11 \end{aligned}$$

So $11^{-1} \equiv -7 + 26 = 19 (\mathrm{mod}\, 26)$. Two parts: 1st: gcd, 2nd: write gcd as linear combo. gcd'ing takes $O(\log^3(M))$.

2nd part: $O(\log(M))$ steps (same as gcd). The worst step is $= 3(26 - 2 \cdot 11) - 1 \cdot 11 = 3 \cdot 26 - 7 \cdot 11$. First copy down 6 numbers $\leq M$. Takes time $6O(\log(M)) \stackrel{\text{rule 1}}{=} O(\log(M))$. Then simplification involves one multiplication $O(\log^2(M)$ and one addition of numbers $\leq M$, which takes time $O(\log(M))$. So the worst step takes time $O(\log(M)) + O(\log^2(M)) + O(\log(M)) \stackrel{\text{rule 3}}{=} O(\log^2(M))$. So writing the gcd as a linear combination has running time

(# steps)(time per step) $= O(\log(M))O(\log^2(M)) \stackrel{rule\,4}{=} O(\log^3(M))$. The total time for the modular inversion is the time to gcd and the time to write it as a linear combination which is $O(\log^3(M)) + O(\log^3(M)) \stackrel{rule\,1\,\text{or}\,3}{=} O(\log^3(M))$.

Problem 3: Assume $B, N \leq M$. Find an upper bound for how long it takes to reduce $B^N \bmod M$ using the repeated squares algorithm on a computer. Solution: There are $n = O(\log(N))$ steps.

Example. Compute $87^{43} \bmod 103$. $43 = (101011)_2 = (n_5 n_4 n_3 n_2 n_1 n_0)_2$.
Step 0. Start with $a = 1$. Since $n_0 = 1$, set $a = 87$.
Step 1. $87^2 \equiv 50$. Since $n_1 = 1$, set $a = 87 \cdot 50 \equiv 24 (\equiv 87^2 \cdot 87)$.
Step 2. $50^2 \equiv 28 (\equiv 87^4)$. Since $n_2 = 0$, $a = 24$.
Step 3. $28^2 \equiv 63 (\equiv 87^8)$. Since $n_3 = 1$, $a = 24 \cdot 63 \equiv 70 (\equiv 87^8 \cdot 87^2 \cdot 87)$.
Step 4. $63^2 \equiv 55 (\equiv 87^{16})$. Since $n_4 = 0$, $a = 70$.
Step 5. $55^2 \equiv 38 (\equiv 87^{32})$. Since $n_5 = 1$, $a = 70 \cdot 38 \equiv 85 (\equiv 87^{32} \cdot 87^8 \cdot 87^2 \cdot 87)$.

There's no obvious worst step, except that it should have $n_i = 1$. Let's consider the running time of a general step. Let S denote the current reduction of B^{2^i}. Note $0 \leq a < M$ and $0 \leq S < M$. For the step, we first multiply $S \cdot S$, $O(\log^2(M))$. Note $0 \leq S^2 < M^2$. Then we reduce $S^2 \bmod M$ ($S^2 \div M$), $O(\log(M^2)\log(M)) \stackrel{\text{rule}\,2}{=} O(\log(M)\log(M)) = O(\log^2(M))$. Let H be the reduction of $S^2 \bmod M$; note $0 \leq H < M$. Second we multiply $H \cdot a$, $O(\log^2(M))$. Note $0 \leq Ha < M^2$. Then reduce $Ha \bmod M$, $O(\log(M^2)\log(M)) = O(\log^2(M))$. So the time for a general step is $O(\log^2(M)) + O(\log^2(M)) + O(\log^2(M)) + O(\log^2(M)) = 4O(\log^2(M)) \stackrel{rule\,1}{=} O(\log^2(M))$.

The total running time for computing $B^N \bmod M$ using repeated squares is (# of steps)(time per step) $= O(\log(N)O(\log^2(M) \stackrel{\text{rule}\,4}{=} O(\log(N)\log^2(M))$. If $N \approx M$ then we simply say $O(\log^3(M))$. End Problem 3.

The running time to compute B^N is $O(N^i \log^j(B))$, for some $i, j \geq 1$ (to be determined in the homework) This is very slow.

Problem 4: Find an upper bound for how long it takes to compute $N!$ using $(((1 \cdot 2) \cdot 3) \cdot 4) \ldots$. Hint: $\log(A!) = O(A\log(A))$ (later).

Example: Let $N = 5$. So find 5!.

$$\begin{aligned} 1 \cdot 2 &= 2 \\ 2 \cdot 3 &= 6 \\ 6 \cdot 4 &= 24 \\ 24 \cdot 5 &= 120 \end{aligned}$$

There are $N-1$ steps, which we round up to N. The worst step is the last which is $[(N-1)!] \cdot N$, $O(\log((N-1)!)\log(N))$. From above we have $\log((N-1)!) \approx \log(N!) = O((N)\log(N))$ which we round up to $O(N\log(N))$. So the worst step takes time $O(N\log(N)\log(N)) = O(N\log^2 N)$.

Since there are about N steps, the total running time is (# steps)(time per step) $= O(N^2\log^2(N))$, which is very slow.

So why is $\log(A!) = O(A\log(A))$? Stirling's approximation says $A! \approx (A/e)^A\sqrt{2A\pi}$ (Stirling). Note $20! = 2.43 \cdot 10^{18}$ and $(20/e)^{20}\sqrt{2 \cdot 20 \cdot \pi} = 2.42 \cdot 10^{18}$. So $\log(A!) = A(\log(A)-$

$\log(e)) + \frac{1}{2}(\log(2) + \log(A) + \log(\pi))$. Thus $\log(A!) = O(A\log(A))$ (the other terms are smaller).
End Problem 4.

Say you have a cryptosystem with a key space of size N. You have a known plaintext/ciphertext pair. Then a brute force attack takes, on average $\frac{N}{2} = O(N)$ steps.

The running time to find a prime factor of N by trial division $(N/2, N/3, N/4, \ldots)$ is $O(\sqrt{N}\log^j(N))$ for some $j \geq 1$ (to be determined in the homework). This is very slow.

Say you have r integer inputs to an algorithm (i.e. r variables $N_1, \ldots, N_r$) (for multiplication: $r = 2$, factoring: $r = 1$, reduce $b^N (\mathrm{mod} M)$: $r = 3$). An algorithm is said to run in polynomial time in the lengths of the numbers (= number of bits) if the running time is $O(\log^{d_1}(N_1) \cdots \log^{d_r}(N_r))$. (All operations involved in encryption and decryption, namely gcd, addition, multiplication, division, repeated squares, inverse mod m, run in polynomial time).

If $n = O(\log(N))$ and $p(n)$ is an increasing polynomial, then an algorithm that runs in time $c^{p(n)}$ for some constant $c > 1$ is said to run in exponential time (in the length of N). This includes trial division and brute force.

Trial division: The $\log^j(N))$ is so insignificant, that people usually just say the running time is $O(\sqrt{N}) = O(N^{1/2}) = O((c^{logN})^{1/2}) = O(c^{logN/2}) = O(c^{n/2})$. Since $\frac{1}{2}n$ is a polynomial in n, this takes exponential time. The running times of computing B^N and $N!$ are also exponential. For AES, the input N is the size of the key space $N = 2^{128}$ and the running time is $\frac{1}{2}N = O(N) = c^{\log(N)}$. The running time to solve the discrete logarithm problem for an elliptic curve over a finite field $\mathbf{F}_q$ is $O(\sqrt{q})$, which is exponential, like trial division factoring.

There is a way to interpolate between polynomial time and exponential time. Let $0 < \alpha < 1$ and $c > 1$. Then $L_N(\alpha, c) = O(c^{(\log^\alpha(N)\mathrm{loglog}^{1-\alpha}(N))}$. Note if $\alpha = 0$ we get $O(c^{\mathrm{loglog}(N)}) = O(\log N)$ is polynomial. If $\alpha = 1$ we get $O(c^{\log N})$ is exponential. If the running time is $L_N(\alpha, c)$ for $0 < \alpha < 1$ then it is said to be subexponential. The running time to factor N using the Number Field Sieve is $L_N(\frac{1}{3}, c)$ for some c. So this is much slower than polynomial but faster than exponential.

The current running time for finding a factor of N using the number field sieve is $L_N(\frac{1}{3}, c)$ for some c. This which is much slower than polynomial but faster than exponential. Factoring a 20 digit number using trial division would take longer than the age of the universe. In 1999, a 155-digit RSA challenge number was factored. In January 2010, a 232 digit (768 bit) RSA challenge number was factored. The number field sieve has been adapted to solving the finite field discrete logarithm problem in $\mathbf{F}_q$. So the running time is also $L_q(\frac{1}{3}, c)$.

The set of problems whose solutions have polynomial time algorithms is called P. There's a large set of problems for which no known polynomial time algorithm exists for solving them (though you can check that a given solution is correct in polynomial time) called NP. Many of the solutions differ from each other by polynomial time algorithms. So if you could solve one in polynomial time, you could solve them all in polynomial time. It is known that, in terms of running times, P$\leq$ NP $\leq$ exponential.

One NP problem: find simultaneous solutions to a system of non-linear polynomial equations mod 2. Like $x_1x_2x_5 + x_4x_3 + x_7 \equiv 0(\mathrm{mod}2)$, $x_1x_9 + x_2 + x_4 \equiv 1(\mathrm{mod}2)$, ... If you

could solve this problem quickly you could crack AES quickly. This would be a lone plaintext attack and an x_i would be the ith bit of the key.

Cryptography

In this section we will introduce the major methods of encryption, hashing and signatures.

6 Simple Cryptosystems

Let $\mathcal{P}$ be the set of possible plaintext messages. For example it might be the set { A, B,. . . ,Z } of size 26 or the set { AA, AB, . . . ,ZZ } of size 26^2. Let $\mathcal{C}$ be the set of possible ciphertext messages.

An enciphering transformation f is a map from $\mathcal{P}$ to $\mathcal{C}$. f shouldn't send different plaintext messages to the same ciphertext message (so f should be one-to-one, or injective). We have $\mathcal{P} \stackrel{f}{\to} \mathcal{C}$ and $\mathcal{C} \stackrel{f^{-1}}{\to} \mathcal{P}$; together they form a cryptosystem. Here are some simple ones.

We'll start with a cryptosystem based on single letters. You can replace letters by other letters. Having a weird permutation is slow, like A→ F, B→ Q, C→ N,. . .. There's less storage if you have a mathematical rule to govern encryption and decryption.

Shift transformation: P is plaintext letter/number A=0, B=1, . . . , Z=25. The Caesar cipher is an example: Encryption is given by $C \equiv P + 3(\text{mod}26)$ and so decryption is given by $P \equiv C - 3(\text{mod}26)$. This is the Caesar cipher. If you have an N letter alphabet, a shift enciphering transformation is $\text{C} \equiv \text{P} + b(\text{mod}N)$ where b is the encrypting key and $-b$ is the decrypting key.

For cryptanalysis, the enemy needs to know it's a shift transformation and needs to find b. In general one must assume that the nature of the cryptosystem is known (here a shift).

Say you intercept a lot of CT and want to find b so you can decrypt future messages. Methods: 1) Try all 26 possible b's. Probably only one will give sensible PT. 2) Use frequency analysis. You know $E = 4$ is the most common letter in English. You have a lot of CT and notice that $J = 9$ is the most common letter in the CT so you try $b = 5$.

An affine enciphering transformation is of the form $C \equiv aP + b(\text{mod}N)$ where the pair (a, b) is the encrypting key. You need $\gcd(a, N) = 1$ or else different PT's will encrypt as the same CT (as there are $N/\gcd(a, N)$ possible aP's).

Example: $C \equiv 4P + 5(\text{mod}\, 26)$. Note $B = 1$ and $O = 14$ go to $9 = J$.

Example: $C \equiv 3P + 4(\text{mod}, 26)$ is OK since $\gcd(3, 26) = 1$. Alice sends the message U to Bob. $U = 20$ goes to $3 \cdot 20 + 4 = 64 \equiv 12(\text{mod}\, 26)$. So U= 20→12 =M (that was encode, encrypt, decode). Alice sends M to Bob. Bob can decrypt by solving for P. $C - 4 \equiv 3P(\text{mod}\, 26)$. $3^{-1}(C - 4) \equiv P(\text{mod}\, 26)$. $3^{-1} \equiv 9\text{mod}\, 26)$ (since $3 \cdot 9 = 27 \equiv 1(\text{mod}\, 26)$). $P \equiv 9(C - 4) \equiv 9C - 36 \equiv 9C + 16(\text{mod}\, 26)$. So $P \equiv 9C + 16(\text{mod}\, 26)$. Since Bob received M= 12 he then computes $9 \cdot 12 + 16 = 124 \equiv 20(\text{mod}\, 26)$.

In general encryption: $C \equiv aP + b(\text{mod}N)$ and decryption: $P \equiv a^{-1}(C - b)(\text{mod}N)$. Here $(a^{-1}, -a^{-1}b)$ is the decryption key.

How to cryptanalyze. We have $N = 26$. You could try all $\phi(26) \cdot 26 = 312$ possible key pairs (a, b) or do frequency analysis. Have two unknown keys so you need two equations.

Assume you are the enemy and you have a lot of CT. You find $Y = 24$ is the most common and $H = 7$ is the second most common. In English, $E = 4$ is the most common and $T = 19$ is the second most common. Let's say that decryption is by $P \equiv a'C + b'(\text{mod}26)$ (where $a' = a^{-1}$ and $b' = -a^{-1}b$). Decrypt $HFOGLH$.

First we find (a', b'). We assume $4 \equiv a'24 + b'(\text{mod}26)$ and $19 \equiv a'7 + b'(\text{mod}26)$. Subtracting we get $17a' \equiv 4 - 19 \equiv 4 + 7 \equiv 11(\text{mod}26)$ $(*)$. So $a' \equiv 17^{-1}11(\text{mod}26)$. We can use the Euclidean algorithm to find $17^{-1} \equiv 23(\text{mod}26)$ so $a' \equiv 23 \cdot 11 \equiv 19(\text{mod}26)$. Plugging this into an earlier equation we see $19 \equiv 19 \cdot 7 + b'(\text{mod}26)$ and so $b' \equiv 16(\text{mod}26)$. Thus $P \equiv 19C + 16(\text{mod}26)$.

Now we decrypt $HFOGLH$ or 7 5 14 6 11 7. We get $19 \cdot 7 + 16 \equiv 19 = T$, $19 \cdot 5 + 16 \equiv 7 = H$,... and get the word THWART. Back at $(*)$, it is possible that you get an equation like $2a' \equiv 8(\text{mod}26)$. The solutions are $a' \equiv 4(\text{mod}13)$ which is $a' \equiv 4$ or $17(\text{mod}26)$. So you would need to try both and see which gives sensible PT.

Let's say we want to impersonate the sender and send the message DONT i.e. 3 14 13 19. We want to encrypt this so we use $C \equiv aP + b(\text{mod}26)$. We have $P \equiv 19C + 16(\text{mod}26)$ so $C \equiv 19^{-1}(P - 16) \equiv 11P + 6(\text{mod}26)$.

We could use an affine enciphering transformation to send digraphs (pairs of letters). If we use the alphabet A - Z which we number 0 - 25 then we can encode a digraph xy as $26x + y$. The resulting number will be between 0 and $675 = 26^2 - 1$. Example: TO would become $26 \cdot 19 + 14 = 508$. To decode, compute $508 \div 26 = 19.54$, then $-19 = .54$, then $\times 26 = 14$. We would then encrypt by $C \equiv aP + b(\text{mod}626)$.

7 Symmetric key cryptography

In symmetric key cryptosystem, Alice and Bob must agree on a secret, shared key ahead of time. We will consider stream ciphers and block ciphers.

8 Finite fields

If p is a prime we rename $\mathbf{Z}/p\mathbf{Z} = \mathbf{F}_p$, the field with p elements $= \{0, 1, \ldots, p-1\}$ with $+, -, \times$. Note all elements α other than 0 have $\gcd(\alpha, p) = 1$ so we can find $\alpha^{-1}(\text{mod}p)$. So we can divide by any non-0 element. So it's like other fields like the rationals, reals and complex numbers.

$\mathbf{F}_p^*$ is $\{1, \ldots, p-1\}$ here we do $\times, \div$. Note $\mathbf{F}_p^*$ has $\phi(p-1)$ generators g (also called primitive roots of p). The sets $\{g, g^2, g^3, \ldots, g^{p-1}\}$ and $\{1, 2, \ldots, p-1\}$ are the same (though the elements will be in different orders).

Example, $\mathbf{F}_5^*$, $g = 2$: $2^1 = 2$, $2^2 = 4$, $2^3 = 3$, $2^4 = 1$. Also $g = 3$: $3^1 = 3$, $3^2 = 4$, $3^3 = 2$, $3^4 = 1$. For $\mathbf{F}_7^*$, $2^1 = 2$, $2^2 = 4$, $2^3 = 1$, $2^4 = 2$, $2^5 = 4$, $2^6 = 1$, so 2 is not a generator. $g = 3$: $3^1 = 3$, $3^2 = 2$, $3^3 = 6$, $3^4 = 4$, $3^5 = 5$, $3^6 = 1$.

9 Finite Fields Part II

Here is a different kind of finite field. Let $\mathbf{F}_2[x]$ be the set of polynomials with coefficients in $\mathbf{F}_2 = \mathbf{Z}/2\mathbf{Z} = \{0, 1\}$. Recall $-1 = 1$ here so $- = +$. The polynomials are

$$0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1, \ldots$$

There are two of degree 0 (0,1), four of degree ≤ 1, eight of degree ≤ 2 and in general the number of polynomials of degree $\leq n$ is 2^{n+1}. They are are $a_n x^n + \ldots + a_0$, $a_i \in \{0, 1\}$. Let's multiply:

```
      x^2 +   x + 1
            x^2 + x
      -------------
      x^3 + x^2 + x
x^4 + x^3 + x^2
------------------
x^4             + x
```

A polynomial is irreducible over a field if it can't be factored into polynomials with coefficients in that field. Over the rationals (fractions of integers), $x^2 + 2$, $x^2 - 2$ are both irreducible. Over the reals, $x^2 + 2$ is irreducible and $x^2 - 2 = (x + \sqrt{2})(x - \sqrt{2})$ is reducible. Over the complex numbers $x^2 + 2 = (x + \sqrt{2}i)(x - \sqrt{2}i)$, so both are reducible.

$x^2 + x + 1$ is irreducible over $\mathbf{F}_2$ (it's the only irreducible quadratic). $x^2 + 1 = (x + 1)^2$ is reducible. $x^3 + x + 1$, $x^3 + x^2 + 1$ are the only irreducible cubics over $\mathbf{F}_2$.

When you take $\mathbf{Z}$ and reduce mod p a prime (an irreducible number) you get $0, \ldots, p-1$, that's the stuff less than p. In addition, $p = 0$ and everything else can be inverted. You can write this set as $\mathbf{Z}/p\mathbf{Z}$ or $\mathbf{Z}/(p)$.

Now take $\mathbf{F}_2[x]$ and reduce mod $x^3 + x + 1$ (irreducible). You get polynomials of lower degree and $x^3 + x + 1 = 0$, i.e. $x^3 = x + 1$. $\mathbf{F}_2[x]/(x^3 + x + 1) = \{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}$ with the usual $+, (-), \times$ and $x^3 = x+1$. Let's multiply in $\mathbf{F}_2[x]/(x^3+x+1)$.

```
      x^2 + x + 1
            x + 1
      -----------
      x^2 + x + 1
x^3 + x^2 + x
-----------------
x^3           + 1
```

But $x^3 = x + 1$ so $x^3 + 1 \equiv (x + 1) + 1 (\mathrm{mod} x^3 + x + 1)$ and $x^3 + 1 \equiv x (\mathrm{mod} x^3 + x + 1)$. So $(x^2 + x + 1)(x + 1) = x$ in $\mathbf{F}_2[x]/(x^3 + x + 1)$. This is called $\mathbf{F}_8$ since it has 8 elements. Notice $x^4 = x^3 \cdot x = (x + 1)x = x^2 + x$ in $\mathbf{F}_8$.

The set $\mathbf{F}_2[x]/$(irreducible polynomial of degree d) is a field called $\mathbf{F}_{2^d}$ with 2^d elements. It consists of the polynomials of degree $\leq d-1$. $\mathbf{F}_{2^d}^*$ is the non-0 elements and has $\phi(2^d - 1)$

generators. x is a generator for $\mathbf{F}_8^*$ described above. $g = x$, x^2, $x^3 = x + 1$, $x^4 = x^2 + x$, $x^5 = x^4 \cdot x = x^3 + x^2 = x^2 + x + 1$, $x^6 = x^3 + x^2 + x = x^2 + 1$, $x^7 = x^3 + x = 1$.

You can represent elements easily in a computer. You could represent $1 \cdot x^2 + 0 \cdot x + 1$ by 101. For this reason, people usually use discrete log cryptosystems with fields of the type $\mathbf{F}_{2^d}$ instead of the type $\mathbf{F}_p$ where $p \approx 2^d \approx 10^{300}$. Over $\mathbf{F}_p$ they are more secure; over $\mathbf{F}_{2^d}$ they are easier to implement on a computer.

In $\mathbf{F}_2[x]/(x^6 + x + 1)$ invert $x^4 + x^3 + 1$. Use the polynomial Euclidean algorithm. $x^6 + x + 1 = q(x^4 + x^3 + 1) + r(x)$ where the degree of $r(x)$ is less than the degree of $x^4 + x^3 + 1$.

```
                                      x^2 + x + 1 = q(x)
                  ------------------------------------
x^4+x^3      +1 | x^6                          + x + 1
                  x^6 + x^5             + x^2
                  --------------------------
                       x^5              + x^2 + x
                       x^5 + x^4              + x
                       ------------------------
                             x^4        + x^2     + 1
                             x^4 + x^3            + 1
                             ----------------------
                                  x^3 + x^2          = r(x)
```

So $\underline{x^6 + x + 1} = (x^2 + x + 1)(\underline{x^4 + x^3 + 1}) + (\underline{x^3 + x^2})$.
Similarly $\underline{x^4 + x^3 + 1} = x(\underline{x^3 + x^2}) + \underline{1}$.
So $1 = (\underline{x^4 + x^3 + 1}) + x(\underline{x^3 + x^2})$
$1 = (\underline{x^4 + x^3 + 1}) + x(\underline{x^6 + x + 1} + (x^2 + x + 1)(\underline{x^4 + x^3 + 1}))$
$1 = 1(x^4 + x^3 + 1) + x(\underline{x^6 + x + 1}) + (x^3 + x^2 + x)(\underline{x^4 + x^3 + 1})$
$1 = (x^3 + x^2 + x + 1)(\underline{x^4 + x^3 + 1}) + x(\underline{x^6 + x + 1})$
$1 = (x^3 + x^2 + x + 1)(x^4 + x^3 + 1) (\mathrm{mod}\, x^6 + x + 1)$.
So $(x^4 + x^3 + 1)^{-1} = x^3 + x^2 + x + 1$ in $\mathbf{F}_2[x]/(x^6 + x + 1) = \mathbf{F}_{64}$. End example.

In $\mathbf{F}_8$ described above, you are working in $\mathbf{Z}[x]$ with two mod's: coefficients are mod 2 and polynomials are mod $x^3 + x + 1$. Note that if $d > 1$ then $\mathbf{F}_{2^d} \neq \mathbf{Z}/2^d\mathbf{Z}$ (in $\mathbf{F}_8$, $1 + 1 = 0$ in $\mathbf{Z}/8\mathbf{Z}$, $1 + 1 = 2$). In much of the cryptography literature, they use $GF(q)$ to denote both $\mathbf{F}_q$ and $\mathbf{F}_q^*$, where q is usually prime or 2^d.

10 Modern stream ciphers

Modern stream ciphers are symmetric key cryptosystems. So Alice and Bob must agree on a key beforehand. The plaintext is turned into ASCII. So the plaintext Go would be encoded as 0100011101101111. There's a given (pseudo)random bit generator. Alice and Bob agree on a seed, which acts as the symmetric/shared/secret key. They both generate the same random bit stream like 0111110110001101, which we call the keystream. Alice gets

the ciphertext by bit-by-bit XOR'ing, i.e. bit-by-bit addition mod 2. $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$.

		PT	0100011101101111	CT	0011101011100010
Example.	keystream $\oplus$	0111110110001101	keystream $\oplus$	0111110110001101	
		CT	0011101011100010	0100011101101111	
				Go	

Let p_i be the ith bit of plaintext, k_i be the ith bit of keystream and c_i be the ith bit of ciphertext. Here $c_i = p_i \oplus k_i$ and $p_i = c_i \oplus k_i$. (See earlier example.)

Here is an unsafe stream cipher used on PC's to encrypt files (savvy users aware it gives minimal protection). Use keyword like Sue 01010011 01110101 01100101. The keystream is that string repeated again and again. At least there's variable key length.

Here is a random bit generator that is somewhat slow, so it is no longer used. Say p is a large prime for which 2 generates $\mathbf{F}_p^*$ and assume $q = 2p+1$ is also prime. Let g generate $\mathbf{F}_q^*$. Say the key is k with $\gcd(k, 2p) = 1$. Let $s_1 = g^k \in \mathbf{F}_q$. (so $1 \leq s_1 < q$) and $k_1 \equiv s_1 (\mathrm{mod}\, 2)$ with $k_1 \in \{0, 1\}$. For $i \geq 1$, let $s_{i+1} = s_i^2 \in \mathbf{F}_q$ with $1 \leq s_i < q$ and $k_i \equiv s_i (\mathrm{mod}\, 2)$ with $k_i \in \{0, 1\}$. It will start cycling because $s_{p+1} = s_2$.

Example. 2 generates $\mathbf{F}_{29}^*$. $2^{28/7} \neq 1$). $g = 2$ also generates $\mathbf{F}_{59}^*$. Let $k = 11$. Then $s_1 = 2^{11} = 42$, $s_2 = 42^2 = 53$, $s_3 = 53^2 = 36$, $s_4 = 36^2 = 57$, ... so $k_1 = 0$, $k_2 = 1$, $k_3 = 0$, $k_4 = 1$, ...

10.1 RC4

RC4 is the most widely used stream cipher. Invented by Ron Rivest (R of RSA) in 1987. The RC stands for *Ron's code*. The pseudo random bit generator was kept secret. The source code was published anonymously on Cypherpunks mailing list in 1994.

Choose n, a positive integer. Right now, people use $n = 8$. Let $l = \lceil (\text{length of PT in bits}/n) \rceil$.

There is a key array $K_0, \ldots, K_{2^n-1}$ whose entries are n-bit strings (which will be thought of as integers from 0 to $2^n - 1$). You enter the key into that array and then repeat the key as necessary to fill the array.

The algorithm consists of permuting the integers from 0 to $2^n - 1$. The permutations are stored in an array $S_0, \ldots, S_{2^n-1}$. Initially we have $S_0 = 0, \ldots, S_{2^n-1} = 2^n - 1$.

Here is the algorithm.

$j = 0$.
For $i = 0, \ldots, 2^n - 1$ do:
  $j := j + S_i + K_i (\mathrm{mod}\, 2^n)$.
  Swap S_i and S_j.
End For
Set the two counters i, j back to zero.
To generate l random n-bit strings, do:
For $r = 0, \ldots, l - 1$ do
  $i := i + 1 (\mathrm{mod}\, 2^n)$.
  $j := j + S_i (\mathrm{mod}\, 2^n)$.
  Swap S_i and S_j.

$t := S_i + S_j (\mathrm{mod}\, 2^n)$.
$KS_r := S_t$.
End For
Then $KS_0KS_1KS_2\ldots$, written in binary, is the keystream.

Do example with $n = 3$.

Say key is 011001100001101 or 011 001 100 001 101 or $[3, 1, 4, 1, 5]$. We expand to $[3, 1, 4, 1, 5, 3, 1, 4] = [K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7]$.

i	j	t	KS_r	S_0	S_1	S_2	S_3	S_4	S_5	S_6	S_7
	0			0	1	2	3	4	5	6	7
0	3			3	1	2	0	4	5	6	7
1	5			3	5	2	0	4	1	6	7
2	3			3	5	0	2	4	1	6	7
3	6			3	5	0	6	4	1	2	7
4	7			3	5	0	6	7	1	2	4
5	3			3	5	0	1	7	6	2	4
6	6			3	5	0	1	7	6	2	4
7	6			3	5	0	1	7	6	4	2
0	0										
1	5	3	1	3	6	0	1	7	5	4	2
2	5	5	0	3	6	5	1	7	0	4	2
3	6	5	0	3	6	5	4	7	0	1	2
4	5	7	2	3	6	5	4	0	7	1	2
5	4	7	2	3	6	5	4	7	0	1	2
6	5	1	6	3	6	5	4	7	1	0	2
7	7	4	7	3	6	5	4	7	1	0	2
0	2	0	5	5	6	3	4	7	1	0	2
1	0	3	4	6	5	3	4	7	1	0	2
2	3	7	2	6	5	4	3	7	1	0	2
3	6	3	0	6	5	4	0	7	1	3	2
4	5	0	6	6	5	4	0	1	7	3	2

The keystream is from the 3-bit representations of $1, 0, 0, 2, 2, 6, 7, 5, 4, 2, 0, 6$, which is 001 000 000 010 010 110 111 101 100 010 000 110 (without spaces).

The index i ensures that every element changes and j ensures that the elements change randomly. Interchanging indices and values of the array gives security.

10.2 Self-synchronizing stream cipher

When you simply XOR the plaintext with the keystream to get the ciphertext, that is called a synchronous stream cipher. Now Eve might get a hold of matched PT/CT strings and find part of the keystream and somehow find the whole keystream. There can be mathematical methods to do this. Also, if Alice accidentally uses the same keystream twice, with two different plaintexts, then Eve can XOR the two ciphertexts together and get the XOR of the two plaintexts, which can be teased apart. One solution is to use old plaintext to encrypt also. This is called a self-synchronizing stream cipher. (I made this one up).

Example. The first real bit of plaintext is denoted p_1.

$$c_i = p_i \oplus k_i \oplus \begin{cases} p_{i-2} \text{ if } p_{i-1} = 0 \\ p_{i-3} \text{ if } p_{i-1} = 1 \end{cases}$$

Need to add $p_{-1} = p_0 = 0$ to the beginning of the plaintext. The receiver uses

$$p_i = c_i \oplus k_i \oplus \begin{cases} p_{i-2} \text{ if } p_{i-1} = 0 \\ p_{i-3} \text{ if } p_{i-1} = 1 \end{cases}$$

Using the plaintext (Go) and keystream from an earlier example, we would have:

```
sender:                                  receiver:
PT          000100011101101111           CT 0010101000001111
keystream     0111110110001101   keystream 0111110110001101
              ----------------             ----------------
CT            0010101000001111        PT  000100011101101111 (Go)
```

One problem with self-synchronizing is that it is prone to error propogation if there are errors in transmission.

10.3 One-time pads

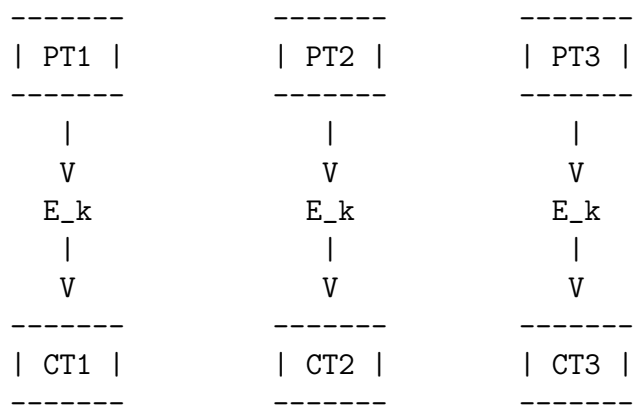
Let's say that each bit of the keystream is truly randomly generated. That implies means that each bit is independent of the previous bits. So you don't start with a seed/key that is short and generate a keystream from it. Ex. Flip a coin. OK if it's not fair (none are). Look at each pair of tosses, if HT write 1, if TH, write 0, if TT or HH, don't write. So HH TH TT HH TH HH HT becomes 001... End ex. This is called a one-time-pad. The keystream must never be used again. Cryptanalysis is provably impossible. This was used by Russians during the cold war and by the phone linking the White House and the Kremlin. It is very impractical.

11 Modern Block Ciphers

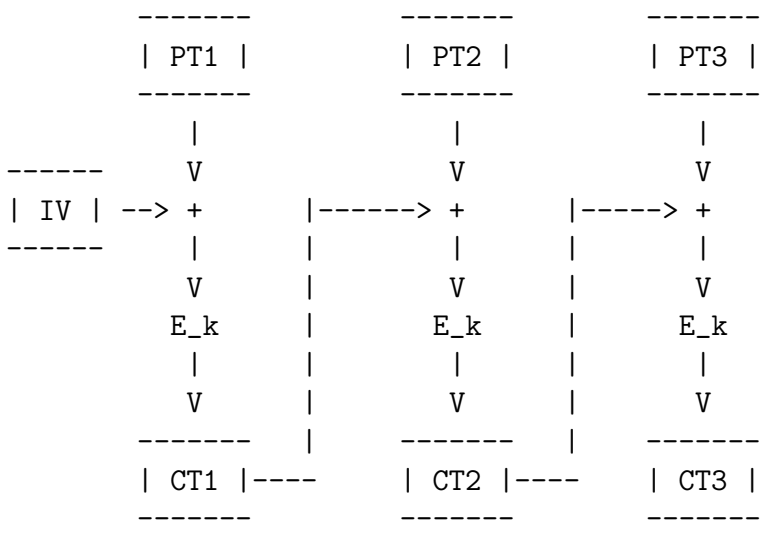
Most encryption now is done using block ciphers. The two most important historically have been the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES). DES has a 56 bit key and 64 bit plaintext and ciphertext blocks. AES has a 128 bit key, and 128 bit plaintext and ciphertext blocks.

11.1 Modes of Operation of a Block Cipher

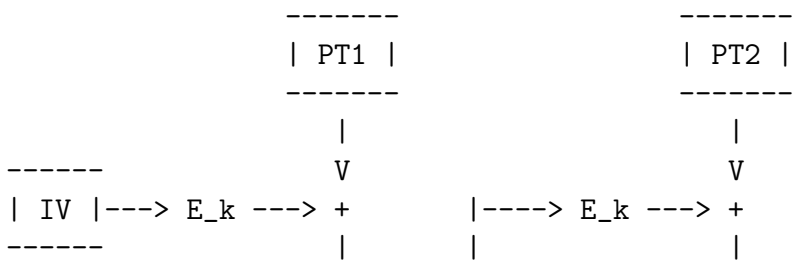
On a chip for a block cipher, there are four modes of operation. The standard mode is the *electronic code book* (ECB) mode. It is the most straightforward but has the disadvantage that for a given key, two identical plaintexts will correspond to identical ciphertexts. If the number of bits in the plaintext message is not a multiple of the block length, then add extra bits at the end until it is. This is called padding.

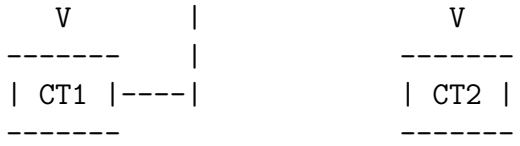


The next mode is the *cipherblock chaining* (CBC) mode. This is the most commonly used mode. Alice and Bob must agree on a non-secret initialization vector (IV) which has the same length as the plaintext. The IV may or may not be secret.

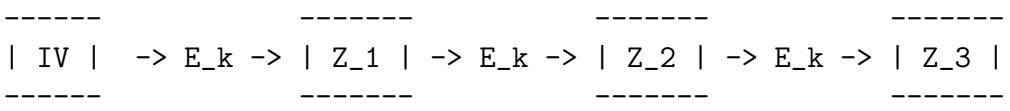


The next mode is the *cipher feedback* (CFB) mode. If the plaintext is coming in slowly, the ciphertext can be sent as soon as as the plaintext comes in. With the CBC mode, one must wait for a whole plaintext block before computing the ciphertext. This is also a good mode of you do not want to pad the plaintext.





The last mode is the *output feedback* (OFB) mode. It is a way to create a keystream for a stream cipher. Below is how you create the keystream.



The keystream is the concatenation of $Z_1Z_2Z_3\ldots$. As usual, this will be XORed with the plaintext. (In the diagram you can add PT_i's, CT_i's and $\oplus$'s.)

11.2 The Block Cipher DES

The U.S. government in the early 1970's wanted an encryption process on a small chip that would be widely used and safe. In 1975 they accepted IBM's Data Encryption Standard Algorithm (DES). DES is a symmetric-key cryptosystem which has a 56-bit key and encrypts 64-bit plaintexts to 64-bit ciphertexts. By the early 1990's, the 56-bit key was considered too short. Surprisingly, Double-DES with two different keys is not much safer than DES, as is explained in Section 30.3. So people started using Triple-DES with two 56 bit keys. Let's say that E_K and D_K denote encryption and decryption with key K using DES. Then Triple-DES with keys K_1 and K_2 is $CT = E_{K_1}(D_{K_2}(E_{K_1}(PT)))$. The reason there is a D_K in the middle is for backward compatibility. Note that Triple-DES using a single key each time is the same thing as Single-DES with the same key. So if one person has a Triple-DES chip and the other a Single-DES chip, they can still communicate privately using Single-DES.

In 1997 DES was brute forced in 24 hours by 500000 computers. In 2008, ATMs worldwide still used Single-DES because ATMs started using Single-DES chips and they all need to communicate with each other and it was too costly in some places to update to a more secure chip.

11.3 The Block Cipher AES

Introduction

However, DES was not designed with Triple-DES in mind. Undoubtedly there would be a more efficient algorithm with the same level of safety as Triple-DES. So in 1997, the National Institute of Standards and Technology (NIST) solicited proposals for replacements of DES. In 2001, NIST chose 128-bit block Rijndael with a 128-bit key to become the Advanced Encryption Standard (AES). (If you don't speak Dutch, Flemish or Afrikaans, then the closest approximation to the pronunciation is Rine-doll). Rijndael is a symmetric-key block cipher designed by Joan Daemen and Vincent Rijmen.

Simplified AES

Simplified AES was designed by Mohammad Musa, Steve Wedig (two former Crypto students) and me in 2002. It is a method of teaching AES. We published the article *A simplified AES algorithm and its linear and differential cryptanalyses* in the journal Cryptologia in 2003. We will learn the linear and differential cryptanalyses in the Cryptanalysis Course.

The Finite Field

Both the key expansion and encryption algorithms of simplified AES depend on an S-box that itself depends on the finite field with 16 elements.

Let $\mathbf{F}_{16} = \mathbf{F}_2[x]/(x^4 + x + 1)$. The word nibble refers to a four-bit string, like 1011. We will frequently associate an element $b_0x^3 + b_1x^2 + b_2x + b_3$ of $\mathbf{F}_{16}$ with the nibble $b_0b_1b_2b_3$.

The S-box

The S-box is a map from nibbles to nibbles. It can be inverted. (For those in the know, it is one-to-one and onto or bijective.) Here is how it operates. As an example, we'll find S-box(0011). First, invert the nibble in $\mathbf{F}_{16}$. The inverse of $x + 1$ is $x^3 + x^2 + x$ so 0011 goes to 1110. The nibble 0000 is not invertible, so at this step it is sent to itself. Then associate to the nibble $N = b_0b_1b_2b_3$ (which is the output of the inversion) the element $N(y) = b_0y^3 + b_1y^2 + b_2y + b_3$ in $\mathbf{F}_2[y]/(y^4 + 1)$. Doing multiplication and addition is similar to doing so in $\mathbf{F}_{16}$ except that we are working modulo $y^4 + 1$ so $y^4 = 1$ and $y^5 = y$ and $y^6 = y^2$. Let $a(y) = y^3 + y^2 + 1$ and $b(y) = y^3 + 1$ in $\mathbf{F}_2[y]/(y^4 + 1)$. The second step of the S-box is to send the nibble $N(y)$ to $a(y)N(y) + b(y)$. So the nibble $1110 = y^3 + y^2 + y$ goes to $(y^3+y^2+1)(y^3+y^2+y)+(y^3+1) = (y^6+y^5+y^4)+(y^5+y^4+y^3)+(y^3+y^2+y)+(y^3+1)$ $= y^2 + y + 1 + y + 1 + y^3 + y^3 + y^2 + y + y^3 + 1 = 3y^3 + 2y^2 + 3y + 3 = y^3 + y + 1 = 1011$. So S-box(0011) = 1011.

Note that $y^4 + 1 = (y + 1)^4$ is reducible over $\mathbf{F}_2$ so $\mathbf{F}_2[y]/(y^4 + 1)$ is not a field and not all of its non-zero elements are invertible; the polynomial $a(y)$, however, is. So $N(y) \mapsto a(y)N(y)+b(y)$ is an invertible map. If you read the literature, then the second step is often described by an affine matrix map.

We can represent the action of the S-box in two ways (note we do not show the intermediary output of the inversion in $\mathbf{F}_{16}^*$). These are called look up tables.

nib	S–box(nib)	nib	S–box(nib)
0000	1001	1000	0110
0001	0100	1001	0010
0010	1010	1010	0000
0011	1011	1011	0011
0100	1101	1100	1100
0101	0001	1101	1110
0110	1000	1110	1111
0111	0101	1111	0111

or $\begin{bmatrix} 9 & 4 & 10 & 11 \\ 13 & 1 & 8 & 5 \\ 6 & 2 & 0 & 3 \\ 12 & 14 & 15 & 7 \end{bmatrix}$.

The left-hand side is most useful for doing an example by hand. For the matrix on the right, we start in the upper left corner and go across, then to the next row and go across etc. The integers 0 - 15 are associated with their 4-bit binary representations. So 0000 = 0 goes to 9 = 1001, 0001 = 1 goes to 4 = 0100, ... , 0100 = 4 goes to 13 = 1101, etc.

Inversion in the finite field is the only non-linear operation in SAES. Note multiplication by a fixed polynomial, $\oplus$ing bits and shifting bits are all linear operations. If all parts of SAES were linear then could use linear algebra on a matched PT/CT pair to easily solve for the key.

Keys

For our simplified version of AES, we have a 16-bit key, which we denote $k_0 \dots k_{15}$. That needs to be expanded to a total of 48 key bits $k_0 \dots k_{47}$, where the first 16 key bits are the same as the original key. Let us describe the expansion. Let $\mathrm{RC}[i] = x^{i+2} \in \mathbf{F}_{16}$. So $\mathrm{RC}[1] = x^3 = 1000$ and $\mathrm{RC}[2] = x^4 = x + 1 = 0011$. If N_0 and N_1 are nibbles, then we denote their concatenation by N_0N_1. Let $\mathrm{RCON}[i] = \mathrm{RC}[i]0000$ (this is a byte, a string of 8 bits). So $\mathrm{RCON}[1] = 10000000$ and $\mathrm{RCON}[2] = 00110000$. These are abbreviations for *round constant*. We define the function RotNib to be $\mathrm{RotNib}(N_0N_1) = N_1N_0$ and the function SubNib to be $\mathrm{SubNib}(N_0N_1) =$S-box$(N_0)$S-box$(N_1)$; these are functions from bytes to bytes. Their names are abbreviations for *rotate nibble* and *substitute nibble*. Let us define an array (vector, if you prefer) W whose entries are bytes. The original key fills $W[0]$ and $W[1]$ in order. For $2 \le i \le 5$,

$$\begin{array}{ll} \text{if } i \equiv 0 (\mathrm{mod}\, 2) & \text{then } W[i] = W[i-2] \oplus \mathrm{RCON}(i/2) \oplus \mathrm{SubNib}(\mathrm{RotNib}(W[i-1])) \\ \text{if } i \not\equiv 0 (\mathrm{mod}\, 2) & \text{then } W[i] = W[i-2] \oplus W[i-1] \end{array} .$$

The bits contained in the entries of W can be denoted $k_0 \dots k_{47}$. For $0 \le i \le 2$ we let $K_i = W[2i]W[2i+1]$. So $K_0 = k_0 \dots k_{15}$, $K_1 = k_{16} \dots k_{31}$ and $K_2 = k_{32} \dots k_{47}$. For $i \ge 1$, K_i is the round key used at the end of the i-th round; K_0 is used before the first round.

Recall $\oplus$ denotes bit-by-bit XORing.

Key Expansion Example

Let's say that the key is 0101 1001 0111 1010. So $W[0] =$ 0101 1001 and $W[1] =$ 0111 1010. Now $i = 2$ so we Rotnib($W[1]$)=1010 0111. Then we SubNib(1010 0111)=0000 0101. Then we XOR this with $W[0] \oplus \mathrm{RCON}(1)$ and get $W[2]$.

$$\begin{array}{rcc} & 0000 & 0101 \\ & 0101 & 1001 \\ \oplus & \underline{1000} & \underline{0000} \\ & 1101 & 1100 \end{array}$$

So $W[2] = 11011100$.

Now $i = 3$ so $W[3] = W[1] \oplus W[2] =$ 0111 1010 $\oplus$ 1101 1100 = 1010 0110. Now $i = 4$ so we Rotnib($W[3]$)=0110 1010. Then we SubNib(0110 1010)=1000 0000. Then we XOR this with $W[2] \oplus \mathrm{RCON}(2)$ and get $W[4]$.

$$\begin{array}{rcc} & 1000 & 0000 \\ & 1101 & 1100 \\ \oplus & \underline{0011} & \underline{0000} \\ & 0110 & 1100 \end{array}$$

So $W[4] = 01101100$.

Now $i = 5$ so $W[5] = W[3] \oplus W[4] = 1010\,0110 \oplus 0110\,1100 = 1100\,1010$.

The Simplified AES Algorithm

The simplified AES algorithm operates on 16-bit plaintexts and generates 16-bit ciphertexts, using the expanded key $k_0 \ldots k_{47}$. The encryption algorithm consists of the composition of 8 functions applied to the plaintext: $A_{K_2} \circ SR \circ NS \circ A_{K_1} \circ MC \circ SR \circ NS \circ A_{K_0}$ (so A_{K_0} is applied first), which will be described below. Each function operates on a state. A state consists of 4 nibbles configured as in Figure 1. The initial state consists of the plaintext as in Figure 2. The final state consists of the ciphertext as in Figure 3.

