CSCE 477/877: Cryptography and Computer Security

Dr. Chris Bourke
Department of Computer Science & Engineering
University of Nebraska—Lincoln
Lincoln, NE 68588, USA
http://cse.unl.edu/~cbourke
cbourke@cse.unl.edu

2015/02/10 11:14:38

Contents

1. Introduction

1.1. Administrivia

- Introduction
- Roll
- Blackboard resources
- Syllabus and policies
- Text books
- Homework now available (due dates: TBD based on pace)
- Project
- Coursera: highly encouraged
- What this course *is*: an introduction (Classical, modern ciphers, PK, hashes, etc.)
- What this course is *not*: not a review of network protocols, not a pen testing, etc.
- Mixture of theory and practice: neither in depth; both in breadth

1.2. What is Security?

"A secure channel over which a message can be transmitted such that no one can eavesdrop"

- Security involves storage, not just communication
- A secure channel is not enough: what about the two end points?
- Also need a mechanism for identification/authentication/non-repudiation

"A system is secure if no *untrusted* party can gain access to information"

- Edward Snowden

"Security is ensuring that secrets are not revealed"

- A locked box may not reveal any secrets, but it does reveal that secrets exist
- A bank likely has money in it
- Existence of communication indicates existence of entities communicating
- Secure communication reveals a lot (you have a mole, existence of a spy etc.); honeypots

"Security is a 'solved' problem"

- Data breaches and not going down
- Potential loss and cost of breaches going up
- Stakes will only increase
- Most breaches are due to poor implementation, human error

- *Not* a solved problem

Things to remember:

- No such thing as perfect security
- Security is a continuous process of escalation
- Risk can only be mitigated, not eliminated
- Catching the "bad" guy is not winning: if they got in, you already failed

Security is a balance:

- Convenience (if cost of usage is too high, users undermine it)
- Efficiency
- Provable vs. Probable security (theorists vs. pragmatists)

The Big 4 and other Important Principles

- Privacy and confidentiality
- Integrity
- Authentication
- Non-repudiation
- Communication scenarios and attack vectors
- Kerckhoff's Principle: "A cryptographic system should be secure even if everything about the system, except the key, is public knowledge."
- Obscurity is not security
- Don't roll your own: in any *real* applications:
  - Don't implement your own adaptations of cryptographic algorithms
  - Don't invent your own algorithms or protocols
  - Don't mix cryptographic primitives into your own protocol
  - DO: rely on public and academic vetting of algorithms and protocols
  - DO: rely on well-established and tested libraries (some are even FIPS/NIST certified)
  - DO: have fun (roll your own for hobby/interest/this course)

Course cavets:

- I'm not a security expert or researcher
- Lack of "current" resources; huge discipline, lots of RFCs, etc.
- Suggestion: take Coursera's Security Specialization (free):
  `https://www.coursera.org/specialization/cybersecurity/7/courses`

1.3. Alphabet Soup

DSA – Digital Signature Algorithm

ECDHE	– Elliptic Curve Diffie-Hellman Ephemeral
ECDSA	– Elliptic Curve Digital Signature Algorithm
FIPS	– Federal Information Processing Standards
ISO	– International Organization for Standardization
NIST	– National Institute of Standards and Technology
NSA	– National Security Administration

$$\begin{tikzcd}
 & & & & \boxed{\text{Oscar}} \arrow[d] & & & \\
\boxed{\text{Alice}} \arrow[r] & x \arrow[r] & e_K(x) \arrow[r] & y \arrow[r] & \boxed{\text{Unsecure Channel}} \arrow[r] & \boxed{\text{Bob}} \arrow[r] & d_K(y) \arrow[r] & x \\
 & & K \arrow[u] \arrow[rrr] & & & \boxed{\text{Secure Channel}} \arrow[u] & &
\end{tikzcd}$$

Figure 1: Secure Communication Scenario

Part I. Classical Ciphers & Cryptanalysis

2. Classical Ciphers

2.1. Basic Definitions

Communication Scenario

- Two parties wish to communicate: Alice wishes to send a message x to Bob
- The problem is that an adversary, Oscar, is attempting to eavesdrop and intercept the message x
- Thus, Alice and Bob *encrypt* the message using an encryption scheme e and a key K
- The function e, d should be assumed public and known to all parties
- The key must be exchanged securely
- The encrypted message y is sent over an insecure channel
- Oscar may have y, but should not (easily) obtain x without K

Formally, a *crypto system* is a 5-tuple: (P, C, K, e_K, d_K):

- P is a finite set of possible plaintexts (or plaintext characters, or blocks of characters)
- C is a finite set of possible ciphertexts (or ciphertext characters, or blocks of characters)
- K is a finite set of possible keys (also called a *keyspace*)
- e_K is a one-to-one (injective) *encryption function*:

$$e_K : P \to C$$

- d_K is the inverse *decryption* function,

$$e_K^{-1} = d_K$$

Properties:

- Inversion implies that
$$e_K(d_K(x)) = x$$
- Typically, $x \in P$ and $y \in C$ are used
- Alternatively,
$$e : P \times K \to C$$
- If $|P| = |C| = n$, there are at least $n!$ possible functions

Typical requirements:

- Computing e, d should be computationally *efficient* to compute *given* K
- Given y, we should not be able to *feasibly* compute K (or x)
- The security of e, d should not be based on obfuscation: assume that e, d are *publicly known*
- No assumptions about our communication channel for y: also assume that attackers have y

2.2. Basic Algebra

Division: Let $a, d, q, r \in \mathbb{Z}$ and suppose

$$a = dq + r$$

Where $0 \leq r < d$, here:

- a is the dividend
- d is the divisor
- q is the quotient
- r is the remainder

Example: 103 divided by 13 has a remainder 12:

$$103 = 13 \cdot 7 + 12$$

Definition 1. Let $a, b \in \mathbb{Z}$, we say that a *divides* b and write

$$a|b$$

if there exists $k \in \mathbb{Z}$ such that
$$b = ka(+0)$$

i.e. there is no remainder after division. Examples: $5|60$, but 3 does not divide 22.

Definition 2. Let $a, b \in \mathbb{Z}$, we say that a is *congruent* to b modulo m and write

$$a \equiv b (\mathrm{mod}\, m)$$

if $m|b-a$.

That is,

$$b - a = mk$$

or

$$a = mk + b$$

(k is negated) b is the remainder of a divided by m.

2.2.1. Algebraic Structures

An *algebraic structure* is a set with one or more *operations* defined on it. Various algebraic structures may be endowed with various properties (axioms).

Let G be an algebraic structure under some operation (usually say, addition $+$).

1. Closure: for any two elements $a, b \in G$, $a + b \in G$
2. Associativity: $(a + b) + c = a + (b + c)$
3. Existence of an additive identity, 0 such that for any $a \in G$, $a + 0 = a = 0 + a$
4. Existence of an additive inverse: for any $a \in G$, there exists a $b \in G$ such that $a + b = 0$
5. Commutativity: for any $a + b = b + a$

We can add a second operation (usually say, multiplication $\cdot$) on G which can have similar properties

6. Closure: for any two elements $a, b \in G$, $a \cdot b \in G$
7. Associativity: $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
8. Commutativity: for any $a \cdot b = b \cdot a$
9. Distributivity: $a \cdot (b + c) = ab + ac$
10. Existence of a multiplicative identity, 1
11. Existence of a multiplicative inverse: for any $a \in G$, there exists an $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = 1$

- An algebraic structure with properties 1–4 is a *group*
- A group with property 5 is an *abelian group*
- A structure with properties 1–10 is a *ring*
- A structure with all properties is a *field*
- A nice visualization/hierarchy can be found in Figure 2

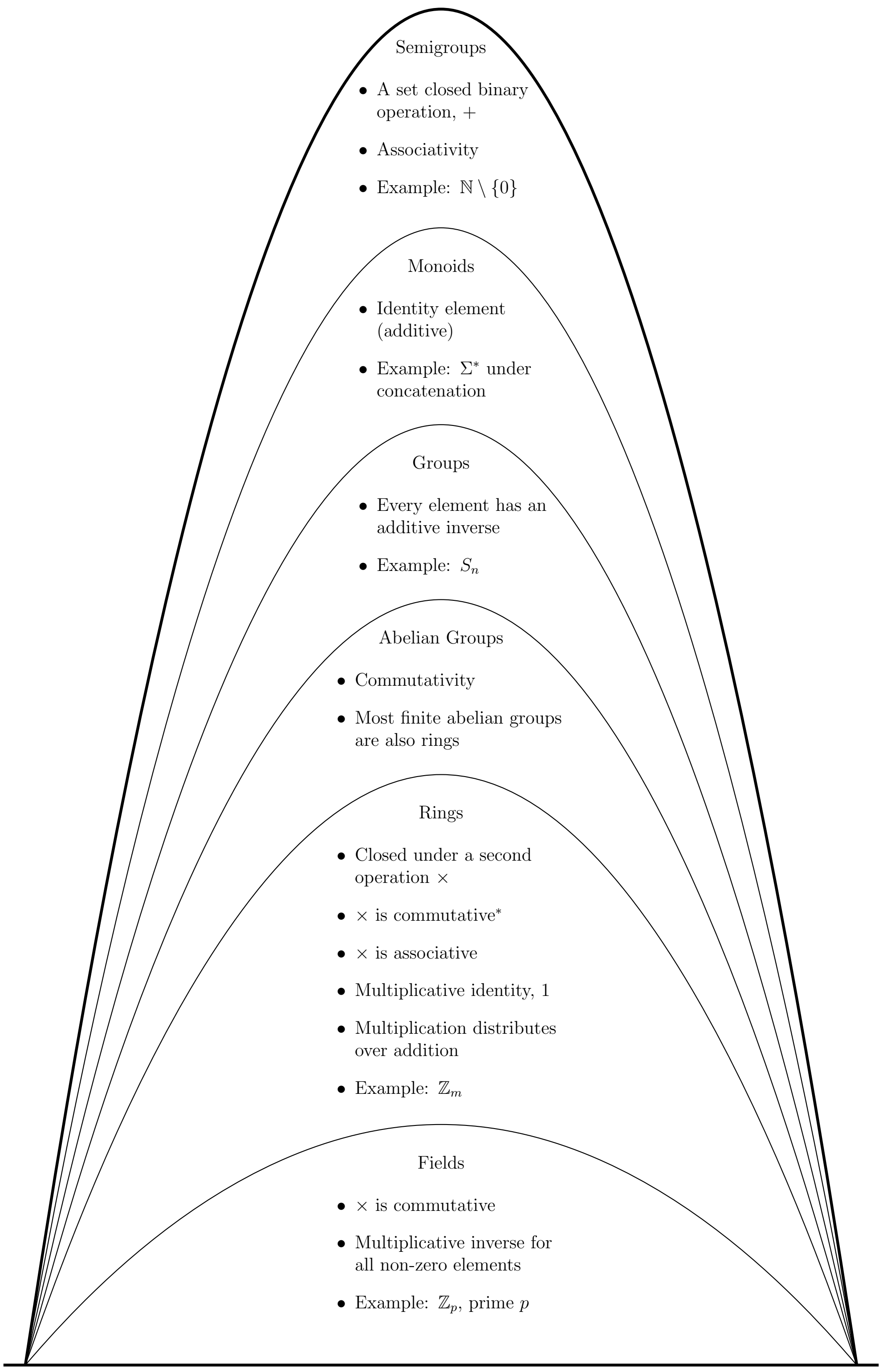

Semigroups
• A set closed binary operation, $+$
• Associativity
• Example: $\mathbb{N} \setminus \{0\}$
Monoids
• Identity element (additive)
• Example: Σ^* under concatenation
Groups
• Every element has an additive inverse
• Example: S_n
Abelian Groups
• Commutativity
• Most finite abelian groups are also rings
Rings
• Closed under a second operation $\times$
• $\times$ is commutative*
• $\times$ is associative
• Multiplicative identity, 1
• Multiplication distributes over addition
• Example: $\mathbb{Z}_m$
Fields
• $\times$ is commutative
• Multiplicative inverse for all non-zero elements
• Example: $\mathbb{Z}_p$, prime p

Figure 2: Various algebraic structures and their relationships and properties

Notes:

- Sometimes a distinction is made between Rings and Commutative Rings, but all Fields are commutative under both operations.
- S_n is the symmetric group on n elements; the set of all permutations under composition (multiplication)
- Many other algebraic structures exist

Motivating examples:

$$Z_m = \{0, 1, 2, \ldots, m-1\}$$

Z_m is closed under addition and multiplication of elements modulo m.

- $\mathbb{Z}_m$ is a ring for any $m \geq 2$
- Example: $\mathbb{Z}_6$: determine some operations/results; what elements have inverses?
- $\mathbb{Z}_p$ is a field for any prime $p \geq 2$
- Example: $\mathbb{Z}_7$: determine some operations/results; what elements have inverses?
- Special example: $\mathbb{Z}_2$: the binary field
  - Addition mod-2 is akin to exclusive-or: $\oplus$
  - multiplication is akin to $\wedge$

Definition 3 (Greatest Common Divisor)**.** Let $a, b \in \mathbb{Z}$ with $a, b \geq 1$. The *greatest common divisor*, written

$$\gcd(a, b)$$

is the largest integer that divides both a, b

Definition 4 (Relatively Prime)**.** Let $a, b \in \mathbb{Z}$ with $a, b \geq 1$. We say that a, b are relatively prime (or cop rime) if their gcd is 1:

$$\gcd(a, b) = 1$$

Observe that if a, b are relatively prime, then neither a divides b nor vice versa.

Theorem 1. Let $a, x, b, m \in \mathbb{Z}, m \geq 2$. Then

$$ax \equiv b \pmod{m}$$

has a unique solution $x \in \mathbb{Z}_m$ if and only if $\gcd(a, m) = 1$

Proof. Omitted

Setting $b = 1$, we have a necessary and sufficient condition for inverses to exist in $\mathbb{Z}_m$:

Corollary 1. Let $x \in \mathbb{Z}_m$, then x has an inverse, $x^{-1} \in \mathbb{Z}_m$ if and only if

$$\gcd(x, m) = 1$$

In a ring, $\mathbb{Z}_m$, how many elements have an inverse?

Theorem 2 (Fundamental Theorem of Arithmetic)**.** For any $m \in \mathbb{Z}$, m can be uniquely written as the product of powers of its prime divisors:

$$m = \prod_{i=1}^{n} p_i^{e_i}$$

Where p_i are m's prime divisors and e_i are their powers.

Theorem 3 (Euler's Totient)**.** Let $m \geq 2$. The number of integers $x \in \mathbb{Z}_m$ relatively prime to m is

$$\phi(m) = \prod_{i=1}^{n} \left(p_i^{e_i} - p_i^{e_i - 1}\right)$$

ϕ is Euler's Totient function.

Examples: $m = 6, 7, 26$

2.2.2. Finding Inverses: Euclid's Algorithm

Euclid's GCD Algorithm can be modified to keep track of additional variables to find integers s, t such that $\gcd(a, b)$ can be expressed as a linear combination of a, b.

Theorem 4. If a and b are positive integers, then there exist integers s, t such that

$$\gcd(a, b) = sa + tb$$

Proof. omitted, proof by strong induction

```
   INPUT  : Two positive integers $a, b$.
   OUTPUT: $r = \gcd(a,b)$ and $s, t$ such that $sa + tb = \gcd(a,b)$.
 1 $a_0 = a$, $b_0 = b$
 2 $t_0 = 0$, $t = 1$
 3 $s_0 = 1$, $s = 0$
 4 $q = \lfloor \frac{a_0}{b_0} \rfloor$
 5 $r = a_0 - qb_0$
 6 WHILE $r > 0$ DO
 7     temp $= t_0 - qt$
 8     $t_0 = t, t =$ temp
 9     temp $= s_0 - qs$
10     $s_0 = s, s =$ temp
11     $a_0 = b_0$, $b_0 = r$
12     $q = \lfloor \frac{a_0}{b_0} \rfloor$, $r = a_0 - qb_0$
13     IF $r > 0$ THEN
14         gcd $= r$
15     END
16 END
17 output gcd, $s, t$
```

Algorithm 1: EXTENDEDEUCLIDEANALGORITHM

a_0	b_0	t_0	t	s_0	s	q	r
27	58	0	1	1	0	0	27
58	27	1	0	0	1	2	4
27	4	0	1	1	-2	6	3
4	3	1	-6	-2	13	1	1
3	1	-6	7	13	-15	3	0

Therefore,

$$\gcd(27, 58) = 1 = (-15)27 + (7)58$$

Example:
Compute $\gcd(25480, 26775)$ and find s, t such that

$$\gcd(25480, 26775) = 25480s + 26775t$$

a_0	b_0	t_0	t	s_0	s	q	r
25480	26775	0	1	1	0	0	25480
26775	25480	1	0	0	1	1	1295
25480	1295	0	1	1	-1	19	875
1295	875	1	-19	-1	20	1	420
875	420	-19	20	20	-21	2	35
420	35	20	-59	-21	62	12	0

Therefore,

$$\gcd(25480, 26775) = 35 = (62)25480 + (-59)26775$$

To find the inverse of $a \in \mathbb{Z}_m$, compute the $\gcd(a, m)$; the coefficient of a, $t = a^{-1}$.

2.3. Substitution Cipher

A substitution cipher defines a permutation that maps plaintext characters to plaintext characters ($P = C$)

$$\pi : P \to P$$

so that the encryption is simply

$$e_K(x) = \pi(x)$$

and the decryption is the inverse permutation

$$d_K(y) = \pi^{-1}(y)$$

- Let $|P| = n$, size of key space: $n!$
- Cyclic permutation notation: (123)(45)
- Tabular notation

2.4. Affine Cipher

Let $P = C = \mathbb{Z}_m$ and let

$$K = \left\{(a, b) \in \mathbb{Z}_m \times \mathbb{Z}_m \middle| \gcd(a, m) = 1\right\}$$

Then the encryption and decryption of the *affine* cipher are:

$$e_K(x) = (ax + b) \bmod m$$

$$d_K(y) = (a^{-1}(y - b)) \bmod m$$

- Known as affine because it is *linear*
- Example?
- Size of key space: $\phi(m) \cdot m$
- Brute Force attack?

Variations

- For $a = 1$, $b \in \mathbb{Z}_m$, we have a simple *shift cipher* or *rotation* cipher
- For $a = 1$, $b = 3$, this is a classical "Caesar Cipher"
- Fix a block size of length n and choose n keys: $K = (k_1, \ldots, k_n)$, then the encryption of a plaintext string $x = x_1, \ldots x_n$ is

  $$e_K(x) = (x_1 + k_1) \bmod m, (x_2 + k_2) \bmod m, \ldots, (x_n + k_n) \bmod m$$

  This is known as a *Vigenere Cipher*

2.5. Permutation Cipher

- Fix a block size n and split the plaintext x
- Let $P = C = (\mathbb{Z}_m)^n$
- Choose a permutation:
  $$\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$$
- Encrypt each block by applying the permutation:
  $$e_K(x_1, \ldots, x_n) = (x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)})$$
- Decryption uses the inverse permutation
  $$d_K(y_1, \ldots, y_n) = \big(y_{\pi^{-1}(1)}, y_{\pi^{-1}(2)}, \ldots, y_{\pi^{-1}(n)}\big)$$

2.6. Hill Cipher

- Block ciphers above still don't have enough "diffusion"
- Each ciphertext character is only dependent on one plaintext character and the key
- A change in one plaintext character only results in the change of one ciphertext character
- We can modify a block cipher so that each ciphertext character in a block depends on all plaintext characters in the block
- This is achieved by use of a matrix as a key. This is call the Hill Cipher (Lester Hill 1929)

Cipher:

- Let $P = C = (\mathbb{Z}_m)^n$
- The block size is n
- And let K be the set of all $n \times n$ nonsingular (invertible) matrices with entries in $\mathbb{Z}_m$

- The encryption/decryption of a block $\mathbf{x} = (x_1, \ldots, x_n)$ is:

$$e_K(\mathbf{x}) = \mathbf{x}K$$

$$d_K(\mathbf{y}) = \mathbf{y}K^{-1}$$

- All matrix operations are performed modulo m

Example: “mom” $\rightarrow \mathbf{x} = (12, 14, 12)$

$$K = \begin{pmatrix} 2 & 3 & 8 \\ 3 & 4 & 7 \\ 11 & 2 & 5 \end{pmatrix}$$

Then $\mathbf{x}K = (16, 12, 20) \rightarrow$ “SOU”

2.6.1. Computing matrix inverses over $\mathbb{Z}_m$

- Given K we want to find K^{-1} such that $KK^{-1} = I$
- However, the operations are over $\mathbb{Z}_m$
- A modification of Gauss-Jordan elimination allows us to find an inverse
- Example (for ease, consider $m = 29$ instead:

$$K = \begin{pmatrix} 10 & 5 & 12 \\ 3 & 14 & 21 \\ 8 & 9 & 11 \end{pmatrix}$$

Over $\mathbb{R}$, the inverse would be:

$$K^{-1} = \begin{pmatrix} 0.050 & -0.076 & 0.091 \\ -0.194 & -0.020 & 0.250 \\ 0.122 & 0.072 & -0.180 \end{pmatrix}$$

but over $\mathbb{Z}_m$:

$$K^{-1} = \begin{pmatrix} 23 & 24 & 24 \\ 19 & 14 & 0 \\ 2 & 8 & 9 \end{pmatrix}$$

x	x^-1
1	1
2	15
3	10
4	22
5	6
7	25
8	11
9	13
12	17
14	27
16	20
18	21
19	26
23	24
28	28

Table 1: Inverses over $\mathbb{Z}_{29}$

Details: setup an augmented matrix:

$$\left[\begin{array}{ccc|ccc} 10 & 5 & 12 & 1 & 0 & 0 \\ 3 & 14 & 21 & 0 & 1 & 0 \\ 8 & 9 & 11 & 0 & 0 & 1 \end{array}\right]$$

- Multiply row 1 by $3 = 10^{-1}$
- subtract $3 \cdot r_1$ from row 2
- subtract $8 \cdot r_1$ from row 3

$$\left[\begin{array}{ccc|ccc} 1 & 15 & 7 & 3 & 0 & 0 \\ 0 & 27 & 0 & 20 & 1 & 0 \\ 0 & 5 & 13 & 5 & 0 & 1 \end{array}\right]$$

- Multiply row 2 by $14 = 27^{-1}$
- subtract $5 \cdot r_2$ from row 3

$$\left[\begin{array}{ccc|ccc} 1 & 15 & 7 & 3 & 0 & 0 \\ 0 & 1 & 0 & 19 & 14 & 0 \\ 0 & 0 & 13 & 26 & 17 & 1 \end{array}\right]$$

- Multiply row 3 by $9 = 13^{-1}$

$$\left[\begin{array}{ccc|ccc} 1 & 15 & 7 & 3 & 0 & 0 \\ 0 & 1 & 0 & 19 & 14 & 0 \\ 0 & 0 & 1 & 2 & 8 & 9 \end{array}\right]$$

- subtract $7 \cdot r_3$ from row 1
- subtract $0 \cdot r_3$ from row 2

$$\left[\begin{array}{ccc|ccc} 1 & 15 & 0 & 18 & 2 & 24 \\ 0 & 1 & 0 & 19 & 14 & 0 \\ 0 & 0 & 1 & 2 & 8 & 9 \end{array}\right]$$

- subtract $15 \cdot r_2$ from row 1

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 23 & 24 & 24 \\ 0 & 1 & 0 & 19 & 14 & 0 \\ 0 & 0 & 1 & 2 & 8 & 9 \end{array}\right]$$

Variation: the Permutation Cipher is a special case of the Hill cipher (permutation matrix: each row/column has exactly one 1)

2.7. Others

- Monoalalphabetic (same cipher alphabet throughout), polyalphabetic (cipher alphabet changes during encryption)
- Playfair (polyalphabetic)

2.8. Cryptanalysis

Types of attacks

- Ciphertext only attack: we only have y
- Known plaintext-ciphertext: we have x, y
- Chosen plaintext/ciphertext attack: Access to e_K, d_K, choose x or y and examine the result

Brute-Force strategy

- Exhaustively search key space and try all keys
- Involves exponential amount of work to generate combinatorial objects
- Should be automated
- Tip: can examine plaintext for common valid English words or formatting

Big Problem:

- In many cryptosystems, different keys can encrypt different plaintexts to the same ciphertext
- Ex: a shift cipher with $k \in \{5, 22\}$ would encrypt "river" and "arena" to the same ciphertext "WNAJW"
- Such keys are called *spurious*: keys that are valid, but incorrect

- The more ciphertext observed the higher chance we have of eliminating spurious keys until the single valid key is found
- With only a small amount of ciphertext, its not possible to find it
- Example: suppose only one bit is sent; how do we know that it decrypts to 0 or 1? With no other information, it is *unbreakable*

2.8.1. Frequency Analysis

- Consider $\mathbb{Z}_{26}$ (corresponding to 26 English language letters)
- For a meaningful plaintext input, the distribution of letters is *not* uniform:
- For a "random" plaintext input, the probability of any given letter would be $\frac{1}{26}$
- Useful in breaking a substitution cipher: given a large enough ciphertext, we can compute the observed distribution (expectation, probability) and make educated guesses
- Most common letters: E, T, A, O, N, I, S, R;
- Letter distribution table: `http://www.macfreek.nl/memory/Letter_Distribution`
- Second level of analysis: digrams: TH, HE, IN, AN, ER, RE, ND, etc.
- Tip: write a program that is able to take a partial permutation and display the output
- Beyond: Trigrams, punctuation, endlines, etc.

2.8.2. Incidence of Coincidence

- For "block" based ciphers, the first step to cryptanalysis is to determine the block size n
- For $n = 2, 3, \ldots$ we can split the ciphertext into n texts:

$$\begin{array}{rcl} \mathbf{y_1} & = & y_1 y_{n+1} y_{2n+1} \cdots, \\ \mathbf{y_2} & = & y_2 y_{n+2} y_{2n+2} \cdots, \\ & \vdots & \\ \mathbf{y_n} & = & y_n y_{n+n} y_{2n+n} \cdots, \end{array}$$

- Now consider: for an English text, what is the probability that 2 randomly chosen characters are the same?
- If the text is random, we would expect it to be

$$26 \cdot \left(\frac{1}{26}\right)^2 \approx 0.038$$

(there are 26 ways to choose the same letter and 26^2 total possible 2-letter combinations)

- Even lower if we consider all possible characters (ex: for $m = 89$, we have 0.010)
- However, if the text is valid English, we would expect something a lot higher (say 0.065 or 0.073 depending on the distribution used).
- Thus, for each block size n, we can compute the observed *Index of Coincidence*:

$$I_c(x) = \frac{\sum_{i=0}^{m} \binom{f_i}{2}}{\binom{n}{2}} = \frac{1}{|x|(|x|-1)} \sum_{i=0}^{m} f_i(f_i - 1)$$

  - Where $|x|$ is the length of the text
  - $|P| = m$ is the size of the alphabet
  - $_i$ is the frequency of the letter i (a *count*, not probability)
- Strategy: for each potential block size $n = 2, 3, \ldots$, compute $I_c(\mathbf{y}_1), \ldots, I_c(\mathbf{y}_n)$
- If n is not our true block size, the Index of Coincidence for each block should be close to random
- If n is our true block size, each index of coincidence should be close to the measured index of English text
- Once the block size has been determined, another frequency analysis attack can be performed on the sunblocks $\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_n}$

Linear & Differential Cryptanalysis

- Linear Cryptanalysis (Matsui early 1990s)
  - You can setup a collection of linear equations involving all possible combinations of plain, cipher, key bits:

$$P_1 \oplus P_4 \oplus C_2 = K_2$$
$$P_3 \oplus P_4 \oplus C_3 = K_1$$

    etc.
  - Ideally, for all possible bits, each equation would be satisfied with probability $\frac{1}{2}$
  - Otherwise, the equation may be *biased*
  - Small enough cipher: examine them all and choose those with the highest bias
  - Otherwise, choice may be tailored to each cipher (exploiting some property)
  - Using high-bias equations, known plaintext-ciphertext pairs can be used to determine the key
  - Essentially an affine (linear) approximation of the cipher function
- Differential Cryptanalysis –more sophisticated; attempts to find and exploit differences in outputs with similar inputs (AES is proven resistant; even DES attack is not highly feasible, requiring 2^{47} chosen plain texts)
- Biham, Shamir (late 1980s) but Known to IBM, NSA when DES was developed (reason for the S-boxes)

- Further reading: http://www.engr.mun.ca/~howard/PAPERS/ldc_tutorial.pdf

2.9. "Perfect" Cryptography & The One-time Pad

Shannon's Definition of *perfect security*:

Definition 5. A crypto system has *perfect security* if $\Pr[x|y] = \Pr[x]$ for all $x \in P, y \in C$.