$b_0b_1b_2b_3$	$b_8b_9b_{10}b_{11}$
$b_4b_5b_6b_7$	$b_{12}b_{13}b_{14}b_{15}$

Figure 1

$p_0p_1p_2p_3$	$p_8p_9p_{10}p_{11}$
$p_4p_5p_6p_7$	$p_{12}p_{13}p_{14}p_{15}$

Figure 2

$c_0c_1c_2c_3$	$c_8c_9c_{10}c_{11}$
$c_4c_5c_6c_7$	$c_{12}c_{13}c_{14}c_{15}$

Figure 3

The Function A_{K_i}: The abbreviation A_K stands for *add key*. The function A_{K_i} consists of XORing K_i with the state so that the subscripts of the bits in the state and the key bits agree modulo 16.

The Function NS: The abbreviation NS stands for *nibble substitution*. The function NS replaces each nibble N_i in a state by S-box(N_i) without changing the order of the nibbles. So it sends the state

N_0	N_2
N_1	N_3

to the state

S-box(N_0)	S-box(N_2)
S-box(N_1)	S-box(N_3)

.

The Function SR: The abbreviation SR stands for *shift row*. The function SR takes the state

N_0	N_2
N_1	N_3

to the state

N_0	N_2
N_3	N_1

.

The Function MC: The abbreviation MC stands for *mix column*. A column $[N_i, N_j]$ of the state is considered to be the element $N_iz + N_j$ of $\mathbf{F}_{16}[z]/(z^2+1)$. As an example, if the column consists of $[N_i, N_j]$ where $N_i = 1010$ and $N_j = 1001$ then that would be $(x^3+x)z + (x^3+1)$. Like before, $\mathbf{F}_{16}[z]$ denotes polynomials in z with coefficients in $\mathbf{F}_{16}$. So $\mathbf{F}_{16}[z]/(z^2+1)$ means that polynomials are considered modulo z^2+1; thus $z^2 = 1$. So representatives consist of the 16^2 polynomials of degree less than 2 in z.

The function MC multiplies each column by the polynomial $c(z) = x^2z + 1$. As an example,

$$[((x^3+x)z + (x^3+1))](x^2z+1) = (x^5+x^3)z^2 + (x^3+x+x^5+x^2)z + (x^3+1)$$

$$= (x^5+x^3+x^2+x)z + (x^5+x^3+x^3+1) = (x^2+x+x^3+x^2+x)z + (x^2+x+1)$$

$$= (x^3)z + (x^2+x+1),$$

which goes to the column $[N_k, N_l]$ where $N_k = 1000$ and $N_l = 0111$.
Note that $z^2+1 = (z+1)^2$ is reducible over $\mathbf{F}_{16}$ so $\mathbf{F}_{16}[z]/(z^2+1)$ is not a field and not all of its non-zero elements are invertible; the polynomial $c(z)$, however, is.

The simplest way to explain MC is to note that MC sends a column

$b_0b_1b_2b_3$
$b_4b_5b_6b_7$

to

$b_0 \oplus b_6$	$b_1 \oplus b_4 \oplus b_7$	$b_2 \oplus b_4 \oplus b_5$	$b_3 \oplus b_5$
$b_2 \oplus b_4$	$b_0 \oplus b_3 \oplus b_5$	$b_0 \oplus b_1 \oplus b_6$	$b_1 \oplus b_7$

.

The Rounds: The composition of functions $A_{K_i} \circ MC \circ SR \circ NS$ is considered to be the i-th round. So this simplified algorithm has two rounds. There is an extra A_K before the first round and the last round does not have an MC; the latter will be explained in the next section.

Decryption

Note that for general functions (where the composition and inversion are possible) $(f \circ g)^{-1} = g^{-1} \circ f^{-1}$. Also, if a function composed with itself is the identity map (i.e. gets you back where you started), then it is its own inverse; this is called an involution. This is true of each A_{K_i}. Although it is true for our SR, this is not true for the real SR in AES, so we will not simplify the notation SR^{-1}. Decryption is then by $A_{K_0} \circ NS^{-1} \circ SR^{-1} \circ MC^{-1} \circ A_{K_1} \circ NS^{-1} \circ SR^{-1} \circ A_{K_2}$.

To accomplish NS^{-1}, multiply a nibble by $a(y)^{-1} = y^2+y+1$ and add $a(y)^{-1}b(y) = y^3+y^2$ in $\mathbf{F}_2[y]/(y^4+1)$. Then invert the nibble in $\mathbf{F}_{16}$. Alternately, we can simply use one of the S-box tables in reverse.

Since MC is multiplication by $c(z) = x^2z + 1$, the function MC^{-1} is multiplication by $c(z)^{-1} = xz + (x^3+1)$ in $\mathbf{F}_{16}[z]/(z^2+1)$.

Decryption can be done as above. However to see why there is no MC in the last round, we continue. First note that $NS^{-1} \circ SR^{-1} = SR^{-1} \circ NS^{-1}$. Let St denote a state. We have $MC^{-1}(A_{K_i}(St)) = MC^{-1}(K_i \oplus St) = c(z)^{-1}(K_i \oplus St) = c(z)^{-1}(K_i) \oplus c(z)^{-1}(St)$ $= c(z)^{-1}(K_i) \oplus MC^{-1}(St) = A_{c(z)^{-1}K_i}(MC^{-1}(St))$. So $MC^{-1} \circ A_{K_i} = A_{c(z)^{-1}K_i} \circ MC^{-1}$.

What does $c(z)^{-1}(K_i)$ mean? Break K_i into two bytes $b_0b_1 \ldots b_7, b_8, \ldots b_{15}$. Consider the first byte

$b_0b_1b_2b_3$
$b_4b_5b_6b_7$

to be an element of $\mathbf{F}_{16}[z]/(z^2+1)$. Multiply by $c(z)^{-1}$, then convert back to a byte. Do the same with $b_8 \ldots b_{15}$. So $c(z)^{-1}K_i$ has 16 bits. $A_{c(z)^{-1}K_i}$ means XOR $c(z)^{-1}K_i$ with the current state. Note when we do MC^{-1}, we will multiply the state by $c(z)^{-1}$ (or more easily, use the equivalent table that you will create in your homework). For $A_{c(z)^{-1}K_1}$, you will first multiply K_1 by $c(z)^{-1}$ (or more easily, use the equivalent table that you will create in your homework), then XOR the result with the current state.

Thus decryption is also

$$A_{K_0} \circ SR^{-1} \circ NS^{-1} \circ A_{c(z)^{-1}K_1} \circ MC^{-1} \circ SR^{-1} \circ NS^{-1} \circ A_{K_2}.$$

Recall that encryption is

$$A_{K_2} \circ SR \circ NS \circ A_{K_1} \circ MC \circ SR \circ NS \circ A_{K_0}.$$

Notice how each kind of operation for decryption appears in exactly the same order as in encryption, except that the round keys have to be applied in reverse order. For the real AES, this can improve implementation. This would not be possible if MC appeared in the last round.

Encryption Example

Let's say that we use the key in the above example 0101 1001 0111 1010. So $W[0] = 0101\,1001$, $W[1] = 0111\,1010$, $W[2] = 1101\,1100$, $W[3] = 1010\,0110$, $W[4] = 0110\,1100$, $W[5] = 1100\,1010$,

Let's say that the plaintext is my name 'Ed' in ASCII: 01000101 01100100 Then the initial state is (remembering that the nibbles go in upper left, **then lower left**, then upper right, then lower right)

0100	0110
0101	0100

Then we do A_{K_0} (recall $K_0 = W[0]W[1]$) to get a new state:

0100 $\oplus$ 0101	0110 $\oplus$ 0111
0101 $\oplus$ 1001	0100 $\oplus$ 1010

$=$

0001	0001
1100	1110

Then we apply NS and SR to get

0100	0100
1100	1111

$\rightarrow SR \rightarrow$

0100	0100
1111	1100

Then we apply MC to get

1101	0001
1100	1111

Then we apply A_{K_1}, recall $K_1 = W[2]W[3]$.

1101 $\oplus$ 1101	0001 $\oplus$ 1010
1100 $\oplus$ 1100	1111 $\oplus$ 0110

$=$

0000	1011
0000	1001

Then we apply NS and SR to get

1001	0011
1001	0010

$\rightarrow SR \rightarrow$

1001	0011
0010	1001

Then we apply A_{K_2}, recall $K_2 = W[4]W[5]$.

1001 $\oplus$ 0110	0011 $\oplus$ 1100
0010 $\oplus$ 1100	1001 $\oplus$ 1010

$=$

1111	1111
1110	0011

So the ciphertext is 11111110 11110011.

The Real AES

For simplicity, we will describe the version of AES that has a 128-bit key and has 10 rounds. Recall that the AES algorithm operates on 128-bit blocks . We will mostly explain the ways in which it differs from our simplified version. Each state consists of a four-by-four grid of bytes.

The finite field is $\mathbf{F}_{2^8} = \mathbf{F}_2[x]/(x^8+x^4+x^3+x+1)$. We let the byte $b_0b_1b_2b_3b_4b_5b_6b_7$ and the element $b_0x^7+\ldots+b_7$ of $\mathbf{F}_{2^8}$ correspond to each other. The S-box first inverts a byte in $\mathbf{F}_{2^8}$ and then multiplies it by $a(y) = y^4+y^3+y^2+y+1$ and adds $b(y) = y^6+y^5+y+1$ in $\mathbf{F}_2[y]/(y^8+1)$. Note $a(y)^{-1} = y^6+y^3+y$ and $a(y)^{-1}b(y) = y^2+1$.

The real ByteSub is the obvious generalization of our NS - it replaces each byte by its image under the S-box. The real ShiftRow shifts the rows left by $0, 1, 2$ and 3. So it sends the state

B_0	B_4	B_8	B_{12}
B_1	B_5	B_9	B_{13}
B_2	B_6	B_{10}	B_{14}
B_3	B_7	B_{11}	B_{15}

to the state

B_0	B_4	B_8	B_{12}
B_5	B_9	B_{13}	B_1
B_{10}	B_{14}	B_2	B_6
B_{15}	B_3	B_7	B_{11}

.

The real MixColumn multiplies a column by $c(z) = (x+1)z^3+z^2+z+x$ in $\mathbf{F}_{2^8}[z]/(z^4+1)$. Also $c(z)^{-1} = (x^3+x+1)z^3+(x^3+x^2+1)z^2+(x^3+1)z+(x^3+x^2+x)$. The MixColumn step appears in all but the last round. The real AddRoundKey is the obvious generalization of our A_{K_i}. There is an additional AddRoundKey with round key 0 at the beginning of the encryption algorithm.

For key expansion, the entries of the array W are four bytes each. The key fills in $W[0], \ldots, W[3]$. The function RotByte cyclically rotates four bytes 1 to the left each, like the action on the second row in ShiftRow. The function SubByte applies the S-box to each byte. $RC[i] = x^i$ in $\mathbf{F}_{2^8}$ and $RCON[i]$ is the concatenation of $RC[i]$ and 3 bytes of all 0's. For $4 \le i \le 43$,

$$\begin{array}{ll} \text{if } i \equiv 0 (\mathrm{mod}\, 4) & \text{then } W[i] = W[i-4] \oplus \mathrm{RCON}(i/4) \oplus \mathrm{SubByte}(\mathrm{RotByte}(W[i-1])) \\ \text{if } i \not\equiv 0 (\mathrm{mod}\, 4) & \text{then } W[i] = W[i-4] \oplus W[i-1]. \end{array}$$

The i-th key K_i consists of the bits contained in the entries of $W[4i] \ldots W[4i+3]$.

AES as a product cipher

Note that there is transposition by row using ShiftRow. Though it is not technically transposition, there is dispersion by column using MixColumn. The substitution is accomplished with ByteSub and AddRoundKey makes the algorithm key-dependent.

Analysis of Simplified AES

We want to look at attacks on the ECB mode of simplified AES.

The enemy intercepts a matched plaintext/ciphertext pair and wants to solve for the key. Let's say the plaintext is $p_0 \ldots p_{15}$, the ciphertext is $c_0 \ldots c_{15}$ and the key is $k_0 \ldots k_{15}$. There are 15 equations of the form

$$f_i(p_0, \ldots, p_{15}, k_0, \ldots k_{15}) = c_i$$

where f_i is a polynomial in 32 variables, with coefficients in $\mathbf{F_2}$ which can be expected to have 2^{31} terms on average. Once we fix the c_j's and p_j's (from the known matched plaintext/ciphertext pair) we get 16 non-linear equations in 16 unknowns (the k_i's). On average these equations should have 2^{15} terms.

Everything in simplified AES is a linear map except for the S-boxes. Let us consider how they operate. Let us denote the input nibble of an S-box by $abcd$ and the output nibble as $efgh$. Then the operation of the S-boxes can be computed with the following equations

$$\begin{aligned} e &= acd + bcd + ab + ad + cd + a + d + 1 \\ f &= abd + bcd + ab + ac + bc + cd + a + b + d \\ g &= abc + abd + acd + ab + bc + a + c \\ h &= abc + abd + bcd + acd + ac + ad + bd + a + c + d + 1 \end{aligned}$$

where all additions are modulo 2. Alternating the linear maps with these non-linear maps leads to very complicated polynomial expressions for the ciphertext bits.

Solving a system of linear equations in several variables is very easy. However, there are no known algorithms for quickly solving systems of non-linear polynomial equations in several variables.

Design Rationale

The quality of an encryption algorithm is judged by two main criteria, security and efficiency. In designing AES, Rijmen and Daemen focused on these qualities. They also instilled the algorithm with simplicity and repetition. Security is measured by how well the encryption withstands all known attacks. Efficiency is defined as the combination of encryption/decryption speed and how well the algorithm utilizes resources. These resources include required chip area for hardware implementation and necessary working memory for software implementation. Simplicity refers to the complexity of the cipher's individual steps and as a whole. If these are easy to understand, proper implementation is more likely. Lastly, repetition refers to how the algorithm makes repeated use of functions.

In the following two sections, we will discuss the concepts security, efficiency, simplicity, and repetition with respect to the real AES algorithm.

Security

As an encryption standard, AES needs to be resistant to all known cryptanalytic attacks. Thus, AES was designed to be resistant against these attacks, especially differential and linear cryptanalysis. To ensure such security, block ciphers in general must have diffusion and non-linearity.

Diffusion is defined by the spread of the bits in the cipher. Full diffusion means that each bit of a state depends on every bit of a previous state. In AES, two consecutive rounds provide full diffusion. The ShiftRow step, the MixColumn step, and the key expansion provide the diffusion necessary for the cipher to withstand known attacks.

Non-linearity is added to the algorithm with the S-Box, which is used in ByteSub and the key expansion. The non-linearity, in particular, comes from inversion in a finite field. This is not a linear map from bytes to bytes. By linear, I mean a map that can be described as map from bytes (i.e. the 8-dimensional vector space over the field $\mathbf{F}_2$) to bytes which can be computed by multiplying a byte by an 8×8-matrix and then adding a vector.

Non-linearity increases the cipher's resistance against cryptanalytic attacks. The non-linearity in the key expansion makes it so that knowledge of a part of the cipher key or a round key does not easily enable one to determine many other round key bits.

Simplicity helps to build a cipher's credibility in the following way. The use of simple steps leads people to believe that it is easier to break the cipher and so they attempt to do so. When many attempts fail, the cipher becomes better trusted.

Although repetition has many benefits, it can also make the cipher more vulnerable to certain attacks. The design of AES ensures that repetition does not lead to security holes. For example, the round constants break patterns between the round keys.

Efficiency

AES is expected to be used on many machines and devices of various sizes and processing powers. For this reason, it was designed to be versatile. Versatility means that the algorithm works efficiently on many platforms, ranging from desktop computers to embedded devices such as cable boxes.

The repetition in the design of AES allows for parallel implementation to increase speed of encryption/decryption. Each step can be broken into independent calculations because of repetition. ByteSub is the same function applied to each byte in the state. MixColumn and ShiftRow work independently on each column and row in the state respectively. The AddKey function can be applied in parallel in several ways.

Repetition of the order of steps for the encryption and decryption processes allows for the same chip to be used for both processes. This leads to reduced hardware costs and increased speed.

Simplicity of the algorithm makes it easier to explain to others, so that the implementation will be obvious and flawless. The coefficients of each polynomial were chosen to minimize computation.

AES vs RC4. Block ciphers more flexible, have different modes. Can turn block cipher into stream cipher but not vice versa. RC4 1.77 times as fast as AES. Less secure.

12 Public Key Cryptography

In a symmetric key cryptosystem, if you know the encrypting key you can quickly determine the decrypting key ($C \equiv aP+b(\mathrm{mod}N)$) or they are the same (modern stream cipher, AES). In public key cryptography, everyone has a public key and a private key. There is no known way of quickly determining the private key from the public key. The idea of public key cryptography originated with Whit Diffie, Marty Hellman and Ralph Merkle.

Main uses of public-key cryptography:
1) Agree on a key for a symmetric cryptosystem.
2) Digital signatures.

Public-key cryptography is rarely used for message exchange since it is slower than symmetric key cryptosystems.

12.1 Encoding for public key cryptography

Sometimes we need to encode some text, a key or a hash in $\mathbf{Z}/n\mathbf{Z}$ or $\mathbf{F}_{2^d}$. We can use the ASCII encoding to turn text into a bit string. Keys and hashes typically are bit strings to begin with. To encode a bit string as an element of $\mathbf{Z}/n\mathbf{Z}$ or $\mathbf{F}_p$ (where p is prime), we

can consider the bit string to be the binary representation of a number m and, if $m < n$ (which it typically will be in applications), then m represents an element of $\mathbf{Z}/n\mathbf{Z}$. In $\mathbf{F}_{2^d} = \mathbf{F}_2[x]/(x^d + \ldots + 1)$ we can encode a bit $b_0 \ldots b_k$ (assuming $k < d$, which is usual in applications) as $b_0x^k + b_1x^{k-1} + \ldots + b_k$. In any case, if the bit string is too long (i.e. $m \geq n$ or $k \geq d$) then it can be broken into blocks of appropriate size).

12.2 RSA

This is named for Rivest, Shamir and Adleman. Recall that if $\gcd(m, n) = 1$ and $a \equiv 1(\mathrm{mod}\phi(n))$ then $m^a \equiv m(\mathrm{mod}n)$.

Bob picks p, q, primes around 10^{150}. He computes $n = pq \approx 10^{300}$ and $\phi(n) = (p-1)(q-1)$. He finds some number e with $\gcd(e, \phi(n)) = 1$ and computes $e^{-1}\mathrm{mod}\phi(n) = d$. Note $ed \equiv 1(\mathrm{mod}\phi(n))$ and $1 < e < \phi(n)$ and $1 < d < \phi(n)$. He publishes (n, e) and keep d, p, q hidden. He can throw out p and q. For key agreement, he may do this once a year or may do it on the fly for each interaction.

Alice wants to send Bob the plaintext message M (maybe an AES key) encoded as a number $0 \leq M < n$. If the message is longer than n (which is rare), then she breaks the message into blocks of size $< n$. Alice looks up Bob's n, e on his website (or possibly in a directory). She reduces $M^e \mathrm{mod}\, n = C$ (that's a trapdoor function) with $0 \leq C < n$. Note $C \equiv M^e (\mathrm{mod}\, n)$. She sends C to Bob.

Bob reduces $C^d \mathrm{mod}\, n$ and gets M. Why? $C^d \equiv (M^e)^d \equiv M^{ed} \equiv M^1 = M (\mathrm{mod}\, n)$.

If Eve intercepts C, it's useless without Bob's d.

Example: Bob chooses $p = 17$, $q = 41$. Then computes $n = pq = 17 \cdot 41 = 697$ and $\phi(n) = (17-1)(41-1) = 640$. He chooses $e = 33$ which is relatively prime to 640. He then computes $33^{-1}\mathrm{mod}\, \underline{640} = 97 = d$. Bob publishes $n = 697$ and $e = 33$.

Alice wants to use $C = aP + b (\mathrm{mod}\, 26)$ with key(s) $C \equiv 7P + 25 (\mathrm{mod}\, 26)$ to send a long message to Bob. She encodes the key as $7 \cdot 26 + 25 = 207$. She computes $207^e \mathrm{mod}\, n = 207^{33} \mathrm{mod}\, 697$. Her computer uses repeated squares to do this. $207^{33} \mathrm{mod}\, 697 = 156$.

Alice sends 156 to Bob. From 156, it is very hard for Eve to determine 207.

Bob gets 156 and computes $156^d \mathrm{mod}\, n = 156^{97} \mathrm{mod}\, 697 = 207$. Then (and this is not part of RSA) breaks $207 = 7 \cdot 26 + 25$. Now (again this is not part of RSA), Alice sends Bob a long message using $C \equiv 7P + 25 (\mathrm{mod}\, 26)$. End example.

Every user has a pair n_A, e_A for Alice, n_B, e_B for Bob, etc. at his/her website or in a directory. n_A, e_A are called Alice's <u>public keys</u>. d_A is called Alice's <u>private key</u>.

When Alice sends a message M to Bob, as above, she computes $M^{e_B} (\mathrm{mod}\, n_B)$. Bob has d_B so can get back to M.

Why is it hard to find d from e and n? Well, $d \equiv e^{-1} (\mathrm{mod}\, \phi(n))$. Finding the inverse is fast (polynomial time). Finding $\phi(n)$ is slow, if all you have is n, since it requires factoring, for which the only known algorithms are subexponential, but not polynomial.

Assume n is know. Then knowing $\phi(n)$ is polynomial time equivalent to knowing p and q. In simpler terms: computing $\phi(n)$ takes about as long as factoring.

Proof. If you know n, p, q then $\phi(n) = (p-1)(q-1)$ which can be computed in $O(\log^2(n))$.

Now let's say you know n, $\phi(n)$. We have $x^2 + (\phi(n) - n - 1)x + n = x^2 + ((p-1)(q-1) - pq - 1)x + pq = x^2 + (pq - p - q + 1 - pq - 1)x + pq = x^2 - (p+q)x + pq = (x-p)(x-q)$.

So we can find p, q by finding the roots of $x^2 + (\phi(n) - n - 1)x + n$. Can find integer roots of a quadratic polynomial in $\mathbf{Z}[x]$ using the quadratic formula. Taking the square root (of a necessarily integer square) and doing the rest of the arithmetic will take little time. End proof.

So finding $\phi(n)$ is as hard as factoring.

Practicalities: You want 1) $\gcd(p-1, q-1)$ to be small. 2) $p-1$ and $q-1$ should each have a large prime factor. 3) p and q shouldn't be too close together, on the other hand, the ratio of bigger/smaller is usually < 4.

There are special factorization algorithms to exploit if any of the three is not true.

4) Usually e is relatively small (which saves time). Often $e = 3$, $17 = 2^4 + 1$ or $65537 = 2^{16} + 1$ (since repeated squares fast for e small, e not have many 1's in binary representation).

For personal use, people use n of 1024 bits, so $n \approx 10^{308}$. For corporate use, people use 1024 or 2048 bits (latter $n \approx 10^{617}$. In the early 1990's, it was common to use 512 bits, so $n \approx 10^{154}$. An RSA challenge number with $n \approx 2^{768} \approx 10^{232}$ was factored in 2009.

In a symmetric key cryptosystem, Alice and Bob must agree on a shared key ahead of time. This is a problem. It requires co-presence (inconvenient) or sending a key over insecure lines (unsafe). We see from the example of RSA, that public key cryptography solves the problem of symmetric key cryptography.

12.3 Finite Field Discrete logarithm problem

Let $\mathbf{F}_q$ be a finite field. Let g generate $\mathbf{F}_q^*$. Let $b \in \mathbf{F}_q^*$. Then $g^i = b$ for some positive integer $i \leq q - 1$. Determining i given $\mathbf{F}_q$, g and b is the finite field discrete logarithm problem (FFDLP).

Example. 2 generates $\mathbf{F}_{101}^*$. So we know $2^i = 3$ (i.e. $2^i \equiv 3 (\mathrm{mod}\, 101)$) has a solution. It is $i = 69$. Similarly, we know $2^i = 5$ has a solution; it is $i = 24$. How could you solve such problems faster than brute force? In Sections 31.1 and 31.3.3 we present solutions faster than brute force. But they are nonetheless not fast. End example.

For cryptographic purposes we take $10^{300} < q < 10^{600}$ where q is a (large) prime or of the form 2^d. Notation, if $g^i = b$ then we write $\log_g(b) = i$. Recall the logarithms you have already learned: $\log_{10}(1000) = 3$ since $10^3 = 1000$ and $\ln(e^2) = \log_e(e^2) = 2$. In the above example, for $q = 101$ we have $\log_2(3) = 69$ (since $2^{69} \equiv 3 (\mathrm{mod}\, 101)$).

The best known algorithms for solving the FFLDP take as long as those for factoring, and so are subexponential (assuming $q = \#\mathbf{F}_q \approx n$).

12.4 Diffie-Hellman key agreement

Diffie-Hellman key agreement over a finite field (FFDH) is commonly used. Most commonly, for each transaction, Alice chooses a $\mathbf{F}_q$ and g (a generator of $\mathbf{F}_q^*$) and a private key a_A with $1 << a_A << q - 1$. She reduces g^{a_A} in $\mathbf{F}_q$ and sends g, the reduction of g^{a_A} and $\mathbf{F}_q$ to Bob. Bob chooses a private key a_B with $1 << a_B << q - 1$. He reduces g^{a_B} in $\mathbf{F}_q$ and sends it to Alice.

Less commonly, there is a fixed $\mathbf{F}_q^*$ and g for all users. Each user has a private key a ($a_A, a_B, a_C, \ldots$) with $1 << a < q-1$ and a public key, which is the reduction of g^a in the field $\mathbf{F}_q$. Each user publishes (the reductions of) $g^{a_A}, g^{a_B}, \ldots$ in a directory or on their websites.

In both cases, they each use the reduction of $g^{a_A a_B}$ in $\mathbf{F}_q$ as their secret shared key.

If Alice and Bob want to agree on a key for AES, they use the reduction of $g^{a_A a_B}$. Alice can compute by raising the reduction of g^{a_B} to a_A. Bob can do the reverse.

Eve has q, g, g^{a_A}, g^{a_B} but can not seem to find $g^{a_A a_B}$ without solving the FFDLP. This often seems amazing. She can find $g^{a_A} g^{a_B} = g^{a_A + a_B}$, but that's useless. To get $g^{a_A a_B}$, she needs to raise g^{a_A}, for example, to a_B. To get a_B she could try to use g and g^{a_B}. But determining a_B from g and g^{a_B} is the FFDLP, for which there is no known fast solution.

Example. $q = p = 97$, $g = 5$. $a_A = 36$ is Alice's private key. $g^{a_A} = 5^{36} \bmod 97 = 50$ so $g^{a_A} = 50$ is Alice's public key. $a_B = 58$ is Bob's private key. $g^{a_B} = 5^{58} \bmod 97 = 44$ so $g^{a_B} = 44$ is Bob's public key.

Alice computes $(g^{a_B})^{a_A} = 44^{36} \equiv 75$ (in $\mathbf{F}_{97}$) (I've changed notation as a reminder) and Bob computes $(g^{a_A})^{a_B} = 50^{58} = 75$.

From 97, 5, 50, 44, Eve can't easily get 75.

Practicalities: The number $q - 1$ should have a large prime factor ℓ (or else there is a special algorithm for solving the FFDLP). In reality, g does not generate all of $\mathbf{F}_q^*$, but instead g generates ℓ elements. The reduction of $g^{a_A a_B}$ will be about the same size as $q \geq 10^{300}$. To turn this into an AES key, they could agree to use the last 128 bits of the binary representation of $g^{a_A a_B}$ if q is prime. If $\mathbf{F}_q = \mathbf{F}_2[x]/(f(x))$ then they could agree to use the coefficients of $x^{127}, \ldots, x^0$ in $g^{a_A a_B}$.

12.5 Lesser used public key cryptosystems

12.5.1 RSA for message exchange

RSA could be used for encrypting a message instead of encrypting an AES key (then there is no need to use AES). Alice encodes a message M as a number $0 \leq M < n_B$ and sends Bob the reduction of $M^{e_B} (\bmod n_B)$.

If the message is too long, she breaks the message into blocks $M_1, M_2, M_3, \ldots$ with each $M_i < n_B$.

12.5.2 ElGamal message exchange

ElGamal message exchange has no patent, so it can be used in PGP. Typically used to exchange a key for a symmetric cryptosystem (like AES; in PGP use CAST-128, like AES) but can be used to send any message.

Alice indicates to Bob that she wants to send him an encrypted message. Bob chooses a finite field $\mathbf{F}_q$ and a generator g of $\mathbf{F}_q^*$. He also chooses a private key a_B with $1 << a_B << q - 1$. Bob sends Alice $\mathbf{F}_q$, g and the reduction of g^{a_B} in $\mathbf{F}_q$.

Alice wants to encrypt the message M to Bob, which she encodes as in Section 12.1. Alice chooses a random session key k with $1 << k < q - 1$. She picks a different k each encryption. She then sends Bob the pair $g^k, M g^{a_B k}$ (each reduced in the finite field).

Alice knows g and g^{a_B} (since it's public) and k so she can compute g^k and $(g^{a_B})^k = g^{a_B k}$ and then multiplies the latter to get $M g^{a_B k}$. Bob receives the pair. He won't find k and won't need to. He first computes $(g^k)^{a_B} = g^{a_B k}$ (he knows a_B, his private key). Then he computes $(g^{a_B k})^{-1}$ (in $\mathbf{F}_q$) and multiplies $(M g^{a_B k})(g^{a_B k})^{-1} = M$.

If Eve finds k (which seems to require solving the FFDLP since g is known and g^k is sent) she could compute $(g^{a_B})k = g^{a_B k}$ then $(g^{a_B k})^{-1}$, then M.

Example: $q = 97$, $g = 5$, $a_B = 58$ is Bob's private key, $g^{a_B} = 44$ is Bob's public key. Alice wants to encrypt $M = 30$ for Bob. She picks a random session key $k = 17$. She computes $g^k = 5^{17} = 83$. She knows $g^{a_B} = 44$ (it's public) and computes $(g^{a_B})^k = 44^{17} = 65$. Then she computes $Mg^{a_B k} = 30 \cdot 65 = 10$. She sends Bob $g^k, Mg^{a_B k} = 83, 10$.

Bob receives $83, 10$. He knows $a_B = 58$ so computes $(g^k)^{a_B} = 83^{58} = 65 = g^{a_B k}$. Then computes $(g^{a_B k})^{-1} = 65^{-1} = 3$ (i.e. $65^{-1} \equiv 3 (\mathrm{mod}\, 97)$), then multiplies $(Mg^{a_B k}(g^{a_B k})^{-1} = 10 \cdot 3 = 30 = M$.

12.5.3 Massey-Omura message exchange

The Massey-Omura cryptosystem (this is not really symmetric or public key). It can be used to send a key or a message. Alice wants to send a message to Bob. Alice chooses a finite field $\mathbf{F}_q$ and a random encrypting key e_A with $\gcd(e_A, q-1) = 1$ and $1 << e_A << q-1$. Alice computes $e_A^{-1} (\mathrm{mod}\, q-1) = d_A$. Alice encodes a plaintext message as an element of the field $M \in \mathbf{F}_q^*$. She sends $\mathbf{F}_q$ and the reduction of M^{e_A} in $\mathbf{F}_q$ to Bob.

Bob chooses a random encrypting key e_B with $\gcd(e_B, q-1) = 1$ and $1 << e_B << q-1$. Both e's are for this encryption only. Bob computes $d_B \equiv e_B^{-1} (\mathrm{mod}\, q-1)$. He computes the reduction of $(M^{e_A})^{e_B} = M^{e_A e_B}$ and sends it back to Alice. Alice computes the reduction of $(M^{e_A e_B})^{d_A} = M^{e_A e_B d_A} \stackrel{*}{=} M^{e_A d_A e_B} = M^{e_B}$ to Bob. Then Bob computes $(M^{e_B})^{d_B} = M$.

The special step is happening at $*$. In a sense, Alice puts a sock on a foot. Bob sticks a shoe over that. Alice then removes the sock, *without* removing the shoe, then Bob removes the shoe. Bob now sees the foot, though Alice never does.

The Massey-Omura cryptosystem was tried with cell-phones.

Example. $p = 677$. Alice sends Bob the digraph SC. Since S= 18 and C= 2 the digraph is encoded as $18 \cdot 26 + 2 = 470 = M$. Alice picks $e_A = 255$, so $d_A = 255^{-1} \mathrm{mod}\, 676 = 395$. Bob picks $e_B = 421$ so $d_B = 421^{-1} \mathrm{mod}\, 676 = 281$. Alice computes $470^{255} \mathrm{mod}\, 677 = 292$ and sends 292 to Bob. Bob computes $292^{421} \mathrm{mod}\, 677 = 156$ and sends 156 to Alice. Alice computes $156^{395} \mathrm{mod}\, 677 = 313$ and sends 313 to Bob. Bob computes $313^{281} \mathrm{mod}\, 677 = 470$ and decodes 470 to SC.

12.6 Elliptic curve cryptography (ECC)

Elliptic curve cryptography is public key cryptography and has much shorter keys (1/6 as many bits now) than than RSA and FFDLP cryptosystems for the seame level of security. This is useful in terms of key agreement and working where there is minimal storage or computations should be kept short, like on smart cards.

12.6.1 Elliptic curves

An elliptic curve is a curve described by an equation of the form $y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6$ and an extra 0-point. Example $y^2 + y = x^3 - x$ is in figure 1 (on a future page). Where did this form come from? It turns out that all cubic curves can be brought

to this form by a change of variables and this is a convenient form for the addition we will describe. For now we will work over the real numbers. We need a zero-point that we will denote $\emptyset$. Bend all vertical lines so they meet at the top and bottom and glue those two endpoints together. It's called the point at ∞ or the 0-point. It closes off our curve. That point completes our curves. It's at the top and bottom of every vertical line.

We can put an addition rule (group structure) for points of an elliptic curve. Rule: Three points lying on a line sum to the 0-point (which we'll denote $\emptyset$).

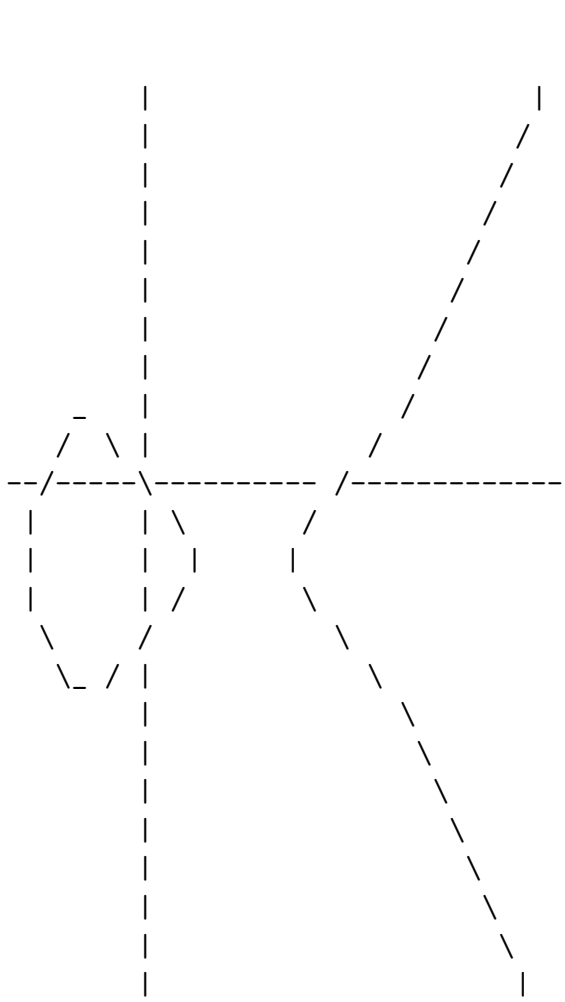
The vertical line L_1 meets the curve at P_1, P_2 and $\emptyset$; so $P_1 + P_2 + \emptyset = \emptyset$, so $P_1 = -P_2$, and $P_2 = -P_1$. Two different points with the same x-coordinate are inverses/negatives of each other. See figure 2.

If you want to add $P_1 + P_2$, points with different x-coordinates, draw a line between them and find the third point of intersection P_3. Note $P_1 + P_2 + P_3 = \emptyset$, $P_1 + P_2 = -P_3$. See figure 3.

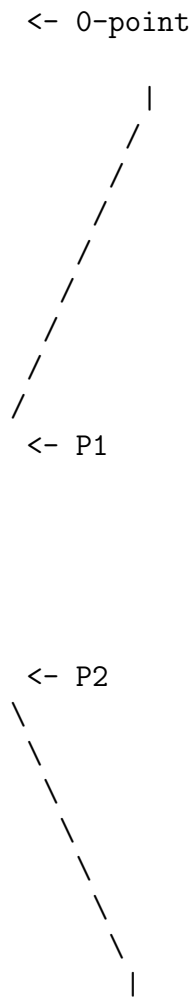
Aside: Where do $y = x^2$ and $y = 2x - 1$ intersect? Where $x^2 = 2x - 1$ or $x^2 - 2x + 1 = (x-1)^2 = 0$. They meet at $x = 1$ twice (from the exponent) and this explains why $y = 2x-1$ is tangent to $y = x^2$. See figure 4.

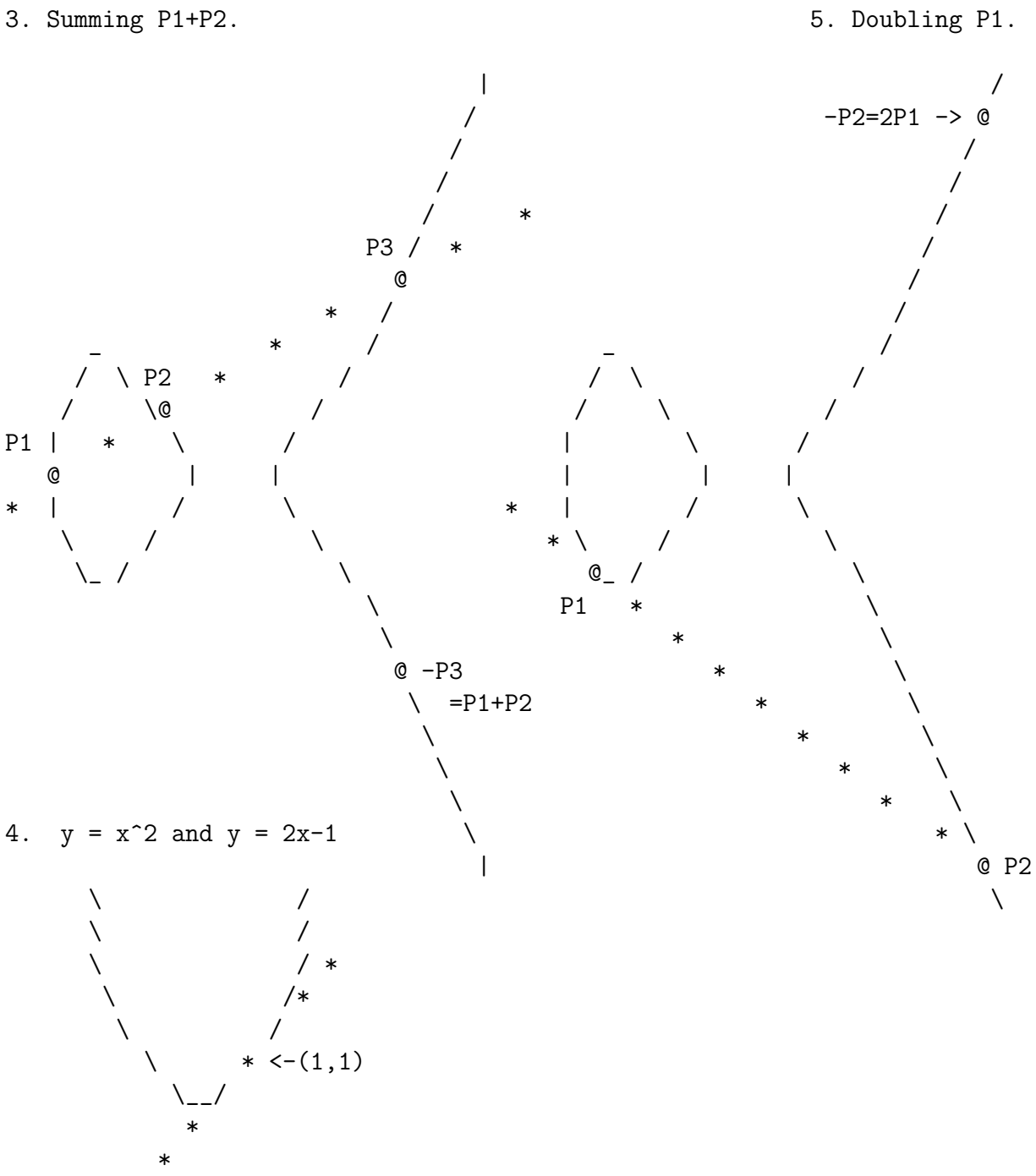
Back to elliptic curves. How to double a point P_1. Draw the tangent line and find the other point of intersection P_2. $P_1 + P_1 + P_2 = \emptyset$ so $2P_1 = -P_2$. See figure 5.

1. An elliptic curve with x & y-axes.
   y^2 + y = x^3 - x



2. EC without axes.
Finding negatives.





This addition is obviously commutative and, though not obvious, it's associative.

Let's do an example. Clearly $P = (1, 0)$ is on the curve $y^2 + y = x^3 - x$. Let's find $2P$. We find the tangent line at P using implicit differentiation. $2y\frac{dy}{dx} + \frac{dy}{dx} = 3x^2 - 1$. So $\frac{dy}{dx} = \frac{3x^2-1}{2y+1}$ and $\frac{dy}{dx}|_{(1,0)} = 2$. The tangent line is $y - 0 = 2(x - 1)$ or $y = 2x - 2$. Where does that intersect $y^2 + y = x^3 - x$? Where $(2x - 2)^2 + (2x - 2) = x^3 - x$ or $x^3 - 4x^2 + 5x - 2 = 0 = (x - 1)^2(x - 2)$. It meets twice where $x = 1$ (i.e. at $(1, 0)$) and once

where $x = 2$. Note that the third point of intersection is on the line $y = 2x - 2$ so it is the point $(2, 2)$. Thus $(1, 0) + (1, 0) + (2, 2) = 2P + (2, 2) = \emptyset$, $(2, 2) = -2P$, $2P = -(2, 2)$. Now $-(2, 2)$ is the other point with the same x-coordinate. If $x = 2$ then we have $y^2 + y = 6$ so $y = 2, -3$ so $2P = (2, -3)$.

To find $3P = P + 2P = (1, 0) + (2, -3)$, we will find the line through $(1, 0), (2, -3)$. It's slope is -3 so $y - 0 = -3(x - 1)$ or $y = -3x + 3$. Where does that line meet $y^2 + y = x^3 - x$? Well $(-3x+3)^2 + (-3x+3) = x^3 - x$ or $x^3 - 9x^2 + 20x - 12 = 0 = (x-1)(x-2)(x-6)$ (note that we knew the line met the curve where $x = 1$ and $x = 2$ so $(x - 1)(x - 2)$ is a factor of $x^3 - 9x^2 + 20x - 12$, so finding the third factor is then easy). The third point of intersection has $x = 6$ and is on $y = -3x + 3$ so it's $(6, -15)$. So $(1, 0) + (2, -3) + (6, -15) = \emptyset$ and $(1, 0) + (2, -3) = -(6, -15)$. What's $-(6, -15)$? If $x = 6$, then $y^2 + y = 210$, $y = -15, 14$ so $-(6, -15) = (6, 14) = P + 2P = 3P$. End of example.

Since adding points is just a bunch of algebraic operations, there are formulas for it. If $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ and $P_1 \neq -P_2$ then $P_1 + P_2 = P_3 = (x_3, y_3)$ where

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}, \quad \nu = \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1}$$

if $x_1 \neq x_2$ and

$$\lambda = \frac{3x_1^2 + 2a_2 x_1 + a_4 - a_1 y_1}{2y_1 + a_1 x_1 + a_3}, \quad \nu = \frac{-x_1^3 + a_4 x_1 + 2a_6 - a_3 y_1}{2y_1 + a_1 x_1 + a_3}$$

if $x_1 = x_2$ and $x_3 = \lambda^2 + a_1\lambda - a_2 - x_1 - x_2$ and $y_3 = -(\lambda + a_1)x_3 - \nu - a_3$ (in either case).

Example. Find $(1, 0) + (2, -3)$ on $y^2 + y = x^3 - x$ using the addition formulas. $a_1 = 0$, $a_3 = 1$, $a_2 = 0$, $a_4 = -1$, $a_6 = 0$, $x_1 = 1$, $y_1 = 0$, $x_2 = 2$, $y_2 = -3$. $\lambda = \frac{-3-0}{2-1} = -3$, $\nu = \frac{0 \cdot 2 - (-3)(1)}{2-1} = 3$. So $x_3 = (-3)^2 + 0(-3) - 0 - 1 - 2 = 6$ and $y_3 = -(-3+0)(6) - 3 - 1 = 14$. So $(1, 0) + (2, -3) = (6, 14)$.

12.6.2 Elliptic curve discrete logarithm problem

Let's work over finite fields. In $\mathbf{F}_p^*$ with $p \neq 2$, a prime, half of the elements are squares. As an example, in $\mathbf{F}_{13}^*$, $1^2 = 1$, $2^2 = 4$, $3^2 = 9$, $4^2 = 3$, $5^2 = 12$, $6^2 = 10$, $7^2 = 10$, $8^2 = 12$, $9^2 = 3$, $10^2 = 9$, $11^2 = 4$, $12^2 = 1$. The equation $y^2 = 12$ has two solutions $y = \pm 5 = 5, 8$.