- Not "perfect" in the practical sense; perfect in the statistical sense
- Given a ciphertext y, it is equally likely that y can be decrypted to any $x \in P$
- Affine cipher is "perfect" with respect to this definition
- Affine is breakable because we use the same key for the entire text which violates this definition
- However, if a different key is used for each character, it *is* prefect (no information bleeds as a result of the distribution of English letters)

Theorem 5. A crypto system provides perfect security if and only if every key is used with equal probability, $\frac{1}{|K|}$ and for every $x \in P$ and every $y \in C$ there is a unique key K such that $e_K(x) = y$

If we use a unique key for each character (or at its simplest, each bit), then we've achieved perfect security. This is known as the *one-time-pad* (Vernam, 1917)

Definition 6 (One-time Pad)**.** Let $P = C = K = \{0,1\}^n = (\mathbb{Z}_2)^n$, define $e_K(x)$ to be the vector sum over $\mathbb{Z}_m$ (i.e. exclusive-or)

$$e_K(\mathbf{x}) = (x_1 \oplus k_1, \ldots, x_n \oplus k_n)$$

Since the exclusive-or is its own inverse,

$$d_K(\mathbf{y}) = e_K(\mathbf{y})$$

Requirements:

- The key must be *random* (see next section on what that means)
- The key must only be used once
- The key must have length equal to the plaintext x
- Clear disadvantages, but it does provide provable security: all decryptions y are equally possible; all (or nearly all) keys are spurious and there is no way to cull them down if its only used once!

Attacks:

1. Cribbing
    - If the key is used more than once then, observe:

$$\begin{aligned} c_1 &= p_1 \oplus k \\ c_2 &= p_2 \oplus k \end{aligned}$$

and thus,

$$X = c_1 \oplus c_2 = p_1 \oplus p_2$$

- That is, we know that the plain texts were either the same or different depending on X
- You can then guess a likely word segment (example: `␣the␣`) and scan/x-or X until it looks like English. This decrypts section by section.
- If enough cipher texts are intercepted, the key can be found for all
- More on cribbing: `https://www.hellboundhackers.org/articles/read-article.php?article_id=691`

2. Lack of "real" randomness
   - If a *weak* source of randomness is used, it is possible to duplicate the sequence of random bits used to form k
   - Example: PRGs

PRG using a linear congruential method: Choose four integers:

- m, the modulus,
- a, the multiplier,
- c the increment and
- x_0 the seed.

Such that the following hold:

- $2 \leq a < m$
- $0 \leq c < m$
- $0 \leq x_o < m$

Our random sequence is:

$$\{x_n\}_{n=1}^{\infty}$$

with $0 \leq x_n \leq m$ by using the congruence

$$x_{n+1} = (ax_n + c) \bmod m$$

- Bad choices have small *period*: $5x_n + 2 \bmod 10$; seed of $x_0 = 6, 2, 2, 2, \ldots$
- A periodic: $m = 17, a = 5, c = 2, x_0 = 3$. Then the sequence is as follows.
  - $x_{n+1} = (ax_n + c) \bmod m$
  - $x_1 = (5 \cdot x_0 + 2) \bmod 17 = 0$
  - $x_2 = (5 \cdot x_1 + 2) \bmod 17 = 2$
  - $x_3 = (5 \cdot x_2 + 2) \bmod 17 = 12$
  - $x_4 = (5 \cdot x_3 + 2) \bmod 17 = 11$
  - $x_5 = (5 \cdot x_4 + 2) \bmod 17 = 6$
  - $x_6 = (5 \cdot x_5 + 2) \bmod 17 = 15$

  - $x_7 = (5 \cdot x_6 + 2) \bmod 17 = 9$
  - $x_8 = (5 \cdot x_7 + 2) \bmod 17 = 13$ etc.

Even with large period, if the seed is known or enough samples are taken, the next value in the sequence can be computed.

This is because the sequence is *deterministic*.

2.9.1. A discussion on Randomness

Which of the following sequences are random?

- $0, 0, 0, 0, 0, 0, 0, 0, \ldots$
- $0, 1, 0, 1, 0, 1, 0, 1, \ldots$
- $0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, \ldots$
- $0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, \ldots$
- The first is just zeros, the second is periodic (though any infinite sequence should allow and in fact we would *expect* them to occur infinitely often as subsequenes)
- The third corresponds to the distribution of primes
- The fourth was generated on http://random.org which uses atmospheric noise to generate random bits

Is randomness simply a matter of distribution of (blocks) of symbols?

- Normal number: all digits are equally likely, all blocks of digits are uniform
- Champernowne constant (base-10)

$$1234567890101112131415161718192 0\ldots$$

- Deterministic

Is something random that "looks" random?

- Digits of π
- Passes some statistical tests (uniformity) but we can predict (with little computational resources) the next digits
- distribution may be uniform, but still be predictable (Champernowne constant, linear congruence, etc.)

No *true* randomness:

- Matter of randomness vs. *unpredictability*
- Even unpredictable processes have bias (coin flips, dice rolls)
- Only "weak" random sources: atmospheric noise, radioactive decay, cosmic background radiation, etc.
- Source is only random (read: unpredictable) because we don't know the underlying process

- PRGs are only random if we don't know the process (seed and formula)
- Computational randomness: Given a resource bound (P, NP, automata, etc), within that bound, can you compute the next bit?
- Using the power of an automaton, we can't (reliably) predict the next bit of primes; using P we can't predict the next bit of a generator based on SAT
- If a natural process could be sufficiently modeled, its (apparent) randomness would go away
- This means we look at randomness versus unpredictability
- Or: looking random enough = indistinguishable from true randomness (using a given amount of resources)
- Lots of research into pseudorandom number generators, extractors, derandomization, etc.
- Our concern (eventually): can we use cryptographically secure PRGs?
  - Given the first k bits in a sequence, no poly-time algorithm should be able to guess the $k+1$-th bit
  - If the state is compromised, the sequence should be one-way (should not be able to recover prior bits in the sequence)
- CSPRNGs are usually derived from block ciphers (AES) run in counter mode

Resources:

- RFC4086: Randomness Requirements for Security `http://tools.ietf.org/html/rfc4086`
- `http://random.org`: online random source (atmospheric noise)
- `/dev/random`: collects randomness from devices (CPU, I/O, network traffic, etc.) as a seed
- Algorithms: Yarrow/SHA1, Fortuna, ARC4, ChaCha20, ANSI X9.17

2.10. Entropy & English

Definition 7 (Entropy)**.** Suppose $\mathbf{X}$ is a discrete random variable with values from X. Then the entropy of $\mathbf{X}$ is defined as

$$H(\mathbf{X}) = -\sum_{x\in X} \Pr[x] \cdot \log_2 (\Pr[x])$$

Defined on $[0,1]$, zero at both ends, maximized 1 at $p = .5$

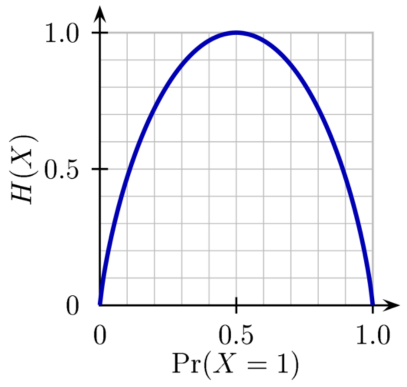


Figure 3: Binary Entropy Graph

Entropy of a language L: defined by *block entropy*:

$$H_L = \lim_{n\to\infty} \frac{H(P^n)}{n}$$

Where P^n is the probability distribution on n-block English (examples: $\Pr[\texttt{qqq}] = 0$, $\Pr[\texttt{the}]$ is higher

Redundancy is defined to be

$$R_L = 1 - \frac{H_L}{\log_2 |P|}$$

- $|P|$ is the size of the plaintext space
- Random language: $H_L = \log |P|, R_L = 0$ (no redundancy)
- Plaintext data is highly structured: it has very low *entropy*
- Random data has very high entropy (no information, no structure, no redundancy)
- Encryption aims to make plaintext indistinguishable from random data: map low entropy data to high entropy data in a reversible way
- English sentences are highly structured: far from uniform distribution, other rules (q is always followed by u), etc., low independence (digrams and trigrams); lots of redundancy
- Example: uniform random 26-letter alphabet should have entropy

$$H(x) = 4.70$$

- English has entropy $1 \le H_L \le 1.5$
- English is 75% redundant information!! ($R_L = 1 - \frac{1.25}{\log 26}$)
- Huffman coding: typical English files can be compressed to 25% of their original size on average!

- Consequence: a ciphertext of sufficient length reduces the likelihood of spurious keys (s_n is the number of spurious keys):

$$s_n \geq \frac{|K|}{|P|^{nR_L}} - 1$$

- The *unicity* distance n_0 is then the expected amount (length, number) of ciphertext examples required for s_n to become zero:

$$n_0 \approx \frac{\log_2 |K|}{R_L \log_2 |P|}$$

- Example: for $R_L = .75$ (English), and $|P| = 26$, $|K| = 26!$, the substitution cipher has

$$n_0 \approx 25$$

  That is, we only need to observe 25 English letters to be able to decrypt with high probability!!
- Contrast with OTP: entropy of keyspace $\log |K|$ grows unbounded: $n_0 \to \infty$
- Unicity distance is only a measure on the *size* of the ciphertext needed to eliminate spurious keys, its *not a guarantee* on the ability to decipher it (just that it can be).

2.11. Summary, Discussion

- Classical ciphers are clearly insecure
- Its not just that key/plain/cipher text spaces are small (they can be *made* large)
- Classical ciphers preserve entropy (information, data)
- We want our encrypted information to be indistinguishable from random noise
- Illustrative examples: Visualizing ciphers using images

Example: Permutation cipher

- Information is only locally scrambled, can still identify global information
- Even when permutation is applied to entire image: information still preserved (distribution of colors)

Example: Substitution cipher

- Most information is preserved; identifiable

Example: Hill cipher

- Simply a combination of substitution/permutation

Example: Padding

- Completely unidentifiable; indistinguishable from the pad itself

Example: Modern encryption scheme (segue)

- Appears nearly random!

Part II. Modern Block Ciphers

- Secure ciphers must posses both "confusion" and "diffusion"
- Diffusion means that the output bits depend on a (large) subset of the input bits
- Confusion means that the output bits depend (in a nontrivial way) on a large subset of the key bits
- This dissipates any non-uniformity (say the structure of English) over the entire ciphertext
- Substitution (exchanging an input for an output) gives confusion
- Permutations (of the input) provide diffusion
- Strict avalanche property: changing i bit in the input changes the j-th bit of the output with probability $\frac{1}{2}$

3. Substitution-Permutation Networks

A substitution-permutation network is a block cipher that:

- Splits an input x into m blocks of ℓ bits
- Then applies to each block a substitution (referred to as an S-box):

$$\pi_S : \{0,1\}^\ell \to \{0,1\}^\ell$$

- Then combine the resulting substituted blocks back into a full block and a permutation is applied:

$$\pi_P : \{1, \dots, \ell m\} \to \{1, \dots, \ell m\}$$

- Each is applied one after the other in N rounds.
- An initial key $K \in \{0,1\}^n$ is used to generate a *key schedule*, $k_1, \dots, k_{N+1}$
- At each round, the key k_i is *mixed* with the block using an exclusive-or operation
- In general, the permutation is not applied to the last round (handled separately)
- Algorithm 3.1 (Stinson)

Motivation:

- Let S be families (sets) of functions. If for any pair of functions $S_1, S_2 \in S$, there exists another function $S_3 \in S$ such that $S_1 \circ S_2 = S_3$ then S is said to be *idempotent*
- Clearly, iterated applications of an idempotent cipher do not add any additional security, only complexity for the encryption/decryption
- Fact: If S, P are both idempotent (as the substitution cipher and permutation ciphers are) and S, P both commute with each other, then $S \circ P$ is idempotent

- Proof:
$$\begin{aligned}(S_1 \circ P_1) \circ (S_2 \circ P_2) &= S_1 \circ S_2 \circ P_1 \circ P_2 \\ &= S_3 \circ P_3 \\ &= S_3 \circ P_3\end{aligned}$$
- Thus, S, P must *not* commute
- Unclear: If S, P are both idempotent and do not commute, does that necessarily mean that $S \circ P$ is not idempotent?
- Likely not true in general (likely we could come up with examples both ways)
- However, we can (likely) construct particular substitutions/permutations whose composition is not idempotent

```
   INPUT   : $x, \pi_S, \pi_P, (K^1, \ldots, K^{Nr+1})$
1  $w^0 \leftarrow x$
2  FOR $r = 1, \ldots, Nr - 1$ DO
3      $u^r \leftarrow w^{r-1} \oplus K^r$
4      FOR $i = 1, \ldots, m$ DO
5          $v^r_{(i)} \leftarrow \pi_S(u^r_{(i)})$
6      END
7      $w^r \leftarrow (v^r_{\pi_P(1)}, \cdots, v^r_{\pi_P(\ell m)})$
8  END
9  $u^{Nr} \leftarrow w^{Nr-1} \oplus K^{Nr}$
10 FOR $i = 1, \ldots, m$ DO
11     $v^{Nr}_{(i)} \leftarrow \pi_S(u^{Nr}_{(i)})$
12 END
13 $y \leftarrow v^{Nr} \oplus K^{Nr+1}$
   OUTPUT: $y$
```

Algorithm 2: Algorithm 3.1 (Stinson): SPN

Example 3.1 from Stinson:

- Let $\ell = m = N = 4$
- That's 4 blocks of 4 bits over 4 rounds
- All S-boxes the same (below)
- π_P as below
- Key schedule: given $k \in \{0, 1\}^{32}$ each round shifts by 4 bits and uses the first 16

z	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
$\pi_S(z)$	e	4	d	1	2	f	b	8	3	a	6	c	5	9	0	7

Table 2: The Substitution (S-box) tableau (in hex)

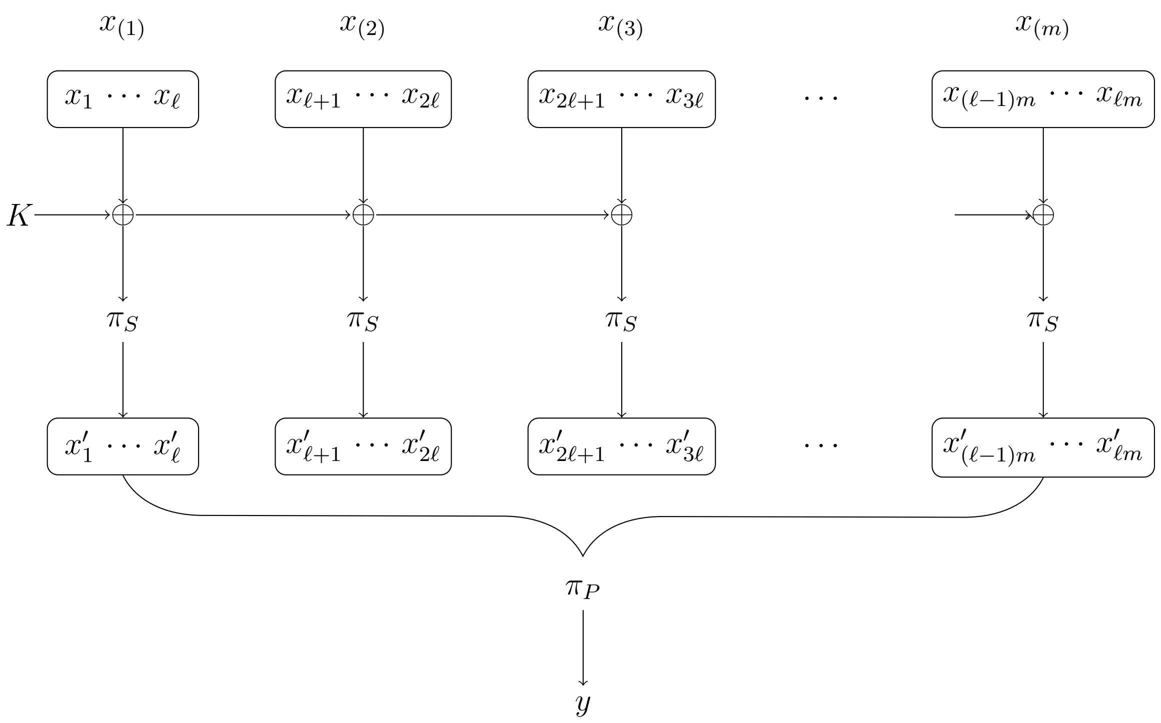

$x_{(1)}$
$x_{(2)}$
$x_{(3)}$
$x_{(m)}$
$x_1 \cdots x_\ell$
$x_{\ell+1} \cdots x_{2\ell}$
$x_{2\ell+1} \cdots x_{3\ell}$
$\cdots$
$x_{(\ell-1)m} \cdots x_{\ell m}$
K
π_S
π_S
π_S
π_S
$x'_1 \cdots x'_\ell$
$x'_{\ell+1} \cdots x'_{2\ell}$
$x'_{2\ell+1} \cdots x'_{3\ell}$
$\cdots$
$x'_{(\ell-1)m} \cdots x'_{\ell m}$
π_P
y

Figure 4: One round of a general substitution-permutation network

z	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\pi_P(z)$	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16

Table 3: The Permutation tableau

An example encryption of the SPN:

```
Substitution-Permutation Network
  blockSize = 4
  numBlocks = 4
  numRounds = 4
  key       = 1100 0100 1011 0110 1000 0110 0011 0010
 Round Keys:
  k_01 = 1100 0100 1011 0110
  k_02 = 0100 1011 0110 1000
  k_03 = 1011 0110 1000 0110
  k_04 = 0110 1000 0110 0011
  k_05 = 1000 0110 0011 0010

w_0 = 1001 1010 0010 1011
Round 1
 K_1 = 1100 0100 1011 0110
 u_1 = 0101 1110 1001 1101
 r_1 = 1111 0000 1010 1001
 w_1 = 1011 1000 1010 1001
```

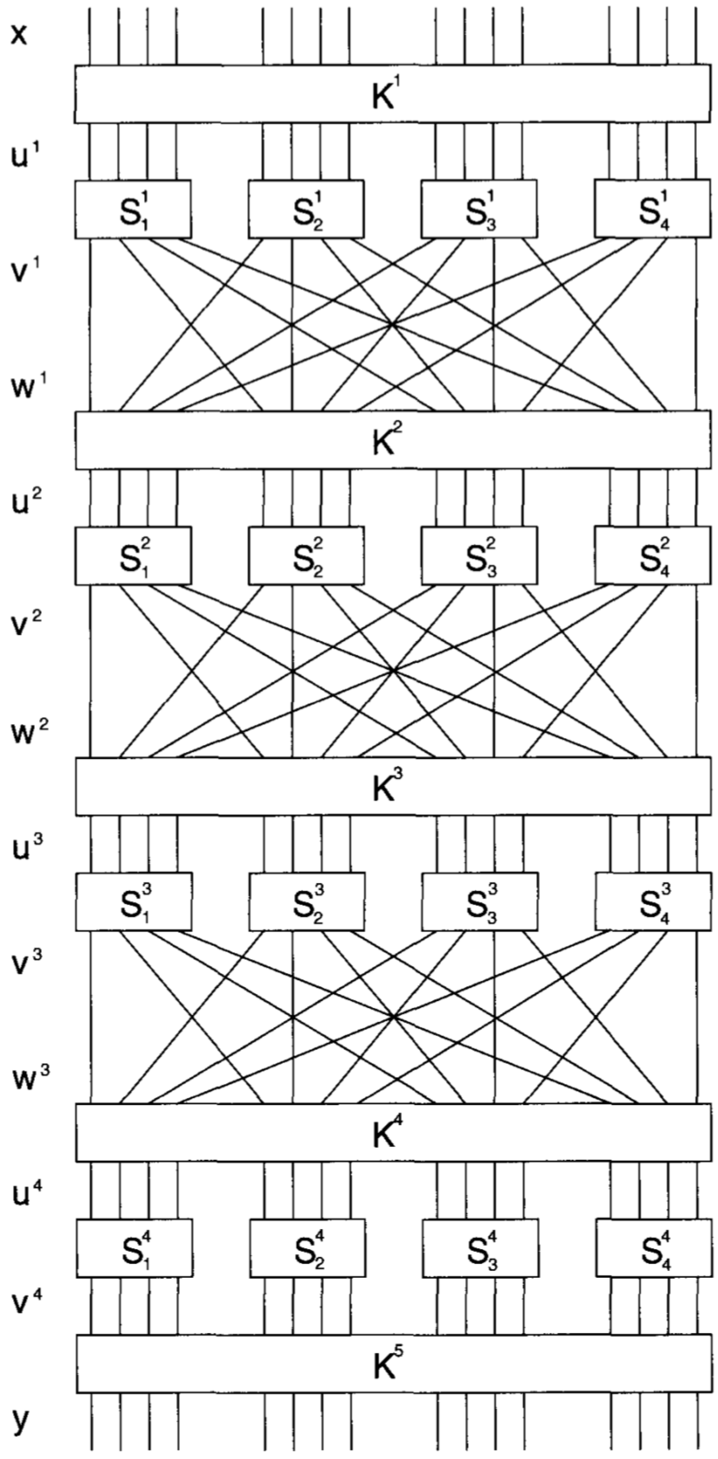


Figure 5: SPN from Stinson (Example 3.1)

```
Round 2
 K_2 = 0100 1011 0110 1000
 u_2 = 1111 0011 1100 0001
 r_2 = 0111 0001 0101 0100
 w_2 = 0000 1011 1000 1110
Round 3
 K_3 = 1011 0110 1000 0110
 u_3 = 1011 1101 0000 1000
 r_3 = 1100 1001 1110 0011
 w_3 = 1110 1010 0011 0101
Round 4 (outside loop)
 w_3 = 1110 1010 0011 0101
 K_4 = 0110 1000 0110 0011
 u_4 = 1000 0010 0101 0110
 r_4 = 0011 1101 1111 1011
 K_5 = 1000 0110 0011 0010
 y   = 1011 1011 1100 1001

k = 1100 0100 1011 0110 1000 0110 0011 0010 (c4b68632)
x = 1001 1010 0010 1011 (9a2b)
y = 1011 1011 1100 1001 (bbc9)
```

- In general, a mapping
$$\{0,1\}^k \to \{0,1\}^k$$
maps $n = 2^k$ elements which requires a look-up table of size
$$n \log n = k \cdot 2^k$$
- Example: 16 bits to 16 bits requires $2^{20} = 1$MB
- Example: 64 bits to 64 bits would require $64 \cdot 2^{64} = 1$ zettabyte

4. DES

- NIST (then NBS) issued RFPs'
- IBM submitted 1974, approved 1976
- Initial skepticism (S-boxes, small key)
- Discovered in 1990 that boxes protected against differential cryptanalysis (known to NSA, IBM)

Operation:

- Plaintext (64 bits) input: x
- Initial Permutation $\mathsf{IP}(x)$
- 16 rounds:

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

Table 4: S-box S_1

- Each round, block is split into 2 half-blocks: L, R swapped, exclusive-ors, a Feistel function is applied (cf. Figure 6)
- Final step: final permutation: $\mathsf{FP} = \mathsf{IP}^{-1}$

Feistel Function:

- (Horst Feistel, IBM): an iterated cipher with a round function f
- Under the assumption that f is cryptographically secure, iterated applications of f provably provide stronger security
- 32 bit half block is Expanded: $E(x)$ to a 48 bit block
- 48 bit expanded block is xored with sub key
- 48 bits is split into 8 6-bit blocks
- 8 different S-boxes s_i are applied to each block: each outputs only 4 bits (32-bit output)
- A permutation P is applied resulting in 48 bit output

S-boxes:

- 8 different ones;
- 4 rows, 16 columns
- Given input $x = b_5b_4b_3b_2b_1b_0$
- First and last bit b_5b_0 specify a row, middle 4 bits $b_4b_3b_2b_1$ specify a column
- Result is the table entry (0–15)
- Example: $s_1 : 011011$

Key:

- Initially 64 bits, every 8th bit is ignored (parity checksum)
- Permuted Choice 1 (PC-1) is applied
- Separated into two 28-bit halves
- 16 rounds: each half is cyclically shifted left (rounds 1, 2, 9, 16 by 1 bit, all others by 2 bits)
- Each round: shift then apply PC-2, use the result as key k_i for $i = 1 \ldots, 16$
- Decryption: same scheme, with key schedule reversed!
- http://en.wikipedia.org/wiki/Cipher_block_chaining#Cipher-block_chaining_