There are efficient algorithms for determining whether or not an element of $\mathbf{F}_p^*$ is a square and if so, what are the square roots. If $p > 3$ then we can find an equation for our elliptic curve of the form $y^2 = x^3 + a_4 x + a_6$, by changing variables, if necessary, and not affect security..

Example. Let E be $y^2 = x^3 + 1$ find $E(\mathbf{F}_5)$ (the points with coordinates in $\mathbf{F}_5$). It helps to know the squares: $0^2 = 0$, $1^2 = 1$, $2^2 = 4$, $3^2 = 4$, $4^2 = 1$.

x	$x^3 + 1$	$y = \pm\sqrt{x^3 + 1}$	points
0	1	$\pm 1 = 1, 4$	$(0, 1), (0, 4)$
1	2	no	
2	4	$\pm 2 = 2, 3$	$(2, 2), (2, 3)$
3	3	no	
4	0	0	$(4, 0)$ and $\emptyset$

We have 6 points in $E(\mathbf{F}_5)$. Over a finite field you can add points using lines or addition formulas. If $G = (2,3)$ then $2G = (0,1)$, $3G = (4,0)$, $4G = (0,4)$ (note it has same x-coordinate as $2G$ so $4G = -2G$ and $6G = \emptyset$), $5G = (2,2)$, $6G = \emptyset$. So $G = (2,3)$ is a generator of $E(\mathbf{F}_5)$.

Example. Let E be $y^2 = x^3+x+1$ over $\mathbf{F}_{109}$. It turns out that $E(\mathbf{F}_{109})$ has 123 points and is generated by $G = (0,1)$. The point $(39,45)$ is in $E(\mathbf{F}_{109})$ since $39^3 + 39 + 1 \bmod 109 = 63$ and $45^2 \bmod 109 = 63$. So $(39,45) = (0,1) + (0,1) + \ldots + (0,1) = n(0,1)$ for some integer n. What is n? That is the discrete log problem for elliptic curves over finite fields (ECDLP). You could solve this by brute force, but not if 109 is replaced by a prime around 10^{50}. Solving the ECDLP for an elliptic curve over $\mathbf{F}_q$ with $q \approx 2^{163}$ (or 2^{192}) is about as hard as solving the FFDLP for $\mathbf{F}_q$ with $q \approx 2^{1024}$ or factoring n with $n \approx 2^{1024}$ (or 2^{8000}, respectively). So we can use shorter keys. Another advantage here is that for a given finite field there can be lots of associated elliptic curves.

It takes one or two points to generate $E(\mathbf{F}_p)$. Consider $y^2 = x^3 + 1$ over $\mathbf{F}_7$. $0^2 = 0$, $(\pm 1)^2 = 1$, $(\pm 2)^2 = 4$, $(\pm 3)^2 = 2$.

x	x^3+1	$y = \pm\sqrt{x^3+1}$	points
0	1	± 1	$(0,1), (0,6)$
1	2	± 3	$(1,3), (1,4)$
2	2	± 3	$(2,3), (2,4)$
3	0	0	$(3,0)$
4	2	± 3	$(4,3), (4,4)$
5	0	0	$(5,0)$
6	0	0	$(6,0)$ and $\emptyset$

So $E(\mathbf{F}_7)$ has 12 points.

	$R = (5,0)$	$2R = \emptyset$
$Q = (1,3)$	$Q + R = (2,3)$	
$2Q = (0,1)$	$2Q + R = (4,4)$	
$3Q = (3,0)$	$3Q + R = (6,0)$	
$4Q = (0,6)$	$4Q + R = (4,3)$	
$5Q = (1,4)$	$5Q + R = (2,4)$	
$6Q = \emptyset$		

All points are of the form $nQ+mR$ with $n \in \mathbf{Z}/6\mathbf{Z}$ and $m \in \mathbf{Z}/2\mathbf{Z}$. Note that the coefficients of $y^2 = x^3+1$ and the coordinates of the points are all defined modulo 7, whereas the points add up modulo 6. In this case, two points together generate. You could still use discrete log with $G = (1,3)$ as a pseudo-generator point, for example. It wouldn't generate all of $E(\mathbf{F}_7)$ but half of it.

On average, $\#E(\mathbf{F}_q) = q + 1$.

12.6.3 Elliptic curve cryptosystems

In the next two sections, we describe the analogues of the Diffie Hellman key agreement system and the ElGamal message exchange system. The section on the Elliptic Curve El-

Gamal message exchange system explains how to encode a plaintext message on a point of an elliptic curve.

12.6.4 Elliptic curve Diffie Hellman

Analogue of Diffie Hellman key exchange for elliptic curves (ECDH). Choose a finite field $\mathbf{F}_q$ with $q \approx 10^{50}$. Note that since this discrete logarithm problem is currently harder to solve than that described earlier in $\mathbf{F}_q^*$, we can pick q smaller than before. Fix some elliptic curve E whose defining equation has coefficients in $\mathbf{F}_q$ and a (pseudo) generator point $G = (x_1, y_1)$ which is in $E(\mathbf{F}_q)$. The point G must have the property that some very high multiple of G is the 0-point $nG = \emptyset$. Recall $nG = G + G + G + \ldots + G$ (n times). The number n should have a very large prime factor and $n \neq q, q+1, q-1$ (otherwise there are special faster algorithms for solving the ECDLP).

Each user has a private key number $a_A, a_B, \ldots$ and a public key point $a_A G, a_B G, \ldots$. Or, Alice specifies $\mathbf{F}_q$, $E(\mathbf{F}_q)$ and G, a (pseudo)-generating point and sends Bob $\mathbf{F}_q$, $E(\mathbf{F}_q)$, G and $a_A G$. Alice and Bob's Diffie-Hellman shared key will be $a_A a_B G$.

Example $q = p = 211$, $E : y^2 = x^3 - 4$, $G = (2, 2)$. It turns out that $241G = \emptyset$. Alice chooses private key $a_A = 121$, so A's public key is $a_A G = 121(2, 2) = (115, 48)$. Bob chooses private key $a_B = 223$, so B's public key is $a_B G = 223(2, 2) = (198, 72)$. Their shared key is $a_A a_B G$. A sends $a_A G$ to Bob, B sends $a_B G$ to Alice.

Then A computes $a_A(a_B G) = 121(198, 72) = (111, 66)$ and B computes $a_B(a_A G) = 223(115, 48) = (111, 66)$. So $(111, 66)$ is their shared key that they could use for a symmetric key cryptosystem. End example.

Aside. Find $120G + 130G$. Note $= 250G = 241G + 9G = 0 + 9G = 9G$. I.e. since $250 (\mathrm{mod}\, 241) = 9$ we have $250G = 9G$. End aside.

Note that trying to figure out what multiple of $G = (2, 2)$ gives you Alice's public key $(115, 48)$ is the ECDLP. Alice can compute $121G$ through repeated doubling. In addition, if $p \approx 10^{50}$, Alice and Bob could agree to use the last 128 bits of the binary representation of the x-coordinate of their shared key as an AES key. If working over $\mathbf{F}_{2^r}$ could use the coefficients of $x^{127}, \ldots, 1$ of the x-coordinate as an AES key.

Practicalities: For FFDH people use $q > 10^{300}$ but for ECDH people use $q > 10^{50}$. Usually $q = 2^r$ with $r \geq 163$. Indeed, adding two points on an elliptic curve is much slower than doing a multiplication in a finite field. However, since q is much smaller for ECDH, that makes up for much of the slow-down.

12.6.5 Elliptic Curve ElGamal Message Exchange

Analogue of ElGamal message exchange. First issue: How to encode a message as a point. Go back to finite fields. If working with $p = 29$ and want to encode the alphabet in $\mathbf{F}_{29}$, then you can encode $A = 0, \ldots, Z = 25$. What to do with the elliptic curve, for example $y^2 = x^3 - 4$ over $\mathbf{F}_{29}$. Ideally you could encode a number as an x-coordinate of a point, but not all numbers are x-coordinates of points (only about half of them). Not all numbers are y-coordinates of points either (only about half of them). Try to encode $I = 8$. $8^3 - 4 = 15 \neq \square \in \mathbf{F}_{29}^*$.

Instead you could work over $p = 263$ (chosen because it's the first prime bigger than $26 \cdot 10$). Encode the message and one free digit as the x-coordinate of a point. With 10 digits to choose from and each having a 50% chance of success, this should work (in real life you might have the last eight bits free so the probability of trouble is essentially 0).

Say you have $p = 263$, $E : y^2 = x^3 - 4$. Message $P = 15$. Find a point $(15a, y)$ on the curve. Try $x = 150$.
$x = 150$, $150^3 - 4 \equiv 180 \neq \square (\text{mod} 263)$.
$x = 151$, $151^3 - 4 \equiv 14 \neq \square (\text{mod} 263)$.
$x = 152$, $152^3 - 4 \equiv 228 \neq \square (\text{mod} 263)$.
$x = 153$, $153^3 - 4 \equiv 39 \equiv 61^2 (\text{mod} 263)$.
A handout shows how to do this in PARI.
So $(153, 61)$ is a point on the curve and all but the last digit of the x-coordinate is our message. If Alice wants to send the message L to Bob, then she picks a random k. Let $a_B G$ be B's public key. Q is the encoded plaintext point. Then Alice sends $(kG, Q + ka_B G)$ to Bob. Bob receives it. Computes $a_B kG$ and subtracts that from $Q + kA_B G$ to get the plaintext point Q.

Example. $p = 263$, $E : y^2 = x^3 - 4$, $G = (2, 2)$, $a_B = 101$, $a_B G = 101(2, 2) = (165, 9)$ (this is Bob's public key point). Alice wants to send the message P encoded as $M = (153, 61)$ to Bob. She picks the random session $k = 191$ and computes $kG = 191(2, 2) = (130, 94)$, $k(a_B G) = 191(165, 9) = (41, 96)$ and $M + ka_B G = (153, 61) + (41, 96) = (103, 22)$.

Alices sends the pair of points $kG, M + ka_B G$ or $(130, 94), (103, 22)$ to Bob.

Bob receives the pair of points and computes $a_B(kG) = 101(130, 94) = (41, 96)$. Then computes $(M + ka_B G) - (a_B kG) = (103, 22) - (41, 96) = (103, 22) + (41, -96) = (103, 22) + (41, 167) = (153, 61)$. And decodes all but the last digit of the x-coordinate to 14 =P.

Again people actually prefer to work over fields of the type $\mathbf{F}_{2^r}$. So if you want to encode the AES key $1101 \ldots 1011$ then you could find a point whose x-coordinate is $1 \cdot x^{135} + 1 \cdot x^{134} + 0 \cdot x^{133} + 1 \cdot x^{132} + \ldots + 1 \cdot x^{11} + 0 \cdot x^{10} + 1 \cdot x^9 + 1 \cdot x^8 + b_7 x^7 + \ldots + b_1 x + b_0$ for some $b_i \in \mathbf{F}_2$. The probability of failure is $1/(2^{2^8})$. There are subexponential algorithms for solving the FFDLP in $\mathbf{F}_q^*$ and for factoring n so people work with $q \approx n > 10^{300}$. The best known algorithm for solving the ECDLP in $E(\mathbf{F}_q)$ takes time $O(\sqrt{q})$, which is exponential. So people work with $q > 10^{50}$. It takes longer to add points on an elliptic curve than to do a multiplication in a finite field (or mod n) for the same size of q. But since q is much smaller for an EC, the slow-down is not significant. In addition, we have smaller keys when working with EC's which is useful in terms of key agreement and working where there is minimal storage or computations should be kept short, like on smart cards.

13 Hash Functions and Message Authentication Codes

A hash is a relatively short record of a msg used to ensure you got msg correctly. Silly ex. During noisy phone call, I give you a 3rd party phone number and the sum of the digits mod 10 (that sum is the hash). You sum digits you heard mod 10. If it agrees with my sum, odds are you got it right. End ex.

A hash function $f(x)$ sends $m + t$ bit strings to t bit strings and, when used in cryptography, should have three properties. A hash algorithm $H(x)$ is built up from a hash

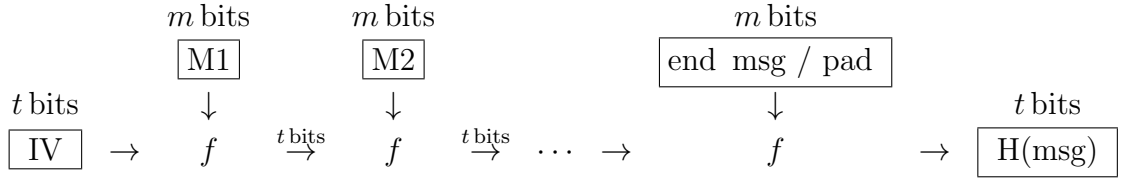
function and sends strings of arbitrary length to t bit stings and should have the same three properties.

A hash algorithm $H(x)$ is said to have the one-way property if given an output y it is difficult to find any input x such that $H(x) = y$. Let's say that the hashes of passwords are stored on a server. When you log in, it computes the hash of the password you just typed and compares it with the stored hash. If someone can solve the one-way problem then she could find your password. A hash algorithm is said to have the weakly collision free property if, given input x, it is difficult difficult to find any $x' \neq x$ such that $H(x) = H(x')$. Let's say that you have a program available for download and you also make its hash available. That way people can download the software, hash it, and confirm that they got the proper software and not something dangerous. If the hash algorithm does not have the weakly collision free algorithm then perhaps he can find a dangerous program that hashes the same way and post his dangerous program and the same hash at a mirror site. It is said to have the strongly collision free property it is difficult to find any x and x' with $x \neq x'$ such that $H(x) = H(x')$. It can be shown (under reasonable assumptions) that strongly collision free implies weakly collision free which implies one-way. Let's say that the hash algorithm does not have the strongly collision free property. In addition, let's assume that Eve can find x and x' with $H(x) = H(x')$ and where she can actually specify ahead of time part of x and part of x' (this is a stronger assumption). Then Eve can a good program and its hash to gain trust. Later she can replace the good program with a bad program with the same hash.

Recently Wang, Feng, Lai, Yu and Lin showed that of the popular hash functions (MD5, SHA-0, SHA-1, SHA-2), all but SHA-2 do not have the strongly collision free property. In addition, SHA-2 is similar to SHA-1, so it might not either. So there was an international competition that in 2012 chose SHA-3.

To create a hash algorithm from a hash function one normally uses a hash function with two inputs: an m-bit string a and a t-bit string b. Then $f(a, b)$ outputs a t-bit string. Let's extend a hash function f to a hash algorithm H. Assume that the M has more than m bits. Break M into m-bit blocks, padding the last block if necessary with 0's. Initially we take b to be a given, known t-bit initialization vector (perhaps all 0's). (For SHA-3, m and t can vary, but one common setting would be $m = 1088$, $t = 256$.)

Example.



If a hash algorithm depends on a secret key, it is called a MAC. To do this, we just replace the known IV with a secret shared key.

Example. f is AES, so $t = m = 128$. Break the message into 128 bit blocks. If the message length is not a multiple of 128 bits then add 0's to the end (padding) so that it is. The key for the first AES is the IV. The key for the second AES is the output of the first AES and so on. The final output is the hash of the message. This is not a secure hash function but it's OK as a MAC.

Let's flesh out this scenario. Alice and Bob used public-key cryptography to agree on two AES keys, key_1 and key_2. Alice sends Bob (in ECB mode, for simplicity) a message encrypted with AES. She breaks the message into n blocks: $PT_1, \ldots, PT_n$. She encrypts each PT_i with AES and using key_1 to get the corresponding ciphertexts $CT_1, \ldots, CT_n$.

Then Alice computes the MAC of $PT_1P_2 \ldots PT_n$ using key_2 and sends the (unencrypted) MAC to Bob.

Bob receives $CT_1, \ldots, CT_n$ and decrypts them using key_1. Now Bob has the PT_i's. Then Bob MACs those PT_i's with key_2 and finds the MAC. Then Bob checks to see if this MAC agrees with the one that Alice sent him. If it does, then he can be sure that no one tampered with the CT_i's during transmission. This is called message integrity.

Without the MAC, Eve could intercept $CT_1, \ldots, CT_n$ along the way and tamper with it (though it probably wouldn't decrypt to something sensible since Eve doesn't know the key).

If Eve tampers with it, she can't create a MAC that will agree with hers. End Example

13.1 The MD5 hash algorithm

One of the most popular hash algorithms at the moment is MD5. The hash algorithms SHA1 and SHA2 are also popular and very similar. SHA stands for *Secure Hash Algorithm*. MD5 is more efficient than the hash algorithm described above using AES. It is based on the following hash function f. The function f takes two inputs: a 128 bit string and a 512 bit strings and its output is a 128 bit strings. Let X be the 512 bit string. For MD5 we will call a 32 bit string a word. So X consists of 16 words. Let $X[0]$ be the first word, $X[1]$ be the next, ..., $X[15]$ be the last word.

Initial buffers

While evaluating f we continually update four buffers: A, B, C, D; each contains one word. There are four constants Const_A = 0x67452301, Const_B = 0xefcdab89, Const_C = 0x98badcfe, Const_D 0x10325476. Together these form the 128 bit initialization vector. The notation 0x indicates that what comes after is a hexadecimal representation which uses the symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f to represent the nibbles 0000, 0001, ..., 1111. So Const_A = 01100111010001010010001100000001. Initially, for the first evaluation of f only, we let $A = \text{Const}_A$, $B = \text{Const}_B$, $C = \text{Const}_C$, $D = \text{Const}_D$. For clarity during the hash function, when we update, A, B, C or D we sometimes give it an index. So initially $A_0 = \text{Const}_A$, $B_0 = \text{Const}_B$, $C_0 = \text{Const}_C$, $D_0 = \text{Const}_D$.

The four functions

Let us define four functions. Each takes 3 words as inputs.

$F(X, Y, Z) = XY \vee \bar{X}Z$,
$G(X, Y, Z) = XZ \vee Y\bar{Z}$,
$H(X, Y, Z) = X \oplus Y \oplus Z$,
$I(X, Y, Z) = Y \oplus (X \vee \bar{Z})$.

Note $\bar{1} = 0$ and $\bar{0} = 1$ (this NOT). Note $0 \vee 0 = 0$, $0 \vee 1 = 1$, $1 \vee 0 = 1$, $1 \vee 1 = 1$ (this is OR).

For example, let us apply F to three bytes (instead of 3 words). Let $X = 00001111$, $Y = 00110011$, $Z = 01010101$.

X	Y	Z	XY	$\bar{X}Z$	$XY \vee \bar{X}Z$
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	0	1

So $F(X, Y, Z) = 01010011$. Note that for 1-bit inputs to F, we had all 8 possibilities 000, ..., 111 and the outputs were 0 half the time and 1 half the time. This is true for G, H and I.

The constants

There are 64 constants $T[1]$, ..., $T[64]$. Let i measure radians. Then $|\sin(i)|$ is a real number between 0 and 1. Let $T[i]$ be the first 32 bits after the decimal point in the binary representation of $|\sin(i)|$. Be careful, here the index starts at 1 because sin(0) is no good.

Rotation notation

If E is a 32-bit string, and $1 \leq n \leq 31$ then $E <<< n$ is a left rotational shift of E by n bits. So if $E = 01000000000000000111111111111111$ then $E <<< 3 = 00000000000000111111111111111010$.

More notation

We let + denote addition modulo 2^{32} where a word is considered to be the binary representation of an integer n with $0 \leq n \leq 2^{32} - 1$.

In computer science $x := y$ means set x equal to the value of y at this moment. So if $x = 4$ at a particular moment and you write $x := x + 1$ then after that step, $x = 5$.

Evaluating f

f takes as input, four 32-bit words that we denote A, B, C, D, and a 512 bit string X.

First the 4 words are stored for later: $AA := A_0$, $BB := B_0$, $CC := C_0$, $DD := D_0$.

Then there are 64 steps in four rounds.

Round 1 consists of 16 steps.

Let [abcd k s i] denote the operation $a := b + ((a + F(b, c, d) + X[k] + T[i]) <<< s)$.

Those 16 steps are

```
[ABCD  0  7  1] [DABC  1 12  2] [CDAB  2 17  3] [BCDA  3 22 4]
[ABCD  4  7  5] [DABC  5 12  6] [CDAB  6 17  7] [BCDA  7 22 8]
[ABCD  8  7  9] [DABC  9 12 10] [CDAB 10 17 11] [BCDA 11 22 12]
[ABCD 12  7 13] [DABC 13 12 14] [CDAB 14 17 15] [BCDA 15 22 16].
```

Round 2 consists of the next 16 steps.

Let [abcd k s i] denote the operation $a := b + ((a + G(b, c, d) + X[k] + T[i]) <<< s)$.

Those 16 steps are

```
[ABCD  1  5 17] [DABC  6  9 18] [CDAB 11 14 19] [BCDA  0 20 20]
[ABCD  5  5 21] [DABC 10  9 22] [CDAB 15 14 23] [BCDA  4 20 24]
[ABCD  9  5 25] [DABC 14  9 26] [CDAB  3 14 27] [BCDA  8 20 28]
[ABCD 13  5 29] [DABC  2  9 30] [CDAB  7 14 31] [BCDA 12 20 32]
```

Round 3 consists of the next 16 steps.

Let [abcd k s i] denote the operation $a := b + ((a + H(b, c, d) + X[k] + T[i]) <<< s)$.

Those 16 steps are

```
[ABCD  5  4 33] [DABC  8 11 34] [CDAB 11 16 35] [BCDA 14 23 36]
[ABCD  1  4 37] [DABC  4 11 38] [CDAB  7 16 39] [BCDA 10 23 40]
[ABCD 13  4 41] [DABC  0 11 42] [CDAB  3 16 43] [BCDA  6 23 44]
[ABCD  9  4 45] [DABC 12 11 46] [CDAB 15 16 47] [BCDA  2 23 48]
```

Round 4 consists of the last 16 steps.

[abcd k s i] denote the operation $a := b + ((a + I(b, c, d) + X[k] + T[i]) <<< s)$.

Those 16 steps are

```
[ABCD  0  6 49] [DABC  7 10 50] [CDAB 14 15 51] [BCDA  5 21 52]
[ABCD 12  6 53] [DABC  3 10 54] [CDAB 10 15 55] [BCDA  1 21 56]
[ABCD  8  6 57] [DABC 15 10 58] [CDAB  6 15 59] [BCDA 13 21 60]
[ABCD  4  6 61] [DABC 11 10 62] [CDAB  2 15 63] [BCDA  9 21 64]
```

Lastly, add in the saved words from the beginning.
A := A + AA, B := B + BB, C := C + CC, D := D + DD.

The output of f is the concatenation ABCD, which has 128 bits.

Clarification

Let us clarify the notation above. First let us look at Step 1. Before Step 1, we have $A = A_0 = 67\ 45\ 23\ 01$, $B = B_0 = \ldots$, $C = C_0 = \ldots$, $D = D_0 = \ldots$. Recall

[abcd k s i] denotes the operation $a := b + ((a + F(b, c, d) + X[k] + T[i]) <<< s)$. And Step 1 is [ABCD 0 7 1].

So that means

$A := B + ((A + F(B, C, D) + X[0] + T[1]) <<< 7)$
or
$A_1 := B_0 + ((A_0 + F(B_0, C_0, D_0) + X[0] + T[1]) <<< 7)$.

So first we evaluate $F(B_0, C_0, D_0)$. Then we turn it into an integer. Then we turn A_0, $X[0]$ and $T[1]$ into integers. Then we add the four integers together modulo 2^{32}. Recall $X[0]$ is the beginning of the 512-bit input message and $T[1]$ comes from sin(1). Take the result and convert it to a 32-bit word and shift left by 7. Turn that back into an integer and turn B_0 into an integer and add those two modulo 2^{32} to get A_1. Turn A_1 back into a 32-bit word. Now $A = A_1$, $B = B_0$, $C = C_0$, $D = D_0$.

Now we move onto Step 2. Recall

[abcd k s i] denotes the operation $a := b + ((a + F(b, c, d) + X[k] + T[i]) <<< s)$. And Step 2 is [DABC 1 12 2]. So that means

$D := A + ((D + F(A, B, C) + X[1] + T[2]) <<< 12)$ or

$D_1 := A_1 + ((D_0 + F(A_1, B_0, C_0) + X[1] + T[2]) <<< 12)$.

Now $A = A_1$, $B = B_0$, $C = C_0$, $D = D_1$.

Hash algorithm

We will call the input to the hash algorithm the message. Let us say that the (padded) message has $3 \cdot 512$ bits and consists of the concatenation of three 512-bit strings $S_1S_2S_3$. First we take S_1 and break it into 16 words: $X[0], \ldots, X[15]$.
We start with the buffers A = 0x67452301, B = 0xefcdab89, C = 0x98badcfe, D = 0x10325476. We go through the algorithm described above. After the 64th steps $A = A_{16}$, $B = B_{16}$, $C = C_{16}$, $D = D_{16}$. Then $A'_{16} = A_{16} + AA$, $B'_{16} = B_{16} + BB$, $C'_{16} = C_{16} + CC$, $D'_{16} = D_{16} + DD$. That gives the 128-bit output ABCD $= A'_{16}B'_{16}C'_{16}D'_{16}$. (Note, I consider the outputs of steps 61 - 64 to be $A_{16}, D_{16}, C_{16}, B_{16}$ and then after adding in $AA = A_0, BB = B_0, CC = C_0, DD = D_0$ the output is $A'_{16}B'_{16}C'_{16}D'_{16}$.)

Next, break S_2 into 16 words: $X[0], \ldots, X[15]$. These will usually be different from the $X[0], \ldots, X[15]$ above. We go through the algorithm described above except the initial values of A, B, C, D come from the output of the previous 64 steps $A = A'_{16}, BB = B'_{16}, C = C'_{16}, D = D'_{16}$, not 0x67452301, ... The the output of the first evaluation of f, namely $A'_{16}B'_{16}C'_{16}D'_{16}$, is (with S_2) the input of the second evaluation of f. Then let $AA = A'_{16}, \ldots$, do the 64 steps and add in $AA, \ldots$. There will again be a 128-bit output ABCD $= A'_{32}B'_{32}C'_{32}D'_{32}$.

Next, break S_3 into 16 words: $X[0], \ldots, X[15]$. We go through the algorithm described above except the initial values of A, B, C, D come from the output of the previous 64 steps, i.e. $A = A'_{32}, B = B'_{32}, C = C'_{32}, D = D'_{32}$.

There will again be a 128-bit output ABCD $= A'_{48}B'_{48}C'_{48}D'_{48}$, which is the output of the algorithm, known as the hash value.

Notes:

1) The message is always padded. First, it is padded so that the number of bits in the message is $448 \equiv -64 (\mathrm{mod}\, 512)$. To pad, you append at the end one 1 and then as many 0's as you need so that the message length will be $-64 (\mathrm{mod}\, 512)$. Let b be the length of the message before padding. If $b \equiv 448 (\mathrm{mod}\, 512)$ to begin with, then append one 1 and 511 0's. Write b in its 64-bit binary representation (so there will probably be a lot of 0's on the left). Now append this 64-bit binary representation of b to the right of our padded message so its length is a multiple of 512.

Let's give an example. Assume the original message has length 1700 bits. Now $1700 \equiv 164 (\mathrm{mod}\, 512)$. We have $448 - 164 = 284$. So we append one 1 and 283 0's on the right. Now the message has 1984 bits. Note $1984 \equiv 448 \equiv -64 (\mathrm{mod}\, 512)$. Now $1700 = (11010100100)_2$. So we append the 64 bit binary representation of 1700, namely
0000000000000000000000000000000000000000000000000000011010100100, to the right and the message now has length $1984 + 64 = 2048 = 4 \cdot 512$.

2) The function f_{hash} is complicated because within each step it cycles through bit-wise operations (F, G, H, I), shifting and additions modulo 2^{32} which operate on entire strings at once (and have carries, unlike bit-wise operations). The fact that all four buffers are constantly updated and the message pieces introduced slowly increases the safety also.

2) It was shown in 2004 (Wang, Feng, Lai, Yu) that MD5 does not have the strongly collision free property. However it may still have the two desired properties: one way and weakly collision free.

13.2 SHA3

The SHA-3 hash algorithm

Bertoni, Daemen (a designer of AES), Peeters and Van Assche submitted Keccak (pronounced *ketsak*) to NIST's hash competition and in 2012 it was selected as SHA-3 (Secure Hash Algorithm 3). Recall SHA-0, SHA-1, SHA-2 and MD5 are all designed similarly and SHA-0, SHA-1 and MD5 were shown not to have the strongly collision free property. SHA-3 is designed very differently from the others. Since many applications of hash algorithms, including certificates and TLS, do not need the strongly collision free property, many will continue using SHA-2 for a while.

Instead of explaining the full generality of Keccak, with its different possible parameters, we will describe what looks like will become the most commonly used. SHA-3 takes inputs of arbitrary length and gives outputs of length 256. The Keccak-256 hash algorithm uses a hash function f, which operates on strings of length 1600. Note, at this writing, NIST and the designers continue to tinker with SHA-3, which may be why the algorithm described below will not give you the sample output at Wikipedia.

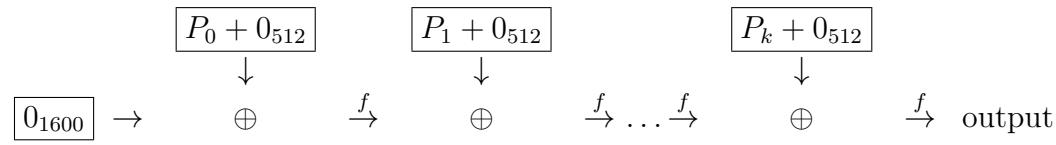
The input to the hash algorithm SHA-3 is padded until its length is a multiple of 1088 (we won't explain where 1088 came from beyond that fact that $1600 - 2 \cdot 256 = 1088$).

You must append at least two 1's, with as many 0's in between (possibly none) so that the padded length is a multiple of 1088.

Example

Let 'length' be the number of bits in the unpadded input message.
If length $\equiv -2$ mod 1088 then append 11.
If length $\equiv -1$ mod 1088 then append 10...01 with 1087 0's.
If length $\equiv 0$ mod 1088 then append 10...01 with 1086 0's.
End Example.

Let's say that the padded input message consists of $P_0, P_1, \ldots, P_k$ where each P_i is a string of 1088 bits. Let 0_ℓ denote a string of ℓ 0's. Let + denote concatenation.



The bits $0 \ldots 255$ of 'output' are the hash.

Now let us describe f. The input and output are both 1-dimensional arrays of length 1600. Let's call the input array $v[0 \ldots 1599]$. First you fill a 3-dimensional array $a[i][j][k]$, for $0 \le i \le 4$, $0 \le j \le 4$, $0 \le k \le 63$, with the entries of v (note, some sources switch i and j). We have $a[i][j][k] = v[64(5j + i) + k]$. Note a is the state that is updated.

Then, to a, we apply 24 rounds (indexed $0, \ldots, 23$), each of which is almost identical. Each round is $\iota \circ \chi \circ \pi \circ \rho \circ \theta$ (recall that means θ comes first). Only ι depends on the round index. After applying the rounds 24 times to the state, we take the final state and turn it back into a 1-dimensional array $v[0 \ldots 1599]$. Now let us describe the functions θ, ρ, π, χ and ι, that update the 3-dimensional array states.

13.2.1 theta

Let a_{in} be the input to θ and a_{out} be the output. For $0 \le i \le 4$, $0 \le j \le 4$, $0 \le k \le 63$, we have $a_{out}[i][j][k] = a_{in}[i][j][k] \oplus (\sum_{j'=0}^{4} a_{in}[i-1][j'][k]) \oplus (\sum_{j'=0}^{4} a_{in}[i+1][j'][k-1])$ where the sums are really $\oplus$'s and the first index (i) works mod 5 and the third index (k) works mod 64. Note, while applying θ, it is important not to continuously update values in the state $a[i][j][k]$.

13.2.2 rho

Let a_{in} be the input to ρ and a_{out} be the output. For $0 \le i \le 4$, $0 \le j \le 4$, $0 \le k \le 63$, we have $a_{out}[i][j][k] = a_{in}[i][j][k - (t+1)(t+2)/2]$ with $0 \le t < 24$ satisfying

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}^t \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} i \\ j \end{bmatrix} \bmod 5$$

or $t = -1$ if $i = j = 0$. (Note the 24 of "$t < 24$" is not the number of rounds. Instead it is $5^2 - 1$, which, with $t = -1$, gives the total number of possible vectors $[i; j]$ mod 5.) Note that the third index, $k - \frac{(t+1)(t+2)}{2}$, is computed mod 64.

For a given i and j, it might be easier to implement ρ using the fact that $\frac{(t+1)(t+2)}{2}$ mod 64 can be found in row i+1 and column j+1 of

$$\begin{bmatrix} 0 & 36 & 3 & 41 & 18 \\ 1 & 44 & 10 & 45 & 2 \\ 62 & 6 & 43 & 15 & 61 \\ 28 & 55 & 25 & 21 & 56 \\ 27 & 20 & 39 & 8 & 14 \end{bmatrix}.$$

13.2.3 pi

Let a_{in} be the input to π and a_{out} be the output. For $0 \le i \le 4$, $0 \le j \le 4$, $0 \le k \le 63$, we have $a_{out}[i][j][k] = a_{in}[i'][j'][k]$ where

$$\begin{bmatrix} i \\ j \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} i' \\ j' \end{bmatrix}.$$

Or (more easy to implement) for $0 \le i' \le 4$, $0 \le j' \le 4$, $0 \le k \le 63$, we have $a_{out}[j'][2i' + 3j'][k] = a_{in}[i'][j'][k]$, where the second index works mod 5.

13.2.4 chi

Let a_{in} be the input to χ and a_{out} be the output. For $0 \le i \le 4$, $0 \le j \le 4$, $0 \le k \le 63$, we have $a_{out}[i][j][k] = a_{in}[i][j][k] \oplus ((a_{in}[i+1][j][k] \oplus 1)(a_{in}[i+2][j][k]))$. In terms of order of operations, do the $\oplus 1$ first, then the multiplication, then the $\oplus$. This first index (i) works mod 5.

13.2.5 iota

Let a_{in} be the input to π and a_{out} be the output. Now ι depends on the round index $0 \le i_r \le 23$. For $0 \le i \le 4$, $0 \le j \le 4$, $0 \le k \le 63$, we have $a_{out}[i][j][k] = a_{in}[i][j][k] \oplus bit[i][j][k]$.

Now for $0 \le \ell \le 6$, we have $bit[0][0][2^\ell - 1] = rc[\ell + 7i_r]$ and all other values of bit are 0. So for each round, we will change at most 7 of the 1600 bits of a.

Now $rc[t]$ is the constant term (bit) of x^t reduced in $\mathbf{F}_2[x]/(x^8 + x^6 + x^5 + x^4 + 1)$.

Example

For round 0 we have

ℓ	$2^\ell - 1$	$t = \ell + 7i_r$	x^t	$bit[0][0][2^\ell - 1]$
0	0	0	1	1
1	1	1	x	0
2	3	2	x^2	0
3	7	3	x^3	0
4	15	4	x^4	0
5	31	5	x^5	0
6	63	6	x^6	0

So we only $\oplus 1$ to $a[0][0][0]$ in round 0. The other 1599 bits remain the same.

For round 1 we have

ℓ	$2^\ell - 1$	$t = \ell + 7i_r$	x^t	$bit[0][0][2^\ell - 1]$
0	0	7	x^7	0
1	1	8	$x^8 = x^6 + x^5 + x^4 + 1$	1
2	3	9	$x^9 = x^7 + x^6 + x^5 + x$	0
3	7	10	$x^{10} = x^8 + x^7 + x^6 + x^2$	
			$= (x^6 + x^5 + x^4 + 1) + x^7 + x^6 + x^2$	
			$= x^7 + x^5 + x^4 + x^2 + 1$	1
4	15	11	$\vdots$	1
5	31	12		0
6	63	13		0

etc. So in round 1, we $\oplus 1$ to $a_{in}[0][0][1]$, $a_{in}[0][0][7]$ and $a_{in}[0][0][15]$ and the other 1597 bits remain the same.
End example.

It's nicer to use the associated Linear (Feedback) Shift Register or LSR (LFSR) to compute $rc[t]$. Let's start with $w = w[0 \dots 7] = [rc[0], rc[1], rc[2], rc[3], rc[4], rc[5], rc[6], rc[7]]$ $= [1, 0, 0, 0, 0, 0, 0, 0]$.
We let $rc[0] = w[0] = 1$.
Then we update $w = [w[1], w[2], w[3], w[4], w[5], w[6], w[7], w[0] \oplus w[4] \oplus w[5] \oplus w[6]] = [0, 0, 0, 0, 0, 0, 0, 1]$

Where did $w[0] \oplus w[4] \oplus w[5] \oplus w[6]$ come from? Recall $x^8 = x^6 + x^5 + x^4 + x^0$.

We let $rc[1] = w[0] = 0$. Then we update
$w = [w[1], w[2], w[3], w[4], w[5], w[6], w[7], w[0] \oplus w[4] \oplus w[5] \oplus w[6]] = [0, 0, 0, 0, 0, 0, 1, 0]$.
We let $rc[2] = w[0] = 0$. Then we update
$w = [w[1], w[2], w[3], w[4], w[5], w[6], w[7], w[0] \oplus w[4] \oplus w[5] \oplus w[6]] = [0, 0, 0, 0, 0, 1, 0, 1]$.
We let $rc[3] = w[0] = 0$. Then we update
$w = [w[1], w[2], w[3], w[4], w[5], w[6], w[7], w[0] \oplus w[4] \oplus w[5] \oplus w[6]] = [0, 0, 0, 0, 1, 0, 1, 1]$.
Then we update $w = [0, 0, 0, 1, 0, 1, 1, 0]$.
Then we update $w = [0, 0, 1, 0, 1, 1, 0, 0]$.

Eventually we get $rc = [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0 \dots$ just like when we used the finite field in the above example.

In pseudocode

```
w=[1,0,0,0,0,0,0,0];
rc[0]=w[0];
for i=1,t do
    w=[w[1],w[2],w[3],w[4],w[5],w[6],w[7],w[0]+w[4]+w[5]+w[6]];
    rc[i]=w[0]
end for
```

Cool facts about LSR's. Note that the output of the LSR is a random looking binary string. This is used in cryptography and elsewhere. It turns out that since x generates $\mathbf{F}_2[x]/(x^8 + x^6 + x^5 + x^4 + 1)^*$, the output of the LSR will repeat every $2^8 - 1$ bits. In addition, if you put those bits in a circle, clockwise, then every byte, other than 00000000, will appear exactly once (reading clockwise).

LSR's were used as random bit generators for stream ciphers in the 1970's (together the finite field and the initial state were the key). Recall that the L stands for linear and with a small amount of known plaintext, you can use linear algebra to crack this stream cipher (cryptographers are still embarrassed). Non-linear shift registers are sometimes used for stream ciphers. There you have multiplication of bits as well as $\oplus$.

14 Signatures and authentication

A bank in Maine agrees to hire an Idaho architecture firm to design their next building. The bank of Maine writes up a contract. Everything is occuring electronically so the Idaho firm wants a digital signature on it. How can this be done? You want the Idaho firm to be sure it's the bank that signed it and not some hacker impersonating that bank.

Making yourself sure that a message came from the proper sender is called authentication. The solution is signatures and certificates. Signatures connect a message with a public key. Certificates connect a public key with an entity. You can use public-key cryptography for signatures.

14.1 Signatures with RSA

Signatures with RSA: Say George Bush (=G) and Tony Blair =(T) are using RSA. In a public key cryptosystem, there is no shared key that only Bush and Blair have. So Osama bin Laden (=L) could e-mail Bush an AES key, encrypted with Bush's public RSA key and then send Bush the message "Lay off Bin Laden, sincerely, Tony Blair" encrypted with AES. How would Bush know who it's from. He must demand a signature.

Case 1, T sends PT msg M to G, no need to encrypt. At end signs M_2 ='Tony Blair'. Wants to make sure G knows its from him. T then computes $M_2^{d_T} \bmod n_T = S$. Could add to end of mgs. Only T can do that. G can verify it's from him by computing $S^{e_T} \bmod n_T$ to get M_2. Eve can read signature too. Also L can cut and paste signature to end of his own message.

Case 2. T creates an AES key and sends $(\text{key}_{\text{AES}})^{e_G} \bmod n_G$ to G. T encrypts message M for G using AES and sends CT to G. T hashes M to get hash(M). T computes $\text{hash}(M)^{d_T} \bmod n_T = S$ and sends to G. G decrypts using RSA to get key_{AES}. G decrypts CT with AES to get M. G hashes (decrypted) M to get hash(M). G compute $S^{e_T} (\bmod n_T)$ and confirms it equals the hash(M) he earlier computed. Only T could have created an S so that $S^{e_T} = \text{hash}(M)$. If it does, then G knows 1) the message was sent by whoever owns the keys e_T and n_T (authentication) and that it was not tampered with by anyone else (integrity). Note that L has access to hash(M). G and T may not want that. So T may encrypt S with RSA or AES.

Case 3. Same as case 2, but T and G do not want L to have access to hash(M) (for whatever reason - maybe T will resend M to someone else). So T wants to encrypt S. Could use AES or RSA for that. For Case 3, uses RSA (AES would be more common in real life).

Case 3a. Assume $n_T < n_G$. Tony wants to sign and encrypt hash(M) using RSA for both. Tony computes $[hash(M)^{d_T} \bmod n_T]^{e_G} \bmod n_G = Y$ and sends that to G. Now G computes $[Y^{d_G} \bmod n_G]^{e_T} \bmod n_T$. Then he checks to see whether or not this equals hash(M).

Case 3b. Assume $n_T > n_G$. There is a problem now. Example: Assume $n_T = 100000$ and $n_G = 1000$ (bad RSA numbers). Assume that $hash(M)^{d_T} \bmod n_T = 10008$. Now when encrypting for G, computations done mod 1000. So after G decrypts (and before he checks the signature) he'll get 8. Then he doesn't know if that should be 8, 1008, 2008, ..., 99008. Solution. Recall message is from T to G. If $n_T > n_G$ then encrypt first then sign. So T computes $[hash(M)^{e_G} \bmod n_G]^{d_T} \bmod n_T = Z$ and sends to G. Now G computes $[Z^{e_T} \bmod n_T]^{d_G} \bmod n_G$.

When sending: Always small n then big n.

Signatures with RSA. Remember: When sending, small n then big n.

Let's say that $n_G = 221, e_G = 187, d_G = 115$ and $n_T = 209, e_T = 191, d_T = 131$.

T wants to sign and encrypt the hash 97 for G. What does he do? In sending, you work with the small n then the big n.

First sign: $97^{d_T} \bmod n_T = 97^{131} \bmod 209 = 108$

Then encrypt: $108^{e_G} \bmod n_G = 108^{187} \bmod 221 = 56$

T sends 56 to G; the enemy sees 56.

G receives 56. The receiver works with the big n and then the small n (since he's undoing).

$56^{d_G} \bmod n_G = 56^{115} \bmod 221 = 108$

$108^{e_T} \bmod n_T = 108^{191} \bmod 209 = 97.$

Now G wants to sign and encrypt the hash 101 for T. In sending, work with small n then big n.

First encrypt: $101^{e_T} \bmod n_T = 101^{191} \bmod 209 = 112$

Then sign: $112^{d_G} \bmod n_G = 112^{115} \bmod 221 = 31$

G sends 31 to T, the enemy sees 31.

T receives 31. The receiver works with the big n and then the small n.

$31^{e_G} \bmod n_G = 31^{187} \bmod 221 = 112$

$112^{d_T} \bmod n_T = 112^{131} \bmod 209 = 101.$

Case 4. Easier (and used frequently on web). Tony encrypts the signed hash using AES instead of RSA.

14.2 ElGamal Signature System and Digital Signature Standard

ElGamal signature scheme (basis for slightly more complicated Digital Signature Standard, DSS). Each user has a large prime p a generator g of $\mathbf{F}_p^*$ a secret key number a and a public key g^a. These should not change frequently.

Let's say Alice wants to sign the hash of a message and send that to Bob. Let S be the encoding of the hash as a number with $1 < S < p$. Alice picks a random session k with $1 << k << p$ and $\gcd(k, p-1) = 1$ and reduces $g^k = r \in \mathbf{F}_p$.

Then she solves $S \equiv a_A r + kx (\mathrm{mod}\, p-1)$ for x. Note x depends both on S and the private key a_A. So $k^{-1}(S - a_A r) \mathrm{mod}\, p - 1 = x$. Note that $g^S \equiv g^{a_A r + kx} \equiv g^{a_A r} g^{kx} \equiv (g^{a_A})^r (g^k)^x \equiv (g^{a_A})^r r^x (\mathrm{mod}\, p)$.

Alice sends r, x, S to Bob as a signature. Bob confirms it's from Alice by reducing $(g^{a_A})^r r^x (\mathrm{mod}\, p)$ and $g^S (\mathrm{mod}\, p)$ and confirms they are the same. Now Bob knows it is from Alice (really whomever has a_A). Why? Only Alice could have solved $k^{-1}(S - a_A r) \mathrm{mod}\, p - 1 = x$ since only she knows a_A. It seems the only way someone can impersonate Alice and create sucha triple is if s/he can solve the FFDLP and find a_A.

Example. Let's say the hash of a message is $316 = S$. Alice's uses $p = 677$, $g = 2$ and private key $a_A = 307$. So her public key is $2^{307} \mathrm{mod}\, 677 = 498$. So $g^{a_A} = 498$. She picks the session $k = 401$ (OK since $\gcd(k, p-1) = 1$).

Alice computes $r = g^k = 2^{401} \mathrm{mod}\, p = 616$ so $r = 616$. She solves $S = a_A r + kx (\mathrm{mod}\, p - 1)$ or $316 = 307 \cdot 616 + 401 \cdot x (\mathrm{mod}\, 676)$. So $401^{-1}(316 - 307 \cdot 616) \mathrm{mod}\, 676 = x$. Now $401^{-1} \mathrm{mod}\, 676 = 617$. So $617(316 - 307 \cdot 616) \mathrm{mod}\, 676 = 512 = x$. Alice sends $(r, x, S) = (616, 512, 316)$.