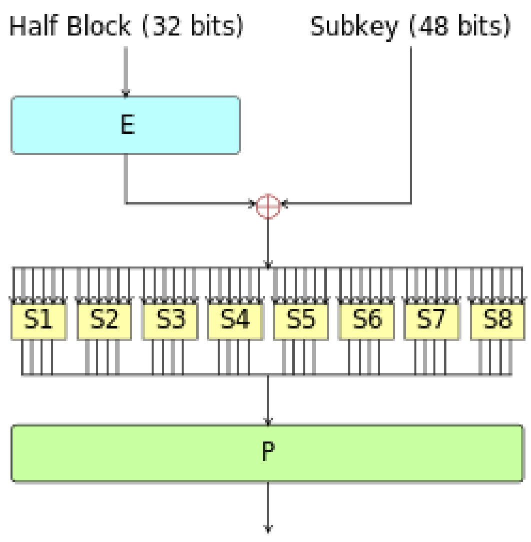


Figure 6: The Feistel Function Visualization

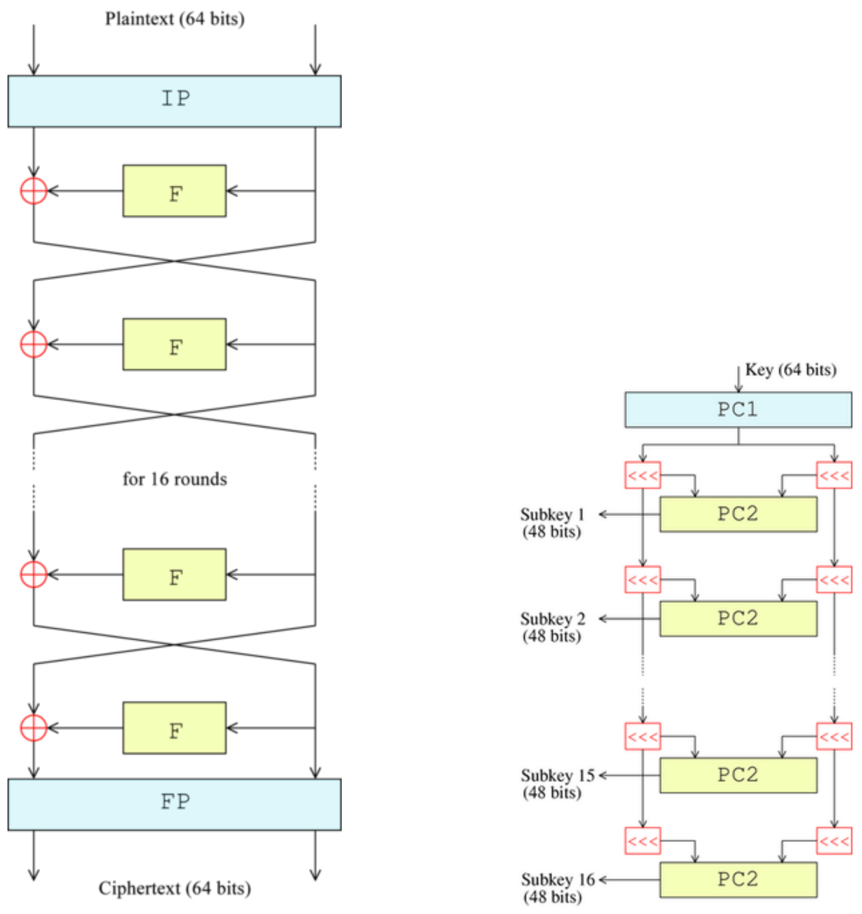


Figure 7: DES main network and key schedule

.28CBC.29

- Efficiency vs safety/advantages
- Contrast: recent CBC attacks: BEAST, Lucky13
- Triple DES: 3 keys $K = (k_1, k_2, k_3)$, 56 bits each (excluding parity)

$$e_K(x) = e_{k_3}(d_{k_2}(e_{k_1}(x)))$$

$$d_K(x) = d_{k_1}(e_{k_2}(d_{k_3}(y)))$$

  Three modes: 3 different keys; $k_1 = k_2 \neq k_3$, all identical

Current state:

- Brute force strategy: try all 2^{56} keys
  - Desktop at 1 million per second: 2283.36 years
  - Supercomputer at 1TFLOP (10^{12} ops per sec): 20.016 hours
- Best attacks require 2^{39} chosen plaintext-ciphertext attacks
  - Desktop at 1 million per second: 6 days, 8 hours
  - Supercomputer at 1TFLOP (10^{12} ops per sec): .5497 seconds
- NIST no longer recommends simple DES (obsolete)
- TDES-mode 1 (112 actual bits of security) is valid until 2030 (reexamined at that point)
- Best known attack for TDES-mode 1 (2014) requires 2^{32} plaintexts, 2^{113} steps, 2^{90} encryptions, 2^{88} memory
  - Desktop at 1 million per second: 3.29×10^{20} years
  - Supercomputer at 1TFLOP (10^{12} ops per sec): 3.29×10^{14} years (329 trillion years)

5. AES

- Late 90s NIST put out RFP for DES replacements
- 15 finalists over several years were vetted publicly
- 2001 NIST adopted (FIPS 197)
- ISO 18033-3 (2002)
- Still approved for top-secret data
- Rijndael cipher; supports 128, 192, 256 bit key modes
- Joan Daemen & Vincent Rijmen
- Name (“Rhine Doll”): portmanteau of author’s name

```
   | 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
---|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
00 |63 7c 77 7b f2 6b 6f c5 30 01 67 2b fe d7 ab 76
10 |ca 82 c9 7d fa 59 47 f0 ad d4 a2 af 9c a4 72 c0
20 |b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 71 d8 31 15
30 |04 c7 23 c3 18 96 05 9a 07 12 80 e2 eb 27 b2 75
40 |09 83 2c 1a 1b 6e 5a a0 52 3b d6 b3 29 e3 2f 84
50 |53 d1 00 ed 20 fc b1 5b 6a cb be 39 4a 4c 58 cf
60 |d0 ef aa fb 43 4d 33 85 45 f9 02 7f 50 3c 9f a8
70 |51 a3 40 8f 92 9d 38 f5 bc b6 da 21 10 ff f3 d2
80 |cd 0c 13 ec 5f 97 44 17 c4 a7 7e 3d 64 5d 19 73
90 |60 81 4f dc 22 2a 90 88 46 ee b8 14 de 5e 0b db
a0 |e0 32 3a 0a 49 06 24 5c c2 d3 ac 62 91 95 e4 79
b0 |e7 c8 37 6d 8d d5 4e a9 6c 56 f4 ea 65 7a ae 08
c0 |ba 78 25 2e 1c a6 b4 c6 e8 dd 74 1f 4b bd 8b 8a
d0 |70 3e b5 66 48 03 f6 0e 61 35 57 b9 86 c1 1d 9e
e0 |e1 f8 98 11 69 d9 8e 94 9b 1e 87 e9 ce 55 28 df
f0 |8c a1 89 0d bf e6 42 68 41 99 2d 0f b0 54 bb 16
```

Figure 8: AES (Rijndael) S-box

5.0.1. Overall Description (128 bit version)

- Given a 128 bit key, expand it into 44 words (4 words = 16 bytes = 128 bits per round) (below)
- Given: a plaintext $x = x_0 x_1 \ldots, x_{15} \in \{0,1\}^{128}$ (16 bytes)
- Initialize State to x as a 4×4 matrix S of bytes:

x_0	x_4	x_8	x_{12}
x_1	x_5	x_9	x_{13}
x_2	x_6	x_{10}	x_{14}
x_3	x_7	x_{11}	x_{15}

- Round 0: AddRoundKey to state
- Round 1–9:
  1. Apply S-box substitution ("SubBytes") to each byte in State (maps 1 byte to 1 byte)
  2. Apply permutation: ShiftRows: left shift the first row 0 times, row 2 once, row 3 twice, row 4 thrice
  3. Apply a MixColumns on state (below)
  4. Apply AddRoundKey to state
- Round 10: Same (skip MixColumns)

5.0.2. Add Round Key

- Each round r, $w[r], w[r+1], w[r+2], w[r+3]$ are used $r = 0, \ldots, 10$
- Each $w[i]$ is a word (4 bytes) and is applied to a *column*
- Best way to visualize: split each word up into 4 bytes, apply to column
- Exclusive-or is applied, mutating the STATE

5.0.3. Algebra

- A *Galois Field* (Gal-o-wa) is any finite field
- The *order* of a field is its size (number of elements in it)
- Fact: the order of any Galois Field must be a prime power, p^n
- Fact: All fields of the same order are isomorphic
- The *characteristic* of a field of order p^n is p: the number of times you need to add any element to get zero (that is $x + x + \cdots + x = xp = 0$)
- Notations:
$$\mathbb{F}_{p^n} \quad GF(p^n)$$
- Examples: $GF(2) = \mathbb{Z}$, $GF(2^8) = GF(256) = \mathbb{Z}_{256} = (\mathbb{Z})^8$
- A polynomial is *irreducible* if it is not the product of any other polynomials (in the ring of polynomials)
- Examples: $x^2 - 1 = (x-1)(x+1)$ is not irreducible, but $x^2 + 1$ is irreducible
- Construction: given an order p^n, we construct a field by: select an irreducible polynomial $p(x)$ of degree n, then
$$\mathbb{F}_{p^n}[x]/p(x)$$
is a field of size p^n with coefficients in $\mathbb{F}_p$
-

5.0.4. Mix Columns

- Linear transformation applied to each column of the state
- Input: (a_0, a_1, a_2, a_3) for each of the columns

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

This can also be seen as the following:

$$
\begin{aligned}
b_0 &= 2a_0 + 3a_1 + 1a_2 + 1a_3 \\
b_1 &= 1a_0 + 2a_1 + 3a_2 + 1a_3 \\
b_2 &= 1a_0 + 1a_1 + 2a_2 + 3a_3 \\
b_3 &= 3a_0 + 1a_1 + 1a_2 + 2a_3
\end{aligned}
$$

- A set matrix is used on each column vector
- The entries are simply 01, 02, 03
- Addition is a simple exclusive-or, however
- Multiplication is in the Galois Field defined by the irreducible polynomial

$$p(x) = x^8 + x^4 + x^3 + x + 1$$

- In general, a binary vector $b_7b_6b_5b_4b_3b_2b_1b_0$ corresponds to the coefficients in the polynomial
$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$$
- Example: `03 = 00000011` times `bf = 10111111`

$$(x+1)(x^7 + x^5 + x^4 + x^3 + x^2 + x + 1)$$

  expanded and with cancellation,

$$x^8 + x^7 + x^6 + 1$$

  after reducing modulo $p(x)$, the remainder is

$$x^7 + x^6 + x^4 + x^3 + x$$

  which is 1101 1010 (`0xda`)
- Addition over $\mathbb{Z}_2$ is simply exclusive-or
- If 01, no change
- If 03: $03 = (02 + 01)$ so it can be broken into a multiplication by 1, then by 2, add the results
- Remains: multiplication by 2.
- Trick: multiplication by 2 is simply a left-shift by 1 bit;
- Prior to the shift, if $b_7 = 1$, then add (xor) the result by

$$00011011 = x^4 + x^3 + x + 1$$

Example:

$$\texttt{0x03} \times \texttt{0xbf} = \texttt{0000 0011} \times \texttt{1011 1111}$$

	`1 0111 1110`	shifted
$\oplus$	`1011 1111`	original
	`1 1100 0001`	overflow, so add
$\oplus$	`0001 1011`	$x^4 + x^3 + x + 1$
	`1101 1010`	$=$ `0xda`

Why this works:

- In general, over $GF(2^n)$,

$$x^n \mod p(x) = p(x) - x^n$$

- Because we only multiply by 1, 2, or 3; the overflow is limited to at most one bit corresponding to x^8
- If there is no overflow, we don't need to do anything
- If there is one bit of overflow, then we need to mod out by $p(x)$
- In general: we may have to perform complicated polynomial long division
- But since it is at most one bit (x^8), we can take

$$p(x) - x^8 = (x^4 + x^3 + x + 1)$$

- Thus, if overflow:

$$x^8 + q(x) \bmod p(x)$$

- As $q(x) < p(x)$, we only need worry about the first term:

$$\begin{aligned} x^8 \bmod p(x) &= p(x) - x^n \\ &= x^8 + x^4 + x^3 + x + 1 - x^8 \\ &= x^4 + x^3 + x + 1 \end{aligned}$$

- Plugging it back in:

$$x^8 + q(x) \bmod p(x) = (x^4 + x^3 + x + 1) + q(x)$$

5.0.5. Key Schedule

```
INPUT : $key[0,\ldots,15]$, 16 bytes = 128 bit key
$Rcon[1\ldots10] = \{01000000, 02000000, 04000000, 08000000,$
    $10000000, 20000000, 40000000, 80000000, 1b000000, 36000000\}$
FOR $i = 0,\ldots,3$ DO
    $w[i] = key[4i], key[4i+1], key[4i+2], key[4i+3]$
    //32 bits
END
FOR $i = 3,\ldots,43$ DO
    $t = w[i-1]$
    IF $i \equiv 0 \bmod 4$ THEN
        $t = \text{SubWord}(\text{RotateWord}(t)) \oplus Rcon[i/4]$
        //rotate word cyclicly rotates left 1 byte; sub word applies
          the s-box to each byte
    END
    $w[i] = w[i-4] \oplus t$
END
```

5.0.6. Conclusion

- Decryption: define inverse and reverse each operation
  - Inverse S-box (see Figure 9): simply just treat entries as row/column entries to reverse
  - Inverse Shift rows: shift right instead
  - Inverse Mix Columns: use the following matrix:

$$\begin{bmatrix} r_0 \\ r_1 \\ r_2 \\ r_3 \end{bmatrix} = \begin{bmatrix} 14 & 11 & 13 & 9 \\ 9 & 14 & 11 & 13 \\ 13 & 9 & 14 & 11 \\ 11 & 13 & 9 & 14 \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} \texttt{0x0E} & \texttt{0x0B} & \texttt{0x0D} & \texttt{0x09} \\ \texttt{0x09} & \texttt{0x0E} & \texttt{0x0B} & \texttt{0x0D} \\ \texttt{0x0D} & \texttt{0x09} & \texttt{0x0E} & \texttt{0x0B} \\ \texttt{0x0B} & \texttt{0x0D} & \texttt{0x09} & \texttt{0x0E} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

$$\begin{aligned} r_0 &= 14a_0 + 11a_1 + 13a_2 + 9a_3 \\ r_1 &= 9a_0 + 14a_1 + 11a_2 + 13a_3 \\ r_2 &= 13a_0 + 9a_1 + 14a_2 + 11a_3 \\ r_3 &= 11a_0 + 13a_1 + 9a_2 + 14a_3 \end{aligned}$$

    Or: you can simply do a look-up of multiplication: http://en.wikipedia.org/wiki/Rijndael_mix_columns
  - Add Round Key: xor is its own inverse