Bob receives and computes $g^S = 2^{316} \mathrm{mod}\, 677 = 424$. $(g^{a_A})^r = 498^{616} \mathrm{mod}\, 677 = 625$. $r^x = 616^{512} \mathrm{mod}\, 677 = 96$. Confirms $g^{a_A r} g^{kx} \equiv 625 \cdot 96 \equiv 424$ is the same as $g^S \mathrm{mod}\, 677 = 424$. End example.

To get DSS, pick a prime $\ell | p - 1$ where $\ell > 10^{55}$ but significantly smaller than p. You choose g with $g^\ell = 1$, $g \neq 1$. This speeds up the algorithm. For further details, see Section 31.1

14.3 Schnorr Authentication and Signature Scheme

Let p be a prime and ℓ be a prime such that $\ell | p - 1$. Pick a such that $a^\ell \equiv 1 (\mathrm{mod}\, p)$. The numbers a, p, ℓ are used by everyone in the system. Each person has a private key s and a public key $v \equiv a^{-s} (\mathrm{mod}\, p)$.

Authentication: Peg (the prover) picks a random $r < \ell$ and computes $x \equiv a^r (\mathrm{mod}\, p)$. She sends x to Vic (the verifier). Vic sends Peg a random number e with $0 \leq e \leq 2^t - 1$ (t will be explained later). Peg computes $y \equiv r + s_P e (\mathrm{mod}\, \ell)$ and sends y to Vic. Vic reduces $a^y v_P^e (\mathrm{mod}\, p)$ and verifies that it equals x. This authenticates that the person who sent y is the person with public key v_P.

Schnorr suggests now using $p \approx 2^{512}$, $\ell \approx 2^{140}$ and $t = 72$.

Signature: Alice wants to sign a message M. She picks a random $r' < \ell$ and computes $x' \equiv a^{r'} (\mathrm{mod}\, p)$. Alice concatenates M and x' and hashes the concatenation to get e'. Alice computes $y' \equiv r' + s_A e' (\mathrm{mod}\, \ell)$. The signature is the triple x', e', y'. Bob has M (either it was encrypted for him or sent in the clear). Bob computes $a^{y'} v_A^{e'} (\mathrm{mod}\, p)$ and verifies it equals x'. Then he verifies that e' is the hash of the concatenation of M and x'.

Computing $y \equiv r + s_P e (\mathrm{mod}\, \ell)$ is fast as it involves no modular inversions. Alice can compute x at any earlier time which speeds up signature time. The security seems based on the difficulty of the FFDLP. The signatures are shorter than with RSA for the same security (since signatures work mod ℓ while the FFDLP most be solved in the larger $\mathbf{F}_p$).

14.4 Pairing based cryptography for digital signatures

Pairing based cryptography is used for short digital signatures (short refers to the length of the key), one round three-way key exchange and identity-based encryption. We will just do signatures.

Let p be a large prime. Let G be a finite dimensional $\mathbf{F}_p$-vector space. So $G \cong \mathbf{Z}/p\mathbf{Z} \times \mathbf{Z}/p\mathbf{Z} \times \cdots \mathbf{Z}/p\mathbf{Z}$. Let H be a cyclic group of order p (a one-dimensional $\mathbf{F}_p$-vector space). We will treat G as a group under addition and H as a group under multiplication. Assume the DLP is hard in both groups. A pairing is a map from $G \times G \to H$. Note, if $g_1, g_2 \in G$ then we denote their pairing by $\langle g_1, g_2 \rangle \in H$. Assume there is a pairing $G \times G \to H$ with the following four properties.
i) For all $g_1, g_2, g_3 \in G$ we have $\langle g_1 + g_2, g_3 \rangle = \langle g_1, g_3 \rangle \langle g_2, g_3 \rangle$.
ii) For all $g_1, g_2, g_3 \in G$ we have $\langle g_1, g_2 + g_3 \rangle = \langle g_1, g_2 \rangle \langle g_1, g_3 \rangle$.
iii) Fix g. If $\langle g, g_1 \rangle = \mathrm{id} \in H$ for all $g_1 \in G$, then g is the identity element of G.
iv) The pairing is easy to evaluate.
(Note, i) - iii) says that m is a non-degenerate bilinear pairing.)

Let $g \in G$ be published and used by everyone.

Alice chooses secret x and publishes $j = g^x$ in a directory:

$\vdots$
Akuzike, i
Alice, j
Arthur, k
$\vdots$

Or, Alice just comes up with a new secret random x for this session and sends $j = g^x$ to Bob.

Alice wants to digitally sign a message $m \in G$ for Bob.

Alice signs message $m \in G$ by computing $s = m^x \in G$.

Alice sends Bob m and s.

Bob confirms it's from Alice by verifying $\langle g, s \rangle = \langle j, m \rangle$.

Proof: $\langle g, s \rangle = \langle g, m^x \rangle = \langle g, m \rangle^x = \langle g^x, m \rangle = \langle j, m \rangle$. End proof.

Bob knows must be from Alice. Only Alice (who knows x) could have found s such that $\langle g, s \rangle = \langle j, m \rangle$.

Implementation. We need to find groups G, H and a pairing with the four properties.

Let $\mathbf{F}_q$ be a finite field and E be an elliptic curve whose coefficients are in $\mathbf{F}_q$. Let $|E(\mathbf{F}_q)| = pn$ where p is a large prime and n is small. Assume $\gcd(p, q) = 1$. Fact: There is a finite extension $\mathbf{F}_{q^r}$ of $\mathbf{F}_q$ such that $E(\mathbf{F}_q)$ has a subgroup isomorphic to $\mathbf{Z}/p\mathbf{Z} \times \mathbf{Z}/p\mathbf{Z}$. We will denote this subgroup by $E[p]$. Fact: Necessarily $r > 1$. Fact: The group $\mathbf{F}_{q^r}^*$ has a

(unique) subgroup of order p. Fact: If $p \not| q-1$ then r is the order of q in $\mathbf{Z}/p\mathbf{Z}^*$. In other words, r is the smallest positive integer such that $p|q^r-1$.

We want to create a pairing from $E[p] \times E[p] \to \mathbf{F}_{q^r}^*$. Actually, the image of the pairing will land in the subgroup of $\mathbf{F}_{q^r}^*$ of order p. Let $R, T \in E[p]$. Want $\langle R, T \rangle$. Find function f_T such that $\mathrm{div}(f_T) = pT - p\mathcal{O}$. That means that $f_T(x,y)$ describes a curve which intersects E only at T and with multiplicity p. To evaluate $\langle R, T \rangle$, we evaluate $(f(U)/f(V))^k$ where $U - V = R$ on the elliptic curve and U and V did not appear in the construction of f_T. Also $k = (q^r-1)/p$.

How to find such an f_T. If $f(x,y)$ is a function of two variables then the curve $f(x,y) = 0$ meets E in certain points with certain multiplicities. The divisor of $f(x,y)$ is the formal sum of those points (not the sum on the elliptic curve) minus the same number of $\mathcal{O}$-points.

Example: E is given by $y^2 = x^3 + 17$. Let $f(x,y) = y + 3x - 1$. Then $f(x,y) = 0$ is $y + 3x - 1 = 0$ or $y = -3x + 1$. To find where line intersects E we solve the equations simultaneously. $y^2 = x^3 + 17$ and $y^2 = (-3x+1)^2$, or $x^3 + 17 = 9x^2 - 6x + 1$ or $x^3 - 9x^2 + 6x - 16 = 0$ or $(x-8)(x-2)(x+1) = 0$. So the line meets the curve in three points, with $x = -1, 2, 8$. To get the y-coordinates we use $y = -3x + 1$. So we get $\mathrm{div}(y + 3x - 1) = (-1, 4) + (2, -5) + (8, -23) - 3(\mathcal{O})$. Let's find $\mathrm{div}(x-2)$. Then $x - 2 = 0$ is $x = 2$. Solve that simultaneously with $y^2 = x^3 + 17$ and get $y^2 = 25$ or $y = \pm 5$. So $\mathrm{div}(x-2) = (2,5) + (2,-5) - 2(\mathcal{O})$.

Fact: If you add (on the elliptic curve) the points in a divisor, you will get $\mathcal{O}$. We will write $\oplus$ for addition on the elliptic curve and $\ominus$ for subtraction on the elliptic curve.

Note: If $\mathrm{div}(fg) = \mathrm{div}(f) + \mathrm{div}(g)$ and $\mathrm{div}(1/f) = -\mathrm{div}(f)$. So $\mathrm{div}(f/g) = \mathrm{div}(f) - \mathrm{div}(g)$.

So $\mathrm{div}(y + 3x - 1)/(x-2) = (-1,4) + (8,-23) - (2,5) - (\mathcal{O})$. And $(-1,4) \oplus (8,-23) \ominus (2,5) \ominus \mathcal{O} = \mathcal{O}$. End example.

Algorithm to find f_T:

Let $p = a_n 2^n + a_{n-1} 2^{n-1} + \ldots + a_1 2 + a_0$ where $a_i \in \{0, 1\}$ (note $a_n = a_0 = 1$). Let $f_T := 1$. Let $R := T$.

For $i = n-1, \ldots, 0$ do

i) Let $f_T := f_T^2$.

ii) Let $l_i = 0$ be the tangent line at R. Note, we have $\mathrm{div}(l_i) = 2(R) + (S_i) - 3(\mathcal{O})$.

iii) Let $v_i := x - x(S_i)$.

iv) Let $R := \ominus S_i$.

v) Let $f_T := (l_i/v_i) f_T$.

vi) If $a_i = 1$, let $m_i = 0$ be the line through R and T. (Note that if $R = T$ then $m_i = 0$ is the tangent line.) Note, we have $\mathrm{div}(m_i) = (R) + (T) + (U_i) - 3(\mathcal{O})$. If $a_i = 0$ let $m_i := 1$.

vii) If $i > 0$ and $a_i = 1$, let $w_i := x - x(U_i)$. If $a_i = 0$ let $w_i := 1$. If $i = 0$ let $w_0 := 1$.

viii) If $a_i = 1$, let $R := \ominus U_i$. If $a_i = 0$, then R remains unchanged.

ix) Let $f_T := (m_i/w_i) f_T$.

End do.

Output f_T. Note, $\mathrm{div}(f_T) = p(T) - p(\mathcal{O})$.

End algorithm.

Example. Let $E : y^2 = x^3 + 3x + 2$ over $\mathbf{F}_{11}$. We have $|E(\mathbf{F}_{11}) = 13$. Let $T = (2, 4)$. Note T has order 13. So $13T = 0$. We have $13 = (1101)_2$. Let $f_T = 1$, $R = T$.

$i = 2$	
$f_T = 1^2 = 1$	
$l_2 = y - 6x + 8$	$\mathrm{div}(y - 6x + 8) = 2(2, 4) + (10, 8) - 3\mathcal{O} = 2(T) + (-2T) - 3(\mathcal{O})$ (Note $y = 6x - 8$ is the tangent line at $(2, 4)$.)
$v_2 = x - 10$	$\mathrm{div}(x - 10) = (10, 8) + (10, 3) - 2\mathcal{O} = (-2T) + (2T) - 2(\mathcal{O})$
$R = (10, 3) = (2T)$	
$f_T = (\frac{l_2}{v_2})1$	$\mathrm{div}(f) = 2(2, 4) - (10, 3) - (\mathcal{O}) = 2(T) - (2T) - (\mathcal{O})$
$a_2 = 1$	
$m_2 = y + 7x + 4$	$\mathrm{div}(y + 7x + 4) = (10, 3) + (2, 4) + (4, 1) - 3(\mathcal{O})$ $= (2T) + (T) + (-3T) - 3(\mathcal{O})$
$w_2 = x - 4$	$\mathrm{div}x - 4 = (4, 1) + (4, 10) - 2(\mathcal{O}) = (-3T) + (3T) - 2(\mathcal{O})$
$R = (4, 10) = (3T)$	
$f_T = (\frac{l_2 m_2}{v_2 w_2})$	$\mathrm{div}(f_T) = 3(2, 4) - (4, 10) - 2(\mathcal{O}) = 3(T) - (3T) - 2(\mathcal{O})$
$i = 1$	
$f_T = (\frac{l_2^2 m_2^2}{v_2^2 w_2^2})$	$\mathrm{div}(f_T) = 6(2, 4) - 2(4, 10) - 4(\mathcal{O}) = = 6(T) - 2(3T) - 4(\mathcal{O})$
$l_1 = y + 9x + 9$	$\mathrm{div}l_2 = 2(4, 10) + (7, 5) - 3(\mathcal{O}) = 2(3T) + (-6T) - 3(\mathcal{O})$
$v_1 = x - 7$	$\mathrm{div}v_2 = (7, 5) + (7, 6) - 2(\mathcal{O}) = (-6T) + (6T) - 2(\mathcal{O})$
$R = (7, 6) = (6T)$	
$f_T = (\frac{l_2^2 m_2^2 l_1}{v_2^2 w_2^2 v_1})$	$\mathrm{div}(f_T) = 6(2, 4) - (7, 6) - 5(\mathcal{O}) = 6(T) - (6T) - 5(\mathcal{O})$
$a_1 = 0$	
$m_1 = 1$	
$w_1 = 1$	
$R = (7, 6) = (6T)$	
$f_T = (\frac{l_2^2 m_2^2 l_1 m_1}{v_2^2 w_2^2 v_1 w_1})$	$\mathrm{div}(f_T) = 6(2, 4) - (7, 6) - 5(\mathcal{O}) = 6(T) - (6T) - 5(\mathcal{O})$
$i = 0$	
$f_T = (\frac{l_2^4 m_2^4 l_1^2 m_1^2}{v_2^4 w_2^4 v_1^2 w_1^2})$	$\mathrm{div}(f_T) = 12(2, 4) - 2(7, 6) - 10(\mathcal{O}) = 12(T) - 2(6T) - 10(\mathcal{O})$
$l_0 = y + 4x + 10$	$\mathrm{div}(l_0) = 2(7, 6) + (2, 4) - 3(\mathcal{O}) = 2(6T) + (-12T) - 3(\mathcal{O})$
$v_0 = x - 2$	$\mathrm{div}(v_0) = (2, 4) + (2, 7) - 2(\mathcal{O}) = (-12T) + (12T) - 2(\mathcal{O})$
$R = (2, 7) = (12T)$	
$f_T = (\frac{l_2^4 m_2^4 l_1^2 m_1^2 l_0}{v_2^4 w_2^4 v_1^2 w_1^2 v_0})$	$\mathrm{div}(f_T) = 12(2, 4) - (2, 7) - 11(\mathcal{O}) = 12(T) - (12T) - 11(\mathcal{O})$
$m_0 = x - 2$	$\mathrm{div}(x - 2) = (2, 7) + (2, 4) - 2(\mathcal{O}) = (12T) + (T) - 2(\mathcal{O})$
$w_0 = 1$	
$f_T = (\frac{l_2^4 m_2^4 l_1^2 m_1^2 l_0 m_0}{v_2^4 w_2^4 v_1^2 w_1^2 v_0 w_0})$	$\mathrm{div}(f_T) = 13(2, 7) - 13(\mathcal{O}) = 13(T) - 13(\mathcal{O})$

So $f_T = (\frac{l_2^4 m_2^4 l_1^2 m_1^2 l_0 m_0}{v_2^4 w_2^4 v_1^2 w_1^2 v_0 w_0})$

Example: $E : y^2 = x^3 + x + 4$ over $\mathbf{F}_7$. Now $|E(\mathbf{F}_7)| = 10$. Let $p = 5$. The smallest positive power of 7 that is $1 (\mathrm{mod}\, 5)$ is 4. Let $K = \mathbf{F}_{7^4} = \mathbf{F}_7[t]/(t^4 + t + 1)$. The points $R = (6t^3 + t^2 + 1, 3t^3 + 5t^2 + 6t + 1)$ and $T = (4, 4)$ each have order 5. We want to find

$\langle R, T\rangle$. First we must find f_T, where $\text{div}(f_T) = 5(4,4) - 5\mathcal{O}$. We have $f_T = (l_1^2 l_0)/(v_1^2)$ where $l_1 = y - 4$, $l_0 = y + 4x + 1$, $v_1 = x - 6$. We must find U, V such that $U \ominus V = R$ and U, V were not involved in finding f_T.

The points that occurred while finding f_T were $(4,4), (4,3), (6,3), (6,4), \mathcal{O}$. We can not let $U = R$ and $V = \mathcal{O}$, but we can let $U = 2R = (6t^3 + 6t^2 + 3t, 5t^3 + 2t^2 + 4t + 6)$ and $V = R$. (Ed, find U and V). We evaluate $f_T(U)/f_T(V)$.
$f_T(U) = (l_1(U))^2 l_0(U)/(v_1(U))^2$.
Now $l_1(U) = (y-4)(U) = (5t^3 + 2t^2 + 4t + 6 - 4)$.
$l_0(U) = (y + 4x + 1)(U) = (5t^3 + 2t^2 + 4t + 6 + 4(6t^3 + 6t^2 + 3t) + 1)$.
$v_1(U) = (x - 6)(U) = (6t^3 + 6t^2 + 3t - 6)$.
$f_T(U) = 6t^3 + 4t^2 + 3t + 1$.
$l_1(V) = (3t^3 + 5t^2 + 6t + 1 - 4)$.
$l_0(V) = (3t^3 + 5t^2 + 6t + 1 + 4(6t^3 + t^2 + 1) + 1)$.
$v_1(V) = (6t^3 + t^2 + 1 - 6)$.
$f_T(V) = t^3 + 5t^2 + t + 1$
$f_T(U)/f_T(V) = 4t^3 + 6t^2$
Now $k = (7^4 - 1)/5 = 480$.
$(f_T(U)/f_T(V))^{480} = 5t^3 + 2t^2 + 6t$.
Note that $(5t^3 + 2t^2 + 6t)^5 = 1$, so it is in the subgroup of order 5. End example.

On an elliptic curve, g^x means xg where x is an integer and g is a point on the elliptic curve.

Applications of Cryptography

15 Public Key Infrastructure

Public key infrastructure (PKI) enables a web of users to make sure of the identities of the other users. This is authentication. If Alice wants to authenticate Bob's identity, then a PKI will have software at Alice's end, and at Bob's end, hardware. A PKI will use certification chains or a web-of-trust and use certain protocols. Below we will explain certification chains and web-of-trust.

15.1 Certficates

Ed uses his Bank of America (BA) ATM card in the Mzuzu branch of the National Bank of Malawi (MB). MB connects to BA. MB creates a random DES key and encrypts it with BA's RSA key and sends to BA. MB will encrypt the message "remove 6000MK from Ed's account and send Bank of Malawi 6000MK to replace our cash" with DES. How does MB know that the RSA public key really belongs to BA and not some hacker in between?

Verisign in Mountain View is a certification authority (CA). They are owned by Symantec and own Geotrust. Here is a simplified sketch of how it works.

BA has created a public key and a document saying "the RSA public key X belongs to Bank of America". Then BA and their lawyers go to a reputable notary public with

identification documents. The notary notarizes the public key document, which is sent to Verisign.

Verisign uses Verisign's private key to sign the following document:
Version: V3
Serial Number: *Hex string*
Issuer: Verisign
Valid from: April 7, 2013
Valid to: April 6, 2014
Subject: bankofamerica.com
Signature Algorithm: SHA-1 (hash)/RSA
Public key: n_{BA}, e_{BA} (often n_{BA} written as hex string)
Certification path: Bank of America, Verisign.
Hash (Fingerprint): SHA-1(everything above this in certificate).
Signature value: $\text{Hash}^{d_{\text{Ver}}} \bmod n_{\text{Ver}}$.

MB also has a Verisign's certificate. Both MB and BA trust Verisign and its public keys. At the start, MB and BA exchange certificates. Each verifies the signature at the end of the other's cert using Verisign's public keys.

There's actually a certification tree with Verisign at the top. At a bank, you may have a sub-CA who uses his or her public key to sign certificates for each branch.

Let's say in the above example that MB's certificate is signed by the Bank of Malawi's sub-CA, and BoM's certificate is signed by Verisign. BA will see on MB's certificate that BoM is above it and Verisign above that. BA will visit all three certificates. Let hash_{BoM} be the hash of BoM's certificate and hash_{MB} be the hash of MB's certificate.
BoM certificate
Public key: n_{BoM}, e_{BoM}
Certification path: BoM, Verisign.
Hash: Hash_{BoM} (i.e. Hash of everything above this).
Signature value: $\text{Hash}_{BoM}^{d_{\text{Ver}}} \bmod n_{\text{Ver}}$.

MB certificate
Public key: n_{MB}, e_{MB}
Certification path: MB, BoM, Verisign.
Hash: Hash_{MB} (i.e. Hash of everything above this).
Signature value: $\text{Hash}_{MB}^{d_{\text{BoM}}} \bmod n_{\text{BoM}}$.

BA hashes top of BoM's certificate and compares with Hash_{BoM}. BA finds n_{Ver}, e_{Ver} on Verisign's certificate. Then BA computes $\text{signature}_{\text{BoM}}^{e_{Ver}} \bmod n_{Ver}$ and compares it with hash_{BoM}. If they are the same, then BA trusts that n_{BoM}, e_{BoM} actually belong to BoM.

Then BA hashes MB certificate and compares with Hash_{MB}. Then BA computes $\text{hash}_{\text{MB}}^{e_{BoM}} \bmod n_{BoM}$ and compares it with hash_{MB} at the bottom of MB's certificate. If they are the same, then BA trusts that n_{MB}, e_{MB} actually belong to MB.

In practice, there are different levels of certification. For a serious one, you really do show ID. You can get a less serious certificate that basically says "the e-mail address eschaefer@gmail.com and the public key n_E, e_E are connected". This less serious one does not certify that the e-mail address is provably connected to the person Ed Schaefer.

Not everyone uses Verisign. The Turkish government and a German bank each have their own CA's. Each can certify the other's certificate. This is called cross-certification.

Verisign can revoke a certificate at any time. Verisign keeps a list on-line of revoked certificates. One reason Verisign would revoke a certificate is if a public key is compromised.

What is the most likely way for a public key to be compromised? From most to least common: 1) Theft, 2) bribery, 3) hacking, 4) cryptanalysis (no known cases).

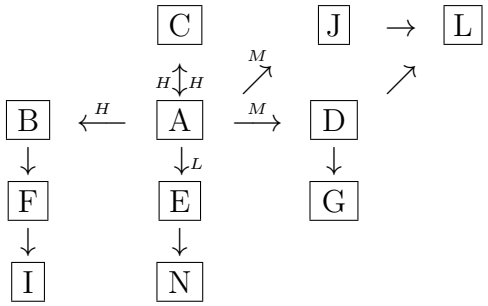
When a URL says https: it has a certificate. To see certificate in Mozilla, left click on box to left of URL. To see certificate in Internet explorer, right click on web page and left click on *properties*. Internet explorer concatenates n, e and e is last 6 hex symbols. Note e is often 3, 17 or 65537 (each of which is very fast with respect to the repeated squares algorithm).

15.2 PGP and Web-of-Trust

PGP started as a free secure e-mail program. Now it implements most kinds of cryptography. It was first distributed by Zimmerman in 1991. It uses public key cryptography, symmetric key cryptography, signatures and hash functions. Most interesting part is that instead of using certification chains it uses a web of trust (though PGP users rarely use it). A web of trust is good for private users and militaries, but not businesses. It allows authentication without certification authorities.

The users themselves decide whom to trust. Each user has a certificate including his/her name and public key. Let $S_X(PK_Y)$ denote X's signature on the hash of the certificate including Y's public key (which I denote PK_Y). Each user has a public key ring. This contains Alice's signature on others' certificate hashes and other's signature's on the hash of Alice's certificate. Each such signed key has certain trust levels assigned to it. Below might be A=Alice's public key ring for a simple web of trust without a central server.

			A's public key ring	
name	Signed key	A trusts legitimacy of key	A trusts key owner to certify other keys	A trusts signer to certify other keys
Bob	$S_A(PK_B)$	High	High	
Cath	$S_A(PK_C)$	High	High	
Dee	$S_A(PK_D)$	High	Medium	
Ed	$S_A(PK_E)$	High	Low	
Cath	$S_C(PK_A)$			High



In the diagram, X $\rightarrow$ Y means that $S_X(PK_Y)$ is in both X's and Y's PKRs. Also X $\stackrel{H}{\rightarrow}$ Y means $S_X(PK_Y)$ is in both PKRs and X trusts Y highly to certify others' public keys.

Say Alice wants to e-mail F. A contacts F and F sends his PKR to A. F's PKR includes $S_B(PK_F)$. Since A highly trusts B to certify others' PKs, A now trusts F's PK.

Alice wants to e-mail I. Now A trusts F's PK but has no reason to trust F to certify I's so A does not trusts I's PK.

Alice wants to e-mail G. G's PKR includes $S_D(PK_G)$. But Alice only has medium trust of D to certify others' PK's. So A does not trust G's PK. Similarly, A would not trust N's PK.

Alice wants to e-mail L. L's PKR includes $S_D(PK_L)$ and $S_J(PK_L)$. Alice medium trusts both D and J to certify others' PK's and two mediums is good enough so Alice trusts L's PK.

When Alice e-mails F, PGP will allow it. When Alice e-mails I, PGP will put a message on Alice's screen that she has no reason to trust I's public key.

Say C wants to e-mail B. B sends C his PKR which includes $S_A(PK_B)$. Now C trusts A to certify others' PKs. So C trusts B's PK. So in this sense, Alice (and so every other user) is like a kind of CA.

Instead of e-mailing key rings, there can be a server that finds the connections and sends information on public key rings.

It can be more complicated (this is possible with PGP). Here is an example of three paths from A to L via E:
$A \stackrel{H}{\to} B$, $B \stackrel{H}{\to} C$, $C \stackrel{H}{\to} D$, $D \stackrel{H}{\to} E$, $E \to L$.
$A \stackrel{H}{\to} F$, $F \stackrel{H}{\to} G$, $G \stackrel{H}{\to} I$, $I \to L$.
$A \stackrel{H}{\to} J$, $K \stackrel{M}{\to} L$. You can assign complicated weights to trust like the first path from A to L has weight .25, the second has weight .33 and the fourth has weight .30. Add them to get weight .85. Maybe you require weight at least 1 to trust $S_E(PK_L)$.

One problem with a web of trust is key revocation. It is impossible to guarantee that no one will use a compromised key (since key certification is ad hoc). However, MIT does keep a list of compromised and revoked keys. PGP supports RSA and Diffie Hellman key agreement, DSS(DSA) and RSA for signing, SHA1 and MD5 for hashing, and AES and other block ciphers for encryption..

16 Internet security

The two main protocols for providing security on the internet (encryption and authentication) are Transport Layer Security and IPSec.

16.1 Transport Layer Security

So how does cryptography actually happen on-line? The process is called the Secure Sockets Layer (SSL), invented by Taher ElGamal. Now being replaced by Transport Layer Security (TLS). When you see https:, the *s* tells us that one of these is being used. The following is a simplification of how it goes. I will use SHA1(key_{MAC},message). To indicate using key_{MAC} as the IV to hash the message with SHA1, which is a hash algorithm. I will use AESENC(key_{AES},message) to denote encrypting a message with AES and the key: key_{AES}.

Bob		Amazon
$\sqsubset$ Bob's cert or PK's$\sqsupset$	$\rightarrow$	
	$\leftarrow$	$\sqsubset$ Amaz's cert $\sqsupset$
check A's cert		
		check B's cert
		create key_{AES}
		create key_{MAC}
	$\leftarrow$	$\sqsubset M_1 := (\text{key}_{\text{AES}}\text{key}_{\text{MAC}})^{e_B} \bmod n_B \sqsupset$
$M_1^{d_B} \bmod n_B$ $(= \text{key}_{\text{AES}}\text{key}_{\text{MAC}})$		
$\sqsubset M_2 := \text{AESENC}(\text{key}_{\text{AES}}, \text{message}) \sqsupset$	$\rightarrow$	
$M_3 := \text{SHA1}(\text{key}_{\text{MAC}}, \text{message})$		
$M_4 := M_3^{d_B} \bmod n_B$		
$\sqsubset M_5 := \text{AESENC}(\text{Key}_{\text{AES}}, M_4) \sqsupset$	$\rightarrow$	
		$\text{AESDEC}(\text{key}_{AES}, M_2)$ $(= \text{message})$
		$M_6 := \text{SHA1}(\text{key}_{MAC}, \text{message})$
		$\text{AESDEC}(\text{key}_{\text{AES}}, M_5)(= M_4)$
		$M_4^{e_B} \bmod n_B (= M_3)$
		Check $M_3 = M_6$

Notes.
1. There are MANY ways this can be done.
2. Most private e-mail addresses and browsers do not have certificates, though one can get a certificate for them.
3. MSG would be the whole transaction ("I want to buy the book Cryptonomicon and here is my credit card number")
4. Instead of AES sometimes DES, 3-DES, RC4, Camelia. Instead of RSA, sometimes Diffie Hellman used for key agreement. Instead of Sha-1, sometimes MD5 used.
5 The part ending with the step $M_1^{d_B} \bmod n_B$ is called the handshake.

Now let's replace Bob by Bank of America and Amazon by Bank of Malawi.
6. Assume that next, BoM sends an encrypted message to BA and then sends an encrypted, signed, hash of that message (so BoM sends things analogous to M_2 and M_5). When BA checks that the two hashes are the same, it knows two things. i) The message was not tampered with (integrity). ii) BoM must have sent it since only they could raise to d_{BoM}. So the signature is verified. From BoM's certificate, BA is certain of connection between BoM and n_{BoM}, e_{BoM}.
7. Let's say BA decides later to deny that they sent the message. This is called repudiation. BM can go to court and argue that BA did agree because only BA could have signed it.
8. Actually, BA could deny it by publishing its private key. Then it could say anyone could have faked the signature. On the other hand, if BM can prove that BA's private key got out after the order, then BA can't repudiate the order.

16.2 IPSec

Internet Protocol Security (IPSec) is a competitor of TLS. It works at a different level than TLS which gives it more flexibility. I do not understand different levels on a computer - it is a

concept from computer engineering. IPSec is, however, less efficient than TLS. It's primary use is in Virtual Private Networks (VPN). A VPN is a private communications network. That means it is used within one company or among a small network of companies. TLS is used by everyone on the internet.

17 Timestamping

If you have a printed document and want to prove that it existed on a certain date, you can get it notarized. This is important for copyrighting a document to prove you originated it. This is more difficult with digital data. If there is a date on it, then the date can be easily replaced by another. The solution is a timestamping service.

Here is a second scenario. If Alice signs a message, and later decides that she does not want that message to be signed (this is a kind of cheating) then she can anonymously publish her private key and say that anyone could have done the signing. This is called repudiation. So if someone receives a signature from Alice, he or she can demand that Alice use a digital timestamping service. That lessens Alice's ability to repudiate the signature.

We will give four protocols for timestamping.

Let's say that Trent is a timestamper. Let $\mathcal{A}$ be Alice's name. Let TTS denote a trusted timestamp. Let's say that Alice's is the nth request for a TTS that Trent has ever received.

Timestamping protocol 1. Alice wants to get a TTS on her document. She computes H, the hash and sends it to Trent. Trent creates a timestamp t = time and date when he received H. Trent computes $TTS = [(\mathcal{A}, H, t, \text{ Trent's address})^{d_T} (\mathrm{mod}\, n_T)$, Trent's public key]. Depending on what you are reading, both t and TTS can be called a timestamp. Alice keeps the TTS. Trent stores nothing.

Digistamp does this. Each costs \$0.40, they've done over a million, none has been called into question.

Problem: Alice could bribe Digistamp to sign with a false t.

Protocol 2. Alice sends H to Trent. Trent creates t and serial # n (serial # 's increment by 1 each time). Trent computes $TTS = (\mathrm{hash}(\mathcal{A}, H, t, n))^{d_T} (\mathrm{mod}\, n_T)$ and sends TTS, t and n to Alice. Every week, Trent publishes last serial number used each day (which Trent signs). Every week, Trent zips collection of week's TTS's and signs that and publishes that. Publications are at web site.

PGP does this and also publishes at alt.security.pgp forum.

Solves the bribing problem.

Problem: Depends on popularity of Trent for trust (must trust that they're not messing with old posts on web site or user group). Hard to have small competitors. Lots of storage.

Protocol 3 Linked timestamping (Haber and Stornetta). Alice sends H_n, the hash of her document, to Trent. Let I_k denote the identity of the person getting the kth timestamp. Note $I_n = \mathcal{A}$. Trent computes
$TTS_n = (n, t_n, I_n, H_n, L_n)^{d_T} (\mathrm{mod}\, n_T)$ where $L_n = (t_{n-1}, I_{n-1}, H_{n-1}, H(L_{n-1}))$. Note L_n connects connects the nth with the $n-1$st. Trent sends Alice TTS_n. Later Trent sends Alice I_{n+1}. In homework you'll describe storage.

Can Alice or Alice and Trent together change t_n later? David, the doubter, can ask Alice for the name of I_{n+1}. David contacts I_{n+1}. Then David raises $(TTS_{n+1})^{e_T} (\mathrm{mod}\, n_T)$. Note,

the fifth entry is $L_{n+1} = t_n, \ldots$. So David sees Alice's timestamp t_n in I_{n+1}'s TTS_{n+1}. David can also contact I_{n-1} if he wants to. David can make sure $t_{n-1} < t_n < t_{n+1}$. This prevents Alice and Trent from colluding to change t_n.

Solves bribing problem. Solves storage problem. Solves problem that trust requires popularity of Trent.

Problem: To check, need to contact people before and after Alice.

Protocol 4. Like Protocol 3, but Trent collects several hashed documents (maybe daily) and signs them all at once.
$TTS_n = (n, t_n, I_{n,1}, H_{n,1}, I_{n,2}, H_{n,2}, \ldots, I_{n,r}, H_{n,r}, L_n)^{d_T} (\mathrm{mod}\, n_T)$ where
$L_n = (t_{n-1}, I_{n-1,1}, H_{n-1,1}, I_{n-1,2}, H_{n-1,2}, \ldots, I_{n-1,q}, H_{n-1,q}, H(L_{n-1}))$. It's probably easier to find any one of several people from the TTS before Alice and any one of them from the TTS after Alice.

The author is not aware of a company implementing Protocols 3 or 4.

Note in all protocols, Alice does not have to release the actual document, but only its hash. For example, she wants a trusted time stamp on an idea, without releasing the idea at that moment.

18 KERBEROS

A protocol is a sequence of steps involving at least two parties that accomplishes a task. KERBEROS is a third-party authentication protocol for insecure, closed systems. Between systems people use fire walls. Note most problems come from people within a system. KERBEROS is used to prove someone's identity (authentication) in a secure manner without using public key cryptography. At SCU, KERBEROS is probably used for access to e-campus, OSCAR, Peoplesoft, Novell, etc. It was developed by MIT and can be obtained free. KERBEROS requires a trusted third party.

It enables a person to use a password to gain access to some service. Let U be the user, C be the client (i.e. the computer that the user is at), AS be the authentication server, TGS be the ticket granting service for AS and S be the service that U wants access to. Note when it's between all between servers, U is not a person and U and C may be the same.

Summary:
i) AS authenticates U's identity to TGS.
ii) TGS gives permission to C to use S.

There are many different implementations of KERBEROS. We will outline a simplification of a basic one.

Initial set-up: If D is a participant (i.e. U, C, AS, TGS, or S), then d is the message that is that participant's username/ID. Let $K_{D,E}$ denote a long term key for D and E and $SK_{D,E}$ denote a session key for D and E. Let $\{$ msg $\}K$ denote a message encrypted with the key K.

Usually a system administrator gives a password to U off line and also gives it to the AS.

U goes to a machine C. Then U logs in by entering u and U's password. C creates $K_{U,AS}$, usually by hashing U's password or hashing U's password and a salt defined by AS (as described in Section 19). The AS typically stores the $K_{U,AS}$ in an encrypted file.

There are two other important keys that are used for a long time. Only AS and TGS have $K_{AS,TGS}$. Only TGS and S have $K_{TGS,S}$.

1. C indicates to AS that U intends to use S by sending to AS: u, c, a, s, t, ℓ, nonce. Note a is the client's network address, t is a simple timestamp (like Thu Apr 24 11:23:04 PST 2008) and ℓ is the requested duration (usually 8 hours). The nonce is a randomly generated string to be used only once.

2. AS creates a session key $SK_{C,TGS}$ and sends C:
$\{\text{nonce}, SK_{C,TGS}, tgs, \{SK_{C,TGS}, c, a, v\}K_{AS,TGS}\}K_{U,AS}$.

C has $K_{U,AS}$ and decrypts. When C sees the same nonce that it sent, it knows that this response corresponds to her request and not an old one being sent by the AS or a man-in-the-middle (someone between C and the AS who pretends to be one or both of the proper parties). Note that the returned nonce, encrypted inside a message with $K_{U,AS}$, also authenticates AS's identity to C.

Note $\{SK_{C,TGS}, c, a, v\}$ is called a ticket-granting ticket (TGT). v gives the expiration of the ticket; so $v = t + \ell$.

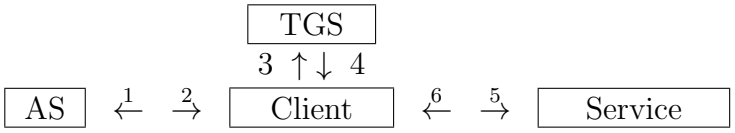
3. C sends TGS: s, $\{SK_{C,TGS}, c, a, v\}K_{AS,TGS}$, $\{c, t\}SK_{C,TGS}$.

Notes: $\{c, t\}$ is called an authenticator.

4. TGS creates a session key $SK_{C,S}$ and sends C: $\{SK_{C,S}\}SK_{C,TGS}$, $\{SK_{C,S}, c, a, v\}K_{TGS,S}$

5. C sends S: $\{SK_{C,S}, c, a, v\}K_{TGS,S}$, $\{c, a, t\}SK_{C,S}$

6. S sends C: $\{t + 1\}SK_{C,S}$.
After this, C has access to S until time v.



For each AS, there is one TGS and there can be several S's for which the TGS grants tickets. This is nice because then only one password can be used to access several services, without having to log in and out. C doesn't know $K_{TGS,S}$ so S trusts the message encrypted with it. The creation of $K_{C,AS}$ involving salting and hashing is not, itself, part of the KERBEROS protocol. The protocol simply assumes the existence of such a secret shared key.

19 Key Management and Salting

A common hacker attack is exploiting sloppy key management. Often bribe or steal to get a key.

Scenario 1. Web: You pick your own password, usually. Uses SSL/TLS so use RSA to agree on AES key which is used to encrypt password (first time and later visits). They store passwords in (hopefully hidden) file with userids. Your password is not hashed. End Scenario 1.

Scenario 2. Non-web systems. The system has a file (may or may not be public) with pairs: userid, hash(password). The system administrator has access to this file. Maybe some day Craig, the password cracker, can get access to this file. He can not use the hashed passwords to get access to your account because access requires entering a password and then having it hashed. End Scenario 2.

In Scenario 2, from the hashes, Craig may be able to determine the original password. Most people's passwords are not random. For example, when I was immature, I used to break into computer systems by guessing that my father's friends' passwords were the names of their eldest daughters. As another example, it would take about 100 seconds on your computer using brute force to determine someone's key if you knew it consisted of 7 lowercase letters (that is 7 bytes; but there are only $26^7 \approx 2^{33}$ of them). But if it consisted of 7 bytes from a pseudo-random bit generator, then it would take a million dollar machine a few hours to brute force it, or your machine a few thousand years. (This is basically like brute-forcing DES since DES has a 7 byte key and running a hash is like running DES. In fact, DES used to be used as a hash were the key was the 0 string and the password was used as the plaintext.) But most people have trouble remembering a password like $8 * \&u!M\}$ and so don't want to use it. They could write it down and put it in their wallet/purse, but that can get stolen. So instead, most passwords are easy to remember.

So Craig can do a dictionary attack. Craig can hash all entries of a dictionary. On-line you can find dictionaries containing all common English words, common proper names and then all of the above entries with i's and l's replaced by 1's and o's replaced by 0's, etc. Craig can even brute force all alpha-numeric strings up to a certain length. Then Craig looks in the password file and finds many matches. This has been used to get tens of thousands of different passwords. Nowadays a single workstation can test 200 million passwords per second.

In 1998, there was an incident where 186000 account names collected and hashed passwords collected. Discovered 1/4 of them using dictionary attack.

Salt is a string that is concatenated to a password. It should be different for each userid. It is public for non-SSL/TLS applications like KERBEROS and UNIX. It might seem like the salt should be hidden. But then the user would need to know the salt and keep it secret. But then the salt may as well just be appended to the password. If the salt were stored on the user's machine instead (so it's secret and the user would not need to memorize it) then the user could not log in from a different machine.

For KERBEROS and UNIX, the system administrator usually gives you your password off-line in a secure way. The system creates your salt.

Scenario 3. (Old UNIX) This is the same as Scenario 2, but the public password file has: username, userid, expiration of password, location information, salt, hash(salt,password). The salt is an unencrypted random string, unique for this userid. Now the dictionary attack won't get lots of passwords. But you can attack a single user as in Scenario 2.

Scenario 4. UNIX. For reasons of backward-compatibility, new Unix-like operating systems need a non-encrypted password file. It has to be similar to the old password file or certain utilities don't work. For example, several utilities need the username to userid map available and look in password file for it. In the password file, where there was once the salt and the hash of a salted password, there is now a *. Unix has a second hidden file called the shadow password file. It is encrypted using a password only known to the system

administrator. The shadow file contains userid, salt, hash(salt,password).

The user doesn't need to look up the salt. If the user connects to UNIX with TLS/SSH, then the password goes, unhashed, through TLS/SSH's encryption. The server decrypts the password, appends the salt, hashes and checks against hash(salt,password) in shadow file.

Scenario 5. KERBEROS uses a non-secret salt which is related to the userid and domain names. If two people have the same password, they won't have the same hash and if one person has two accounts with the same password, they won't have the same hash. The authentication server keeps the hash secret, protected by a password known only to the authentication server.

End scenarios.

A single key or password should not be used forever. The longer it is used, the more documents there are encrypted with it and so the more damage is done if it is compromised. The longer it is used, the more tempting it is to break it and the more time an attacker has to break it.

Good way to generate key, easy to remember, hard to crack. You the words of a song you know. Decide how to capitalize and add punctuation. Then use the first letters of each word and the punctuation. So from the Black-eyed Peas song *I've got a feeling* you could use the lyrics "Fill up my cup, mazel tov! Look at her dancing, " to get the password Fumc,mt!Lahd,

20 Quantum Cryptography

There are two ways of agreeing on a symmetric key that do not involve co-presence. The first is public key cryptography, which is based on mathematics. The second is quantum cryptography. It currently works up to 150 kilometers and is on the market but is not widely used. The primary advantage of quantum cryptography is the ability to detect eavesdropping.

A photon has a polarization. A polarization is like a direction. The polarization can be measured on any basis in two-space: rectilinear (horizontal and vertical), diagonal (slope 1 and slope -1), etc. If you measure a photon in the wrong basis then you get a random result and you disturb all future measurements.

Here is how it works. Alice and Bob need to agree on a symmetric key. Alice sends Bob a stream of photons. Each photon is randomly assigned a polarization in one of the four directions: $|$, $-$, $\backslash$, $/$. We will have $| = 1$, $- = 0$, $\backslash = 1$, $/ = 0$. Let's say that Alice sends: $\backslash\ /\ |\ |\ /\ \backslash\ |\ -\ -\ \backslash\ -\ |\ /$.

Bob has a polarization detector. For each photon, he randomly chooses a basis: rectilinear or diagonal. Say his choices are $\times\ +\ +\ \times\ \times\ +\ +\ +\ \times\ \times\ \times\ +\ +$ Each time he chooses the right basis, he measures the polarization correctly. If he measures it wrong, then he will get a random measurement. His detector might output $\backslash\ -\ |\ \backslash\ /\ /\ |\ -\ /\ \backslash\ \backslash\ |\ |$.

Alice sends	$\backslash$	$/$	$\|$	$\|$	$/$	$\backslash$	$\|$	$-$	$-$	$\backslash$	$-$	$\|$	$/$
Bob sets	$\times$	$+$	$+$	$\times$	$\times$	$+$	$+$	$+$	$\times$	$\times$	$\times$	$+$	$+$
Correct	♡		♡		♡		♡	♡		♡		♡	
Bob gets	$\backslash$	$-$	$\|$	$\backslash$	$/$	$-$	$\|$	$-$	$/$	$\backslash$	$\backslash$	$\|$	$\|$

Notice that when Bob correctly sets the basis, Alice and Bob have the same polarization, which can be turned into a 0 or 1. Looking at the second and last photons, we see an example

of the randomness of Bob's measurement if the basis is chosen incorrectly.

Now Bob contacts Alice, in the clear, and tells her the basis settings he made. Alice tells him which were correct. The others are thrown out.

Alice sends	\	|	/	|	−	\	|
Bob gets	\	|	/	|	−	\	|

Those are turned into 0's and 1's

Alice sends	1	1	0	1	0	1	1
Bob gets	1	1	0	1	0	1	1

On average, if Alice sends Bob $2n$ bits, they will end up with n bits after throwing out those from the wrong basis settings. So to agree on a 128 bit key, on average Alice must send 256 bits.

What if Eve measures the photons along the way. We will focus on the photons for which Bob correctly guessed the basis. For half of those, Eve will guess the wrong basis. Whenever Eve measures in the wrong basis, she makes Bob's measurement random, instead of accurate.

Alice sends	\	|	/	|	−	\	|
Eve sets	×	×	×	×	+	+	+
Pol'n now	\	/	/	\	−	|	|
Bob sets	×	+	×	+	+	×	+
Bob gets	\	−	/	|	−	/	|

Alice sends	1	1	0	1	0	1	1
Bob gets	1	0	0	1	0	0	1

Note for the second and fourth photon, since Eve set the basis incorrectly, Bob gets a random (and half the time wrong) bit. So if Eve is eavesdropping then we expect her to get the wrong basis sometimes and some of those times Bob will get the wrong polarization.