    - Same key schedule, but applied in reverse
- 192 and 256 bit versions: 12 and 14 rounds respectively (larger key schedule)
- Freakin' Awesome visualization (take care, contains errors: see `rijndael_ingles2004.swf`)
- RFC for Galois Counter Mode https://tools.ietf.org/html/rfc5288
- Best attacks (2006): 7, 8, 9 rounds for each version; 2011: full round attack requires 2^{126} (128-bit version), $2^{254.4}$ (256-bit version)
- Brute force at 1TFLOPs: 7.8×10^8 *times* the age of the universe (128 bit); 2.7×10^{47} times the age of the universe (256 bit version)
- Even best attack at 1TFLOPs: 2×10^8, 8.8×10^{46} (128, 256 version respectively)

```
   | 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
---|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
00 |52 09 6a d5 30 36 a5 38 bf 40 a3 9e 81 f3 d7 fb
10 |7c e3 39 82 9b 2f ff 87 34 8e 43 44 c4 de e9 cb
20 |54 7b 94 32 a6 c2 23 3d ee 4c 95 0b 42 fa c3 4e
30 |08 2e a1 66 28 d9 24 b2 76 5b a2 49 6d 8b d1 25
40 |72 f8 f6 64 86 68 98 16 d4 a4 5c cc 5d 65 b6 92
50 |6c 70 48 50 fd ed b9 da 5e 15 46 57 a7 8d 9d 84
60 |90 d8 ab 00 8c bc d3 0a f7 e4 58 05 b8 b3 45 06
70 |d0 2c 1e 8f ca 3f 0f 02 c1 af bd 03 01 13 8a 6b
80 |3a 91 11 41 4f 67 dc ea 97 f2 cf ce f0 b4 e6 73
90 |96 ac 74 22 e7 ad 35 85 e2 f9 37 e8 1c 75 df 6e
a0 |47 f1 1a 71 1d 29 c5 89 6f b7 62 0e aa 18 be 1b
b0 |fc 56 3e 4b c6 d2 79 20 9a db c0 fe 78 cd 5a f4
c0 |1f dd a8 33 88 07 c7 31 b1 12 10 59 27 80 ec 5f
d0 |60 51 7f a9 19 b5 4a 0d 2d e5 7a 9f 93 c9 9c ef
e0 |a0 e0 3b 4d ae 2a f5 b0 c8 eb bb 3c 83 53 99 61
f0 |17 2b 04 7e ba 77 d6 26 e1 69 14 63 55 21 0c 7d
```

Figure 9: Inverse Rijndael S-Box

5.1. Modes of Operation

Any block cipher can be made to operate in different modes for different applications

- ECB (Electronic Code Book) Mode: each block of data is independently
    - In general, same problems as classic cipher: changes are only local; global info can still be discerned
    - Blocks of the same data have the same ciphertext block
    - Good for parallel processing
- CBC (Cipher Block Chaining) Mode: each cipher block is xored withe the next plaintext block

  - Define an Initialization Vector (IV): y_0
    - IV need not be kept secret
    - Only purposes: repeated encryption of the same plaintext with the same key should not produce the same ciphertext
    - Should *not* reuse the same IV
  - Then the cipher stream $y_1 y_2 \cdots$ is constructed by the rule:

$$y_i = e_K(y_{i-1} \oplus x_i)$$

- OFB (output feedback) Mode: an independent key stream is generated
  - Define an Initialization Vector (IV): z_0 and generate a key stream:

$$z_1 z_2, \cdots$$

  - Key stream is generated by the rule:

$$z_i = e_K(z_{i-1})$$

  - Then the encryption sequence is:

$$y_i = x_i \oplus z_i$$

  - Still: local changes only affect 1 block of the ciphertext
- CFB (Cipher feedback) Mode:
  - Initialization vector set: y_0
  - Key stream is generated by:

$$z_i = e_K(y_{i-1})$$

  - Cipher stream defined as:

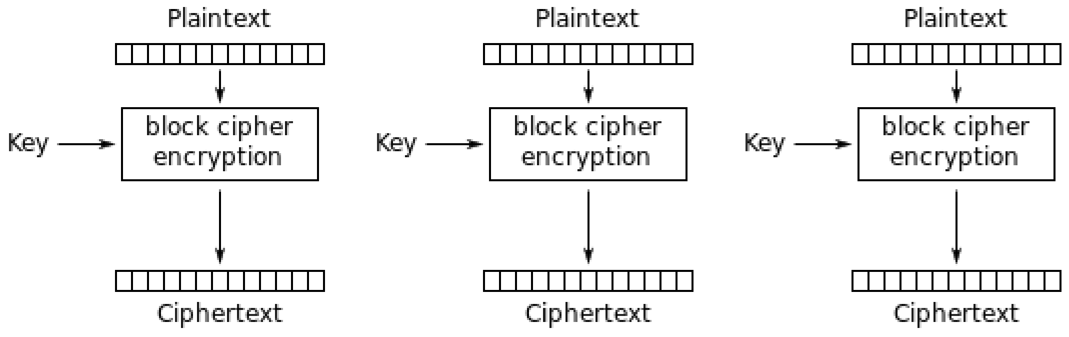
$$y_i = x_i \oplus z_i$$

  - CBC, CFB: useful for producing a MAC (message authentication code) for integrity/authenticity (but not secrecy or nonrepudiation)
- CTR (Counter) Mode: a counter
  - A nonce n (one-time use random number) is combined with a simple counter (or high-period sequence generator) $c_1 c_2 \cdots$
  - Nonce and counter combined and encrypted:

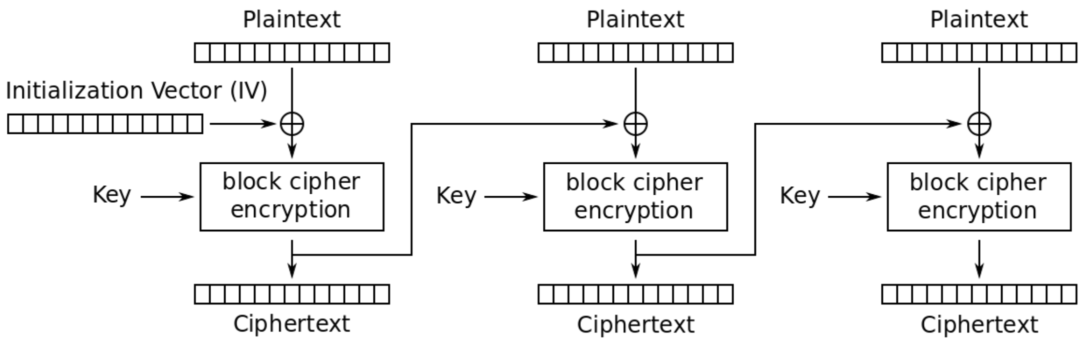
$$z_i = e_K(n \oplus c_i)$$

  - Ciphertext sequence is thus

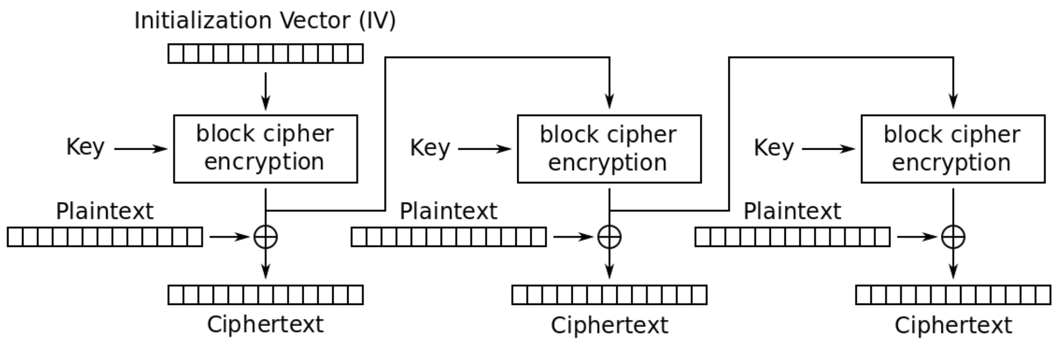
$$y_i = x_i \oplus z_i$$



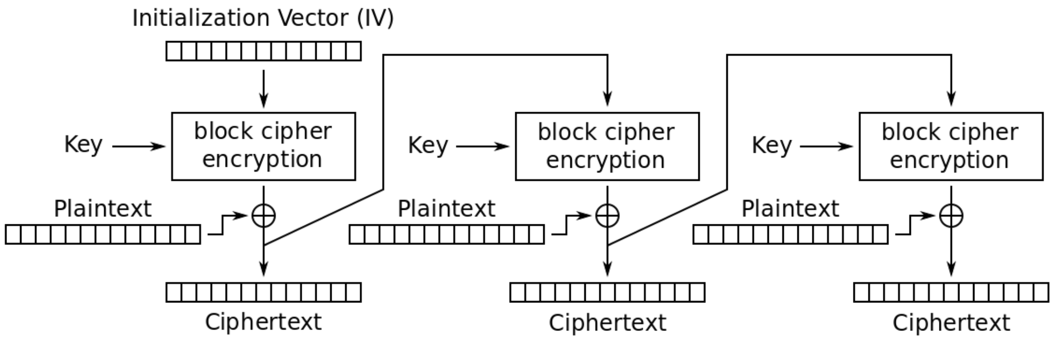
Electronic Codebook (ECB) mode encryption



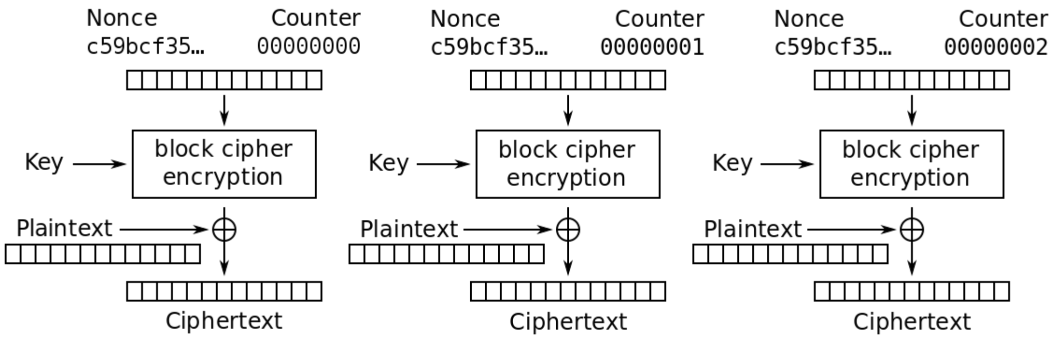
Cipher Block Chaining (CBC) mode encryption



Output Feedback (OFB) mode encryption



Cipher Feedback (CFB) mode encryption



Counter (CTR) mode encryption

5.1.1. Galois Counter Mode

TODO

Part III. Hash Functions

6. Basics

- A *hash function* is a function that maps a “large” domain to a small co-domain
- For our purposes:
  $$h : \{0,1\}^n \rightarrow \{0,1\}^m$$
  or
  $$h : \{0,1\}^{\leq n} \rightarrow \{0,1\}^m$$
  or
  $$h : \{0,1\}^{<\infty} \rightarrow \{0,1\}^m$$
- Keyed hash functions: h_k for some key k
- Desired properties:
  - Functions are not bijective: thus not invertible
  - Uniformity: mapping should equally distribute (hash) elements in the domain over elements in the codomain
  - Efficient to compute

Applications:

- Hash table-based data structures (maps, caches, etc.)
- Message Authentication Codes (MAC): digests
- Checksums: Data Integrity
  - Alice computes the hash of a message/file/data
  - Alice sends the message and hash to Bob
  - Bob computes the hash of the received message
  - If the same: high confidence that the message was authentic and not tampered with
  - If different: file integrity was compromised (or the hash)
  - Oscar can still modify *both* and tamper with the message/file (but it should be difficult)
- Secure Password storage
  - A user’s provides a password
  - Usually, the system adds (appends) a random *salt*
  - The password and salt are hashed and the resulting hash value is stored (salt is also stored)

  - Authentication is repeated by asking for the password, hash it, and compare to the stored hash
  - The actually password is never stored and only ever exists in memory

6.1. Types of Attacks

Cryptographic Properties:

- Efficient to compute (or maybe not!)
- Small changes in the input should permeate throughout the output (large changes in the output): called the *avalanche* effect
- It should be infeasible to generate a message that has a given hash
- It should be infeasible to modify a message without changing its hash
- It should be infeasible to find two different messages with the same hash

Preimage attack:
Given: A hash function $h : X \to Y$ and $y \in Y$
Find: $x \in X$ such that $h(x) = y$

- Effective for password cracking (you have the hash, you need to find the password or provide an alternative, equivalent password)

Second Preimage attack:
Given: A hash function $h : X \to Y$ and $x \in X$
Find: $x' \in X$ such that $x \neq x'$ and $h(x) = h(x')$

Collision attack:
Given: A hash function $h : X \to Y$
Find: $x, x' \in X$ such that $h(x) = h(x')$

- Collision is the least restrictive problem; intuitively it is the "least hard"
- Find two messages that have the same hash
- Compromises digital signatures: you can get Alice to sign document A and then use the signature on document B if $h(A) = h(B)$
- More practically: *chosen-prefix collision* attack: fix p_1, p_2 and find m_1, m_2 such that

$$h(p_1 + m_1) = h(p_2 + m_2)$$

- Allows one to append garbage data (which may be ignored) to certificates or files with different functionality (in the prefix)
- Recall that problem A reduces to a problem B and we write

$$A \leq_{\mathsf{P}} B$$

if there is a polynomial algorithm mapping yes/no instances of A to yes/no instances of B

- Alternatively: $A \leq_\mathsf{P} B$ if you can solve A with an oracle for B (as a subroutine)
- If a hash is collision resistant, this implies second-preimage resistance: Collision $\leq_\mathsf{P}$ SecondPreimage
  - Select $x \in X$, compute $y = h(x)$, use Second Preimage solution to find $x' \neq x$ with the same hash, output (x, x')
- If a hash is collision resistant, this implies pre image resistance if $|X| \geq 2|Y|$ (very weak, almost universal assumption)
  - Select $x \in X$, compute $y = h(x)$, use Preimage solution to find x' with $h(x') = y$, if $x \neq x'$, output (x, x')
- Reductions details omitted (see Stinson)
- In general, collision attacks are not harder than preimage attacks (no constraint on the hash y)
- General rule: if collisions can be found, time to retire the hash

Birthday attack

- Brute Force strategy to find collisions: try all 2^n possibilities ($2^n = |X|$)
- Can we do better? Yes: probabilistic pigeonhole principle
- Scenario: suppose we throw n objects into m boxes with uniform $\frac{1}{m}$ probability
- Then one box will have more than one object with probability

$$1 - \frac{m!}{(m-n)!m^n}$$

- Birthday paradox: for $n = 23, m = 365$, probability is 50.7%

It will be easier to calculate the probability of no collisions since:

$$\Pr[\text{at least 1 box contains at least 2 elements}] = 1 - \Pr[\text{all boxes have at most 1 element}]$$

Derivation:

- Fix an ordering on the objects:

$$o_1, o_2, \ldots, o_n$$

- Consider the n-permutations of m boxes:

$$P(m, n) = \frac{m!}{(m-n)!} = m(m-1)(m-2)\cdots(m-n+1)$$

- These are the pigeonholes that we will evenly distribute n objects into.
- Permutation: order is important because the objects are distinct.