To detect eavesdropping, Alice and Bob agree to check on some of the bits, which are randomly chosen by Alice. For example, in the above, they could both agree to expose, in the clear, what the first three bits are. Alice would say 110 and Bob would say 100 and they would know that they had been tampered with. They would then have to start the whole process again and try to prevent Eve from eavesdropping somehow.

If those check-bits agreed, then they would use the remaining four bits for their key. Of course there is a possibility that Alice and Bob would get the same three bits even though Eve was eavesdropping. So in real life, Alice and Bob would tell each other a lot more bits to detect eavesdropping. The probability that a lot of bits would all agree, given that Eve was eavesdropping, would then be very small. If they disagreed, then they would know there was eavesdropping. If those all agreed, then with very high probability, there was no eavesdropping. So they would throw the check-bits away and use as many bits as necessary for the key.

Eve can perform a man-in-the-middle attack and impersonate Alice with Bob and impersonate Bob with Alice. So quantum cryptography needs some kind of authentication.

Quantum cryptography is considered safer than public key cryptography and has a built-in eavesdropping detection. However, it is difficult to transmit a lot of information this

way, which is why it would be used for agreeing on a symmetric key (like for AES). At the moment, there are physics implementation issues that have been discovered so that the current implementation of quantum cryptography tend to be insecure.

21 Blind Signatures

Here you want a signer to sign a document but the signer not see the document. The analogy: You put a document with a piece of carbon paper above it and place them both in an envelope. You have the signer sign the outside of the envelope so that the document gets signed. How to do with RSA. Say Alice wants Bob to sign a message M without Bob knowing about M. Alice finds a random number r with $gcd(r, n_B) = 1$. Then Alice computes $M' := Mr^{e_B} \bmod n_B$ and sends it to Bob. Bob then computes $M'' := (M')^{d_B} \bmod n_B$ and sends it to Alice. Note $M'' \equiv (M')^{d_B} \equiv (Mr^{e_B})^{d_B} \equiv M^{d_B} r^{e_B d_B} \equiv M^{d_B} r (\bmod n_B)$. Now Alice computes $M''' := M''r^{-1} \bmod n_B$. Note $M''' \equiv M''r^{-1} \equiv M^{d_B} r r^{-1} \equiv M^{d_B} (\bmod n_B)$. So Alice now has Bob's signature on M. But Bob has not seen M and does not seem able to compute it.

Attack on RSA blind signatures. Let's say that n_B is used both for Bob's signing as well as for people encrypting messages for Bob. Let's say that Carol encrypts the message Q for Bob. She sends Bob, $C := Q^{e_B} \bmod n_B$. Eve comes up with a random r and computes $Q' := Cr^{e_B} \bmod n_B$ and sends it to Bob for signing. Bob computes $Q'' := (Q')^{d_B} \bmod n_B$ and sends it to Eve. Note $Q'' \equiv (Q')^{d_B} \equiv C^{d_B} (r^{e_B})^{d_B} \equiv (Q^{e_B})^{d_B} (r^{e_B})^{d_B} \equiv Q^{e_B d_B} r^{e_B d_B} \equiv Qr (\bmod n_B)$. So Eve computes $Q''r^{-1} \bmod n_B$. Note $Q''r^{-1} \equiv Qrr^{-1} \equiv Q (\bmod n_B)$. So now Eve knows Carol's plaintext message Q. So n_B, e_B, d_B should only be used for signing, not encrypting.

22 Digital Cash

If you use a credit card, ATM card or a check to make a purchase, a large institution knows whom you bought something from and how much you spent. Sometimes you prefer privacy. In addition, in such instances, you do not receive the goods until the transaction has been verified through a bank or credit card company. Cash gives privacy and is immediately accepted. However cash is not good for long distance transactions and you can be robbed and sometimes run out. The solution is digital cash.

Let's say Alice wants to buy a $20 book from Carol with digital cash. Alice gets a signed, digital $20 bill from the bank, gives it to Carol and then Carol can deposit it in the bank. This system has some requirements. 1) Forgery is hard. So Eve should not be able to create a signed digital bill without having the bank deduct that amount from her account. 2) Alice should not be able to use the same signed, digital bill twice. So this should be prevented or noticed. 3) When Carol deposits the bill, the bank should not know that it came from Alice. 4) Should be so trustworthy that Carol does not need to check with the bank to accept the bill.

Here is a first attempt at a solution. We'll attack this.

Alice creates a message M="This is worth \$20, date, time, S". Here S is a long random serial number used to distinguish this bill from others. She creates a random r with $\gcd(r, n_B) = 1$. Alice sends $M' := Mr^{e_B} \bmod n_B$ to the bank and tells them to deduct \$20 from her account. The bank signs blindly as above and sends Alice $M'' := (M')^{d_B} \bmod n_B$. Alice computes $M''' := M''r^{-1} \equiv M^{d_B} (\bmod n_B)$. Alice computes $(M''')^{e_B} \equiv (M^{d_B})^{e_B} (\bmod n_B)$ and confirms it equals M. Alice sends M, M''' to Carol. Carol computes $(M''')^{e_B} \bmod n_B$ and confirms it equals M. Carol reads "This is worth \$20 ...". Carol sends M''' to the Bank. The bank computes $(M''')^{e_B} \bmod n_B = M$ and puts \$20 in Carol's account. Only then does Carol give the book to Alice.

Problem 1. Alice could create the message M="This is worth \$20 ..." and tell the bank to deduct \$5 from her account. Since the bank can not figure out M during the interaction with Alice, the bank can not detect the cheating.

Solution 1. The bank has several different triples n_{Bi}, e_{Bi}, d_{Bi} where i is an amount of money. So perhaps the bank has public keys for \$0.01, \$0.02, \$0.05, \$0.10,\$0.20, \$0.50, \$1, \$2, \$5, \$10, \$20, \$50, \$100, \$200, \$500. The bank will sign with $d_{B\$5}$, $n_{B\$5}$. If Carol computes $(M''')^{e_{B\$20}} (\bmod n_{B\$20})$ she will get a random string. Note, if the book costs \$54, Alice can get three bills signed: \$50 + \$2 + \$2 and send them all to Carol.

Solution 2 (preferred as it will work well with eventual full solution). Alice blinds 100 different messages M_i="This is worth \$20, date, time, S_i" for $i = 1, \ldots, 100$, each with a different r_i (S_i is the serial number). The bank randomly picks one of them and signs it and asks Alice to unblind the rest. (In homework, you will determine how to unblind.) The other 99 had better say "This is worth \$20 ..., S_i". You can increase the number 100 to make it harder for Alice to cheat.

Problem 2. Alice can buy another book from David for \$20 using the same $M''' = M^{d_B}$ again. The bank will notice that the serial number has been used twice, but not know that it was Alice who cheated. Putting a serial number in M that is tied to Alice will not help because then Alice loses her anonymity.

Problem 3. Slow like a credit card. Carol should have the bank check the message' serial number against its database to make sure it hasn't been sent before. This is called on-line digital cash because Carol has to get confirmation from the bank first, on-line, that they will put the \$20 in her account.

Solutions to Problems 2 and 3. Off-line digital cash. Uses random identity string (RIS) on the bill.

The RIS 1) must be different for every payment 2) only Alice can create a valid RIS 3) if the bank gets two identical bills with different extended RIS' then Alice has cheated and the bank should be able to identify her 4) if the bank received two identical bills with the same extended RIS' then Carol has cheated.

Let H be a hash function with one-way and weakly collision free properties.

Withdrawal:

1) For $i = 1$ to 100, Alice prepares bills of \$20 which look like $M_i =$ "I am worth \$20, date, time, $S_i, y_{i,1}, y'_{i,1}, y_{i,2}, y'_{i,2}, \ldots, y_{i,50}, y'_{i,50}$" where $y_{i,j} = H(x_{i,j})$, $y'_{i,j} = H(x'_{i,j})$ where $x_{i,j}$ and $x'_{i,j}$ are random bitstrings such that $x_{i,j} \oplus x'_{i,j} =$"Alice" for each i and j. Note $y_{i,1}, y'_{i,1}, y_{i,2}, y'_{i,2}, \ldots, y_{i,50}, y'_{i,50}$ is the RIS.

2) Alice blinds all 100 messages to get M'_i and sends to the bank. Alice tells the bank to deduct \$20 from her account.

3) The bank randomly chooses one of the blinded messages (say 63) asks Alice to unblind the other 99 of the M_is.

4) Alice unblinds the other 99 messages ands also sends all of the $x_{i,j}$ and $x'_{i,j}$ for all 99 messages.

5) The bank checks that the other 99 are indeed \$20 bills and for the other 99 that $y_{i,j} = H(x_{i,j})$ and $y'_{i,j} = H(x'_{i,j})$ and $x_{i,j} \oplus x'_{i,j} =$ Alice.

6) The bank signs the 63rd blinded message M'_{63} and gets M''_{63} and sends it to Alice.

7) Alice multiplies M''_{63} by r_{63}^{-1} and gets M'''_{63}.

Payment

1) Alice gives M_{63}, M'''_{63} to Carol.

2) Carol checks the signature on M'''_{63}. I.e. Carol computes $(M'''_{63})^{e_B} \bmod n_B$ and confirms it equals M_{63}.

3) Carol sends Alice a random bit string of length 50: $b_1, \ldots, b_{50}$.

4) For $j = 1, \ldots, 50$: If $b_j = 0$, Alice sends Carol $x_{63,j}$. If $b_j = 1$, Alice sends Carol $x'_{63,j}$.

5) For $j = 1, \ldots, 50$: Carol checks that $y_{63,j} = H(x_{63,j})$ if $b_j = 0$ or $y'_{63,j} = H(x'_{63,j})$ if $b_j = 1$. If the above equalities hold, Carol accepts the bill.

6) Carol sends Alice the book.

Deposit

1) Carol sends the bank: M_{63}, M'''_{63} and the $x_{63,j}$'s and $x'_{63,j}$'s that Alice sent to Carol. The $y_{63,j}$'s and $y'_{63,j}$'s and those $x_{63,j}$'s and $x'_{63,j}$'s that are revealed form an extended RIS.

2) The bank verifies the signature on M'''_{63}.

3) The bank checks to see if the bill (indentified by S_{63}) is already in their database. If it is not, it puts \$20 in Carol's account and records M_{63}, and the $x_{63,j}$s and $x'_{63,j}$s that Carol sent.

If the bill is in the database and the $x_{63,j}$'s and $x'_{63,j}$'s are the same on both, then the bank knows Carol is cheating by trying to deposit the same bill twice.

Let's say that Alice tries to send the same bill to Carol and later to Dave. Carol sends Alice 11001... and Dave sends 11100.... Now those strings differ at the 3rd place. So Alice will send Carol $x_{63,3}$ and send David $x'_{63,3}$. Carol will send $M_{63}, M'''_{63}, x'_{63,1}, x'_{63,2}, x_{63,3}, x_{63,4}, x'_{63,5}, \ldots$ to the bank. After Carol has sent hers in, the bank will record $M_{63}, M'''_{63}, x'_{63,1}, x'_{63,2}, x_{63,3}, x_{63,4}, x'_{63,5}, \ldots$ under S_{63}. Later Dave will send $M_{63}, M'''_{63}, x'_{63,1}, x'_{63,2}, x'_{63,3}, x_{63,4}, x_{63,5}, \ldots$ to the bank. However, the bank will see S_{63} in the database from when Carol sent the bill in. They will note $x'_{63,3} \neq x_{63,3}$ and compute $x'_{63,3} \oplus x_{63,3}$=Alice and know Alice used the same bill twice. Note for this scenario, Alice could try sending the same bill to Carol at a later time. But Carol would use a different bit-string each time so the bank would still notice.

Used in Singapore, Hong Kong, the Netherlands and Dubai for transit, parking and shops. How is it different than a gift card or an SCU access card with money on it? Digital cash can be used on-line.

23 Bitcoin

In other digital cash systems, if Alice gives money to Bob, then the bank knows how much Bob spent and Alice got, though not that those were connected. Bitcoin is an electronic (digital) cash system, which does not involve a bank. Even greater privacy (like cash). Alice and Bob unknown to all (even each other). Only the amounts are public. Community decides which transactions are valid.

Uses ECDSA. Let G generate $E(\mathbf{F}_q)$ with $q = 2^{256}$. Private keys are integers n and public keys are points $P := nG$.

Uses own hash algorithm built up from other hash functions like 256 bit SHA-2. Like MD5, the messages are broken into 512 bit pieces. Unlike MD5, the hash value has 160 bits.
Ex:

$tran_q$ was earlier. A received 100.

A wants to give the 100 to B. B sends hash(P_{B_r}) to A. Note $P_{A_q} = n_{A_q}G$ and $P_{B_r} = n_{B_r}G$ are public key points on an elliptic curve. Each user uses a new public key point for each transaction.

This is $tran_r$:

a. hash(ST_{q0}) // ST_{q0} is a simplification of $tran_q$

b. 0 // index from $tran_q$ pointing to 100

c. P_{A_q}

d. sign(n_{A_q},ST_{r0} // ST_{r0} =abcef from $tran_r$

e. 100 // index 0 for $tran_r$

f. hash(P_{B_r})

g. hash(ST_{r0}) // This ends $tran_r$.

B sends $tran_r$ to node(s).

(Each) node checks hash(ST_{q0}) against database to make sure it's accepted by majority.

Node finds hash(ST_{q0}) is in accepted $tran_q$.

Node looks in $tran_q$ and confirms index 0 value in $tran_q$ is at least 100=e (in $tran_r$).

Node hashes P_{A_q} from c in $tran_r$ and compares with the hash(P_{A_q}) in $tran_q$.

Node hashes ST_{r0} =abcef in $tran_r$ and compares with g in $tran_r$.

Node computes verifysign(P_{A_q},d) and compares with g (both in $tran_r$).

B wants to use 100 from $tran_r$ and 20 from earlier $tran_p$ and give 110 to C, 10 back to himself.

B and C exchange public keys P_{B_r}, P_{B_p}, P_{B_s}, P_{C_s}.

This is $tran_s$:

a. hash(ST_{r0}) // g from $tran_r$

b. 0 // index from e in $tran_r$, pointing to 100

c. P_{B_r}

d. sign(n_{B_r},ST_{s0}) // ST_{s0} =abcij from $tran_s$

e. hash(ST_{p0})

f. 0 // index from $tran_p$, pointing to 20

g. P_{B_p}

h. sign(n_{B_p},ST_{s1}) // ST_{s1} =efglm from $tran_s$

i. 110 // index 0 for $tran_s$

j. hash(P_{C_s})

k. hash(ST_{s0})

ℓ. 10 // index 1 for $tran_s$

m. hash(P_{B_s})

n. hash(ST_{s1}) // This ends $tran_s$.

C sends $tran_s$ to node(s).

Node checks hash(ST_{r0}) and hash(ST_{p0}) against database to make sure they're each accepted by majority.

Node finds hash(ST_{r0}) is in accepted $tran_r$ and hash(ST_{p0}) is in accepted $tran_p$.

Node looks in $tran_r$ and $tran_p$ and sums the values from index 0 from both (and gets $100 + 20 = 120$).

Node looks at values in i and ℓ3of $tran_s$ and sums the values and ensures that it is at most the sum from the previous step.

Node hashes P_{B_r} from c in $tran_s$ and compares with f in $trans_r$. Nodes hashes P_{B_p} from g in $trans_s$ and compares with $tran_p$.

Node hashes ST_{s0} =abcij from $tran_s$ and compares with k in $tran_s$. Node hashes ST_{s1} =efglm from $tran_s$ and compares with n in $tran_s$.
Node computes verifysign(P_{B_r},d) and compares with k. Node computes verifysign(P_{B_p},h) and compares with n. All happens in $tran_s$.

Real life wouldn't be 10 and 110. Would be 110 and something less than 10. Rest is taken by successful node as transaction fee.

End ex.

When Alice signs ST_{r0}, she is agreeing that she is sending 100BTC to Bob.

How to prevent Alice from double spending? There is a network of nodes (in tens of thousands). Each tries to create a successful block. The reward is a new Bitcoin as well as all the transaction fees. A block contains 1) the hash of the previous successful block, 2) all the transactions since the previous successful block and 3) a random string (called a nonce). Each node creates a block and hashes it. Note the hash value can be considered an integer n with $0 \leq n \leq 2^{160} - 1$. Bitcoin software says: The hash of your new block must be at most, say, $2^{115} + 2^{114} + 2^{112}$ (more later) (in particular, starts with 44 zeros). Each node tries new nonces until it finds a successful hash. Then the node publishes the successful block. Other nodes confirm hash(block) $< 2^{115} + 2^{114} + 2^{112}$ and check all of the transactions as described in the example. It is expected to take, on average, 10 minutes for some node to create a successful block. The number where I put $2^{115} + 2^{114} + 2^{112}$ decreases every two weeks based on a) number of nodes b) computer speeds (using a Poisson random variable).

If two nodes create a block nearly simultaneously, then some nodes will work on one block, some on the other. Whichever is successful, that chain gets longer. Eventually shorter chains are abandoned. Once a block is 6-deep in a chain, people tend to trust it. This is why you can not immediately spend Bitcoins you have gotten through a Bitcoin transaction.

A successful node receives a certain amount of Bitcoin from the Bitcoin server. For 2009 - 2012 the node gets 50BTC. For 2013 - 2016 the node gets 25BTC. Every four years, it halves. The node also receives all of the transaction fees.

What if there are several evil nodes and one of them finds a nonce for a block with a bad transaction and other evil nodes continue this chain. Important assumption: most nodes are good. Only the evil nodes will accept a bad block. The good ones won't work on the next nonce for that block. Since most nodes are good, the evil nodes' chains won't increase as fast as the good nodes' chains and people accept the good chains.

Notes:

1BTC = 10^8 Satoshi's.

The minimum transaction fee is .0005 BTC.

The author can think of two reasons that the output contains the hash of a public key and

not the public key. The first is that it decreases storage. The second (less likely reason) is that if the public key itself were public for a long time (maybe a few years if it takes that long for someone to try to redeem Bitcoins), that could decrease its security.

At one point in 2011, the exchange rate was 1 BTC = \$0.30. In December 2013, it was 1 BTC = \$1200. The total number will not exceed 21 million BTC.

If there are multiple inputs or outputs in a single transaction then people will know that certain public keys are connected.

24 Secret Sharing

Let us assume that you have five managers at Coca Cola in Atlanta who are to share the secret recipe for Coca Cola. You do not want any one of them to have the secret, because then he could go off and sell it to Pepsi. You could encode the secret as a bit-string S of length l. Then you could come up with four random bit-strings of length l, R_1, R_2, R_3, R_4. You could give the first four managers R_1, R_2, R_3, R_4 (one each) and to the fifth you would give $R_1 \oplus R_2 \oplus R_3 \oplus R_4 \oplus S$. Note that no one has the secret S, but if you XOR all five together, you get S. No four people can get S. But what if one of the managers dies and his piece is lost. Maybe you want it so that any three together could get the secret.

Key escrowing is an example of this. There is a secret key that K. Someone comes up with a random string R and then R is given to one trusted entity (like a major bank) and $K \oplus R$ is given to another trusted entity. If someone needs the secret key for some reason, one can go through a legal process to get R and $K \oplus R$.

LaGrange interpolating polynomial scheme for secret sharing

This was invented by Shamir (the S of RSA). Let S be a secret encoded as a positive integer. Let p be a prime with $p > S$. The secret holder give shares of her secret to her managers. The secret holderwants any three (say) of the managers to have access to the secret. So she creates a polynomial with three coefficients $f(x) = ax^2 + bx + S$. The numbers a and b are chosen randomly from 0 to $p-1$. The polynomial will remain secret.

She gives the first manager p, the index 1, and the reduction of $f(1) (\mathrm{mod}\, p)$, the second manager p, the index 2, and the reduction of $f(2) (\mathrm{mod}\, p)$, etc. Not only will this work for five managers but for n managers as long as $n \geq 3$, assuming you want any three managers together to be able to get the secret. Example. The secret is 401. She picks $p = 587$ and $a = 322$, $b = 149$. So $f(x) = 322x^2 + 149x + 401 (\mathrm{mod}\, 587)$. Then $f(1) = 285$, $f(2) = 226$, $f(3) = 224$, $f(4) = 279$, $f(5) = 391$. The first, fourth and fifth managers collude to get the secret. They know $f(x) = ax^2 + bx + S (\mathrm{mod}\, 587)$ but do not know a, b, S. They do know $f(1) = a + b + S = 285$, $16a + 4b + S = 279$ and $25a + 5b + S = 391$. They now solve

$$\begin{bmatrix} 1 & 1 & 1 \\ 16 & 4 & 1 \\ 25 & 5 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ S \end{bmatrix} = \begin{bmatrix} 285 \\ 279 \\ 391 \end{bmatrix} (\mathrm{mod}\, 587)$$

to find a, b, S. (Basic linear algebra works over any field.) They are only interested in S. In PARI use m=[1,1,1;16,4,1;25,5,1]*Mod(1,587), v=[285;279;391], matsolve(m,v).

If she wants any m managers to have access, she creates a polynomial of degree $m-1$ with m coefficients.

25 Committing to a Secret

Committing to a secret (Torben Pedersen, 1998). Let p be a large prime and g generate $\mathbf{F}_p^*$. Assume h is provided by some higher authority and h also generates and that you do not know the solution to $g^x = h$. The values g, h and p are public.

Assume $0 << s << p-1$ is a secret that you want to commit to now and that you will later expose. You want to convince people that you have not changed s when you reveal your secret later. For example, s might be a bid. Choose a random t with $0 << t << p-1$ and compute $E(s,t) = g^s h^t$ (in $\mathbf{F}_p$) and publish $E(s,t)$. That is your commitment to s. You can later open this commitment by revealing s and t. The verifier can check that indeed $g^s h^t$ is the same as your published $E(s,t)$. Note 1) $E(s,t)$ reveals no information about s. 2) You (as the committer) can not find s', t' such that $E(s,t) = E(s',t')$.

In real life, you choose p, with $p > 2^{512}$, such that $q|p-1$ where q is prime and $q > 2^{200}$. large. You pick g and h such that $g^q = 1$, $h^q = 1$ and $g \neq 1$, $h \neq 1$. It adds some speed and does not seem to affect security as explained in Section 31.1.

26 Digital Elections

We will describe Scantegrity II (Invisible Ink) by Chaum, Carback, Clark, Essex, Popoveniuc, Rivest, Ryan, Shen, Sherman. For simplicity, assume the voters are voting for a single position with a slate of candidates.

Election preparation

Assume N is the number of candidates and B is the number of ballots to be created. For m voters we need $B = 2m$ ballots (because of audit ballots). The election officials (EO)'s secret-share a seed and enough of them input their shares into a trusted work-station (WS) to start a (pseudo-)random number generator which creates BN different, random alpha-numeric confirmation codes (CC)'s (like WT9).

The work-station creates Table P, which is never published.

Table P (not published)

Ballot ID	Alice	Bob	Carol
001	WT9	7LH	JNC
002	KMT	TC3	J3K
003	CH7	3TW	9JH
004	WJL	KWK	H7T
005	M39	LTM	HNN

When I say "not published" it also means the EO's don't see it either. Each row is used to create a single ballot.

Then the WS creates Table Q. It is the same as Table P except with the rows permuted randomly. The columns no longer correspond to single candidates.

Table Q (not published)

001	7LH	WT9	JNC
002	J3K	TC3	KMT
003	9JH	CH7	3TW
004	KWK	H7T	WJL
005	M39	HNN	LTM

The work station creates random $t_{i,j}$'s and computes commitments to each CC in Table Q and the EO's have the WS publish the commitments at the election website.

Table Q commitments (published)

001	$g^{7LH}h^{t_{1,1}}$	$g^{WT9}h^{t_{1,2}}$	$g^{JNC}h^{t_{1,3}}$
002			
003	$\vdots$	$\vdots$	$\vdots$
004			
005	$g^{M39}h^{t_{5,1}}$	$g^{HNN}h^{t_{5,2}}$	$g^{LTM}h^{t_{5,3}}$

Here, for example, g^{7LH} assumes you have encoded 7LH as an integer.

The WS creates Tables S and R.

Table S (published at website)

Alice	Bob	Carol

Table R (not published)

Flag	Q-ptr	S-ptr
	005, 1	2, 1
	003, 3	4, 2
	002, 1	4, 3
	001, 3	3, 3
	001, 2	4, 1
	005, 3	3, 2
	004, 2	5, 3
	003, 1	2, 3
	004, 3	3, 1
	002, 3	1, 1
	001, 1	2, 2
	002, 2	5, 2
	004, 1	1, 2
	003, 2	5, 1
	005, 2	1, 3

Table S begins blank and will record votes. Each row of Table R corresponds to a CC

from Q. Each row of R has three entries. Column 1 is first empty. Once a vote has been cast, it contains a flag 0 (unselected) or 1 (selected). Column 2 contains a pointer to Table Q of the form (ID, col of Q). They are entered in random order. Column 3 of R contains pointers to all pairs (row of S, col of S). The S-pointers are not entirely random. For example, S-pointers (1,2), (2,2), (3,2), (4,2), (5,2) must be in the same row as the Q-pointers to CC's for Bob (namely 7LH, TC3, 3TW, KWK, LTM, which have Q-pointers (001,1), (002,2), (003,3), (004,1), (005,3)). Given that, they are randomized.

The WS creates commitments to all the pointers and the EO's have the WS publish them (in their same positions) in Table R-commitments at the election website.

Table R commitments (published)

Flag	Q-ptr	S-ptr
	$g^{005,1}h^{u_{1,1}}$	$g^{2,1}h^{u_{1,2}}$
	⋮	⋮
	$g^{005,2}h^{u_{15,1}}$	$g^{1,3}h^{u_{15,2}}$

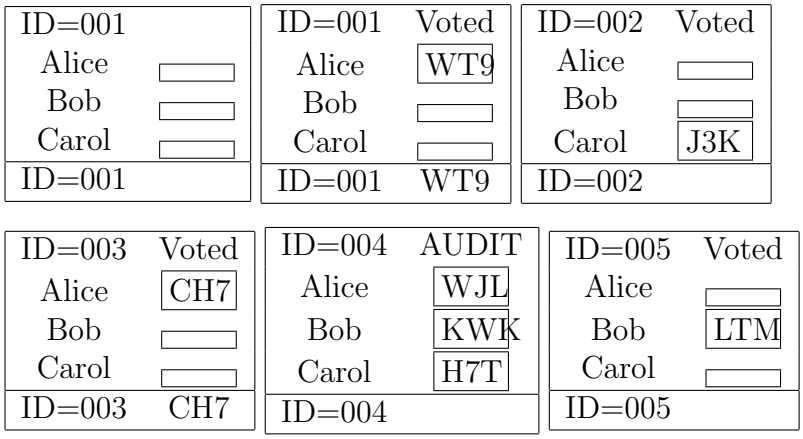
During the election:

The ballots have the CC's from Table P written in invisible ink inside bubbles. The voter is given a decoder pen to rub the bubble for her chosen candidate. The CC then appears. She then has the option to copy the CC onto the bottom part, which is the receipt. She tears off the receipt and keeps it. The poll worker (PW) stamps the top and helps her enter the ballot into an optical scanner. The top is kept there as a paper record, as one way of auditing the election afterwards, if necessary.

The voter has the option to instead ask for two ballots. She then picks one to be the official ballot and one to be an audit ballot. The PW then audit-stamps the audit ballot. The voter can expose (decode) all of the CC's on the audit ballot. The PW helps her enter the audit ballot into the optical scanner which reads the data and notes it is an audit ballot. She may leave with the audit ballot. The audit ballot does not count toward the actual votes.

As an example, assume 001 voted for Alice so CC is WT9, 002 for Carol so CC is J3K, 003 for Alice so CC is CH7, 004 was an audit with WJL, KWK and H7T (from the same person who has ID=003), 005 for Bob so CC is LTM.

Empty Ballot Filled ballots



Person 1 gets ballot 001 and votes for Alice, WT9. She copies "WT9" at the bottom, tears it off and keeps it. She gives the top to the PW.

Person 2 gets ballot 002, votes for Carol, J3K and goes home with no receipt.

Person 3 asks for two ballots (003 and 004). He decides which is real and which is the audit ballot. The PW stamps "audit" on the audit ballot. On the real ballot, he votes for Alice, CH7. On the audit ballot he exposes all three: WJL, KWK, H7T. He copies "CH7" onto the first receipt. He enters the top of both ballots into the optical scanner. He takes home the whole audit ballot and the first receipt.

Person 4 votes for Bob, LTM and goes home with no the receipt.

The WS, after the election:

Person 1 voted for Alice, WT9. The WS takes WT9 and looks it up in Table Q. It's ID 001, column 2. The WS looks in the Q-pointers of Table R for 001,2 and puts a flag there. Next to it in column 3 is the S-pointer 4,1, so the WS puts a 1 in row 4, column 1 of the S-table. Note that this is indeed Alice's column. The WS does the same for Person 2, 3 and 4's votes.

The fourth ballot is an audit ballot. These votes are not recorded. The WS takes WJL, KWK, H7T and looks in Table Q. They are ID 004, columns 1, 2, 3. The WS indicates to itself in Table R that those Q-pointers are audits.

Table R (not published)

Flag	Q-ptr	S-ptr
0	005, 1	2, 1
0	003, 3	4, 2
1	002, 1	4, 3
0	001, 3	3, 3
1	001, 2	4, 1
1	005, 3	3, 2
0	004, 2(A)	5, 3
0	003, 1	2, 3
0	004, 3(A)	3, 1
0	002, 3	1, 1
0	001, 1	2, 2
0	002, 2	5, 2
0	004, 1(A)	1, 2
1	003, 2	5, 1
0	005, 2	1, 3

Table S

	Alice	Bob	Carol
		1	
	1		1
	1		
Sum	2	1	1

The WS sums the columns of Table S. Note that the rows of S in which there are 1's do not tell you who cast each vote. This is because of the (constrained) randomness of the third row of Table R. Tables P, Q and R remain hidden inside the main-frame.

The EO's after the election:

The EO's have the WS publish to the election website part of Table Q: The ID's, the CC's corresponding to votes, the CC's corresponding to audits and the $t_{i,j}$'s for each published CC.

Table Q, partly opened

001		WT9, $t_{1,2}$	
002	J3K, $t_{2,1}$		
003		CH7, $t_{3,2}$	
004	KWK, $t_{4,1}$	H7T, $t_{4,2}$	WJL, $t_{4,3}$
005			LTM, $t_{5,3}$

For each row of Table R, a random, publicly verifiable bit generator (called a coin-flip) decides if the EO's will have the WS open the commitment to the Q-pointer or the S-pointer. One suggestion is to use closing prices of major stock indices as a source. Note that if either the second or third column of Table R is not in a random order, then an observer could make a connection between an ID number and the candidate that person voted for. If the ballot was for auditing, the EO's have the WS open the commitments for all the CC's in Table Q

and the Q- and S-pointers in Table R.

Table R, partly opened

Flag	Q-ptr	S-ptr
0		2, 1, $u_{1,2}$
0	003, 3, $u_{2,1}$	
1		4, 3, $u_{3,2}$
0		3, 3, $u_{4,2}$
1	001, 2, $u_{5,1}$	
1	005, 3, $u_{6,1}$	
0	004, 2, $u_{7,1}$ A	5, 3, $u_{7,2}$
0		2, 3, $u_{8,2}$
0	004, 3, $u_{9,1}$ A	3, 1, $u_{9,2}$
0	002, 3, $u_{10,1}$	
0	001, 1, $u_{11,1}$	
0	002, 2, $u_{12,1}$	
0	004, 1, $u_{13,1}$ A	1, 2, $u_{13,2}$
1		5, 1, $u_{14,2}$
0	005, 2, $u_{15,1}$	

The voters after the election:

Each voter may check her ID number and her vote's CC are there in Table Q. Any interested party can check i) the commitments for Table Q agree with the opened parts of Table Q, ii) the commitments for Table R agree with the opened parts of Table R, iii) that each revealed Q-pointer in Table R either connects a iii-1) revealed code in Table Q to a flag in Table R (Ex 001, 2) or iii-2) a hidden code in Table Q to an unflagged element of Table R (Ex 003, 3) and iv) each revealed S-pointer in Table R connects a iv-1) flag in R to a flag in S (Ex 4,3) or iv-2) an unflagged element of R to an unflagged element of S (Ex 2,1) and v) that the vote tallies published agree with the sums in Table S and the number of votes agrees with the number of flags in Table R.

Person 3 can check, for example, that on his audit ballot, WJL (vote for Alice) was Q-pointer (004,3), which in the partially-opened Table R is next to S-pointer (3,1) and that indeed column 1 contains votes for Alice.

If Cody the coercer tries to coerce Dan on how to vote, Dan can show Cody Dan's CC. Note that no information at the election website can connect Dan's CC with the candidate for whom Dan voted.

It is possible that poll workers could fill in a few ballots. But the poll workers have to keep records of the names of the people in their districts who voted. So they will have to add names to this record. If too many ballots are filled out by poll workers, then auditors might get suspicious and determine that some of the people the poll workers claim voted, did not vote.

Scantegrity II has been used in a 2009 Takoma Park MD city election.

Cryptanalysis

27 Basic Concepts of Cryptanalysis

Cryptosystems come in 3 kinds:
1. Those that have been broken (most).
2. Those that have not yet been analyzed (because they are new and not yet widely used).
3. Those that have been analyzed but not broken. (RSA, Discrete log cryptosystems, AES).

3 most common ways to turn ciphertext into plaintext:
1. Steal/purchase/bribe to get key
2. Exploit sloppy implementation/protocol problems (hacking/cracking). Examples: someone used spouse's name as key, someone sent key along with message
3. Cryptanalysis

There are three kinds of cryptanalysis.
Ciphertext only attack. The enemy has intercepted ciphertext but has no matching plaintext. You typically assume that the enemy has access to the ciphertext. Two situations:
a) The enemy is aware of the nature of the cryptosystem, but does not have the key. True with most cryptosystems used in U.S. businesses.
b) The enemy is not aware of the nature of the cryptosystem. The proper users should never assume that this situation will last very long. The Skipjack algorithm on the Clipper Chip is classified, for example. Often the nature of a military cryptosystem is kept secret as long as possible. RSA has tried to keep the nature of a few of its cryptosystems secret, but they were published on Cypherpunks.
Known plaintext attack. The enemy has some matched ciphertext/plaintext pairs. The enemy may well have more ciphertext also.
Chosen plaintext attack. Here we assume that the enemy can choose the plaintext that he wants put through the cryptosystem. Though this is, in general, unrealistic, such attacks are of theoretic interest because if enough plaintext is known, then chosen plaintext attack techniques may be useable. However this is an issue with with smart cards.

As in the first cryptography course, we will not spend much time on classical cryptanalysis, but will instead spend most of our time looking at current cryptanalytic methods.

28 Historical Cryptanalysis

Designers of cryptosystems have frequently made faulty assumptions about the difficulty of breaking a cryptosystem. They design a cryptosystem and decide "the enemy would have to be able to do x in order to break this cryptosystem". Often there is another way to break the cryptosystem. Here is an example. This is a simple substitution cipher where one replaces every letter of the alphabet by some other letter of the alphabet. For example A $\rightarrow$ F, B $\rightarrow$ S, C $\rightarrow$ A, D $\rightarrow$ L .... We will call this a monalphabetic substitution cipher. There are about $1.4 \cdot 10^{26}$ permutations of 26 letters that do not fix a letter. The designers of this cryptosystem reasoned that there were so many permutations that this cryptosystem was safe. What they did not count on was that there is much regularity within each language.

In classical cryptanalysis, much use was made of the regularity within a language. For example, the letters, digraphs and trigraphs in English text are not distributed randomly. Though their distributions vary some from text to text. Here are some sample percentages (assuming spaces have been removed from the text): E - 12.3, T - 9.6, A - 8.1, O - 7.9, ..., Z - 0.01. The most common digraphs are TH - 6.33, IN, 3.14, ER - 2.67, ..., QX - 0. The most common trigraphs are THE - 4.73, ING - 1.42, AND - 1.14, ... Note there are tables where the most common digraphs are listed as TH, HE, IN, ER, so it does depend on the sample. What do you notice about these percentages? AES works on ASCII 16-graphs that include upper and lower case, spaces, numerals, punctuation marks, etc.

The most common reversals are ER/RE, ES/SE, AN/NA, TI/IT, ON/NO, etc. Note that they all involve a vowel.

If there is a lot of ciphertext from a monalphabetic substitution cipher, then you can just count up the frequencies of the letters in the ciphertext and guess that the most commonly occurring letter is E, the next most is T, etc. If there is not much ciphertext, then you can still often cryptanalyze successfully. For example, if you ran across XMOX XMB in a text, what two words might they be? (that the, onto one). Find a common word that fits FQJFUQ (people).

Let's cryptanalyze this: GU P IPY AKJW YKN CJJH HPOJ RGNE EGW OKIH-PYGKYW HJVEPHW GN GW DJOPZWJ EJ EJPVW P AGUUJVJYN AVZIIJV MJN EGI WNJH NK NEJ IZWGO REGOE EJ EJPVW EKRJSJV IJPWZVJA KV UPV PRPB

(Note second and last words)

28.1 The Vigenère cipher

Encryption of single letters by $C \equiv P+b \pmod{26}$ (where b is a constant) is a monalphabetic shift cipher. If $b = 3$ we get the Caesar cipher. We can cycle through some finite number of monalphabetic shift ciphers. This is the Vigenère cipher (1553). There would be a key word, for example TIN. T is the 19th letter of the alphabet, I is the 7th and N is the 13th (A is the 0th). So we would shift the first plaintext letter by 19, the second by 7, the third by 13, the fourth by 19, etc. See the Vigenère square on the next page. In order to encrypt CRYPTO with the key TIN you first encrypt C with T. Look in the square on the next page in row C, column T (or vice versa). You would get VZLIBB. Note that the letter B appears twice in the ciphertext. If the proper addressee has the same key then she can make the same table and decryption is easy.

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
BCDEFGHIJKLMNOPQRSTUVWXYZA
CDEFGHIJKLMNOPQRSTUVWXYZAB
DEFGHIJKLMNOPQRSTUVWXYZABC
EFGHIJKLMNOPQRSTUVWXYZABCD
FGHIJKLMNOPQRSTUVWXYZABCDE
GHIJKLMNOPQRSTUVWXYZABCDEF
HIJKLMNOPQRSTUVWXYZABCDEFG
IJKLMNOPQRSTUVWXYZABCDEFGH
JKLMNOPQRSTUVWXYZABCDEFGHI
KLMNOPQRSTUVWXYZABCDEFGHIJ
LMNOPQRSTUVWXYZABCDEFGHIJK
MNOPQRSTUVWXYZABCDEFGHIJKL
NOPQRSTUVWXYZABCDEFGHIJKLM
OPQRSTUVWXYZABCDEFGHIJKLMN
PQRSTUVWXYZABCDEFGHIJKLMNO
QRSTUVWXYZABCDEFGHIJKLMNOP
RSTUVWXYZABCDEFGHIJKLMNOPQ
STUVWXYZABCDEFGHIJKLMNOPQR
TUVWXYZABCDEFGHIJKLMNOPQRS
UVWXYZABCDEFGHIJKLMNOPQRST
VWXYZABCDEFGHIJKLMNOPQRSTU
WXYZABCDEFGHIJKLMNOPQRSTUV
XYZABCDEFGHIJKLMNOPQRSTUVW
YZABCDEFGHIJKLMNOPQRSTUVWX
ZABCDEFGHIJKLMNOPQRSTUVWXY
```

Let's say that we receive the following text that we know was encrypted with the Vigenère cipher.

```
wzggqbuawq pvhveirrbv nysttaknke nxosavvwfw frvxqumhuw
wqgwtgziih locgpnhjmn nmtzqboavv abcuawohbv rjTAMPOvkl
gpigfsmfmw vnniyhzyrv qkkiqywweh vjrjwgWEWG Zhcxucakep
wpsnjhvama hkmehnhuww vtzguwaclz stsvfxlplz muywzygagk
aofkioblwi argtvrgzit xeofswcrqb tllcmiabfk ttbwbfenvz
snlytxahuw vgtzstghut vrzwrcglpr ariltwxwTA MPOtgvwlqh
vkhkynwpmp vmwgbjxqnb tnuxhkwasa gvbwbntswm pwfdmhxnce
zinbdsqarv aihojmneqo alfwmpomqd qgmkuwvgfg husrfaqggg
vavwzyahgg wbrgjjbake axkgovnkww kdwiwhdnbo aumggbgbmv
exaoogypWE WGZvgymfrf gglbcuaq
```

How could we determine the length of the keyword? There are two methods we will look at. They were invented by Kasiski (19th century) and Friedman (20th century).

The Kasiski text. Let's consider a frequent trigraph like THE and let's say that the keyword is 5 letters long. If the trigraph THE starts at the n and mth positions in the plaintext and $n \not\equiv m (\mathrm{mod}\, 5)$ then they will be encrypted differently. If $n \equiv m (\mathrm{mod}\, 5)$, then they will be encrypted the same way. Keyword VOICE. Plaintext THEONETHE becomes ciphertext OVMQRZHPG whereas plaintext THEBOTHE becomes OVMDSOVM. Of course this would work for AND, ING or any other frequent trigraph. For any given pair of the same plaintext trigraphs, we expect that one fifth of the time they will be separated by a distance a multiple of 5 and so will be encrypted identically. With enough ciphertext we can look for repeated trigraphs and compute the distance between them. These distances should be multiples of the key length usually.

Note the repeated appearances of WEWGZ and TAMPO that are $322 = 2 \cdot 7 \cdot 23$ and $196 = 2^2 \cdot 7^2$ apart. Repeated appearances of HUWW and UWVG are $119 = 7 \cdot 17$ and $126 = 2 \cdot 3^2 \cdot 7$ apart. These distances should be multiples of the length of the keyword. We suspect the keylength is 7 or 14 if HUWW is a coincidence. We must be cautious though because there can be coincidences like the fact that the two AVV's are 43 apart. So if we write a computer program to get the greatest common divisor of all of the distances it would output 1.

The Friedman test gives you an estimate of the length of the keyword. Note that if we have a monalphabetic shift cipher, and draw a histogram of the letter appearances in the ciphertext, it will look like a histogram of the letters in the plaintext, only shifted over. Sorted, the percentages would be (12.31, 9.59, 8.05, 7.94, ...20, .20, .1, .09) for ETAO, ...QXJZ. If we have a two alphabet shift cipher then, for example, the frequency of A appearing in the ciphertext is the average of the frequencies of the two letters sent to A. Sorted the percentages might be (8.09, 7.98, 7.565, 7.295, ...1.115, 1.04, 0.985, 0.31). If there is a ten alphabet shift cipher then for each ciphertext letter, the frequency of appearance is the average of the ten frequencies of the letters mapping to it. Sorted, the percentages might be (4.921, 4.663, 4.611, 4.589, ...3.284, 3.069, 3.064, 2.475). Note that if we consider the frequencies of each letter in the ciphertext, that the mean, regardless of the number of alphabets, is 1/26. But the variance is smaller, the more alphabets there are. So we can use the variance to estimate the number of alphabets used (i.e. the length of the keyword).

We need a statistic like variance. If one selects a pair of letters from the text (they need not be adjacent), what is the probability that both are the same letter? Say we have an n letter text (plaintext or ciphertext) and there are n_0 A's, n_1 B's, ..., n_{25} Z's. So $\sum n_i = n$. The number of pairs of A's is $n_0(n_0-1)/2$, etc. So the total number of pairs of the same letter is $\sum n_i(n_i-1)/2$. The total number of pairs in the text is $n(n-1)/2$. So the probability that a pair of letters (in an n letter text) is the same is

$$\frac{\sum_{i=0}^{25} \frac{n_i(n_i-1)}{2}}{n(n-1)/2} = \frac{\sum_{i=0}^{25} n_i(n_i-1)}{n(n-1)} = I.$$

I is called the observed index of coincidence (IOC) of that text. What is the expected IOC of standard English plaintexts? Let p_0 be the probability that the letter is A, $p_0 \approx .0805$, etc. The probability that both letters in a pair are A's is p_0^2. The probability that both are Z's is p_{25}^2. So the probability that a pair of letters is the same is $\sum p_i^2 \approx .065$ for English. For a random string of letters $p_0 = \ldots = p_{25} = 1/26$ then $\sum p_i^2 = 1/26 \approx 0.038$. (Note that we

would get a random string of letters from a Vigenère cipher with infinite key length if the key is random.) For any monalphabetic substitution cipher, the expected IOC is approximately .065.

To make this a little more understandable, let's consider an easier example than English. Create a new language $\alpha, \beta, \gamma, \delta$ with letter frequencies .4, .3, .2, .1. The expected IOC is 0.3. Shift one, and then the ciphertext letter frequencies are .1, .4, .3, .2 and again the expected IOC is 0.3. If we encrypt with the Vigenère cipher and key $\beta\gamma$ (i.e. shift one, then two, then one, ...) then the frequency of α in the ciphertext is $1/2(.1) + 1/2(.2) = .15$ of β is $1/2(.4) + 1/2(.1) = .25$ of γ is $1/2(.3) + 1/2(.4) = .35$ and of $\delta = 1/2(.2) + 1/2(.3) = .25$. Then the expected IOC is 0.27. Note it becomes smaller, the longer the key length. Note you can use the observed IOC to determine if ciphertext comes from a monalphabetic or polyalphabetic cipher.

Back to English. Let's say we have ciphertext of length n with a keyword of length k, which we want to determine. For simplicity, assume $k|n$ and there is no repeated letter in the keyword. We can write the ciphertext in an array as follows (of course, we don't really know k yet).

$$\begin{array}{ccccc} c_1 & c_2 & c_3 & \dots & c_k \\ c_{k+1} & c_{k+2} & c_{k+3} & \dots & c_{2k} \\ \vdots & & & & \\ & & & & c_n \end{array}$$

We have n/k rows. Each column is just a monalphabetic shift. Two letters chosen in one column have probability $\approx .065$ of being the same. Two letters in different columns have probability ≈ 0.038 of being the same.

What's the expected IOC? The number of pairs from the same column is $n((n/k)-1)/2 = n(n-k)/(2k)$. The number of pairs from different columns is $n(n-(n/k))/2 = n^2(k-1)/(2k)$. The expected number of pairs of the same letter is

$$A = 0.065\left(\frac{n(n-k)}{2k}\right) + 0.038\left(\frac{n^2(k-1)}{2k}\right).$$

The probability that any given pair consists of the same two letters is

$$\frac{A}{n(n-1)/2} = \frac{1}{k(n-1)}[0.027n + k(0.038n - 0.065)].$$

This is the expected IOC. We set this equal to the observed IOC and solve for k.

$$k \approx \frac{.027n}{(n-1)I - 0.038n + 0.065} \quad \text{where} \quad I = \sum_{i=0}^{25} \frac{n_i(n_i-1)}{n(n-1)}$$

In our example $I = 0.04498, k \approx 3.844$. Thus we get an estimate of the key length of 3.844 (it is actually 7 - this is the worst I've ever seen the Friedman test perform).

Solving for the key. Considering the results of the (Friedman and) Kasiski tests, let's assume that the key length is 7. Now we want to find the key, i.e. how much each is shifted.

We can write a program to give us a histogram of the appearance of each letter of CT in the 1st, 8th, 15th, 22nd, etc. positions. In PARI we type k=7, l=478, \r ctnum2.txt, \r hist.txt The output is 7 vectors of length 26. The nth is the histogram giving the frequency of each letter in the nth column if we write the CT as 7 columns. So the first vector gives the histogram of the appearance of each letter of CT in the 1st, 8th, 15th, 22nd, etc. positions.

w	z	g	g	q	b	u
a	w	q	p	v	h	e
i	r	r	b	v	n	y

Here it is with the number of appearances for the letters A - Z
$[10, 0, 0, 1, 1, 3, 7, 0, 0, 5, 7, 3, 2, 2, 0, 0, 1, 0, 4, 1, 2, 3, 10, 0, 1, 6]$. We know that E, T, A, are the most frequently appearing letters. The distances between them are A - 4 - E - 15 - T - 7 - A. So for keylength 7 and ciphertext of length 478, I will assume that each of these letters must show up at least 3 times in each of the seven sets/histograms. So we look in the histogram for three letters, each of which appears at least 3 times and which have distances 4 - 15 - 7 apart. If there is more than one such triple, then we will sum the number of appearances of each of the 3 letters and assign higher preferences to the shifts giving the greatest sums.

For the histogram above we note that a shift of 6 has appearance sum $7 + 7 + 6 = 20$ whereas a shift of 18 has sum 17. We can similarly make a histogram for the ciphertext letters in the 2nd, 9th, 16th, etc. positions and for the 3rd, ..., the 4th, ..., the 5th, ..., the 6th, ... and the 7th, ... positions. For the second, the only shift is 4. For the third, the shift of 0 has sum 11 and 2 has sum 17. For the fourth, the shift 7 has sum 14 and shift 19 has sum 20. For the fifth, shift 8 has sum 16 and shift 11 has sum 12. For the sixth, shift 2 has sum 13 and shift 14 has sum 22. For the seventh, shift 1 has sum 17 and shift 13 has sum 21. So our first guess is that the shifts are [6,4,2,19,8,14,13]. We can decrypt using this as a keyword and the first seven letter of plaintext are QVENINH. That Q seems wrong. The other guess for the first shift was 18. Let's try that. We get EVEN IN HIS OWN TIME ....

29 Cryptanalysis of modern stream ciphers

We will discuss the cryptanalysis of two random bit generators from the 1970's. Each is a known plaintext attack. To crack the first one, we need to learn about continued fractions

29.1 Continued Fractions

A simple continued fraction is of the form

$$a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \cfrac{1}{a_n}}}}$$

$a_i \in \mathbf{Z}$ and $a_i \neq 0$ if $i > 1$. This is often written $[a_1, a_2, \ldots, a_n]$ for typesetting purposes. Every rational number (fraction of integers) has a simple continued fraction. Example $27/8 = 3 + 1/(2 + 1/(1 + 1/(2))) = [3, 2, 1, 2]$. If we let the expansion continue forever we get

something looking like $\alpha = a_1 + 1/(a_2 + 1/(a_3 + \ldots = [a_1, a_2, \ldots$. This is called an infinite simple continued fraction. α is defined to be $\lim_{n\to\infty}[a_1, a_2, \ldots, a_n]$ which is a real number. Conversely, every irrational number has a unique infinite simple continued fraction. The rational numbers
$a_1, a_1 + 1/a_2$, $a_1 + 1/(a_2 + 1/a_3)$, (or $[a_1], [a_1, a_2], [a_1, a_2, a_3]$) are called the partial quotients or convergents of α.

The convergents are very good rational approximations of α. Conversely, every "very good" rational approximation of α is a convergent (I won't explain "very good" precisely). For example, 22/7 is a famous very good rational approximation to π. Indeed it is a convergent. We have $\pi = 3 + 1/(7 + 1/(15 + \ldots$. The first three convergents are 3, $3 + 1/7 = 22/7 = 3.14\ldots$, and $3 + 1/(7 + 1/15) = 333/106 = 3.1415\ldots$. If α is irrational and $|\alpha - \frac{a}{b}| < \frac{1}{2b^2}$ where $a, b \in \mathbf{Z}$, $b \geq 1$ then $\frac{a}{b}$ is a convergent to α.

Given α, here's how to find the a_i's. Let $\lfloor\alpha\rfloor$ be the greatest integer $\leq \alpha$. So $\lfloor\pi\rfloor = 3$, $\lfloor 5\rfloor = 5$, $\lfloor -1.5\rfloor = -2$. Let $\alpha_1 = \alpha$ and $a_1 = \lfloor\alpha_1\rfloor$. Let $\alpha_2 = 1/(\alpha_1 - a_1)$ and $a_2 = \lfloor\alpha_2\rfloor$. Let $\alpha_3 = 1/(\alpha_2 - a_2)$ and $a_3 = \lfloor\alpha_3\rfloor$, etc.

Here is a cool example. Let $e = 2.71828\ldots$. Then $\alpha_1 = e$, $a_1 = \lfloor\alpha_1\rfloor = 2$. $\alpha_2 = \frac{1}{e-2} = 1.39\ldots$, $a_2 = \lfloor 1.39\ldots\rfloor = 1$. $\alpha_3 = \frac{1}{1.39\ldots-2} = 2.55\ldots$, $a_3 = \lfloor 2.55\ldots\alpha_2\rfloor = 1$, $\alpha_4 = \frac{1}{2.55\ldots-2} = 1.82\ldots$, $a_4 = \lfloor 1.82\ldots\alpha_2\rfloor = 1$, $\alpha_5 = \frac{1}{1.82\ldots-1} = 1.22\ldots$, $a_5 = \lfloor 1.22\ldots\alpha_2\rfloor = 1$, $\alpha_6 = \frac{1}{1.22\ldots-1} = 4.54\ldots$, $a_6 = \lfloor 4.54\ldots\alpha_2\rfloor = 4$, etc. In fact, the continued fraction is $[2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10, \ldots]$, the pattern continues.

29.2 b/p Random Bit Generator

This is a (pseudo)-random bit generator that was used in the 1970's. Let p be a large prime for which 2 generates $\mathbf{F}_p^*$. Choose $1 \leq b < p$ and let b/p be the key. Write out the base 2 "decimal" expansion of $b/p = 0.k_1k_2k_3k_4\ldots$ (where k_i is a bit). Then $k_1k_2k_3\ldots$ is the keystream. It repeats every $p - 1$ bits.

Example. Let $p = 11$. Since $2^2 \neq 1$ and $2^5 \neq 1 \in \mathbf{F}_{11}$ we see that 2 generates $\mathbf{F}_{11}^*$. Let the key be 5/11. In base 2 that is 101/1011. Here's how you expand that to a "decimal".

```
         0.01110...
     -----------
1011 | 101.00000
        10 11
       ------
        10 010
         1 011
        ------
           1110
           1011
           ----
             110 etc.
```

For our homework, we'll do something awkward that's easier to calculate. Let p be a prime for which 10 generates $\mathbf{F}_p^*$. Choose $1 \leq b < p$. Write out the decimal expansion of

$b/p = 0.n_1n_2n_3\ldots$ (with $0 \le n_i \le 9$). Then $n_1n_2n_3\ldots$ is the keystream. Since 10 generates, it repeats every $p-1$ digits (as seldom as possible).

Example, $b = 12, p = 17$, $b/p = .70588235294117647058\ldots$, $n_1 = 7$, $n_2 = 0$, etc. Say you have plaintext like $MONTEREY$. Turn each letter into a pair of digits $1214131904170424 = p_1p_2p_3\ldots$, $0 \le p_i \le 9$, so $p_1 = 1$, $p_2 = 2$, $p_3 = 1$, etc.

To encrypt, $c_i = p_i + n_i(\text{mod}10)$, and the ciphertext is $c_1c_2c_3\ldots$. To decrypt $p_i = c_i - n_i(\text{mod}10)$.

```
Example       (encryption)                    (decryption)
             PT 12141319                        CT 82629544
    + keystream 70588325                  - keystream 70588235
    --------------------                  --------------------
             CT 82629544                        PT 12141319
 (note there's no carrying)                      M O N T
```

Here's a known plaintext attack. Say the number p has l digits and Eve has the ciphertext and the first $2l+1$ plaintext digits/bits. She can find b and p using simple continued fractions.

Say Eve has the ciphertext and the first 3n "bits" of plaintext. So she can get the first $3n$ "bits" of keystream. She finds the first convergent to the keystream that agrees with the first $2n+1$ "bits" of the keystream. She sees if it agrees with the rest. If $n > \log(p)$ then this will succeed.

In the following example, we have the first 18 digits of PT, so $n = 6$.

```
      CT  5309573992060 746098818740243
      PT  0200170405201 11704
keystream 5109403597869 63905
```

We find the continued fraction of .510940359786963905 and get
$[0, 1, 1, 22, 2, 1, 5, 1, 1, 3, 2, 4, 308404783, 1, 2, 1, 2\ldots]$. The convergents are $0, 1, 1/2, 23/45, 47/92, \ldots,$. The convergent $[0, 1, 1, \ldots 1, 3, 2] = 6982/13665 = .51094035858\ldots$ is not right but the next one $[0, 1, 1, \ldots 1, 3, 2, 4] = 30987/60647 = .510940359786963095 8815769987$. It is the first one that agrees with the first 13 digits of keystream and it also agrees with the following 5 so we are confident that it is right.

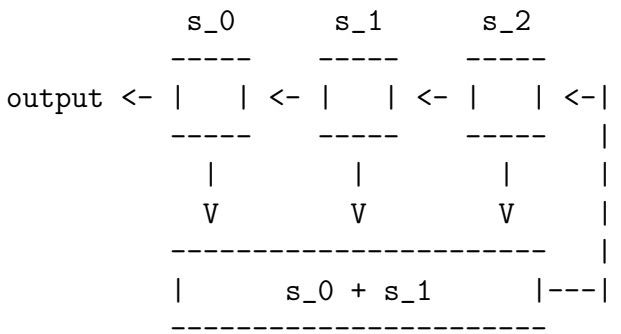
```
      CT  5309573992060 746098818740243
      PT  0200170405201 117040003081306 = CAREFULREADING
keystream 5109403597869 639058815769947
```

Actually we can intercept any consecutive bits, not just the first ones, it is just a bit more complicated. It still works as long as $n \ge p$.

29.3 Linear Shift Register Random Bit Generator

The following random bit generator is quite efficient, though as we will see, not safe. It is called the linear shift register (LSR) and was popular in the late 1960's and early 1970's. They are still used for hashing and check sums and there are non-linear shift registers still being used as random bit generators.

Here's an example.



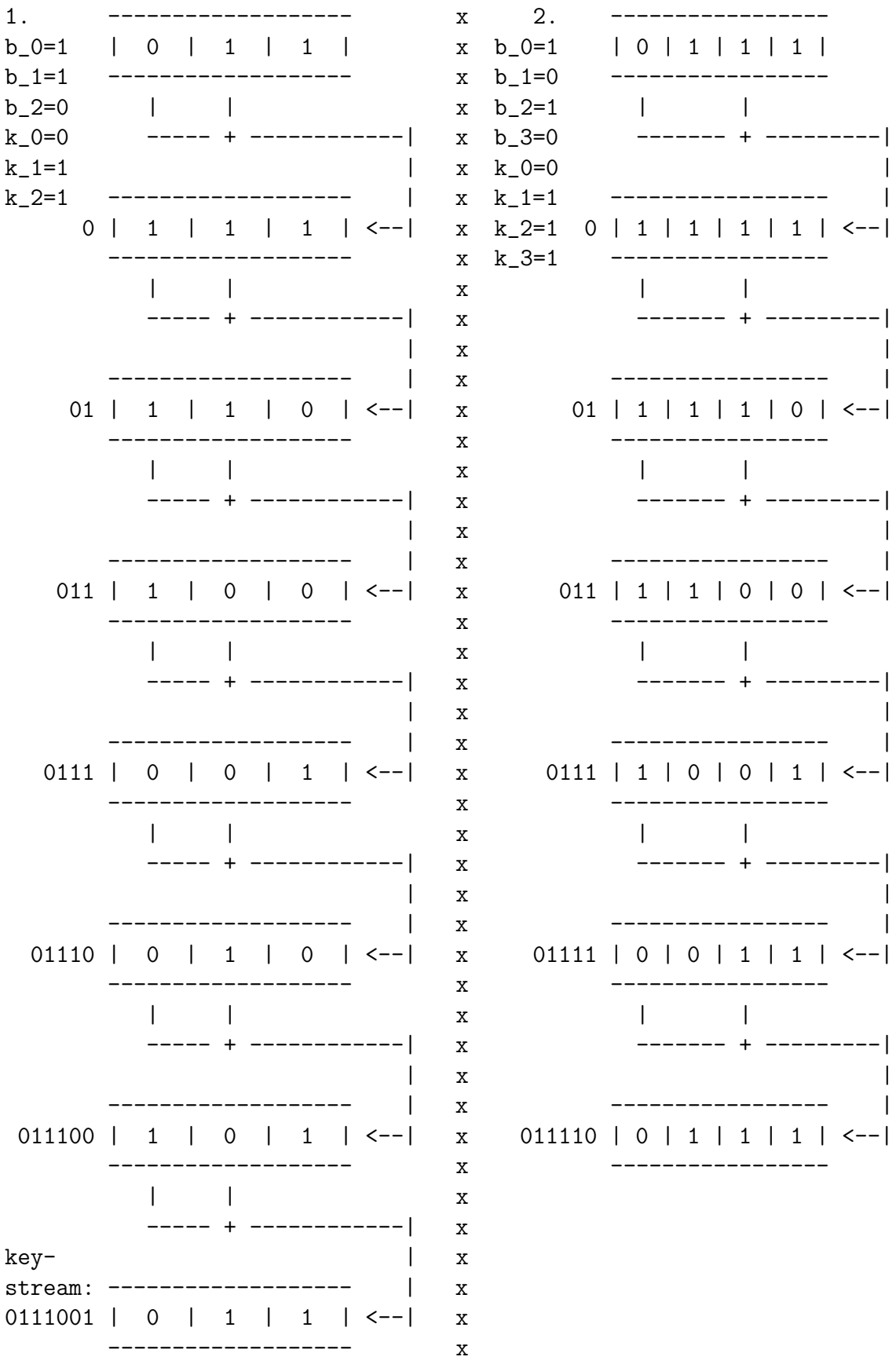
The output is the keystream. Let's start with an initial state of $(s_0, s_1, s_2) = (0, 1, 1)$. This is in figure 1 on the next page. Starting with figure 1, we will make the small boxes adjacent and include down-arrows only at the boxes contributing to the sum. At the bottom of the figure you see that we come back to the initial state so the keystream will start repeating the same 7 bit string.

That was a 3-stage LSR. For an n-stage LSR (32 is the most common for cryptography), the key/seed is $2n$ bits called $b_0, \ldots, b_{n-1}, k_0, \ldots, k_{n-1}$, all $\in \{0, 1\}$ where $b_0 \neq 0$ and not all of the k_i's are 0's.

The first set of $(s_0, \ldots s_{n-1})$ is called the initial state and it's $(k_0, \ldots, k_{n-1})$. In the last example we had $(k_0, k_1, k_2) = (0, 1, 1)$. The function giving the last bit of the next state is $f(s_0, \ldots, s_{n-1}) = b_0 s_0 + b_1 s_1 + \ldots + b_{n-1} s_{n-1}$. In the last example, $f = s_0 + s_1 = 1s_0 + 1s_1 + 0s_2$ so $(b_0, b_1, b_2) = (1, 1, 0)$. The state is $(s_0, \ldots, s_{n-1})$ and we move from state to state. At each state, s_i is the same as s_{i+1} from the previous state, with the exception of s_{n-1} which is the output of f.

For a fixed f (i.e. a fixed set of b_i's) there are 2^n different initial states for an n-stage LSR. We say that 2 states are in the same orbit if one state leads to another. $000\ldots0$ has its own orbit. In the example, all other seven states are in one single orbit. So our keystream repeats every $7 = 2^3 - 1$ (which is best possible).

Let's consider the 4-stage LSR with $(b_0, b_1, b_2, b_3) = (1, 0, 1, 0)$. We'll find the orbit size of the state $(0, 1, 1, 1)$. See figure 2. The orbit size is 6. So the keystream repeats every 6. We would prefer a keystream to repeat every $2^4 - 1 = 15$. A 4-stage LSR with $(b_0, b_1, b_2, b_3) = (1, 0, 0, 1)$ has orbit sizes of 15 and 1.


1.
b_0=1
b_1=1
b_2=0
k_0=0
k_1=1
k_2=1
0
01
011
0111
01110
011100
key-
stream:
0111001
2.
b_0=1
b_1=0
b_2=1
b_3=0
k_0=0
k_1=1
k_2=1
k_3=1
0
01
011
0111
01111
011110

Let's consider the 5-stage LSR with $(b_0, b_1, b_2, b_3, b_4) = (1, 1, 1, 0, 1)$. Find the orbit size of state $(1, 1, 0, 0, 1)$. We see it has orbit length $31 = 2^5 - 1$. The output keystream is 1100110111110100010010101100001. Note that all states other than $(0, 0, 0, 0, 0)$ appear. That also means that all possible consecutive strings of length 5 of 0's and 1's (other than 00000) appear exactly once in the above keystream.

How to tell if there are two orbits of sizes 1 and $2^n - 1$. Let $g(x) = b_0 + b_1x^1 + b_2x^2 + \ldots + b_{n-1}x^{n-1} + x^n$. You get a maximal length orbit (size $2^n - 1$) exactly when $g(x)$ is primitive mod 2 (i.e. over $\mathbf{F}_2$). We say $g(x)$ is primitive mod 2 if it is irreducible mod 2 and x is a generator of $\mathbf{F}_{2^n}^* = \mathbf{F}_2[x]/(g(x))^*$. If $2^n - 1$ is prime, then it is sufficient that $g(x)$ be irreducible.

So we want to pick $(b_0, \ldots, b_{n-1})$ so that $g(x)$ is irreducible and x generates $\mathbf{F}_{2^n}$. That way the keystream repeats as seldom as possible. For a 32-stage LSR there are about 67 million different irreducible polynomials.

Note in the first example we had $(b_0, b_1, b_2) = (1, 1, 0)$ which corresponds to $1 + x + x^3$ which is irreducible and it did have a maximal length orbit. In the second example we had $(b_0, b_1, b_2, b_3) = (1, 0, 1, 0)$ which corresponds to $1 + x^2 + x^4 = (1 + x + x^2)^2$. The polynomial isn't irreducible and we didn't get a maximal length orbit. The third example was $(b_0, b_1, b_2, b_3, b_4) = (1, 1, 1, 0, 1)$ which corresponds to $1 + x + x^2 + x^4 + x^5$ which is irreducible and again there was a maximal length orbit.

As an example, encrypt Hi with $(b_0, \ldots, b_4)(k_0, \ldots, k_4) = (1, 1, 1, 0, 1)(1, 1, 0, 0, 1)$. We get the keystream of the third example. The plaintext Hi = 0100100001101001 in ASCII.

```
      PT 0100100001101001
keystream 1100110111110100
      CT 1000010110011101
```

Note CT+keystream=PT too and PT+CT=keystream. This is a nice property of XOR.

How can we crack this? Let's say we intercept CT and the first $2n$ bits of PT (so this is a known plaintext attack). Then we can get the first $2n$ bits of keystream $k_0, \ldots, k_{2n-1}$. Then we can generate the whole $2^n - 1$ keystream. Let's say that the proper users are using $f = s_0 + s_1 + s_2 + s_4$ as in the third example, though we don't know this. We do know $k_0k_1k_2k_3k_4k_5k_6k_7k_8k_9 = 1100110111$.

know	don't know	can write
k_5	$= k_0 + k_1 + k_2 + k_4$	$= b_0k_0 + b_1k_1 + b_2k_2 + b_3k_3 + b_4k_4$
k_6	$= k_1 + k_2 + k_3 + k_5$	$= b_0k_1 + b_1k_2 + b_2k_3 + b_3k_4 + b_4k_5$
k_7	$= k_2 + k_3 + k_4 + k_6$	$= b_0k_2 + b_1k_3 + b_2k_4 + b_3k_5 + b_4k_6$
k_8	$= k_3 + k_4 + k_5 + k_7$	$= b_0k_3 + b_1k_4 + b_2k_5 + b_3k_6 + b_4k_7$
k_9	$= k_4 + k_5 + k_6 + k_8$	$= b_0k_4 + b_1k_5 + b_2k_6 + b_3k_7 + b_4k_8$

In general $k_{t+n} = b_0k_t + b_1k_{t+1} + \ldots + b_{n-1}k_{t+n-1}$. We have n linear equations (over $\mathbf{F}_2$) in n unknowns (the b_i's). We re-write this as a matrix equation.

$$\begin{bmatrix} k_0 & k_1 & k_2 & k_3 & k_4 \\ k_1 & k_2 & k_3 & k_4 & k_5 \\ k_2 & k_3 & k_4 & k_5 & k_6 \\ k_3 & k_4 & k_5 & k_6 & k_7 \\ k_4 & k_5 & k_6 & k_7 & k_8 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} k_5 \\ k_6 \\ k_7 \\ k_8 \\ k_9 \end{bmatrix}$$

We then fill in the known values for the k_i's and get

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

We solve this matrix equation over $\mathbf{F}_2$ and get $b_0b_1b_2b_3b_4 = 11101$.

Now you know the b_i's and you know the initial state (the first k_i's) so you can create the keystream yourself. End of example.

In general, we have

$$\begin{bmatrix} k_0 & k_1 & k_2 & \dots & k_{n-1} \\ k_1 & k_2 & k_3 & \dots & k_n \\ k_2 & k_3 & k_4 & \dots & k_{n+1} \\ & & \vdots & & \\ k_{n-1} & k_n & k_{n+1} & \dots & k_{2n-2} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \end{bmatrix} = \begin{bmatrix} k_n \\ k_{n+1} \\ k_{n+2} \\ \vdots \\ k_{2n-1} \end{bmatrix}$$

Example. You know Alice and Bob are using a 6 stage LSR and this message starts with To (in ASCII) and you intercept the ciphertext 1011 0000 1101 1000 0010 0111 1100 0011.

```
CT 10110000110110000010011111000011
PT 0101010001101111
      T       o
   ----------------
KS 1110010010110111
```

We have

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Solving we get $b_0b_1b_2b_3b_4b_5 = 110000$. Since we know the b_i's and the first 6 bits of keystream, we can create the whole keystream and decrypt the rest of the plaintext.

In Pari, we type