- We place each object into distinct pigeonhole with probability $\frac{1}{m}$, so in total:

$$\left(\frac{1}{m}\right)^n = \frac{1}{m^n}$$

- Thus the probability is:

$$1 - \frac{m!}{(m-n)!m^n}$$

Alternate Derivation

- Consider choosing n bins (from a total of m bins to map n objects to:

$$\binom{m}{n} = \frac{n!}{(m-n)!n!}$$

- But now consider actually mapping objects $o_1, \ldots, o_n$ to each bin
- Here, order matters, but in addition, we only want to map one object to one bucket
- That is, we want to count the total number of one-to-one functions from $o_1, \ldots, o_n$ to the n bins we chose.
- Thus:

$$\frac{n!}{(m-n)!n!} \cdot n! = \frac{n!}{(m-n)!}$$

- Again, viewing allocation of objects to buckets, how many functions in total are there?

$$m^n$$

- Thus the probability of a random allocation resulting in all bins having 0 or 1 objects is

$$\frac{n!}{(m-n)!} \div m^n = \frac{m!}{(m-n)!m^n}$$

Finally:

- Hash scenario: evaluate q hashes (among 2^n possibilities) to find a collision
- How large does q need to be to provide a better than 50% probability of finding a collision
- Details: using a Taylor Series approximation for e^{-x}, solving and estimating (omitted):

$$q \approx 1.17\sqrt{|Y|}$$

- If $|Y| = 2^m$ then we only need to evaluate

$$\sqrt{2^m} = 2^{m/2}$$

hashes to find a collision with 50% expected success

- Examples: MD4 ($m = 160$), SHA-3 ($m = 256$)
- Any attack that does better than a birthday attack, even if impractical compromises a hash function

Summary

- For pre image and 2nd pre image attacks, the ideal strength is 2^n for brute force attacks
- For collision attacks, the ideal strength is $2^{n/2}$ for birthday attacks
- For a MAC attack: attacker would want to deduce the key or produce a new, forged message agnostic of the key

Bad Example

- Consider the simple hash function:
$$h : \mathbb{Z} \to \mathbb{Z}_7$$
defined by
$$h(x) = 3x + 5 \bmod 7$$
- In general, hash functions of the form $h(x) = ax + b \bmod m$ are okay for (say) hash tables, but they are not cryptographically secure
- They are not collision resistant: for any $x \in \mathbb{Z}$, $7x$ hashes to the same value; in general for any $x \in \mathbb{Z}$, mx hashes to the same value
- In general, failure to be collision resistant does not necessarily imply failure to be pre image resistant, but in this case it is.
- Find a pre image: given $y = h(x)$, simply solve:
$$a^{-1}(km - b) = x$$
for any $k \in \mathbb{Z}$ to get one.

6.2. Resources

Online Calculators:

- MD4: http://hash.online-convert.com/md4-generator
- MD5: http://www.md5.cz/
- MD5 Decoder: http://md5decoder.org/
- SHA-1: http://www.sha1-online.com/
- SHA-256: http://www.xorbin.com/tools/sha256-hash-calculator
- SHA-3: http://www.fishtrap.co.uk/online-sha3/
- JavaScript Crypto! https://code.google.com/p/crypto-js/

7. MD4 & MD5

MD4:

- 1990, Ronald Rivest (of RSA) RFC1320
- $\{0,1\}^{\leq 2^{64}-1} \to \{0,1\}^{128}$
- 3 rounds with 16 operations each: some nonlinear, some mixing, some constants
- Used for passwords in Windows NT, XP, Vista, 7!!!
- Thoroughly broken:
  - 1995: Full collision attack
  - 2007: collision attack in microseconds
  - 2008: Theoretical pre image attack (2^{102})
  - 2011: RFC 6150 retired MD4

MD5:

- 1992 (Rivest, RFC 1321)
- $\{0,1\}^{\leq 2^{64}-1} \to \{0,1\}^{128}$
- 4 rounds, some other improvements
- Often used in password storage (common, but bad practice)
- Broken:
  - 1996: flaw, not fatal at the time
  - 2004: collision resistance broken by birthday attack (1 hr on a cluster)
  - Creation of 2 files with same checksum
  - 2005-2008: more advances, SSL certificate validity exploit
  - 2009: Theoretical pre image attack in $2^{123.4}$
  - 2011: RFC 6151 recommends against continued use
  - 2013: collision attack in 2^{18}: < 1 second
  - Chosen prefix attack in a few hours on a laptop (2^{39})

8. SHA1

8.1. Merkle-Damgård Constructions

- Message is padded (with MD-compliant conditions) to have n blocks of equal length
- Padding is simply 0^d+ binary representation of d
- State is initialized with Initial Vector (IV), 0^{m+1}
- Padded input is split into blocks of length $t-1$

- Each block is fed, along with the previous result into a one-way compression function $f : \{0,1\}^{m+t} \to \{0,1\}^m$
- If f is collision resistant, the Merkle-Demgård hash construction is collision resistant

```
$n \leftarrow |x|$
$k \leftarrow \lceil n/(t-1) \rceil$
$d \leftarrow k(t-1) - n$
FOR $i = 1, \dots, k-1$ DO
    $y_i \leftarrow x_i$
END
$y_k \leftarrow x_k \,|\, 0^d$
$y_{k+1} \leftarrow$ binary representation of $d$
//IV is of $m+1$ zeros, $y_1$ is of length $t-1$, so $z_1$ is of length $m+t$
$z_1 \leftarrow 0^{m+1} \,|\, y_1$
$g_1 \leftarrow f(z_1)$
FOR $i = 1, \dots, k$ DO
    $z_{i+1} \leftarrow g_i \,|\, 1 \,|\, y_{i+1}$
    $g_{i+1} \leftarrow f(z_{i+1})$
END
$h(x) \leftarrow g_{k+1}$
```

Algorithm 3: Merkle Damgård Hash Construction

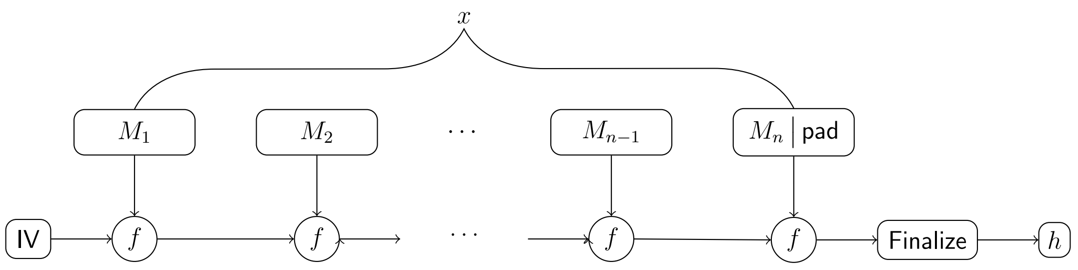


Figure 10: A general Merkle-Damgård Hash Function Construction

- 1995, FIPS 180-1
- Digest size: 160 bits, 80 rounds, ideal secure bits: 80
- Still (as of 2014) the most widely used hash algorithm
- 2005: first theoretical attacks (53/80 rounds in less than $2^{160/2}$ evaluations; full in less than 2^{69})
- 2010: NIST requires move to SHA-2 for government use

- 2012: SHA1 attack that requires $2^{58.5}$ cycles for near-collision, 2^{61} for full (Stevens: cost \$2.77M)
- Better article/Chrome starts to deprecate: http://googleonlinesecurity.blogspot.com/2014/09/gradually-sunsetting-sha-1.html
- Industry leaders (Microsoft, Google, Mozilla) have scheduled disuse by 2017

Basics:

- Padded with zeros and count so that there are n blocks, each 512 bits
- Each round: each block is split into 16 words (32 bits each)
- 16 words are expanded to 80 (for rounds)
- Previous digest (160 bits) is mixed with next block (512 bits) to create a new block (160 bits)
- Uses combinations of additions, bit/block choosing, exclusive-or, majority, cyclic shifts, mixing of magic numbers

```
INPUT  : $x \in \{0,1\}^{\leq 2^{64}-1}$
1 $d \leftarrow (447 - |x|) \bmod 512$
2 $\ell \leftarrow$ binary representation of $|x|$, padded with leading zeros so $|\ell| = 64$
3 $y \leftarrow x \,|\, 1 \,|\, 0^d \,|\, \ell$
```

Algorithm 4: SHA-1-Pad(x)

After padding, we have y and break it up into n blocks of 512 bits each:

$$y = M_1 \,|\, M_2 \,|\, \cdots \,|\, M_n$$

Round Functions: $f_0, \ldots, f_{79}$:

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee (\neg B \wedge D) & 0 \leq t \leq 19 & \text{Choose} \\ B \oplus C \oplus D & 20 \leq t \leq 39 & \text{exclusive-or} \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & 40 \leq t \leq 59 & \text{majority} \\ B \oplus C \oplus D & 60 \leq t \leq 79 & \text{exclusive-or} \end{cases}$$

Nothing-up-my-sleeve keys in hex:

$$K_i = \begin{cases} \texttt{5a827999} & 0 \leq t \leq 19 & 2^{30} \cdot \sqrt{2} \\ \texttt{6ed9eba1} & 20 \leq t \leq 39 & 2^{30} \cdot \sqrt{3} \\ \texttt{8f1bbcdc} & 40 \leq t \leq 59 & 2^{30} \cdot \sqrt{5} \\ \texttt{ca62c1d6} & 60 \leq t \leq 79 & 2^{30} \cdot \sqrt{10} \end{cases}$$

Note:

- $X + Y$ is integer division modulo 2^{32}
- $\mathsf{ROTL}^s(X)$ is the cyclic left-shift of X by s positions

```
INPUT  : $x \in \{0,1\}^{\leq 2^{64}-1}$
1 $y = M_1 \mid M_2 \mid \cdots \mid M_n \leftarrow \mathsf{SHA\text{-}1\text{-}Pad}(x)$
  //initial values same as MD5, lower order bits of counting up and
    down
2 $H_0 \leftarrow$ 67452301
3 $H_1 \leftarrow$ efcdab89
4 $H_2 \leftarrow$ 98badcfe
5 $H_3 \leftarrow$ 10325476
6 $H_4 \leftarrow$ c3d2e1f0
7 FOR $i = 1, \ldots, n$ DO
      //Each $W_i$ is a word, 32 bits
8     $M_i = W_0 \mid W_1 \mid \cdots \mid W_{15}$
9     FOR $t = 16, \ldots, 79$ DO
10        $W_t \leftarrow \mathsf{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$
11    END
12    $A \leftarrow H_0$
13    $B \leftarrow H_1$
14    $C \leftarrow H_2$
15    $D \leftarrow H_3$
16    $E \leftarrow H_4$
17    FOR $t = 0, \ldots, 79$ DO
18        $temp \leftarrow \mathsf{ROTL}^5(A) + f_t(B,C,D) + E + W_t + K_t$
19        $E \leftarrow D$
20        $D \leftarrow C$
21        $C \leftarrow \mathsf{ROTL}^{30}(B)$
22        $B \leftarrow A$
23        $A \leftarrow temp$
24    END
25    $H_0 \leftarrow H_0 + A$
26    $H_1 \leftarrow H_1 + B$
27    $H_2 \leftarrow H_2 + C$
28    $H_3 \leftarrow H_3 + D$
29    $H_4 \leftarrow H_4 + E$
30 END
OUTPUT: $H_0 \mid H_1 \mid H_2 \mid H_3 \mid H_4$
```

Algorithm 5: SHA-1(x)

9. SHA-2

- Family of functions: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256
- NSA/NIST/FIPS (FIPS 180-4)
- Digest sizes: 224, 256, 384, or 512 bits
- Rounds: 64 or 80
- 256/512 structures are same, operating on 32 or 64 bits with different constants/shifts
- Other variations are truncated versions (smaller hash in the end) with different IVs
- Similar to SHA-1: Choose, Majority, x-or, parities, shifts, constants, etc.
- SHA-2 is the minimum current standard (112 secure bits, as of 2010)
- Best attacks: (Pseudo) Collision breaks 46 rounds of SHA-256; Preimage attacks 52 rounds of SHA-256, 57 rounds of SHA-512
- Pseudocode and Summary: `http://en.wikipedia.org/wiki/SHA-2`
- NIST Description: `http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512`
  `pdf`

9.1. SHA-512

The following pages are an excerpt of the SHA-512 version from the NIST Description publication originally pull from: `http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf`

3. SHA-512

3.1. Overview

SHA-512 is a variant of SHA-256 which operates on eight 64-bit words. The message to be hashed is first

(1) padded with its length in such a way that the result is a multiple of 1024 bits long, and then

(2) parsed into 1024-bit *message blocks* $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$.