```
s=[1,1,1,0,0,1,0,0,1,0,1,1]
\r mat6.txt
matsolve(m*Mod(1,2),v)
b=[1,1,0,0,0,0]
k=[1,1,1,0,0,1]
\r ctlsr.txt
\r declsr6.txt
```

We see that the plaintext is ToAl

What if we only knew the first character of plaintext is T. Then we would have to brute force the remaining 4 bits to get a total of 12 keybits. Note that only some of those possibilities would result in an invertible matrix over $\mathbf{F}_2$. Of those, we can see which give keystreams that give sensible plaintext.

End example.

There are people using non-linear shift registers as random bit generators. Example $f = s_0 s_2 s_7 + s_1 + s_2 s_4 + s_4$.

30 Cryptanalysis of block ciphers

30.1 Brute Force Attack

A brute force attack is a known plaintext attack. Eve has one known plaintext/ciphertext pair. Let us say the keys are b bits. She encrypts the plaintext with all 2^b keys and sees which gives the ciphertext. There will probably be few keys that do it. If there is more than one candidate, then she tries the few candidates on a second plaintext/ciphertext pair. Probably only one will work. She expects success half way through, so the number of attempts is one half the size of the key space. So for DES this is 2^{55} attempts and for AES it is 2^{127} attempts.

30.2 Standard ASCII Attack

Single DES is a block cipher with a 64 bit plaintext, a 64 bit ciphertext and a 56 bit key. Here is a ciphertext-only attack if the plaintext is standard (not extended) ASCII. So the first bit of each byte is a 0. Decrypt CT_1 with all 2^{56} keys. Find which ones give standard ASCII PT. That should be $1/2^8$ of them or 2^{48} keys. Decrypt CT_2 with the 2^{48} keys. Again only $1/2^8$ of those or 2^{40} should give standard ASCII. After decrypting CT_3 with 2^{40} keys will have 2^{32} keys, ... After CT_7 will probably only have one key left. This attack requires $2^{56} + 2^{48} + 2^{40} + 2^{32} + \ldots + 2^8$ attempts. That number $\approx 2^{56}$ so only twice as bad as known PT attack.

30.3 Meet-in-the-Middle Attack

The meet-in-the-middle attack on a double block cipher was invented by Diffie and Hellman. It is a known plaintext attack also. Let E_{K_1} denote encryption with key K_1 and a given block cipher. Then $CT = E_{K_2}(E_{K_1}(PT))$ (where PT = plaintext and CT = ciphertext) is a double block cipher.

Eve needs two known plaintext/ciphertext pairs. She encrypts the first plaintext (with the single block cipher) with all keys and saves the results. She then decrypts the matching ciphertext with all keys and saves the results. She then finds the key pairs $\mathrm{key}_{1,n_1}, \mathrm{key}_{2,n_2}$ where $E_{\mathrm{key}_{1,n_1}}(PT_1) = D_{\mathrm{key}_{2,n_2}}(CT_1)$ (and D denotes decryption). For each successful key pair, she computes $E_{\mathrm{key}_{2,n_2}}(E_{\mathrm{key}_{1,n_1}}(PT_2))$ to see if she gets CT_2. There will probably be only one pair for which this works. This attack requires a huge amount of storage. But note, for

a block cipher with a b-bit key, that the number of steps is about $3(2^b)$. So a double block cipher is not much more secure than a single block cipher with a single key.

You can use a meet-in-the-middle attack to show that Triple-DES with three keys is not much more secure than Triple-DES with two keys. Two keys are easier to agree on, so Triple-DES is done with two keys, as described in Section 11.2.

Example. Let's say that we have a block cipher with two-bit plaintexts, ciphertexts and keys. The entire cipher is described by the following table with the entry in the 4 by 4 array being the ciphertext for the given plaintext and key.

PT\ key	00	01	10	11
00	01	10	00	11
01	10	01	11	10
10	00	11	01	01
11	11	00	10	00

Let's say that Alice is using double encryption with two keys. You know that when she encrypts $\mathrm{PT}_1 = 10$ she gets $\mathrm{CT}_1 = 00$ and when she encrypts $\mathrm{PT}_2 = 00$ she gets $\mathrm{CT}_2 = 01$.

You first single encrypt $\mathrm{PT}_1 = 10$ with all keys and get

$$\begin{array}{cccc} & & 10 & \\ E_{00}\swarrow & E_{01}\swarrow & & \searrow E_{10} \quad \searrow E_{11} \\ 00 & 11 & & 01 \quad 01 \end{array}$$

You then single decrypt $\mathrm{CT}_1 = 00$ with all keys and get

$$\begin{array}{cccc} 10 & 11 & & 00 \quad 11 \\ D_{00}\nwarrow & D_{01}\nwarrow & & \nearrow D_{10} \quad \nearrow D_{11} \\ & & 00 & \end{array}$$

We now search for matches where $E_{i,j}(10) = D_{k,l}(00)$. There are three of them as shown in the next diagram, one going from 00 to 00 and two connecting 11 to 11.

$$\begin{array}{cccccc} & & & 10 & & \\ E_{00}\swarrow & & E_{01}\swarrow & & \searrow E_{10} & \quad \searrow E_{11} \\ 00 & & 11 & & 01 & \quad 01 \\ & \searrow & \downarrow & \searrow & & \\ & & \downarrow & & \searrow & \\ & & \downarrow & \searrow & & \searrow \\ 10 & & 11 & & 00 & \quad 11 \\ D_{00}\nwarrow & & D_{01}\nwarrow & & \nearrow D_{10} & \quad \nearrow D_{11} \\ & & & 00 & & \end{array}$$

Let's consider the arrow pointing straight down. We have $E_{01}(E_{01}(\mathrm{PT}_1 = 10) = 11) = 00 = \mathrm{CT}_1$. The arrow going from 00 to 00 indicates that $E_{10}(E_{00}(10)) = 00$ and the other arrow going from 11 to 11 indicates that $E_{11}(E_{01}(10)) = 00$. So Alice used the two keys $(01, 01)$, $(00, 10)$ or $(01, 11)$.

Recall that when Alice double encrypted $\mathrm{PT}_2 = 00$ she got $\mathrm{CT}_2 = 01$. Now $E_{01}(E_{01}(00)) = 11 \neq \mathrm{CT}_2$, $E_{10}(E_{00}(00)) = 11 \neq \mathrm{CT}_2$ and $E_{11}(E_{01}(00)) = 01 = \mathrm{CT}_2$. So Alice's key pair is $(01, 11)$.

30.4 One-round Simplified AES

We will use one-round simplified AES in order to explain linear and differential cryptanalysis.

One-round Simplified AES

Recall S-box is a map from 4 bit strings to 4 bit strings. It is not a linear map. So, for example, S-box$(b_0, b_1, b_2, b_3) \neq b_0 + b_1, b_1 + b_2 + b_3, b_0, b_1 + b_2$. We have a key $K_0 = k_0k_1k_2k_3k_4k_5k_6k_7k_8k_9k_{10}k_{11}k_{12}k_{13}k_{14}k_{15}$. To expand the key, we

	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}
				$\swarrow$	$\searrow$			
	k_{12}	k_{13}	k_{14}	k_{15}	k_8	k_9	k_{10}	k_{11}
	S	b	o	x	S	b	o	x
	n_0	n_1	n_2	n_3	n_4	n_5	n_6	n_7
	1	0	0	0	0	0	0	0
$\oplus$	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7
$=$	k_{16}	k_{17}	k_{18}	k_{19}	k_{20}	k_{21}	k_{22}	k_{23}
$\oplus$	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}
$=$	k_{24}	k_{25}	k_{26}	k_{27}	k_{28}	k_{29}	k_{30}	k_{31}

We have $K_1 = k_{16}k_{17}k_{18}k_{19}k_{20}k_{21}k_{22}k_{23}k_{24}k_{25}k_{26}k_{27}k_{28}k_{29}k_{30}k_{31}$.
For linear cryptanalysis later note: $k_{16} + k_{24} = k_8$ and $k_{17} + k_{25} = k_9$, ..., $k_{23} + k_{31} = k_{15}$.
Let's recall encryption:

$p_0p_1p_2p_3$	$p_8p_9p_{10}p_{11}$
$p_4p_5p_6p_7$	$p_{12}p_{13}p_{14}p_{15}$

$\xrightarrow{A_{K_0}}$

$p_0+k_0 \;\; p_1+k_1 \;\; p_2+k_2 \;\; p_3+k_3$	$p_8+k_8 \;\; p_9+k_9 \;\; p_{10}+k_{10} \;\; p_{11}+k_{11}$
$p_4+k_4 \;\; p_5+k_5 \;\; p_6+k_6 \;\; p_7+k_7$	$p_{12}+k_{12} \;\; p_{13}+k_{13} \;\; p_{14}+k_{14} \;\; p_{15}+k_{15}$

Let S-box$(p_0p_1p_2p_3 + k_0k_1k_2k_3) = m_0m_1m_2m_3$, and so on. After NS the state is then

$\xrightarrow{NS}$

$m_0m_1m_2m_3$	$m_8m_9m_{10}m_{11}$
$m_4m_5m_6m_7$	$m_{12}m_{13}m_{14}m_{15}$

$\xrightarrow{SR}$

$m_0m_1m_2m_3$	$m_8m_9m_{10}m_{11}$
$m_{12}m_{13}m_{14}m_{15}$	$m_4m_5m_6m_7$

After MC the state is then

$\xrightarrow{MC}$

$m_0+m_{14} \;\; m_1+m_{12}+m_{15} \;\; m_2+m_{12}+m_{13} \;\; m_3+m_{13}$	$m_6+m_8 \;\; m_4+m_7+m_9 \;\; m_4+m_5+m_{10} \;\; m_5+m_{11}$
$m_2+m_{12} \;\; m_0+m_3+m_{13} \;\; m_0+m_1+m_{14} \;\; m_1+m_{15}$	$m_4+m_{10} \;\; m_5+m_8+m_{11} \;\; m_6+m_8+m_9 \;\; m_7+m_9$

.

Lastly, we add K_1. The state is then

$\xrightarrow{A_{K_1}}$

$m_0+m_{14}+k_{16} \;\; \ldots \; m_3+m_{13}+k_{19}$	$m_6+m_8+k_{24} \;\; \ldots \; m_5+m_{11}+k_{27}$
$m_2+m_{12}+k_{20} \;\; \ldots \; m_1+m_{15}+k_{23}$	$m_4+m_{10}+k_{28} \;\; \ldots \; m_7+m_9+k_{31}$

$=$

$c_0c_1c_2c_3$	$c_8c_9c_{10}c_{11}$
$c_4c_5c_6c_7$	$c_{12}c_{13}c_{14}c_{15}$

We'll denote the cells:

cell 1	cell 3
cell 2	cell 4

30.5 Linear Cryptanalysis

Linear cryptanalysis is a known plaintext attack. It was invented by Matsui and published in 1992. You need a lot of matched plaintext/ciphertext pairs from a given key. Let's say that the plaintext block is $p_0 \ldots p_{n-1}$, the key is $k_0 \ldots k_{m-1}$ and the corresponding ciphertext block is $c_0 \ldots c_{n-1}$.

Say that the linear equation $p_{\alpha_1} + p_{\alpha_2} + \ldots + p_{\alpha_a} + c_{\beta_1} + \ldots + c_{\beta_b} + k_{\gamma_1} + \ldots + k_{\gamma_g} = x$ (where $x = 0$ or 1, $1 \leq a \leq n$, $1 \leq b \leq n$, $1 \leq g \leq m$) holds with probability $p > .5$ over all plaintext/key pairs. So $x + p_{\alpha_1} + \ldots + c_{\beta_b} = k_{\gamma_1} + \ldots k_{\gamma_g}$ with probability $p > .5$.

Compute $x + p_{\alpha_1} + \ldots + c_{\beta_b}$ over all intercepted plaintext/ciphertext pairs. If it's 0 most of the time, assume $k_{\gamma_1} + \ldots k_{\gamma_g} = 0$. If it's 1 most of the time, assume $k_{\gamma_1} + \ldots k_{\gamma_g} = 1$. Now you have a relation on key bits. Try to get several relations.

		linear	non-linear
x_0	x_1	$x_0 + x_1$	$x_0 x_1$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Linear cryptanalysis of 1 round Simplified AES

Let S-box$(a_0 a_1 a_2 a_3) = b_0 b_1 b_2 b_3$. We have

a_0	a_1	a_2	a_3	b_0	b_1	b_2	b_3
0	0	0	0	1	0	0	1
0	0	0	1	0	1	0	0
0	0	1	0	1	0	1	0
0	0	1	1	1	0	1	1
0	1	0	0	1	1	0	1
0	1	0	1	0	0	0	1
0	1	1	0	1	0	0	0
0	1	1	1	0	1	0	1
1	0	0	0	0	1	1	0
1	0	0	1	0	0	1	0
1	0	1	0	0	0	0	0
1	0	1	1	0	0	1	1
1	1	0	0	1	1	0	0
1	1	0	1	1	1	1	0
1	1	1	0	1	1	1	1
1	1	1	1	0	1	1	1

I computed the output of all linear combinations of the a_i's and b_i's. Here are a few of the relations I found with probabilities higher than .5 (there are no relations of probability $> .75$).

$$a_3 + b_0 = \quad 1110111101011011 \quad = 1, p = 12/16 = .75 \quad (1)$$
$$a_0 + a_1 + b_0 = \quad 1011010111111110 \quad = 1, p = .75 \quad (2)$$
$$a_1 + b_1 = \quad 0100011010000000 \quad = 0, p = .75 \quad (3)$$
$$a_0 + a_1 + a_2 + a_3 + b_1 \quad = 0, p = .75 \quad (4)$$
$$a_1 + a_2 + b_0 + b_1 \quad = 1, p = .75 \quad (5)$$
$$a_0 + b_0 + b_1 \quad = 1, p = .75 \quad (6)$$

Finding Equations

In order to find equations of the form $p_{\alpha_1} + p_{\alpha_2} + \ldots + p_{\alpha_a} + c_{\beta_1} + \ldots + c_{\beta_b} + k_{\gamma_1} + \ldots + k_{\gamma_g} = x$, we need to eliminate the m_i's and n_i's. We will first eliminate the n_i's.

$$
\begin{aligned}
c_0 + k_{16} + c_8 + k_{24} = c_0 + c_8 + k_8 &= m_0 + m_{14} + m_6 + m_8 && (7)\\
c_1 + k_{17} + c_9 + k_{25} = c_1 + c_9 + k_9 &= m_1 + m_{12} + m_{15} + m_4 + m_7 + m_9 && (8)\\
c_2 + c_{10} + k_{10} &= m_2 + m_{12} + m_{13} + m_4 + m_5 + m_{10} && (9)\\
c_3 + c_{11} + k_{11} &= m_3 + m_{13} + m_5 + m_{11} && (10)\\
c_4 + c_{12} + k_{12} &= m_2 + m_{12} + m_4 + m_{10} && (11)\\
c_5 + c_{13} + k_{13} &= m_0 + m_3 + m_{13} + m_5 + m_8 + m_{11} && (12)\\
c_6 + c_{14} + k_{14} &= m_0 + m_1 + m_{14} + m_6 + m_8 + m_9 && (13)\\
c_7 + c_{15} + k_{15} &= m_1 + m_{15} + m_7 + m_9. && (14)
\end{aligned}
$$

The right-hand side of each of those eight equations involves all four cells of a state. So to use these equations as they are and combine them with equations like 1 through 6 would give us equations with probabilities very close to .5 (we will see later where this tendency toward .5 comes from when combining equations).

Instead we notice that the bits appearing on the right-hand side of Equation 7 are a subset of those appearing in Equation 13. Similarly, the bits appearing on the right-hand side of equations 10, 11 and 14 are subsets of those appearing in Equations 12, 9 and 8, respectively. So if we add the Equations 7 and 13, then 10 and 12, then 11 and 9 and lastly 14 and 8, we get

$$
\begin{aligned}
c_0 + c_6 + c_8 + c_{14} + k_8 + k_{14} &= m_1 + m_9 && (15)\\
c_3 + c_5 + c_{11} + c_{13} + k_{11} + k_{13} &= m_0 + m_8 && (16)\\
c_2 + c_4 + c_{10} + c_{12} + k_{10} + k_{12} &= m_5 + m_{13} && (17)\\
c_1 + c_7 + c_9 + c_{15} + k_9 + k_{15} &= m_4 + m_{12}. && (18)
\end{aligned}
$$

Equations 15 - 18 are always true.

Let us consider Equation 16. We want to replace $m_0 + m_8$ with expressions in the p_i's and k_i's that hold with some probability higher than .5. Within a cell, both m_0 and m_8 correspond to the bit b_0 in the nibble $b_0b_1b_2b_3$. So we can use Equations 1 and 2.

Let's first use Equation 1 for both cells 1 and 3.

$p_0 + k_0$ $p_1 + k_1$ $p_2 + k_2$ $p_3 + k_3$	$p_8 + k_8$ $p_9 + k_9$ $p_{10} + k_{10}$ $p_{11} + k_{11}$
$p_4 + k_4$ $p_5 + k_5$ $p_6 + k_6$ $p_7 + k_7$	$p_{12} + k_{12}$ $p_{13} + k_{13}$ $p_{14} + k_{14}$ $p_{15} + k_{15}$

$\xrightarrow{NS}$

$m_0m_1m_2m_3$	$m_8m_9m_{10}m_{11}$
$m_4m_5m_6m_7$	$m_{12}m_{13}m_{14}m_{15}$

Recall the notation S-box$(a_0a_1a_2a_3) = b_0b_1b_2b_3$.

Equation 1 is $a_3 + b_0 = 1, p = .75$ (for cells 1 and 3).

$p_3 + k_3 + m_0 = 1, p = .75$

$p_{11} + k_{11} + m_8 = 1, p = .75$

$p_3 + p_{11} + k_3 + k_{11} + m_0 + m_8 = 0, p = (.75)^2 + (.25)^2 = .625$

$p_3 + p_{11} + k_3 + k_{11} = m_0 + m_8, p = .625$

Recall

$c_3 + c_5 + c_{11} + c_{13} + k_{11} + k_{13} = m_0 + m_8$ (16) always

Combining
$c_3 + c_5 + c_{11} + c_{13} + k_{11} + k_{13} = p_3 + p_{11} + k_3 + k_{11}, p = .625$
$p_3 + p_{11} + c_3 + c_5 + c_{11} + c_{13} = k_3 + k_{13}$ (19), $p = .625$
This is our first of the kind of equation needed to do linear cryptanalysis.

Next we use Equation 1 on cell 1 and Equation 2 on cell 3 (we will again combine with Equation 16).

$p_0 + k_0$ $p_1 + k_1$ $p_2 + k_2$ $p_3 + k_3$	$p_8 + k_8$ $p_9 + k_9$ $p_{10} + k_{10}$ $p_{11} + k_{11}$
$p_4 + k_4$ $p_5 + k_5$ $p_6 + k_6$ $p_7 + k_7$	$p_{12} + k_{12}$ $p_{13} + k_{13}$ $p_{14} + k_{14}$ $p_{15} + k_{15}$

$\xrightarrow{NS}$

$m_0 m_1 m_2 m_3$	$m_8 m_9 m_{10} m_{11}$
$m_4 m_5 m_6 m_7$	$m_{12} m_{13} m_{14} m_{15}$

Recall the notation S-box$(a_0 a_1 a_2 a_3) = b_0 b_1 b_2 b_3$.
Equation 1 is $a_3 + b_0 = 1, p = .75$ (for cell 1).
Equation 2 is $a_0 + a_1 + b_0 = 1, p = .75$ (for cell 3).
$p_3 + k_3 + m_0 = 1, p = .75$
$p_8 + k_8 + p_9 + k_9 + m_8 = 1, p = .75$
$p_3 + p_8 + p_9 + c_3 + c_5 + c_{11} + c_{13} = m_0 + m_8, p = .625$
Recall
$c_3 + c_5 + c_{11} + c_{13} + k_{11} + k_{13} = m_0 + m_8$ (16) always
Combining
$p_3 + p_8 + p_9 + c_3 + c_5 + c_{11} + c_{13} = k_3 + k_8 + k_9 + k_{11} + k_{13}$ (20), $p = .625$
This is the second of the kind of equation needed to do linear cryptanalysis.

If we use Equation 2 on both cells 1 and 3 we get $p_0 + p_1 + p_8 + p_9 + c_3 + c_5 + c_{11} + c_{13} = k_0 + k_1 + k_8 + k_9 + k_{11} + k_{13}$ (21).
Using Equation 2 on cell 1 and Equation 1 on cell 3 is now redundant as it gives the same information as we get by combining equations 19, 20 and 21.

For Equation 18 we can also use Equations 1 and 1, 1 and 2, and 2 and 2 on cells 2 and 4, respectively, to get equations 22 - 24.
For Equation 15 we can use Equations 3 and 3, 3 and 4, and 4 and 4 on cells 1 and 3, respectively, to get equations 25 - 27.
For Equation 17 we can use Equations 3 and 3, 3 and 4, and 4 and 4, on cells 2 and 4, respectively, to get equations 28 - 29.

If we add Equations 15 and 16 we get $c_0 + c_3 + c_5 + c_6 + c_8 + c_{11} + c_{13} + c_{14} + k_8 + k_{11} + k_{13} + k_{14} = m_0 + m_1 + m_8 + m_9$.

$p_0 + k_0$ $p_1 + k_1$ $p_2 + k_2$ $p_3 + k_3$	$p_8 + k_8$ $p_9 + k_9$ $p_{10} + k_{10}$ $p_{11} + k_{11}$
$p_4 + k_4$ $p_5 + k_5$ $p_6 + k_6$ $p_7 + k_7$	$p_{12} + k_{12}$ $p_{13} + k_{13}$ $p_{14} + k_{14}$ $p_{15} + k_{15}$

$\xrightarrow{NS}$

$m_0 m_1 m_2 m_3$	$m_8 m_9 m_{10} m_{11}$
$m_4 m_5 m_6 m_7$	$m_{12} m_{13} m_{14} m_{15}$

Recall the notation S-box$(a_0 a_1 a_2 a_3) = b_0 b_1 b_2 b_3$.
Equation 5 is $a_1 + a_2 + b_0 + b_1 = 1, p = .75$ (for cells 1 and 3).

$p_1 + k_1 + p_2 + k_2 + m_0 + m_1 = 1$, $p = .75$

$p_9 + k_9 + p_{10} + k_{10} + m_8 + m_9 = 1$, $p = .75$.

$p_1 + p_2 + p_9 + p_{10} + k_1 + k_2 + k_9 + k_{10} = m_0 + m_1 + m_8 + m_9$, $p = .675$.

Recall

$c_0 + c_3 + c_5 + c_6 + c_8 + c_{11} + c_{13} + c_{14} + k_8 + k_{11} + k_{13} + k_{14} = m_0 + m_1 + m_8 + m_9$, (15) + (16) always.

Combining

$p_1+p_2+p_9+p_{10} +c_0+c_3+c_5+c_6+c_8+c_{11}+c_{13}+c_{14} = k_1+k_2+k_8+k_9+k_{10}+k_{11}+k_{13}+k_{14}$ (31), $p = .675$.

It is tempting to use Equation 6 until one notices that it is the same as Equation 2 + Equation 3, and hence is redundant. If we add 17 and 18 we can also use Equation 5 on cells 2 and 4 to get equation (32).

The 14 equations are

$$
\begin{aligned}
p_3 + p_{11} + c_3 + c_5 + c_{11} + c_{13} &= k_3 + k_{13} && (19)\\
p_3 + p_8 + p_9 + c_3 + c_5 + c_{11} + c_{13} &= k_3 + k_8 + k_9 + k_{11} + k_{13} && (20)\\
p_0 + p_1 + p_8 + p_9 + c_3 + c_5 + c_{11} + c_{13} &= k_0 + k_1 + k_8 + k_9 + k_{11} + k_{13} && (21)\\
p_7 + p_{15} + c_1 + c_7 + c_9 + c_{15} &= k_7 + k_9 && (22)\\
p_7 + p_{12} + p_{13} + c_1 + c_7 + c_9 + c_{15} &= k_7 + k_9 + k_{12} + k_{13} + k_{15} && (23)\\
p_4 + p_5 + p_{12} + p_{13} + c_1 + c_7 + c_9 + c_{15} &= k_4 + k_5 + k_9 + k_{12} + k_{13} + k_{15} && (24)\\
p_1 + p_9 + c_0 + c_6 + c_8 + c_{14} &= k_1 + k_8 + k_9 + k_{14} && (25)\\
p_1 + p_8 + p_9 + p_{10} + p_{11} + c_0 + c_6 + c_8 + c_{14} &= k_1 + k_9 + k_{10} + k_{11} + k_{14} && (26)\\
p_0 + p_1 + p_2 + p_3 + p_8 + p_9 + p_{10} + p_{11} + c_0 + c_6 + c_8 + c_{14} &= k_0 + k_1 + k_2 + k_3 + k_9 + k_{10} + k_{11} + k_{14} && (27)\\
p_5 + p_{13} + c_2 + c_4 + c_{10} + c_{12} &= k_5 + k_{10} + k_{12} + k_{13} && (28)\\
p_5 + p_{12} + p_{13} + p_{14} + p_{15} + c_2 + c_4 + c_{10} + c_{12} &= k_5 + k_{10} + k_{13} + k_{14} + k_{15} && (29)\\
p_4 + p_5 + p_6 + p_7 + p_{12} + p_{13} + p_{14} + p_{15} + c_2 + c_4 + c_{10} + c_{12} &= k_4 + k_5 + k_6 + k_7 + k_{10} + k_{13} + k_{14} + k_{15} && (30)\\
p_1 + p_2 + p_9 + p_{10} + c_0 + c_3 + c_5 + c_6 + c_8 + c_{11} + c_{13} + c_{14} &= k_1 + k_2 + k_8 + k_9 + k_{10} + k_{11} + k_{13} + k_{14} && (31)\\
p_5 + p_6 + p_{13} + p_{14} + c_1 + c_2 + c_4 + c_7 + c_9 + c_{10} + c_{12} + c_{15} &= k_5 + k_6 + k_9 + k_{10} + k_{12} + k_{13} + k_{14} + k_{15}; && (32)
\end{aligned}
$$

they each hold with probability .625.

This set of linear equations can be represented by the matrix equation

$$
\begin{bmatrix}
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0\\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1\\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0\\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1\\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1\\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0\\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1
\end{bmatrix}
\begin{bmatrix}
k_0\\ k_1\\ k_2\\ k_3\\ k_4\\ k_5\\ k_6\\ k_7\\ k_8\\ k_9\\ k_{10}\\ k_{11}\\ k_{12}\\ k_{13}\\ k_{14}\\ k_{15}
\end{bmatrix}
=
\begin{bmatrix}
p_3 + p_{11} + c_3 + c_5 + c_{11} + c_{13}\\
\dots\\
\\
\\
\\
\\
\\
\\
\\
\\
\\
\\
\\
\\
\\
p_5 + p_6 + \dots c_{15}
\end{bmatrix}
$$

The rows are linearly independent, so there is no redundancy of information in the 14 equations.

The Attack

Eve takes the known plaintext/ciphertext pairs and evaluates $p_3 + p_{11} + c_3 + c_5 + c_{11} + c_{13}$ for all of them. If it's usually 0, then she puts 0 at the top of that vector. If it's usually 1, then she puts 1.

For simplicity, assume Eve has gotten all 14 choices of 0 or right 1. She now has 14 equations in 16 unknowns $(k_0, \ldots, k_{15})$, so there are $2^{16-14} = 4$ possible solutions for the key. The simplest way to extend the 14 rows of the matrix to a basis (of the 16-dimensional $\mathbf{F}_2$-vector space) is to include the vectors associate to k_0 and k_4. Eve uses the matrix equation

$$\begin{bmatrix} 0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&1&0&0&0&0&1&1&0&1&0&1&0&0 \\ 1&1&0&0&0&0&0&0&1&1&0&1&0&1&0&0 \\ 0&0&0&0&0&0&0&1&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&1&0&0&1&1&0&1 \\ 0&0&0&0&1&1&0&0&0&1&0&0&1&1&0&1 \\ 0&1&0&0&0&0&0&0&1&1&0&0&0&0&1&0 \\ 0&1&0&0&0&0&0&0&0&1&1&1&0&0&1&0 \\ 1&1&1&1&0&0&0&0&0&1&1&1&0&0&1&0 \\ 0&0&0&0&0&1&0&0&0&0&1&0&1&1&0&0 \\ 0&0&0&0&0&1&0&0&0&0&1&0&0&1&1&1 \\ 0&0&0&0&1&1&1&1&0&0&1&0&0&1&1&1 \\ 0&1&1&0&0&0&0&0&1&1&1&1&0&1&1&0 \\ 0&0&0&0&0&1&1&0&0&1&1&0&1&1&1&1 \\ 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix} \begin{bmatrix} k_0 \\ k_1 \\ k_2 \\ k_3 \\ k_4 \\ k_5 \\ k_6 \\ k_7 \\ k_8 \\ k_9 \\ k_{10} \\ k_{11} \\ k_{12} \\ k_{13} \\ k_{14} \\ k_{15} \end{bmatrix} = \begin{bmatrix} p_3 + p_{11} + c_3 + c_5 + c_{11} + c_{13} \\ \ldots \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ p_5 + p_6 + \ldots c_{15} \\ * \\ * \end{bmatrix}$$

In the vector at right, in the first 14 places, she puts in 0, or 1, whichever appeared more commonly. She brute forces the remaining two bits.

Let us say that Eve wants to be 95% certain that all 14 of the bit choices are correct. We can approximate the number n of different plaintexts and corresponding ciphertexts she will need for this certainty level. For $i = 19, \ldots, 32$, let $\hat{p}_i$ denote the random variable whose value is equal to the proportion of the n plaintexts and corresponding ciphertexts for which the left-hand side of Equation i is equal to the correct value (0 or 1) of the right-hand side, for the given key. For each i, the expected value of $\hat{p}_i$ is .625 and its variance is $(.625)(1 - .625)/n$. Therefore the standard deviation of $\hat{p}_i$ is $\sqrt{15/(64n)}$.

We want $\text{Prob}(\hat{p}_i > .5 \text{ for } i = 19 \ldots, 32) = .95$. We will assume that the $\hat{p}_i$'s are independent since we do not know any better and it is probably close to true. Thus we have $.95 = \text{Prob}(\hat{p}_i > .5 \text{ for } i = 19 \ldots, 32) = \text{Prob}(\hat{p}_i > .5)^{14}$, for any given i. Therefore we want $\text{Prob}(\hat{p}_i > .5) = \sqrt[14]{.95} = .99634$, for any given i.

For sufficiently large n, the random variable $\hat{p}_i$ is essentially normal. So we will standardize $\hat{p}_i$ by subtracting off its expected value and dividing by its standard deviation, which will give us (approximately) the standard normal random variable denoted Z.

We have

$$.99634 = \text{Prob}(\hat{p}_i > .5) = \text{Prob}\left(\frac{\hat{p}_i - .625}{\sqrt{\frac{15}{64n}}} > \frac{.5 - .625}{\sqrt{\frac{15}{64n}}}\right) = \text{Prob}\left(Z > \frac{-\sqrt{n}}{\sqrt{15}}\right).$$

Therefore

$$1 - .99634 = .00366 = \text{Prob}\left(Z < \frac{-\sqrt{n}}{\sqrt{15}}\right).$$

By the symmetry of the probability density function for Z we have

$$.00366 = \text{Prob}\left(Z > \frac{\sqrt{n}}{\sqrt{15}}\right).$$

Looking in a Z-table, we see $\text{Prob}(Z > 2.685) \approx .00366$. To solve for n, we set $\sqrt{n}/\sqrt{15} = 2.685$. This gives us $n = 15(2.685)^2$; rounding up we get $n = 109$.

A generalization of the above argument shows that for a certainty level of c with $0 < c < 1$ (we chose $c = .95$) we have $n = 15z^2$ where z is the value in a Z-table corresponding to $1 - \sqrt[14]{c}$. In fact only $n = 42$ plaintexts and corresponding ciphertexts are needed for the certainty level to go above $c = .5$. If none of the keys works, she can get more plaintexts and corresponding ciphertexts. Alternately, she can try switching some of her 14 choices for the bits. Her first attempt would be to switch the one that had occurred the fewest number of times.

Speed of attack

If Eve has a single plaintext and corresponding ciphertext, then she can try all $2^{16} = 65536$ keys to see which sends the plaintext to the ciphertext. If more than one key works, she can check the candidates on a second plaintext and corresponding ciphertext. Undoubtedly only one key will work both times. This is called a pure brute force attack. On average, she expects success half way through, so the expected success occurs after 2^{15} encryptions.

For our linear cryptanalytic attack to succeed with 95% certainty, Eve needs to compute the values of the 14 different $\sum_{i \in S_1} p_i \oplus \sum_{j \in S_2} c_j$'s for each of the 109 plaintexts and corresponding ciphertexts. Then she only needs to do a brute force attack with four possible keys.

Thus this linear cryptanalytic attack seems very attractive compared to a pure brute force attack for one-round simplified AES. However, when you add rounds, you have to do more additions of equations in order to eliminate unknown, intermediary bits (like the m_i's and n_i's) and the probabilities associated to the equations then tend toward .5 (as we saw our probabilities go from .75 to .625). The result is that many more plaintexts and corresponding ciphertexts are needed in order to be fairly certain of picking the correct bit values for the $\sum_{l \in S_3} k_l$'s.

30.6 Differential cryptanalysis

Differential cryptanalysis (Biham, Shamir) is chosen plaintext attack. Usually unrealistic. You can use if have enormous amount of known PT (with enough, you'll find ones you would

have chosen). (Smart cards/cable box). Eve picks 2 PT's that differ in specified bits and same at specified bits and looks at difference in corresponding two CT's and deduces info about key.

Diff'l cry's of 1-round simplified Rijndael: $A_{K_1} \circ MC \circ SR \circ NS \circ A_{K_0}$. One key used to encrypt 2 PT's $p_0 \dots p_{15}$ & $p_0^* \dots p_{15}^*$ with $p_8 \dots p_{15} = p_8^* \dots p_{15}^*$ to get 2 CT's $c_0 \dots c_{15}$ & $c_0^* \dots c_{15}^*$. Want $p_0 p_1 p_2 p_3 \neq p_0^* p_1^* p_2^* p_3^*$ as nibbles (OK if some bits same). Want $p_4 p_5 p_6 p_7 \neq p_4^* p_5^* p_6^* p_7^*$.

$p_0\ p_1\ p_2\ p_3$	$p_8\ p_9\ p_{10}\ p_{11}$
$p_4\ p_5\ p_6\ p_7$	$p_{12}\ p_{13}\ p_{14}\ p_{15}$

$p_0^*\ p_1^*\ p_2^*\ p_3^*$	$p_8\ p_9\ p_{10}\ p_{11}$
$p_4^*\ p_5^*\ p_6^*\ p_7^*$	$p_{12}\ p_{13}\ p_{14}\ p_{15}$

A_{K_0}

$p_0+k_0\ \ p_1+k_1\ \ p_2+k_2\ \ p_3+k_3$	$p_8+k_8\ \ p_9+k_9\ \ p_{10}+k_{10}\ \ p_{11}+k_{11}$
$p_4+k_4\ \ p_5+k_5\ \ p_6+k_6\ \ p_7+k_7$	$p_{12}+k_{12}\ \ p_{13}+k_{13}\ \ p_{14}+k_{14}\ \ p_{15}+k_{15}$

and

$p_0^*+k_0\ \ p_1^*+k_1\ \ p_2^*+k_2\ \ p_3^*+k_3$	$p_8+k_8\ \ p_9+k_9\ \ p_{10}+k_{10}\ \ p_{11}+k_{11}$
$p_4^*+k_4\ \ p_5^*+k_5\ \ p_6^*+k_6\ \ p_7^*+k_7$	$p_{12}+k_{12}\ \ p_{13}+k_{13}\ \ p_{14}+k_{14}\ \ p_{15}+k_{15}$

NS (S-box each nybble)

$m_0m_1m_2m_3$	$m_8m_9m_{10}m_{11}$
$m_4m_5m_6m_7$	$m_{12}m_{13}m_{14}m_{15}$

$m_0^*m_1^*m_2^*m_3^*$	$m_8m_9m_{10}m_{11}$
$m_4^*m_5^*m_6^*m_7^*$	$m_{12}m_{13}m_{14}m_{15}$

SR (shiftrow)

$m_0m_1m_2m_3$	$m_8m_9m_{10}m_{11}$
$m_{12}m_{13}m_{14}m_{15}$	$m_4m_5m_6m_7$

$m_0^*m_1^*m_2^*m_3^*$	$m_8m_9m_{10}m_{11}$
$m_{12}m_{13}m_{14}m_{15}$	$m_4^*m_5^*m_6^*m_7^*$

MC (mixcolumn)

$m_0+m_{14}\ \ m_1+m_{12}+m_{15}\ \ m_2+m_{12}+m_{13}\ \ m_3+m_{13}$	$m_6+m_8\ \ m_4+m_7+m_9\ \ m_4+m_5+m_{10}\ \ m_5+m_{11}$
$m_2+m_{12}\ \ m_0+m_3+m_{13}\ \ m_0+m_1+m_{14}\ \ m_1+m_{15}$	$m_4+m_{10}\ \ m_5+m_8+m_{11}\ \ m_6+m_8+m_9\ \ m_7+m_9$

and

$m_0^*+m_{14}\ \ m_1^*+m_{12}+m_{15}\ \ m_2^*+m_{12}+m_{13}\ \ m_3^*+m_{13}$	$m_6^*+m_8\ \ m_4^*+m_7^*+m_9\ \ m_4^*+m_5^*+m_{10}\ \ m_5^*+m_{11}$
$m_2^*+m_{12}\ \ m_0^*+m_3^*+m_{13}\ \ m_0^*+m_1^*+m_{14}\ \ m_1^*+m_{15}$	$m_4^*+m_{10}\ \ m_5^*+m_8+m_{11}\ \ m_6^*+m_8+m_9\ \ m_7^*+m_9$

AK_1

$m_0+m_{14}+k_{16}\ \ m_1+m_{12}+m_{15}+k_{17}\ \ m_2+m_{12}+m_{13}+k_{18}\ \ m_3+m_{13}+k_{19}$
$m_2+m_{12}+k_{20}\ \ m_0+m_3+m_{13}+k_{21}\ \ m_0+m_1+m_{14}+k_{22}\ \ m_1+m_{15}+k_{23}$

$m_6+m_8+k_{24}\ \ m_4+m_7+m_9+k_{25}\ \ m_4+m_5+m_{10}+k_{26}\ \ m_5+m_{11}+k_{27}$
$m_4+m_{10}+k_{28}\ \ m_5+m_8+m_{11}+k_{29}\ \ m_6+m_8+m_9+k_{30}\ \ m_7+m_9+k_{31}$

and

$m_0^*+m_{14}+k_{16}\ \ m_1^*+m_{12}+m_{15}+k_{17}\ \ m_2^*+m_{12}+m_{13}+k_{18}\ \ m_3^*+m_{13}+k_{19}$
$m_2^*+m_{12}+k_{20}\ \ m_0^*+m_3^*+m_{13}+k_{21}\ \ m_0^*+m_1^*+m_{14}+k_{22}\ \ m_1^*+m_{15}+k_{23}$

$m_6^*+m_8+k_{24}\ \ m_4^*+m_7^*+m_9+k_{25}\ \ m_4^*+m_5^*+m_{10}+k_{26}\ \ m_5^*+m_{11}+k_{27}$
$m_4^*+m_{10}+k_{28}\ \ m_5^*+m_8+m_{11}+k_{29}\ \ m_6^*+m_8+m_9+k_{30}\ \ m_7^*+m_9+k_{31}$

This equals

$c_0c_1c_2c_3$	$c_8c_9c_{10}c_{11}$
$c_4c_5c_6c_7$	$c_{12}c_{13}c_{14}c_{15}$

$c_0^*c_1^*c_2^*c_3^*$	$c_8^*c_9^*c_{10}^*c_{11}^*$
$c_4^*c_5^*c_6^*c_7^*$	$c_{12}^*c_{13}^*c_{14}^*c_{15}^*$

XOR the next to the lasts (with m_i's and k_i's) together and get

$m_0+m_0^*\ \ m_1+m_1^*\ \ m_2+m_2^*\ \ m_3+m_3^*$	don't care
don't care	$m_4+m_4^*\ \ m_5+m_5^*\ \ m_6+m_6^*\ \ m_7+m_7^*$

which equals the XOR of the last one (with the c_i's).

$c_0+c_0^*\ \ c_1+c_1^*\ \ c_2+c_2^*\ \ c_3+c_3^*$	don't care
don't care	$c_{12}+c_{12}^*\ \ c_{13}+c_{13}^*\ \ c_{14}+c_{14}^*\ \ c_{15}+c_{15}^*$

We have $c_0c_1c_2c_3 + c_0^*c_1^*c_2^*c_3^*$
$= m_0m_1m_2m_3 + m_0^*m_1^*m_2^*m_3^*$
$= \text{Sbox}(p_0p_1p_2p_3 + k_0k_1k_2k_3)+\text{Sbox}(p_0^*p_1^*p_2^*p_3^* + k_0k_1k_2k_3)$.

So **if** $p_8 \dots p_{15} = p_8^* \dots p_{15}^*$ then $\text{Sbox}(p_0p_1p_2p_3 + k_0k_1k_2k_3)+\text{Sbox}(p_0^*p_1^*p_2^*p_3^* + k_0k_1k_2k_3) = c_0c_1c_2c_3 + c_0^*c_1^*c_2^*c_3^*$. (I). ((Box, including if statement.))

All known except $k_0k_1k_2k_3$. Note equation DEPENDS ON $p_8 \dots p_{15} = p_8^* \dots p_{15}^*$.

Similarly $c_{12}c_{13}c_{14}c_{15}+ c_{12}^*c_{13}^*c_{14}^*c_{15}^*$
$= m_4m_5m_6m_7 + m_4^*m_5^*m_6^*m_7^*$

$\text{Sbox}(p_4p_5p_6p_7 + k_4k_5k_6k_7)+ \text{Sbox}(p_4^*p_5^*p_6^*p_7^* + k_4k_5k_6k_7)$.

So if $p_8 \dots p_{15} = p_8^* \dots p_{15}^*$ then $\text{Sbox}(p_4p_5p_6p_7 + k_4k_5k_6k_7)+ \text{Sbox}(p_4^*p_5^*p_6^*p_7^* + k_4k_5k_6k_7) = c_{12}c_{13}c_{14}c_{15}+ c_{12}^*c_{13}^*c_{14}^*c_{15}^*$. (II).

All known except $k_4k_5k_6k_7$.

Ex: Eve encrypts

	p_0			p_{15}	
ASCII No =	0100	1110	0110	1111	and
CT =	0010	0010	0100	1101	

	p_0^*			p_{15}^*
ASCII to =	0111	0100	0110	1111
CT =	0000	1010	0001	0001

Eq'n I: $\text{Sbox}(0100 + k_0k_1k_2k_3) +\text{Sbox}(0111 + k_0k_1k_2k_3) = 0010 + 0000 = 0010$.

cand	A	B			
$k_0k_1k_2k_3$	+0100	+0111	Sbox(A)	Sbox(B)	Sbox(A)+Sbox(B)
0000	0100	0111	1101	0101	1000
0001	0101	0110	0001	1000	1001
0010	0110	0101	1000	0001	1001
0011	0111	0100	0101	1101	1000
0100	0000	0011	1001	1011	0010
0101	0001	0010	0100	1010	1110
0110	0010	0001	1010	0100	1110
0111	0011	0000	1011	1001	0010
1000	1100	1111	1100	0111	1011
1001	1101	1110	1110	1111	0001
1010	1110	1101	1111	1110	0001
1011	1111	1100	0111	1100	1011
1100	1000	1011	0110	0011	0101
1101	1001	1010	0010	0000	0010
1110	1010	1001	0000	0010	0010
1111	1011	1000	0011	0110	0101

So $k_0k_1k_2k_3 \in \{0100, 0111, 1101, 1110\}$.

Note that set of cand's is determined by 2nd and 3rd columns. Those are pairs with XOR=0011. So if $p_0p_1p_2p_3 + p_0^*p_1^*p_2^*p_3^* = 0011$ will get same cand's. So find new pair with $p_0p_1p_2p_3 + p_0^*p_1^*p_2^*p_3^* \neq 0011, (0000)$.

In PARI we
? p1=[0,1,0,0]
? p2=[0,1,1,1]
ctxor=[0,0,1,0]
? \r xor.txt
((Output:))
[0,1,0,0]
⋮
[1,1,1,0]

Eve encrypts

	p_0			p_{15}	
ASCII Mr =	0100	1101	0111	0010	and
CT =	1101	0101	1101	0000	

	p_0^*			p_{15}^*
ASCII or =	0110	1111	0111	0010
CT =	1100	0001	0100	1111

I: Sbox$(0100 + k_0k_1k_2k_3)$ $+Sbox(0110 + k_0k_1k_2k_3) = 1101 + 1100 = 0001$. We get the candidates $k_0k_1k_2k_3 \in \{1101, 1111\}$. The intersection of two sets is $k_0k_1k_2k_3 = 1101$.

Using II, and these two matched PT/CT pairs, Eve determines $k_4k_5k_6k_7 = 1100$.

To get $k_8k_9k_{10}k_{11}$ and $k_{12}k_{13}k_{14}k_{15}$, need two more eq'ns III and IV (to be determined in HW).

With enough rounds (> 3 DES, > 1 Rijndael) impossible to find such equations with probability 1. Instead find such eq'ns with lower prob'y p. Then right keybits appear in proportion p of candidate sets. Others show up randomly and since so many (real DES/Rijndael), will appear much less often.

31 Attacks on Public Key Cryptography

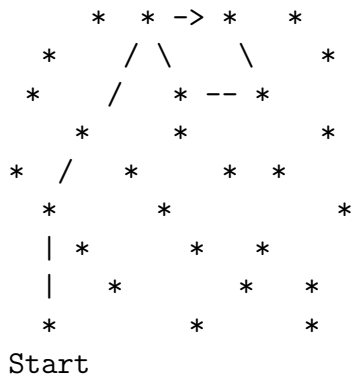
In this section, we present algorithms for factoring, the finite field discrete logarithm problem and the elliptic curve discrete logarithm problem that run faster than brute force.

31.1 Pollard's ρ algorithm

The birthday paradox says that if there are more than 23 people in a room, then odds are that two have the same birthday. In general, if $\alpha\sqrt{n}$ items are drawn with replacement from a set of size n, then the probability that two will match is approximately $1 - e^{-\alpha^2/2}$.

So if you pick $\sqrt{\ln(4)n} \approx 1.2\sqrt{n}$ items, odds are 50/50 that two will match. So you need $\sqrt{365\ln(4)} \approx 23$ birthdays.

If you take a random walk through a set of size n, then you expect after $1.2\sqrt{n}$ steps that you'll come back to a place you've been before. Exploiting this is called Pollard's ρ-method. The number of expected steps before returning to some previous point is $O(\sqrt{n})$. Below is a random walk that shows why it's called the ρ-method (note shape).



A factoring algorithm based on this was the first algorithm significantly faster than trial division. It is still best on numbers in the 8 - 15 digit range. We want to factor n. Iterate a function, like $f(x) = x^2 + 1 (\mathrm{mod} n)$ (starting with, say $x = 0$) and you get a random walk through $\mathbf{Z}/n\mathbf{Z}$. If $n = pq$, you hope that modulo p you come back (to a place you've been before) and mod q that you don't. Example: Let's say we want to factor 1357. We'll use the map $x^2 + 1$ so $a_{m+1} = a_m^2 + 1 (\mathrm{mod}\, 1357)$ where we start with $a_0 = 0$.

Here's an algorithm with little storage and no look-up.
Step 1) Compute a_1, a_1, a_2, $\gcd(a_2 - a_1, n)$, store a_1, a_2
Step 2) Compute a_2, a_3, a_4, $\gcd(a_4 - a_2, n)$, store a_2, a_4 and delete a_1, a_2
Step 3) Compute a_3, a_5, a_6, $\gcd(a_6 - a_3, n)$, store a_3, a_6 and delete a_2, a_4
Step 4) Compute a_4, a_7, a_8, $\gcd(a_8 - a_4, n)$, store a_4, a_8 and delete a_3, a_6
Step 5) Compute a_5, a_9, a_{10}, $\gcd(a_{10} - a_5, n)$, store a_5, a_{10}, and delete a_4, a_8, etc.

i	$2i-1$	$2i$	a_i	a_{2i-1}		a_{2i}	$\gcd(a_{2i} - a_i, 1357)$
1	1	2	1	1	$\to$	2	1
			$\downarrow$		$\swarrow$		
2	3	4	2	5	$\to$	26	1
			$\downarrow$		$\swarrow$		
3	5	6	5	677	$\to$	1021	1
			$\downarrow$		$\swarrow$		
4	7	8	26	266	$\to$	193	1
			$\downarrow$		$\swarrow$		
5	9	10	677	611	$\to$	147	1
			$\downarrow$		$\swarrow$		
6	11	12	1021	1255	$\to$	906	23

So $23|1357$ and $1357/23 = 59$. Note that computing $x^2 + 1 (\mathrm{mod}\, n)$ is fast and gcd'ing is fast.

Why did this work? Let's look behind the scenes at what was happening modulo 23 and modulo 59.

$a_i \bmod 1357$	$a_i \bmod 23$	$a_i \bmod 59$
$a_1 = 1$	1	1
$a_2 = 2$	2	2
$a_3 = 5$	5	5
$a_4 = 26$	3	26
$a_5 = 677$	10	28
$a_6 = 1021$	9	18
$a_7 = 266$	13	30
$a_8 = 193$	9	16
$a_9 = 611$	13	21
$a_{10} = 147$	9	29
$a_{11} = 1255$	13	16
$a_{12} = 906$	9	21

Note $906 - 1021 \equiv 9 - 9 \equiv 0 (\mathrm{mod}\, 23)$, $\equiv 21 - 18 \equiv 3 (\mathrm{mod}\, 59)$. So $23|906 - 1021$, $59 \not| 906 - 1021$. So $\gcd(906 - 1021, 1357) = 23$.

Wouldn't it be faster just to make the list $a_1, \ldots, a_8$ and gcd at each step with all the previous a_i's? No. If the ρ (when we get back, modulo p to where we've been before) happens after m steps (above we see it happened for $m = 8$) then we need $1 + 2 + \ldots + (m-1) \approx \frac{m^2}{2}$ gcd's and a lot of storage. The earlier algorithm has only about $4m$ steps.

How long until we ρ? Assume $n = pq$ and $p < \sqrt{n}$. If we come back to somewhere we have been before mod p after m steps then $m = O(\sqrt{p}) = O(\sqrt[4]{n}) = O(e^{\frac{1}{4}\log n})$. Trial division takes $O(\sqrt{n}) = O(e^{\frac{1}{2}\log n})$ steps. In each case, we multiply by $O(\log^i(n))$ for $i = 2$ or 3 to get the running time. This multiplier, however, is insignificant. The number field sieve takes time $e^{O(\log n)^{1/3}(\log\log n)^{2/3})}$.

We can use the same idea to solve the discrete log problem for elliptic curves over finite fields. This is still the best known algorithm for solving the ECDLP. Let $E : y^2 = x^3 + 17x + 1$ over $\mathbf{F}_{101}$. The point $G = (0, 1)$ generates $E(\mathbf{F}_{101})$. In addition, $103G = \emptyset$ so the multiples of the points work modulo 103. The point $Q = (5, 98) = nG$ for some n; find n. Let $x(\text{point})$ denote the x-coordinate of a point, so $x(Q) = 5$.

Let's take a random walk through $E(\mathbf{F}_{101})$. Let $v_0 = [0, 0]$ and $P_0 = \emptyset$. The vector $v_i = [a_i, b_i]$ means $P_i = a_i Q + b_i G$ where a_i, b_i are defined modulo 103. Here is the walk:
If $x(P_i) \leq 33$ or $P_i = \emptyset$ then $P_{i+1} = Q + P_i$ and $v_{i+1} = v_i + [1, 0]$.
If $33 < x(P_i) < 68$ then $P_{i+1} = 2P_i$ and $v_{i+1} = 2v_i$.
If $68 \leq x(P_i)$ then $P_{i+1} = G + P_i$ and $v_{i+1} = v_i + [0, 1]$.

When $P_{2j} = P_j$, quit. Then $P_{2j} = a_{2j}Q + b_{2j}G = a_jQ + b_jG = P_j$. So $(a_{2j} - a_j)Q = (b_j - b_{2j})G$ and $Q = (b_j - b_{2j})(a_{2j} - a_j)^{-1}G$ where $(a_{2j} - a_j)^{-1}$ is reduced modulo 103. Note the step $P_{i+1} = 2P_i$ depends on the fact that $\gcd(2, \#E(\mathbf{F}_p)) = 1$. If $2|\#E(\mathbf{F}_p)$, then replace

2 with the smallest prime relatively prime to $\#E(\mathbf{F}_p)$.

		$P_i = a_iQ + b_iG$
i	P_i	$v_i = [a_i, b_i]$
	0	$[0,0]$
1	$(5,98)$	$[1,0]$
2	$(68,60)$	$[2,0]$
3	$(63,29)$	$[2,1]$
4	$(12,32)$	$[4,2]$
5	$(8,89)$	$[5,2]$
6	$(97,77)$	$[6,2]$
7	$(62,66)$	$[6,3]$
8	$(53,81)$	$[12,6]$
9	$(97,77)$	$[24,12]$
10	$(62,66)$	$[24,13]$
11	$(53,81)$	$[48,26]$
12	$(97,77)$	$[96,52]$

Note that $P_{12} = P_6$ so $6Q + 2G = 96Q + 52G$. Thus $-90Q = 50G$ and $Q = (-90)^{-1}50G$. We have $(-90)^{-1}50 \equiv 91(\text{mod}103)$ so $Q = 91G$. Of course we really compute P_1, P_1, P_2 and compare P_1, P_2. Then we compute P_2, P_3, P_4 and compare P_2, P_4. Then we compute P_3, P_5, P_6 and compare $P_3, P_6, etc.$.

We can use a similar idea to solve the FFDLP. Instead of doubling (i.e. squaring here) in random walk, we raise to a power relatively prime to $\#\mathbf{F}_q^*$ for $q = 2^r$ or q a prime. Example. $g = 2$ generates $\mathbf{F}_{101}^*$. We have $y = 86 = g^x$. Find x.

We'll take a random walk through $\mathbf{F}_{101}^*$. Let $c_0 = 1$ and $v_0 = [0,0]$. The vector $v_i = [a_i, b_i]$ means $c_i = y^{a_i}g^{b_i} = 86^{a_i}2^{b_i}$. Note a_i, b_i are defined modulo 100.
If $c_i \leq 33$ then $c_{i+1} = c_i y = 86c_i$ and $v_{i+1} = v_i + [1,0]$.
If $33 < c_i < 68$ then $c_{i+1} = c_i^3$ and $v_{i+1} = 3v_i$ (we used 3 since it is relatively prime to 100, the group order).
If $68 \leq c_i$ then $c_{i+1} = c_i g = 2c_i$ and $v_{i+1} = v_i + [0,1]$.

When $c_{2j} = c_j$ you quit. Then $c_{2j} = y^{a_{2j}}g^{b_{2j}} = y^{a_j}g^{b_j} = c_j$. So $y^{a_{2j}-a_j} = g^{b_j-b_{2j}}$ and $y = g^{(b_j-b_{2j})(a_{2j}-a_j)^{-1}}$, where the inversion in the exponent happens modulo 100.

			$c_i = 86^{a_i}2^{b_i}$
	i	c_i	$v_i = [a_i, b_i]$
	0	1	[0,0]
	1	86	[1,0]
	2	71	[1,1]
	3	41	[1,2]
We have	4	39	[3,6]
	5	32	[9,18]
	6	25	[10,18]
	7	29	[11,18]
	8	70	[12,18]
	9	39	[12,19]
	10	32	[36,57]

So $32 = 86^{36}2^{57} = 86^{9}2^{18}$ and $86^{27} = 2^{18-57}$. Now $18 - 57 \equiv 61 (\mathrm{mod}\, 100)$ so $86^{27} = 2^{61}$ and $86 = 2^{61(27^{-1})}$. Now $61(27^{-1}) \equiv 61(63) \equiv 43 (\mathrm{mod}\, 100)$ so $86 = 2^{43}$ in $\mathbf{F}_{101}$. Note that if the entries in the vector $[a_i, b_i]$ become 100 or larger, you reduce them modulo 100. Had $(a_{2j} - a_j)$ not been invertible modulo 100, then we would find all solutions to $(a_{2j} - a_j)x = b_j - b_{2j} (\mathrm{mod}\, 100)$.

Often, when the security of a cryptosystem depends on the difficulty of solving the discrete logarithm problem in $\mathbf{F}_q^*$, you choose q so that there is a prime ℓ with $\ell | q - 1 = \#\mathbf{F}_q^*$ of a certain size. In this case, you find an element $h \in \mathbf{F}_q^*$ such that $h^\ell = 1$, with $h \neq 1$. All computations are done in the subset (subgroup) of $\mathbf{F}_q^*$ generated by h. If ℓ is sufficiently smaller than q, then encryption, decryption, signing or verifying a signature will be faster. Note that Pollard's ρ-algorithm can be done in the subset of elements in $\mathbf{F}_q^*$ generated by h. So this subset has size ℓ. So the running time depends on ℓ. The number field sieve adaptation of the index calculus algorithm, runs in sub-exponential time, so is much faster than Pollard's ρ algorithm for a whole finite field. The index calculus algorithm, however, can not be adapted to the subset generated by h. So its running time depends on q. You can choose ℓ so that the Pollard's ρ algorithm in the subset generated by h takes the same amount of time as the index calculus algorithm in $\mathbf{F}_q^*$. This increases speed without compromising security. Note that the number field sieve and the index calculus algorithm are described in Sections 31.3.3 and 31.2.6, though the number field sieve adaptation of the index calculus algorithm is not.

31.2 Factoring

The most obvious way of cracking RSA is to factor a user's $n = pq$ into the primes p and q. When we talk about the problem of factoring, we assume that we are looking for a single non-trivial factor of a number n, so we can assume n is odd. So $105 = 7 \cdot 15$ is a successful factorization, despite the fact that 15 is not prime.

The best known algorithm for factoring an RSA number is the number field sieve. However, factoring appears in other cryptographic contexts as well. Given the Pohlig-Hellman algorithm described in Section 31.3.2, we can se that it is important to be able to factor the size of a group of the form $\mathbf{F}_q^*$ or $E(\mathbf{F}_q)$. That way we can ensure that the discrete logarithm problem is difficult in those groups. Typically, factoring algorithms other than the number field sieve are used for such factorizations. For that reason, we will look at other factoring algorithms (like those using continued fractions and elliptic curves) that are used in these other cryptographic contexts.

Trial division is **very** slow, but still the fastest way of factoring integers of fewer than 15 digits.

Most of the better factoring algorithms are based on the following. For simplification we will assume that $n = pq$ where p and q are different odd primes. Say we find x and y such that $x^2 \equiv y^2 (\mathrm{mod} n)$. Assume $x \not\equiv \pm y (\mathrm{mod} n)$. Then $n | x^2 - y^2$ so $n | (x + y)(x - y)$. So $pq | (x + y)(x - y)$. We hope that $p | (x + y)$ and $q | (x - y)$. If so, $\gcd(x + y, n) = p$ (and gcd'ing is fast) and $q = n/p$. If not, then $n = pq | x + y$ or $n = pq | x - y$ so $x \equiv \pm y (\mathrm{mod}\, n)$ and you try again.

Note that n need not be the product of exactly two primes. In general, all the arguments above go through for more complicated n. In general $\gcd(x - y, n)$ is some divisor of n.

Here are some algorithms for finding x and y.

31.2.1 Fermat Factorization

This algorithm exploits the fact that small integers are more likely than large integers to be squares. It does not help to let $x = 1, 2, 3, \ldots$ since the reduction of $x^2 (\mathrm{mod}\, n)$ is equal to x^2. So we need to find x such that $x^2 > n$ (or $x > \sqrt{n}$) and we want the reduction of $x^2 (\mathrm{mod}\, n)$ to be small modulo n so we pick x just larger than $\sqrt{n}$.

The notation $\lceil x \rceil$ denotes the smallest number that is greater than or equal to x. So $\lceil 1.5 \rceil = 2$ and $\lceil 3 \rceil = 3$. First compute $\lceil \sqrt{n} \rceil$. Then compute $\sqrt{\lceil \sqrt{n} \rceil^2 - n}$. If it's not an integer then compute $\sqrt{(\lceil \sqrt{n} \rceil + 1)^2 - n}$. If it's not an integer then compute $\sqrt{(\lceil \sqrt{n} \rceil + 2)^2 - n}$, etc. until you get an integer. Example. $n = 3229799$, $\sqrt{n} \approx 1797.16$ so $\lceil \sqrt{n} \rceil = 1798$. $1798^2 - n = 3005$, but $\sqrt{3005} \notin \mathbf{Z}$. $1799^2 - n = 6602$, but $\sqrt{6602} \notin \mathbf{Z}$. $1800^2 - n = 10201$ and $\sqrt{10201} = 101$. Thus $1800^2 - n = 101^2$ and $1800^2 - 101^2 = n$ so $(1800 + 101)(1800 - 101) = n = 1901 \cdot 1699$.

Fermat factorization tends to work well when n is not too much larger than a square.

31.2.2 Factor Bases

In most modern factoring algorithms (continued fraction, quadratic sieve and number field sieve), you want to find x's so that the reduction of $x^2 (\mathrm{mod}\, n)$ is smooth. That is because it is easier to multiply smooth numbers together in the hopes of getting a square. One way to make that likely is for the reductions to be small. So you could choose x^2 near a multiple of n. So $x^2 \approx kn$ for $k \in \mathbf{Z}$ and $x \approx \sqrt{kn}$. We then create a factor base consisting of the primes $\leq b$, our smoothness bound. The best choice of b depends on some complicated number theory. The bound b will increase with n.

We want to exploit any $x^2 \approx kn$ whether $x^2 > kn$ or $x^2 < kn$. Now if $x^2 < kn$ then we should have $x^2 \equiv -1 \cdot l (\mathrm{mod}\, n)$ where l is small. So we include -1 in our factor base as well. For each x with $x^2 \approx kn$, compute the reduction of $x^2 (\mathrm{mod}\, n)$. If the reduction r is just under n, then you will instead use $-1 \cdot (n - r)$. Now factor the reduction over the factor base using trial division. If you can not, then try another x. Continue until some product of some subset of the reductions is a square.

Here is a simple algorithm exploiting this idea. Choose the closest integers to $\sqrt{kn}$ for $k = 1, 2, \ldots$. The number of close integers depends on n in an ugly way. Let x be an integer near $\sqrt{kn}$.

Example. $n = 89893$, use $b = 20$ and the four closest integers to each $\sqrt{kn}$. We have $\sqrt{n} \approx 299.8$.

		-1	2	3	5	7	11	13	17	19
$299^2 \equiv -492$	not factor									
$300^2 \equiv 107$	not factor									
$298^2 \equiv -1089$	$-1 \cdot 3^2 \cdot 11^2$	1	0	0	0	0	0	0	0	0
$301^2 \equiv 708$	not factor									
$\sqrt{2n} \approx 424.01$										
$424^2 \equiv -10$	$-1 \cdot 2 \cdot 5$	1	1	0	1	0	0	0	0	0
$425^2 \equiv 839$	not factor									
$423^2 \equiv -857$	not factor									
$426^2 \equiv 1690$	$= 2 \cdot 5 \cdot 13^2$	0	1	0	1	0	0	0	0	0

In order to get the product of the reductions to be a square, we need to find some subset of the vectors on the right that sum to 0 modulo 2. We see that $298^2 \cdot 424^2 \cdot 426^2 \equiv -1 \cdot 3^2 \cdot 11^2 \cdot -1 \cdot 2 \cdot 5 \cdot 2 \cdot 5 \cdot 13^2 (\mathrm{mod}\, n)$. So $(298 \cdot 424 \cdot 426)^2 \equiv (-1 \cdot 2 \cdot 3 \cdot 5 \dot{1}1 \cdot 13)^2 (\mathrm{mod}\, n)$. Recall that if $x^2 \equiv y^2 (\mathrm{mod}\, n)$ and $x \not\equiv \pm y (\mathrm{mod}\, n)$ then $\gcd(x+y, n)$ is a non-trivial factor of n¿ Now you reduce what is inside the parentheses. We have $298 \cdot 424 \cdot 426 \equiv 69938 (\mathrm{mod}\, n)$ and $-1 \cdot 2 \cdot 3 \cdot 5 \cdot 11 \cdot 13 \equiv 85603 (\mathrm{mod}\, n)$. Note that $\gcd(69938 + 85603, n) = 373$ and $n/373 = 241$.

As a note, this example is unrepresentative of the usual case in that we used each vector that arose. You might have gotten vectors $v_1 = [1, 0, 0, 0, 0, 0, 0 \ldots$, $v_2 = [1, 1, 0, 1, 0, 0, 0 \ldots$, $v_3 = [0, 0, 1, 0, 1, 0, 0 \ldots$, $v_4 = [1, 0, 0, 0, 0, 1, 0 \ldots$, $v_5 = [0, 1, 0, 1, 0, 0, 0, \ldots$ (where $\ldots$ means 0's). Then we would need to find a subset whose sum is 0 modulo 2. Note that if we let M be the matrix whose columns are $v_1, \ldots, v_5$, we can compute the null space of M modulo 2. Any non-0 vector in the null space gives a trivial linear combination of the v_i's. The null space of M mod 2 has $(1, 1, 0, 0, 1)$ as a basis, so $v_1 + v_2 + v_5 = 0$ modulo 2.

There are several algorithms which improve ways of finding b_i's so that the $b_i^2 (\mathrm{mod}\, n)$'s are smooth. This includes the continued fraction factoring algorithm, the quadratic sieve and the number field sieve.

31.2.3 Continued Fraction Factoring

This was the best factoring algorithm around 1975. See Section 29.1 for an explanation of continued fractions. It is still the fastest algorithm for factoring some medium sized integers. We want x^2 near a multiple of n. Let's say that b/c is a convergent to $\sqrt{n}$'s simple continued fraction. Then $b/c \approx \sqrt{n}$ so $b^2/c^2 \approx n$ so $b^2 \approx c^2 n$ so b^2 is near a multiple of n. So $b^2 (\mathrm{mod} n)$ is small.

Let $n = 17873$. The simple continued fraction expansion of $\sqrt{n} = 133.689939 \ldots$ starts $[133, 1, 2, 4, 2, 3, 1, 2, 1, 2, 3, 3, \ldots$. We will use the factor base $\{-1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29\}$ and omit the 0's in our chart.

				-1	2	3	5	7	11	13	17	19	23	29
$[133] = 133$	$133^2 \equiv$	$-184 =$	$-1 \cdot 2^3 \cdot 23$	1	1								1	
$[133, 1] = 134$	$134^2 \equiv$	$83 =$	n.f.											
$[133, 1, 2] = \frac{401}{3}$	$401^2 \equiv$	$-56 =$	$-1 \cdot 2^3 \cdot 7$	1	1			1						
$[133, 1, 2, 4] = \frac{1738}{13}$	$1738^2 \equiv$	$107 =$	n.f.											
$[133, \ldots, 2] = \frac{3877}{29}$	$3877^2 \equiv$	$-64 =$	$-1 \cdot 2^6$	1										
$[133, \ldots, 3] = \frac{13369}{100}$	$13369^2 \equiv$	$161 =$	$7 \cdot 23$					1					1	

Note $\frac{401}{3} \approx 133.67$, $\frac{1738}{13} \approx 133.692$, $\frac{3877}{29} \approx 133.6896$, etc.

$(133 \cdot 401 \cdot 13369)^2 \equiv (-1 \cdot 2^3 \cdot 7 \cdot 23)^2 (\mathrm{mod}\, n)$. Now $133 \cdot 401 \cdot 13369 \equiv 1288$ and $-1 \cdot 2^3 \cdot 7 \cdot 23 \equiv 16585$ but $1288 \equiv -16585$. That's bad. It means $\gcd(16585 + 1288, n) = n$ and $\gcd(16585 - 1288, n) = 1$. So we get no factors. We continue.

				-1	2	3	5	7	11	13	17	19	23	29
$[133, \ldots, 1] = \frac{17246}{129}$	$17246^2 \equiv$	$-77 =$	$-1 \cdot 7 \cdot 11$	1				1	1					
$[133, \ldots, 2] = \frac{47861}{358}$	$47861^2 \equiv$	$149 =$	n.f.											
$[133, \ldots, 1] = \frac{65107}{487}$	$65107^2 \equiv$	$-88 =$	$-1 \cdot 2^3 \cdot 11$	1	1				1					

$(401 \cdot 3877 \cdot 17246 \cdot 65107)^2 \equiv (-1^2 \cdot 2^6 \cdot 7 \cdot 11)^2 (\mathrm{mod}\, n)$. Now $401 \cdot 3877 \cdot 17246 \cdot 65107 \equiv 7272$ and $-1^2 \cdot 2^6 \cdot 7 \cdot 11 \equiv 4928$. We have $7272 - 4928 = 2344$ and $\gcd(2344, n) = 293$ and $n/293 = 61$. Both 293 and 61 are prime.

31.2.4 H.W. Lenstra Jr.'s Elliptic Curve Method of Factoring

This algorithm is often best if n's smallest prime factor is between 13 and 65 digits and the next smallest prime factor is a lot bigger. Let's start with a motivating example.

Example. Let's use the elliptic curve $y^2 = x^3 + x + 1$ to factor 221. Clearly the point $R = (0, 1)$ is on this elliptic curve. Modulo 221 you could compute $2R = (166, 137)$, $3R = (72, 169)$, $4R = (109, 97)$, and $5R = (169, 38)$. To compute $6R = R + 5R$, you first find the slope $\frac{38-1}{169-0} = \frac{37}{169} (\mathrm{mod}\, 221)$. So we need to find $169^{-1} (\mathrm{mod}\, 221)$. We do the Euclidean algorithm and discover that $\gcd(221, 169) = 13$. So $13|221$ and $221/13 = 17$.

What happened behind the scenes?

	mod 13	mod 17
R	(0,1)	(0,1)
$2R$	(10,7)	(13,1)
$3R$	(7,0)	(4,16)
$4R$	(10,6)	(9,12)
$5R$	(0,12)	(16,4)
$6R$	$\emptyset$	(10,12)

Note that $18R = \emptyset (\mathrm{mod}\, 17)$. We succeeded since modulo 13, $6R = \emptyset$ but modulo 17, $6R \neq \emptyset$.

Note modulo any prime, some multiple of R is 0. End example.

For simplicity, we will consider the elliptic curve method of factoring $n = pq$, though the method works for arbitrary positive integers. Choose some elliptic curve E and point R on it modulo n. Find a highly composite number like $t!$ (the size of t depends on n) and hope that $t!R = \emptyset$ modulo one prime (say p) but not the other. Then gcd(denominator of the slope used in the computation of $t!R, n) = p$ and you get a factor.

Why $t!$? There's some m with $mR = 0 \pmod{p}$. If $m|t!$ (which is somewhat likely) then $t! = lm$ and so $t!R = lmR = l(mR) = l\emptyset = \emptyset$.

There are two ways this can fail. 1) $t!R$ is not $\emptyset$ mod p or q (like $2!R$ in the last example). 2) $t!R$ is $\emptyset$ mod p and q so gcd(denominator,n) $= n$. If you fail, choose a new E and R. With most other factoring algorithms you do not have such choices. For example, you could use the family $E : y^2 = x^3 + jx + 1$ and $R = (0, 1)$ for various j.

Example. Let $n = 670726081$, $E : y^2 = x^3 + 1$ and $R = (0, 1)$ Then $(100!)R = \emptyset$. So 2) happened. Now use $E : y^2 = x^3 + x + 1$ and the same R. Then $(100!)R = (260043248, 593016337)$. So 1) happened. Now use $E : y^2 = x^3 + 2x + 1$ and the same R. In trying to compute $(100!)R$, my computer gave an error message since it could not invert 54323 $(\mathrm{mod}\, n)$. But this failure brings success since $\gcd(54323, n) = 54323$ and $n/54323 = 12347$.

31.2.5 Number Fields

We will study number fields so as to have some understanding of the number field sieve.

Let $\mathbf{Z}$ denote the integers, $\mathbf{Q}$ denote the rationals (fractions of integers), and $\mathbf{R}$ denote the real numbers. Let $i = \sqrt{-1}$, $i^2 = -1$, $i^3 = -i$, $i^4 = 1$. The set $\mathbf{C} = \{a + bi |\, a, b \in \mathbf{R}\}$ is the set of complex numbers. $\pi + ei$ is complex. Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ with $a_i \in \mathbf{Z}$. Then we can write $f(x) = a_n(x - \alpha_1)(x - \alpha_2) \cdot \ldots \cdot (x - \alpha_n)$ with $\alpha_i \in \mathbf{C}$. The set of $\{\alpha_i\}$ is unique. The α_i's are called the roots of f. α is a root of $f(x)$ if and only if $f(\alpha) = 0$.

$\mathbf{Q}(\alpha)$ is called a number field. It is all numbers gotten by combining the rationals and α using $+, -, \times, \div$.

Example. $f(x) = x^2 - 2 = (x + \sqrt{2})(x - \sqrt{2})$. Take $\alpha = \sqrt{2}$. $\mathbf{Q}(\sqrt{2}) = \{a + b\sqrt{2} |\, a, b \in \mathbf{Q}\}$. Addition and subtraction are obvious in this set. $(a + b\sqrt{2})(c + d\sqrt{2}) = (ac + 2bd) + (ad + bc)\sqrt{2} \in \mathbf{Q}(\sqrt{2})$. To divide $(a + b\sqrt{2})/(c + d\sqrt{2})$:

$$\frac{a + b\sqrt{2}}{c + d\sqrt{2}} = \frac{(a + b\sqrt{2})(c - d\sqrt{2})}{(c + d\sqrt{2})(c - d\sqrt{2})} = \frac{ac - 2bd}{c^2 - 2d^2} + \frac{bc - ad}{c^2 - 2d^2}\sqrt{2} \in \mathbf{Q}(\sqrt{2}).$$

Example. $g(x) = x^3 - 2$, $\alpha = 2^{1/3}$. $\mathbf{Q}(\alpha) = \{a + b \cdot 2^{1/3} + c \cdot 2^{2/3} |\, a, b, c \in \mathbf{Q}\}$. You can add, subtract, multiply and divide in this set also (except by 0). The division is slightly uglier.

Every element of a number field is a root of a polynomial with integer coefficients ($a_i \in \mathbf{Z}$), which as a set are relatively prime, and positive leading coefficient ($a_n > 0$). The one with the lowest degree is called the minimal polynomial of α.

Example. Find the minimal polynomial of $\alpha = 2^{1/3} + 1$. We can be clever here. Note $(\alpha - 1)^3 = 2$, $\alpha^3 - 3\alpha^2 + 3\alpha - 1 = 2$, $\alpha^3 - 3\alpha^2 + 3\alpha - 3 = 0$. The minimal polynomial is $f(x) = x^3 - 3x^2 + 3x - 3$. Clearly $f(\alpha) = 0$ so α is a root of f.

If the leading coefficient of the minimal polynomial is 1 then α is called an algebraic integer. This agrees with the usual definition for rational numbers. The minimal polynomial of 5 is $1x - 5$ and the minimal polynomial of $3/4$ is $4x - 3$.

In a number field K if $\alpha = \beta\gamma$ and all three are algebraic integers, then we say $\beta|\alpha$. In a number field K, we call an algebraic integer α prime if $\alpha|\beta\gamma$ implies that $\alpha|\beta$ or $\alpha|\gamma$, where β, γ are algebraic integers in K. Not all number fields have "enough" primes sadly. This fact makes the number field sieve difficult to implement.

For example, $\mathbf{Q}(\sqrt{-5})$ is one of the problem number fields. The integers are of the form $\{a + b\sqrt{-5} \mid a, b \in \mathbf{Z}\}$. We have $6 = (1 + \sqrt{-5})(1 - \sqrt{-5}) = 2 \cdot 3$. Now 2 is irreducible in this number field; i.e. we can only factor it: $2 = 2 \cdot 1 = -2 \cdot -1$. However 2 is not prime. Notice that $2|(1 + \sqrt{-5})(1 - \sqrt{-5})$ but $2 \not| 1 + \sqrt{-5}$ and $2 \not| 1 - \sqrt{-5}$. What I mean by $2 \not| 1 + \sqrt{-5}$

is that $(1+\sqrt{-5})/2$ has minimal polynomial $2x^2-2x+3$ so it is not an algebraic integer. Notice that we also do not have unique factorization here.

In $\mathbf{Z}$, we say that we have unique factorization (as in the fundamental theorem of arithmetic (see page 5 from the cryptography class). On the other hand $14 = 7\cdot 2 = -7\cdot -2$. We can say that 7 and -7 are associated primes because their quotient is a unit (an invertible algebraic integer).

From now on, for simplicity, we will always work in the number field $\mathbf{Q}(i) = \{a+bi|\, a,b \in \mathbf{Q}\}$. The algebraic integers in $\mathbf{Q}(i)$ are $\{a+bi|\, a,b\in\mathbf{Z}\}$. This set is usually denoted $\mathbf{Z}[i]$. This is a well-behaved number field. The units are $\{\pm 1, \pm i\}$. If $p \in \mathbf{Z}_{>0}$ is prime, and $p \equiv 3(\text{mod}4)$ then p is still prime in $\mathbf{Z}[i]$. If $p \equiv 1(\text{mod}4)$ then we can write $p = a^2+b^2$ with $a,b\in\mathbf{Z}$ and then $p = (a+bi)(a-bi)$ and $a+bi$ and $a-bi$ are non-associated primes. So there are two primes "over p". Note $17 = 4^2+1^2 = (4+i)(4-i)$ also $17 = (1+4i)(1-4i)$. That's OK since $(1-4i)i = 4+i$ and i is a unit so $1-4i$ and $4+i$ are associated. We can denote that $1-4i \sim 4+i$. The number i is a unit since it's minimal polynomial is x^2+1, so it's an algebraic integer, and $i\cdot i^3 = 1$ so $i|1$. In fact, $1-4i \sim 4+i \sim -1+4i \sim -4-i$ and $1+4i \sim 4-i \sim -1-4i \sim -4+i$ (since $\pm i, \pm 1$ are units). However none of the first four are associated to any of the latter 4. Among associates, we will always pick the representative of the form $a \pm bi$ with $a \geq b \geq 0$.

$2 = (1+i)(1-i)$ but $(1-i)i = 1+i$ so $1-i \sim 1+i$ so there is one prime $(1+i)$ over 2.

Let us list some primes in $\mathbf{Z}$ and their factorizations in $\mathbf{Z}[i]$: $2 = (1+i)^2 i^3$, $3 = 3$, $5 = (2+i)(2-i)$, $7 = 7$, $11 = 11$, $13 = (3+2i)(3-2i)$, $17 = (4+i)(4-i)$, $19 = 19$, $23 = 23$, $29 = (5+2i)(5-2i)$, $31 = 31$, $37 = (6+i)(6-i)$.

There is a norm map $\mathbf{Q}(i) \xrightarrow{N} \mathbf{Q}$ by $N(a+bi) = a^2+b^2$ so $N(2+i) = 5$, $N(7) = 49$. If $a+bi \in \mathbf{Z}[i]$, p is a prime in $\mathbf{Z}$ and $p|N(a+bi)$ (so $p|a^2+b^2$) then a prime lying over p divides $a+bi$. This helps factor algebraic integers in $\mathbf{Z}[i]$.

Factor $5+i$. Well $N(5+i) = 26$ so all factors are in the set $\{i, 1+i, 3+2i, 3-2i\}$. Now $3+2i|5+i$ if $(5+i)/(3+2i)$ is an integer.

$$\frac{5+i}{3+2i}\left(\frac{3-2i}{3-2i}\right) = \frac{17}{13} + \frac{-7}{13}i$$

so $3+2i \not| 5+i$.

$$\frac{5+i}{3-2i}\left(\frac{3+2i}{3+2i}\right) = \frac{13}{13} + \frac{13}{13}i = 1+i$$

so $(5+i) = (3-2i)(1+i)$.

Factor $7+i$. Well $N(7+i) = 50 = 2\cdot 5^2$. $(7+i)/(2+i) = 3-i$ and $N(3-i) = 10$. $(3-i)/(2+i) = (1-i)$ and $N(1-i) = 2$. $(1-i)/(1+i) = -i = i^3$ so $7+i = i^3(1+i)(2+i)^2$.

The following is even more useful for factoring in $\mathbf{Z}(i)$. If $a+bi \in \mathbf{Z}[i]$ and $\gcd(a,b) = 1$ and $N(a+bi) = p_1^{\alpha_1}\cdot \ldots \cdot p_r^{\alpha_r}$ where the p_i's are positive prime numbers then $p_i \not\equiv 3(\text{mod}4)$ and $a+bi = i^{\alpha_0}\pi_1^{\alpha_1}\cdot\ldots\cdot\pi_r^{\alpha_r}$ where π_i is one or the other of the primes over p_i. You never get both primes over p_i showing up.

In the last case, $N(7+i) = 2^1\cdot 5^2$. So we know that $7+i = i^{\alpha_0}(1+i)^1(2\pm i)^2$. So we need only determine α_0 and $\pm$. Here's another example. $N(17-6i) = 325 = 5^2\cdot 13$ so $17-6i = i^{\alpha_0}(2\pm i)^2(3\pm 2i)$, and the $\pm$'s need not agree.

If α and β are elements of $\mathbf{Q}(i)$ then $N(\alpha\beta) = N(\alpha)N(\beta)$.

31.2.6 The Number Field Sieve

The number field sieve (Pollard, Adleman, H. Lenstra) is currently the best known factoring algorithm for factoring a number n if $n > 10^{130}$ and the smallest prime dividing n is at least 10^{65}. RSA numbers are of this type. The number RSA-193 was factored in 2005 using the number field sieve. That number has 193 digits.

Choose a degree d (it depends on n, $d \approx \sqrt{\ln(n)/\ln\ln(n)}$). Let $m = \lfloor \sqrt[d]{n} \rfloor$ and expand n in base m. So $n = m^d + a_{d-1}m^{d-1} + \ldots + a_0$ with $0 \leq a_i < m$. Let $f(x) = x^d + a_{d-1}x^{d-1} + \ldots + a_0$. Let α be a root of f. We work in $\mathbf{Q}(\alpha)$.

Let's factor 2501. We have $\sqrt{\frac{\ln(n)}{\ln(\ln(n))}} \approx 1.95$ so we let $d = 2$. $\lfloor \sqrt{2501} \rfloor = 50$ and $2501 = 50^2 + 1$ so $f(x) = x^2 + 1$ a root of which is i. Note 50 acts like i in $\mathbf{Z}/2501\mathbf{Z}$ since $50^2 \equiv -1 (\mathrm{mod} 2501)$. Define maps $h : \mathbf{Z}[i] \to \mathbf{Z}$ by $h(a+bi) = a+b50$ and $\hat{h} : \mathbf{Z}[i] \to \mathbf{Z}/2501\mathbf{Z}$ by $\hat{h}(a+bi) = a+b50 (\mathrm{mod}\, 2501)$. The map $\hat{h}$ has the properties that $\hat{h}(\alpha+\beta) = \hat{h}(\alpha)+\hat{h}(\beta)$ and $\hat{h}(\alpha\beta) = \hat{h}(\alpha)\hat{h}(\beta)$ for all $\alpha, \beta \in \mathbf{Z}[i]$. The map h has the former, but not the latter property.

Find numbers of the form $\alpha = a + bi$ with $a, b \in \mathbf{Z}$, $a \geq 0$, $b \neq 0$ and $\gcd(a, b) = 1$ where $a + bi$ is smooth in $\mathbf{Z}[i]$ and $h(a + bi)$ is smooth in $\mathbf{Z}$. These are needles in a haystack, which is why factoring is still difficult. We need to find $\alpha_1, \ldots, \alpha_r$ where $\alpha_1\alpha_2 \cdots \alpha_r = \beta^2$ in $\mathbf{Z}[i]$ and $h(\alpha_1)h(\alpha_2) \cdots h(\alpha_r) = t^2$ in $\mathbf{Z}$.

Let us explain how this helps. We have $h(\beta)^2 = h(\beta)h(\beta) \equiv \hat{h}(\beta)\hat{h}(\beta) \equiv \hat{h}(\beta^2) \equiv \hat{h}(\alpha_1\alpha_2 \cdots \alpha_r) \equiv \hat{h}(\alpha_1) \cdots \hat{h}(\alpha_r) \equiv h(\alpha_1) \cdots h(\alpha_r) = t^2$. Now reduce $h(\beta)$ and t mod n (both are in $\mathbf{Z}$). Now $h(\beta))^2 \equiv t^2 (\mathrm{mod}\, n)$. If we have $h(\beta) \not\equiv \pm t (\mathrm{mod}\, n)$, then $\gcd(h(\beta) + t, n)$ is a non-trivial factor of n.

Now let's factor 2501 with the number field sieve. We'll use the factor base $i, 1 + i, 2 \pm i, 3 \pm 2i, 4 \pm i, 5 \pm 2i$ for the algebraic integers. These lie over $1, 2, 5, 13, 17, 29$. The primes 3, 7, 13 are not in the factorbase since they do not exploit the h map and so give no information. We'll use the factor base
$-1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29$ for the integers. We have $h(a + bi) = a + b \cdot 50$.

α	factor α	$h(\alpha)$	factor $h(\alpha)$	
i	$= i$	50	$= 2 \cdot 5^2$	
$1+i$	$= 1+i$	51	$= 3 \cdot 17$	
$2+i$	$= 2+i$	52	$= 2^2 \cdot 13$	$*$
$4+i$	$= 4+i$	54	$= 2 \cdot 3^3$	$*$
$7+i$	$= i^3(1+i)(2+i)^2$	57	$= 3 \cdot 19$	$*$
$1-i$	$= i^3(1+i)$	-49	$= -1 \cdot 7^2$	$*$
$2-i$	$= 2-i$	-48	$= -1 \cdot 2^4 \cdot 3$	
$4-i$	$= 4-i$	-46	$= -1 \cdot 2 \cdot 23$	$*$
$5-i$	$= i^3(3+2i)(1+i)$	-45	$= -1 \cdot 3^2 \cdot 5$	
$8-i$	$= (3-2i)(2+i)$	-42	$= -1 \cdot 2 \cdot 3 \cdot 7$	
$5+2i$	$= 5+2i$	105	$= 3 \cdot 5 \cdot 7$	
$1-2i$	$= i^3(2+i)$	-99	$= -1 \cdot 3^2 \cdot 11$	$*$
$9-2i$	$= (4+i)(2-i)$	-91	$= -1 \cdot 7 \cdot 13$	
$5-2i$	$= 5-2i$	-95	$= -1 \cdot 5 \cdot 19$	
$5-3i$	$= i^3(1+i)(4+i)$	-145	$= -1 \cdot 5 \cdot 29$	
$3+4i$	$= (2+i)^2$	203	$= 7 \cdot 29$	
$2+5i$	$= i(5-2i)$	252	$= 2^2 \cdot 3^2 \cdot 7$	
$3+5i$	$= (4+i)(1+i)$	253	$= 11 \cdot 23$	$*$
$3-5i$	$= i^3(1+i)(4-i)$	-247	$= -1 \cdot 13 \cdot 19$	$*$

These α's would be stored as vectors with entries mod 2. So the last one would be $3-5i \sim (1,1,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,1,0,0)$ corresponding to $(i, 1+i, 2+i, 2-i, 3+2i, 3-2i, 4+i, 4-i, 5+2i, 5-2i, -1, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29)$. Then do linear algebra to find relations. We find found that if we add all the vectors with $*$'s you get the 0 vector. So multiplying the corresponding α's we get a square in the algebraic integers and multiplying the corresponding $h(\alpha)$'s we get a square in the integers. The product of the starred algebraic integers is $i^{12}(1+i)^4(2+i)^4(4+i)^2(4-i)^2$ and the product of the corresponding integers is $(-1)^4 \cdot 2^4 \cdot 3^6 \cdot 7^2 \cdot 11^2 \cdot 13^2 \cdot 19^2 \cdot 23^2$.

Let

$$\beta = i^6(1+i)^2(2+i)^2(4+i)(4-i) = 136 - 102i.$$

$$\text{So } \; h(\beta) = 136 - 102 \cdot 50 = -4964 \equiv 38 (\text{mod}\, 2501).$$

$$\text{Let } \; t = (-1)^2 \cdot 2^2 \cdot 3^3 \cdot 7 \cdot 11 \cdot 13 \cdot 19 \cdot 23 = 47243096 \equiv 1807 (\text{mod}\, 2501).$$

Thus $38^2 \equiv 1444 \equiv 1807^2 (\text{mod}\, 2501)$. $\gcd(1807 - 38, 2501) = 61$ and $2501/61 = 41$ so $2501 = 61 \cdot 41$.

The word *sieve* refers to a way of choosing only certain α's that have a higher probability of being smooth and having $h(\alpha)$ be smooth. For example, a number n with $1 \leq n \leq 1000$ that is divisible by 36 has a higher chance of being 11-smooth than an arbitrary number in that range. You can find conditions on a and b, modulo 36, to guarantee that $h(a+bi)$ is a multiple of 36.

31.3 Solving the Finite Field Discrete Logarithm Problem

We will look at two methods for solving the FFLDP. The first is the Pollig-Hellman method which is useful if the size of $\mathbf{F}_q^*$ is smooth. Note, this algorithm can be adapted to solving the ECDLP as well. The second is the index calculus method, for which there is no known adaptation to the ECDLP.

31.3.1 The Chinese Remainder Theorem

First we need to learn the Chinese Remainder Theorem. Let $m_1, m_2, \ldots, m_r$ be pairwise co-prime (gcd=1) integers. The system of congruences $x \equiv a_1 (\mathrm{mod} m_1)$, $x \equiv a_2 (\mathrm{mod} m_2)$, $\ldots$,$x \equiv a_r (\mathrm{mod} m_r)$ has a unique solution $x (\mathrm{mod} m_1 m_2 \cdot \ldots \cdot m_r)$. Example: If $x \equiv 1 (\mathrm{mod} 7)$ and $x \equiv 2 (\mathrm{mod} 4)$, then $x \equiv 22 (\mathrm{mod} 28)$. In words, you could say that if you know something modulo m and you know it also modulo n then you can know it modulo mn.

Here is a cute example. If we square $625^2 = 390625$. In fact, if the last three digits of any positive integer are 000, 001 or 625 and you square it, the square will have that property. Are there any other 3 digit combinations with that property? We want to solve $x^2 \equiv x (\mathrm{mod}\, 1000)$. Instead we can solve $x^2 \equiv x (\mathrm{mod}\, 8)$ and $x^2 \equiv x (\mathrm{mod}\, 125)$. Clearly $x \equiv 0, 1 (\mathrm{mod}\, 8)$ and $x \equiv 0, 1 (\mathrm{mod}\, 125)$ work. So we get four solutions $x \equiv 0 (\mathrm{mod}\, 8)$ and $x \equiv 0 (\mathrm{mod}\, 125)$ so $x \equiv 0 (\mathrm{mod}\, 1000)$, $x \equiv 1 (\mathrm{mod}\, 8)$ and $x \equiv 1 (\mathrm{mod}\, 125)$ so $x \equiv 1 (\mathrm{mod}\, 1000)$, $x \equiv 1 (\mathrm{mod}\, 8)$ and $x \equiv 0 (\mathrm{mod}\, 125)$ so $x \equiv 625 (\mathrm{mod}\, 1000)$, $x \equiv 0 (\mathrm{mod}\, 8)$ and $x \equiv 1 (\mathrm{mod}\, 125)$ so $x \equiv 376 (\mathrm{mod}\, 1000)$. Indeed $376^2 = 141376$. End cute example.

Here is an algorithm for finding such an x. We want a term that is $a_1 (\mathrm{mod} m_1)$ and 0 mod the rest of the m_i's. So we can use the term $a_1 m_2 m_3 \ldots m_r \cdot b_1$ where $m_2 m_3 \ldots m_r b_1 \equiv 1 (\mathrm{mod} m_1)$ so let $b_1 = (m_2 \ldots m_r)^{-1} (\mathrm{mod} m_1)$. We want a term that is $a_2 (\mathrm{mod} m_2)$ and 0 mod the rest. Use $a_2 m_1 m_3 m_4 \ldots m_r b_2$ where $b_2 = (m_1 m_3 m_4 \ldots m_r)^{-1} (\mathrm{mod} m_2)$, etc. So

$$x = (a_1 m_2 m_3 \ldots m_r b_1) + (a_2 m_1 m_3 \ldots m_r b_2) + \ldots + (a_r m_1 m_2 \ldots m_{r-1} b_r) (\mathrm{mod} m_1 m_2 \ldots m_r).$$

Example: Solve $x \equiv 2 (\mathrm{mod} 3)$, $x \equiv 3 (\mathrm{mod} 5)$, $x \equiv 9 (\mathrm{mod} 11)$.
$b_1 = (5 \cdot 11)^{-1} (\mathrm{mod} 3) = 1^{-1} (\mathrm{mod} 3) = 1$
$b_2 = (3 \cdot 11)^{-1} (\mathrm{mod} 5) = 3^{-1} (\mathrm{mod} 5) = 2$
$b_3 = (3 \cdot 5)^{-1} (\mathrm{mod} 11) = 4^{-1} (\mathrm{mod} 11) = 3$
so $x = 2(5 \cdot 11)1 + 3(3 \cdot 11)2 + 9(3 \cdot 5)3 = 713 \equiv 53 (\mathrm{mod} 165)$.

Aside: Using the Chinese Remainder Theorem to speed up RSA decryption

Let $n = pq$. Assume $M^e \equiv C (\mathrm{mod}\, n)$. Then decryption is the computation of $C^d \equiv M (\mathrm{mod}\, n)$ which takes time $O(\log^3(n))$. This is a slow $O(\log^3(n))$. Instead, let $M (\mathrm{mod}\, p) = M_p$, $M (\mathrm{mod}\, q) = M_q$, $C (\mathrm{mod}\, p) = C_p$, $C (\mathrm{mod}\, q) = C_q$, ((ask them for modulus:)) $d (\mathrm{mod}\, p - 1) = d_p$, $d (\mathrm{mod} q - 1) = d_q$. We have $M_p, M_q, C_p, C_q, d_p, d_q < 2n^{1/2}$. You can pre-compute d_p and d_q. The reductions C_p and C_q take time $O(\log^2(n^{1/2}))$, which will be insignificant. Computing $C_p^{d_p} (\mathrm{mod}\, p) = M_p$ and $C_q^{d_q} (\mathrm{mod}\, q) = M_q$ each take time $O(\log^3(n^{1/2}))$. So each of these computations takes ((ask them)) 1/8 as long as computing $C^d (\mathrm{mod}\, n)$. Using the Chinese remainder theorem to determine M from $M_p (\mathrm{mod}\, p)$ and $M_q (\mathrm{mod}\, q)$ takes time $O(\log^3(n^{1/2}))$, but is slower than computing $C_p^{d_p} \equiv M_p (\mathrm{mod}\, p)$. In practice, the two repeated squares modulo p and q and using the Chinese remainder theorem algorithm take about half as long as computing $C^d \equiv M (\mathrm{mod}\, n)$.

31.3.2 The Pollig-Hellman algorithm

First a rigorous and then an imprecise definition. Let r be a positive integer. An integer n is $\underline{r\text{-smooth}}$ if all of the prime divisors of n are $\leq r$. So one million is 6 smooth. An integer is said to be smooth if all of its prime divisors are relatively small, like one million or $2520 = 2^3 \cdot 3^2 \cdot 5 \cdot 7$.

The Pohlig-Hellman algorithm is useful for solving the discrete logarithm problem in a group whose size is smooth. It can be use in $\mathbf{F}_q^*$ where q is a prime number or the power of a prime in general. It only works quickly if $q-1$ is smooth. So for discrete logarithm cryptosystems, q should be chosen so that $q-1$ has a large prime factor. First I will show the algorithm, then give an example of the algorithm, then explain why it works.

Let g be a generator of $\mathbf{F}_q^*$. We are given $y \in \mathbf{F}_q^*$ and we want to solve $g^x = y$ for x. Let $q-1 = p_1^{\alpha_1} \cdot \ldots \cdot p_r^{\alpha_r}$ where the p_i's are primes. For each prime p_i, we want to find a solution of $z^{p_i} = 1$ (with $z \neq 1$) in F_q^*. Such a solution is called a $\underline{\text{primitive } p_i\text{th root of unity}}$ and is denoted ζ_{p_i}. Recall for any $w \in \mathbf{F}_q^*$, we have $w^{q-1} = 1$. Since $q-1$ is the smallest power of g giving 1, we see that $\zeta_{p_i} = g^{(q-1)/p_i}$ is a primitive p_ith root of unity.

For each prime p_i we precompute all solutions to the equation $z^{p_i} = 1$. These are $\zeta_{p_i}^0 = 1$, $\zeta_{p_i}^1$, ..., $\zeta_{p_i}^{p_i-1}$. These values are stored with the corresponding exponents of ζ_{p_i}.

Recall that in $\mathbf{F}_q^*$, exponents work mod $q-1$. So once we find $x(\mathrm{mod} p_i^{\alpha_i})$ we can use the Chinese Remainder Theorem to find x. So now we want to find $x(\mathrm{mod} p^{\alpha})$ (where we drop the subscript). Let's say we write x (reduced mod p^{α}) in base p. Mind you we don't yet know how to do this, but the base p expansion does exist: $x \cong x_0 + x_1 p + x_2 p^2 + \ldots + x_{\alpha-1} p^{\alpha-1} (\mathrm{mod}\, p^{\alpha})$, with $0 \leq x_i < p$. Now we start determining the x_i's. Note if we are dealing with Alice's private key, then Alice knows x and the x_i's and Eve does not know any of them (yet).
Find $y^{(q-1)/p}(\mathrm{mod} q)$. It's in the list of ζ_p^i. We have $y^{(q-1)/p} \equiv \zeta_p^{x_0}(\mathrm{mod} q)$. Now we know x_0.
Let $y_1 \equiv y/(g^{x_0})(\mathrm{mod} q)$. Find $y_1^{(q-1)/p^2} \equiv \zeta_p^{x_1}(\mathrm{mod} q)$. Now we know x_1.
Let $y_2 \equiv y/(g^{x_0+x_1 p})$. Find $y_2^{(q-1)/p^3} \equiv \zeta_p^{x_2}$. Now we know x_2.
Let $y_3 \equiv y/(g^{x_0+x_1 p+x_2 p^2})$. Find $y_3^{(q-1)/p^4} \equiv \zeta_p^{x_3}$. Now we know x_3. Etc.

Let's do an example. Let $q = 401$, $g = 3$ and $y = 304$. We want to solve $3^x = 304(\mathrm{mod} 401)$. We have $q-1 = 2^4 \cdot 5^2$. First find $x(\mathrm{mod} 16)$. First we pre-compute $g^{(400/2)} = \zeta_2 = \zeta_2^1 = 400$ and $\zeta_2^2 = \zeta_2^0 = 1$ (this is all mod 401). We have $x = x_0 + x_1 2 + x_2 4 + x_3 8 (\mathrm{mod} 16)$ and want to find the x_i's.

$$y^{(q-1)/p} = 304^{400/2} \equiv 400(\mathrm{mod} 401) = \zeta_2^1 \quad \text{so } x_0 = 1.$$

$\frac{y}{g^{x_0}} = 304/3^1 \equiv 235(\mathrm{mod} 401) = y_1.$ $\quad y_1^{(q-1)/p^2} = 235^{400/(2^2)} \equiv 400 = \zeta_2^1 \quad \text{so } x_1 = 1.$

$\frac{y}{g^{x_0+x_1 p}} = 304/(3^{1+1\cdot 2}) \equiv 338 = y_2.$ $\quad y_2^{(q-1)/p^3} = 338^{400/(2^3)} \equiv 1 = \zeta_2^0 \quad \text{so } x_2 = 0.$

$\frac{y}{g^{x_0+x_1 p+x_2 p^2}} = 304/(3^{1+1\cdot 2+0\cdot 4}) \equiv 338 = y_3.$ $\quad y_3^{(q-1)/p^4} = 336^{400/(2^4)} \equiv 400 = \zeta_2^1 \quad \text{so } x_3 = 1.$

Thus $x = 1 + 1\cdot 2 + 0\cdot 4 + 1\cdot 8 = 11(\mathrm{mod} 16)$. This was four steps instead of brute forcing 2^4.

Now we find $x(\mathrm{mod} 25)$. First we pre-compute $g^{400/5} = \zeta_5 = 72$, $\zeta_5^2 = 372$, $\zeta_5^3 = 318$, $\zeta_5^4 = 39$, and $\zeta_5^5 = \zeta_5^0 = 1$. We have $x = x_0 + x_1 5(\mathrm{mod} 25)$.

$$y^{(q-1)/p} = 304^{400/5} \equiv 372 = \zeta_5^2 \quad \text{so } x_0 = 2.$$

$y/(g^{x_0}) = 304/(3^2) \equiv 212 = y_1.$ $\quad y_1^{(q-1)/p^2} = 212^{400/5^2} \equiv 318 = \zeta_5^3 \quad \text{so } x_1 = 3.$

Thus $x \equiv 2 + 3 \cdot 5 = 17(\text{mod}25)$. This was two steps instead of brute forcing 5^2. If $x \equiv 11(\text{mod}16)$ and $17(\text{mod}25)$ then, from the Chinese Remainder Theorem algorithm $x = 267$. So $3^{267} = 304(\text{mod}401)$. We have a total of $2 + 4 + 5 + 2 = 13$ steps (the 2 and 5 are from pre-computing) instead of brute forcing $400 = 2^4 \cdot 5^2$.

Why does this work? Let's look at a simpler example. Let $q = 17$, $g = 3$. We have $q - 1 = 16 = 2^4$. $3^{1 \cdot 8} = 16 = \zeta_2^1$ and $3^{0 \cdot 8} = 1 = \zeta_2^0$.

Note that $w^{16} \equiv 1(\text{mod}\,17)$ for any $1 \leq w \leq 16$. We have $3^{11} \equiv 7(\text{mod}\,17)$ and assume that 11 is unknown. So $3^{1+1\cdot 2+0\cdot 4+1\cdot 8} \equiv 7(\text{mod}\,17)$.

know		same, but not know	
7,	$7^8 \equiv 16 = (3^8)^1$	$3^{1+1\cdot 2+0\cdot 4+1\cdot 8}$,	$3^{1\cdot 8+16n} = 3^{1\cdot 8}(3^{16})^n = (3^8)^1$
$\frac{7}{3^1}$,	$(\frac{7}{3^1})^4 = 16 = (3^8)^1$	$3^{1\cdot 2+0\cdot 4+1\cdot 8}$,	$3^{1\cdot 8+16n} = (3^8)^1$
$\frac{7}{3^{1+1\cdot 2}}$,	$(\frac{7}{3^{1+1\cdot 2}})^2 = 1 = (3^8)^0$	$3^{0\cdot 4+1\cdot 8}$,	$3^{0\cdot 8+16n} = (3^8)^0$
$\frac{7}{3^{1+1\cdot 2+0\cdot 4}}$,	$(\frac{7}{3^{1+1\cdot 2+0\cdot 4}})^1 = 16 = (3^8)^1$	$3^{1\cdot 8} = (3^8)^1$	

Let's say that $q - 1 = 2^{200}$. Then Pollard's ρ algorithm would take 2^{100} steps and this would take $2 + 100$ steps.

31.3.3 The Index Calculus Algorithm

The index calculus algorithm is a method of solving the discrete logarithm problem in fields of the type $\mathbf{F}_q$, where q is prime, or a prime power. We will do an example in a field of the form $\mathbf{F}_{2^d}$. For homework you will do an example in a field of the form $\mathbf{F}_p$ for p a prime.

Recall $\mathbf{F}_2[x]/(x^3 + x + 1) = \{a_0 + a_1x + a_2x^2 | a_i \in \mathbf{F}_2\}$. We can call this field $\mathbf{F}_8$. We have $2 = 0$ and $x^3 + x + 1 = 0$ so $x^3 = -x - 1 = x + 1$. $\mathbf{F}_8^*$ is $\mathbf{F}_8$ without the 0. There are $8 - 1 = 7$ elements of $\mathbf{F}_8^*$ and they are generated by x. We see $x^1 = 1$, $x^2 = x^2$, $x^3 = x + 1$, $x^4 = x^2 + x$, $x^5 = x^2 + x + 1$, $x^6 = x^2 + 1$ and $x^7 = 1$. Note $x^{12} = x^7 \cdot x^5 = x^5$ so exponents work modulo 7 $(= \#\mathbf{F}_8^*)$.

Recall $\log_b m = a$ means $b^a = m$ so $\log_x(x^2 + x + 1) = 5$ since $x^5 = x^2 + x + 1$. We will usually drop the subscript x. The logs give exponents so the logs work mod 7. Note $(x^2 + 1)(x + 1) = x^2$. Now $\log(x^2 + 1)(x + 1) = \log(x^2 + 1) + \log(x + 1) = 6 + 3 = 9$ whereas $\log(x^2) = 2$ and that's OK since $9 \equiv 2(\text{mod}7)$.

Let's do this in general. Let $f(x) \in \mathbf{F}_2[x]$ have degree d and be irreducible mod 2. We have $\mathbf{F}_q = \mathbf{F}_2[x]/(f(x))$ where $q = 2^d$. Let's say g generates $\mathbf{F}_q^*$. If $g^n = y$ we say $\log_g y = n$ or $\log y = n$. We have $\log(uv) \equiv \log(u) + \log(v)(\text{mod}q - 1)$ and $\log(u^r) \equiv r\log(u)(\text{mod}q - 1)$.

The discrete log problem in $\mathbf{F}_q^*$ is the following. Say $g^n = y$. Given g and y, find n mod $q - 1$, i.e. find $n = \log_g y$. Note $\log_g g = 1$.

For the index calculus algorithm, choose m with $1 < m < d$ (how these are chosen is based on difficult number theory and statistics, for $d = 127$, choose $m = 17$).

Part 1. Let $h_1, \ldots, h_k$ be the set of irreducible polynomials in $\mathbf{F}_2[x]$ of degree $\leq m$. Find the log of every element of $\{h_i\}$. To do this, take powers of g like g^t and hope $g^t = h_1^{\alpha_1} h_2^{\alpha_2} \cdot \ldots \cdot h_k^{\alpha_k}$ (some of the α_i's may be 0). In other words, we want g^t to be m-smooth. Log both sides. We get $t = \alpha_1 \log(h_1) + \ldots + \alpha_r \log(h_r)$. That's a linear equation in $\log(h_i)$ (the only unknowns). Find more such linear equations until you can solve for the $\log(h_i)$'s. Once done, all of the $a_i = \log(h_i)$ are known.

Part 2. Compute $y(g^t)$ for various t until $yg^t = h_1^{\beta_1} \cdot \ldots \cdot h_r^{\beta_r}$. Then $\log y + t\log g = \beta_1 \log h_1 + \ldots + \beta_r \log h_r$ or $\log y + t = \beta_1 a_1 + \ldots + \beta_r a_r$. The only unknown here is $\log y$. When working in a finite field, people often use *ind* instead of *log*.

Here's an example. ((See the handout))

Let $f(x) = x^{11}+x^4+x^2+x+1$. This is irreducible mod 2. Work in the field $\mathbf{F}_2[x]/(f(x)) = \mathbf{F}_q$ where $q = 2^{11}$. We note $g = x$ is a generator for $\mathbf{F}_q^*$. We'll choose m=4.

We want to solve $g^n = y = x^9 + x^8 + x^6 + x^5 + x^3 + x^2 + 1$ for n. I.e. find log(y). The first part has nothing to do with y. Let

$$\begin{array}{llll} 1 = log(x) & a = log(x+1) & c = log(x^2+x+1) & d = log(x^3+x+1) \\ e = log(x^3+x^2+1) & h = log(x^4+x+1) & j = log(x^4+x^3+1) & k = log(x^4+x^3+x^2+x+1). \end{array}$$

We search through various g^t's and find

$$\begin{array}{ll} g^{11} = (x+1)(x^3+x^2+1) & 11 = a + e (\mathrm{mod}\, 2047 = q-1) \\ g^{41} = (x^3+x^2+1)(x^3+x+1)^2 & 41 = e + 2d \\ g^{56} = (x^2+x+1)(x^3+x+1)(x^3+x^2+1) & 56 = c + d + e \\ g^{59} = (x+1)(x^4+x^3+x^2+x+1)^2 & 59 = a + 2k \\ g^{71} = (x^3+x^2+1)(x^2+x+1)^2 & 71 = e + 2c \end{array}$$

Note that although we have four relations in a, c, d, e (the first, second, third and fifth), the fifth relation comes from twice the third minus the second, and so is redundant. Thus we continue searching for relations.

$g^{83} = (x^3+x+1)(x+1)^2$, $83 = d + 2a$.

Now the first, second, third and the newest are four equations (mod 2047) in four unknowns that contain no redundancy. In other words, the four by four coefficient matrix for those equations is invertible modulo 2047. We solve and find $a = 846$, $c = 453$, $d = 438$, $e = 1212$. Now we can solve for k: $k = (59 - a)/2 (\mathrm{mod} q - 1) = 630$. Now let's find h and j. So we need only look for relations involving one of those two.

$g^{106} = (x+1)(x^4+x^3+1)(x^4+x^3+x^2+x+1)$ so $106 = a + j + k$ and $j = 677$.

$g^{126} = (x^4+x+1)(x^4+x^3+x^2+x+1)(x+1)^2$ so $126 = h + k + 2a$ and $h = 1898$.

So $a = 846$, $c = 453$, $d = 438$, $e = 1212$, $h = 1898$, $j = 677$, $k = 630$.

Now move onto the second part. We compute yg^t for various t's. We find $y(g^{19}) = (x^4+x^3+x^2+x+1)^2$. So log(y) + 19log(g) = 2k. Recall log(g)=log(x)=1. So log(y) $= 2k - 19 \equiv 1241$ mod 2047 and so $x^{1241} = y$.

The number field sieve can be combined with the index calculus algorithm. As a result, the running time for solving the FFDLP is essentially the same as for factoring.

Appendix A. ASCII Encoding

ASCII

	00100000	32	8	00111000	56	P	01010000	80	h	01101000	104
!	00100001	33	9	00111001	57	Q	01010001	81	i	01101001	105
″	00100010	34	:	00111010	58	R	01010010	82	j	01101010	106
#	00100011	35	;	00111011	59	S	01010011	83	k	01101011	107
$	00100100	36	<	00111100	60	T	01010100	84	l	01101100	108
%	00100101	37	=	00111101	61	U	01010101	85	m	01101101	109
&	00100110	38	>	00111110	62	V	01010110	86	n	01101110	110
′	00100111	39	?	00111111	63	W	01010111	87	o	01101111	111
(	00101000	40	@	01000000	64	X	01011000	88	p	01110000	112
)	00101001	41	A	01000001	65	Y	01011001	89	q	01110001	113
∗	00101010	42	B	01000010	66	Z	01011010	90	r	01110010	114
+	00101011	43	C	01000011	67	[	01011011	91	s	01110011	115
,	00101100	44	D	01000100	68	\	01011100	92	t	01110100	116
-	00101101	45	E	01000101	69	]	01011101	93	u	01110101	117
.	00101110	46	F	01000110	70	∧	01011110	94	v	01110110	118
/	00101111	47	G	01000111	71	_	01011111	95	w	01110111	119
0	00110000	48	H	01001000	72	‘	01100000	96	x	01111000	120
1	00110001	49	I	01001001	73	a	01100001	97	y	01111001	121
2	00110010	50	J	01001010	74	b	01100010	98	z	01111010	122
3	00110011	51	K	01001011	75	c	01100011	99	{	01111011	123
4	00110100	52	L	01001100	76	d	01100100	100	|	01111100	124
5	00110101	53	M	01001101	77	e	01100101	101	}	01111101	125
6	00110110	54	N	01001110	78	f	01100110	102	∼	01111110	126
7	00110111	55	O	01001111	79	g	01100111	103			

References

[1] Beutelspacher, A., *Cryptology*. Washington: The Mathematical Association of America, 1994.

[2] Blum, L., Blum, M. & Shub, M., *Comparison of two pseudo-random number generators*. in Conf. Proc. Crypto 82, 1982, 61–78.

[3] Brassard, G., *Modern cryptology: A tutorial*. Lecture Notes in Computer Science **325**, New York: Springer-Verlag, 1994.

[4] Chaum, D., Carback, R., Clark, J., Essex, A., Popoveniuc, S., Rivest, R.L., Ryan, P.Y.A., Shen, E., Sherman, A.T., *Scantegrity II: End-to-end verifiability for optical scan election systems using invisible ink confirmation codes*, Proceedings of USENIX/ACCURATE EVT 2008.

[5] Koblitz, N., *A course in number theory and cryptography*. New York: Springer Verlag, 1987.

[6] Konheim, A.G., *Cryptography: A primer*. New York: John Wiley & Sons, Inc., 1981.

[7] Matsui, M., *Linear cryptanalysis method for DES cipher*. In *Advances in Cryptography - Eurocrypt '93*, Springer-Verlag, Berlin, 1993, 386 - 397.

[8] Pomerance, C., *The number field sieve. Mathematics of Computation, 1943 - 1993: a half century of computional mathematics* (Vancouver, BC, 1993), Proc. Sympos. Appl. Math., **48**, Amer. Math. Soc. Providence, RI 1994, 465 - 480.

[9] Schneier, B., *Applied cryptography*. New York: John Wiley & Sons, Inc., 1996.

[10] RSA Laboratories, *Answers to frequently asked questions about today's cryptography, version 3.0*. RSA Data Security, Inc., 1996.