The message blocks are processed one at a time: Beginning with a fixed initial hash value $H^{(0)}$, sequentially compute

$$H^{(i)} = H^{(i-1)} + C_{M^{(i)}}(H^{(i-1)}),$$

where C is the SHA-512 *compression function* and $+$ means word-wise mod 2^{64} addition. $H^{(N)}$ is the **hash** of M.

3.2. Description of SHA-512

The SHA-512 compression function operates on a 1024-bit *message block* and a 512-bit *intermediate hash value.* It is essentially a 512-bit block cipher algorithm which encrypts the intermediate hash value using the message block as key. Hence there are two main components to describe: (1) the SHA-512 compression function, and (2) the SHA-512 message schedule.
We will use the following notation:

$\oplus$	bitwise XOR
$\wedge$	bitwise AND
$\vee$	bitwise OR
$\neg$	bitwise complement
$+$	mod 2^{64} addition
R^n	right shift by n bits
S^n	right rotation by n bits

Table 2: Notation

For SHA-512, all of these operators act on 64-bit words.

The **initial hash value** $H^{(0)}$ is the following sequence of 64-bit words (which are obtained by taking the fractional parts of the square roots of the first eight primes):

$$\begin{aligned}
H_1^{(0)} &= \texttt{6a09e667f3bcc908} \\
H_2^{(0)} &= \texttt{bb67ae8584caa73b} \\
H_3^{(0)} &= \texttt{3c6ef372fe94f82b} \\
H_4^{(0)} &= \texttt{a54ff53a5f1d36f1} \\
H_5^{(0)} &= \texttt{510e527fade682d1} \\
H_6^{(0)} &= \texttt{9b05688c2b3e6c1f} \\
H_7^{(0)} &= \texttt{1f83d9abfb41bd6b} \\
H_8^{(0)} &= \texttt{5be0cd19137e2179}
\end{aligned}$$

Preprocessing

Computation of the hash of a message begins by preparing the message:

1. Pad the message in the usual way: Suppose the length of the message M, in bits, is ℓ. Append the bit "1" to the end of the message, and then k zero bits, where k is the smallest non-negative solution to the equation $\ell + 1 + k \equiv 896 \bmod 1024$. To this append the 128-bit block which is equal to the number ℓ written in binary. For example, the (8-bit ASCII) message "`abc`" has length $8 \cdot 3 = 24$ so it is padded with a one, then $896 - (24 + 1) = 871$ zero bits, and then its length to become the 1024-bit padded message

$$\texttt{01100001 01100010 01100011 1 } \underbrace{00\cdots0}_{\texttt{871}} \ \underbrace{00\cdots0011000}_{\texttt{128}}.$$

   The length of the padded message should now be a multiple of 1024 bits.

2. Parse the message into N 1024-bit blocks $M^{(1)}, M^{(2)}, \ldots, M^{(N)}$. The first 64 bits of message block i are denoted $M_0^{(i)}$, the next 64 bits are $M_1^{(i)}$, and so on up to $M_{15}^{(i)}$. We use the big-endian convention throughout, so within each 64-bit word, the left-most bit is stored in the most significant bit position.

Main loop

The hash computation proceeds as follows:

```
For $i = 1$ to $N$   ($N$ = number of blocks in the padded message)
{
        • Initialize registers $a, b, c, d, e, f, g, h$ with the $(i-1)^{\text{st}}$ intermediate hash value (=
        the initial hash value when $i = 1$) •
        $a \leftarrow H_1^{(i-1)}$
        $b \leftarrow H_2^{(i-1)}$
            $\vdots$
        $h \leftarrow H_8^{(i-1)}$

        • Apply the **SHA-512 compression function** to update registers $a, b, \ldots, h$ •
        For $j = 0$ to 79
        {
            Compute $Ch(e, f, g)$, $Maj(a, b, c)$, $\Sigma_0(a)$, $\Sigma_1(e)$, and $W_j$ (see Definitions below)
            $T_1 \leftarrow h + \Sigma_1(e) + Ch(e, f, g) + K_j + W_j$
            $T_2 \leftarrow \Sigma_0(a) + Maj(a, b, c)$
            $h \leftarrow g$
            $g \leftarrow f$
            $f \leftarrow e$
            $e \leftarrow d + T_1$
            $d \leftarrow c$
            $c \leftarrow b$
            $b \leftarrow a$
            $a \leftarrow T_1 + T_2$
        }
        • Compute the $i^{\text{th}}$ **intermediate hash value** $H^{(i)}$ •
        $H_1^{(i)} \leftarrow a + H_1^{(i-1)}$
        $H_2^{(i)} \leftarrow b + H_2^{(i-1)}$
            $\vdots$
        $H_8^{(i)} \leftarrow h + H_8^{(i-1)}$
}
```

$H^{(N)} = (H_1^{(N)}, H_2^{(N)}, \ldots, H_8^{(N)})$ is the **hash** of M.

Definitions

Six logical functions are used in SHA-512. Each of these functions operates on 64-bit words and produces a 64-bit word as output. Each function is defined as follows:

$$\begin{aligned}
Ch(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z)\\
Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)\\
\Sigma_0(x) &= S^{28}(x) \oplus S^{34}(x) \oplus S^{39}(x)\\
\Sigma_1(x) &= S^{14}(x) \oplus S^{18}(x) \oplus S^{41}(x)\\
\sigma_0(x) &= S^{1}(x) \oplus S^{8}(x) \oplus R^{7}(x)\\
\sigma_1(x) &= S^{19}(x) \oplus S^{61}(x) \oplus R^{6}(x)
\end{aligned}$$

Expanded message blocks $W_0, W_1, \ldots, W_{79}$ are computed as follows via the **SHA-512 message schedule**:

```
$W_j = M_j^{(i)}$ for $j = 0, 1, \ldots, 15$, and
For $j = 16$ to 79
{
    $W_j \leftarrow \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16}$
}
```

Definitions, continued

A sequence of constant words, $K_0, \ldots, K_{79}$, is used in SHA-512. In hex, these are given by

```
428a2f98d728ae22 7137449123ef65cd b5c0fbcfec4d3b2f e9b5dba58189dbbc
3956c25bf348b538 59f111f1b605d019 923f82a4af194f9b ab1c5ed5da6d8118
d807aa98a3030242 12835b0145706fbe 243185be4ee4b28c 550c7dc3d5ffb4e2
72be5d74f27b896f 80deb1fe3b1696b1 9bdc06a725c71235 c19bf174cf692694
e49b69c19ef14ad2 efbe4786384f25e3 0fc19dc68b8cd5b5 240ca1cc77ac9c65
2de92c6f592b0275 4a7484aa6ea6e483 5cb0a9dcbd41fbd4 76f988da831153b5
983e5152ee66dfab a831c66d2db43210 b00327c898fb213f bf597fc7beef0ee4
c6e00bf33da88fc2 d5a79147930aa725 06ca6351e003826f 142929670a0e6e70
27b70a8546d22ffc 2e1b21385c26c926 4d2c6dfc5ac42aed 53380d139d95b3df
650a73548baf63de 766a0abb3c77b2a8 81c2c92e47edaee6 92722c851482353b
a2bfe8a14cf10364 a81a664bbc423001 c24b8b70d0f89791 c76c51a30654be30
d192e819d6ef5218 d69906245565a910 f40e35855771202a 106aa07032bbd1b8
19a4c116b8d2d0c8 1e376c085141ab53 2748774cdf8eeb99 34b0bcb5e19b48a8
391c0cb3c5c95a63 4ed8aa4ae3418acb 5b9cca4f7763e373 682e6ff3d6b2b8a3
748f82ee5defb2fc 78a5636f43172f60 84c87814a1f0ab72 8cc702081a6439ec
90befffa23631e28 a4506cebde82bde9 bef9a3f7b2c67915 c67178f2e372532b
ca273eceea26619c d186b8c721c0c207 eada7dd6cde0eb1e f57d4f7fee6ed178
06f067aa72176fba 0a637dc5a2c898a6 113f9804bef90dae 1b710b35131c471b
28db77f523047d84 32caab7b40c72493 3c9ebe0a15c9bebc 431d67c49c100d4c
4cc5d4becb3e42b6 597f299cfc657e2a 5fcb6fab3ad6faec 6c44198c4a475817.
```

These are the first sixty-four bits of the fractional parts of the cube roots of the first eighty primes.

3.3. Diagrams

The SHA-512 compression function is pictured below:

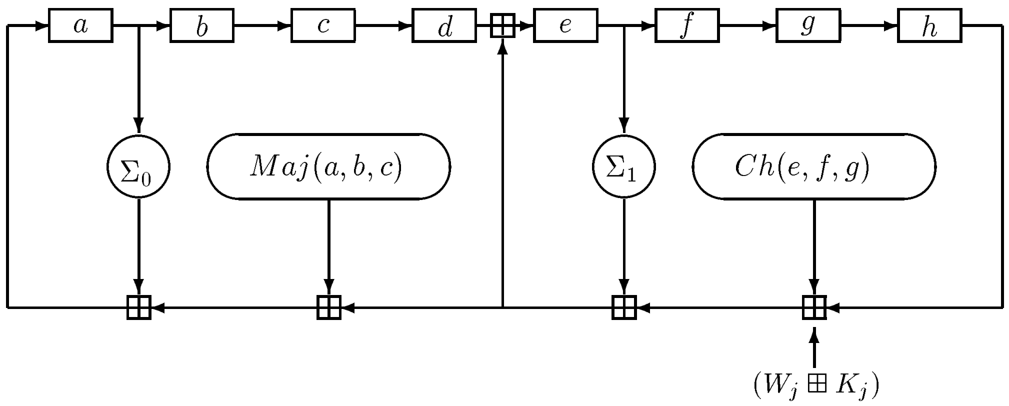


Figure 3: j^{th} internal step of the SHA-512 compression function C

where the symbol $\boxplus$ denotes mod 2^{64} addition.

The message schedule can be drawn as follows:

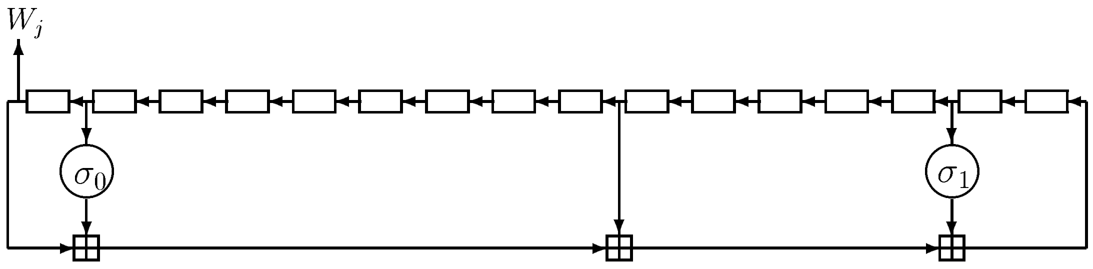


Figure 4: SHA-512 message schedule

The registers here are loaded with $W_0, W_1, \ldots, W_{15}$.

10. SHA3: Keccek

History:

- Theoretical attacks for SHA1, 2 and age led NIST to make RFC: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html for new function, 2008
- Round 1: Feb 2009, Round 2: July 2009; Final Round: Dec 2010; Winner: Oct 2012
- Winner: Keccak ("ket-chak")
- Page; http://keccak.noekeon.org/
- Keccak family of hash functions defines various internal state size

$$s \in \{25 \times 2^0, 25 \times 2^1, 25 \times 2^2, 25 \times 2^3, 25 \times 2^4, 25 \times 2^5, 25 \times 2^6\}$$

$$s \in \{25, 50, 100, 200, 400, 800, 1600\}$$

- SHA-3 restricts attention to state size $s = 25 \times 2^6 = 25 \times 64 = 1600$, with various digest sizes $224, 256, 384, 512$ providing security bits 112, 128, 192, 256
- Optional Shake128/Shake256 can be used to extend digest size to arbitrary length
- Careful note: some conversion between big-endian at the byte and word level may be necessary
- We'll restrict attention to the 512 version

Resources:

- Original SHA-3 RFC: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html
- FIPS 202 standardization still in draft: http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf
- Nice Presentation: https://www.fbi.h-da.de/fileadmin/personal/h.baier/Lectures-summe SS-13-Cryptography/lecture_crypto_ss13_chapter09-sha3-handout.pdf
- Home Page: http://keccak.noekeon.org/
- Reference: http://keccak.noekeon.org/Keccak-reference-3.0.pdf

10.0.1. Sponge Function

- Sponge Function: finite state, variable length input, variable length output
  - Internal state is initialized to zeros 0^{1600}
  - Input is padded with 10*1 to create $r = 576$-bit blocks P_i
  - Absorption Phase: For each block P_i:
    - pad: $P_i \mid 0^{b-r} = P_i \mid 0^{1024}$
    - Take the exclusive-or of the current state and the padded block (the message never affects that last 1024 bit capacity directly)

Label/Digest size	Block size (rate, r)	Capacity ($c = b - r$)
224	1152	448
256	1088	512
384	832	768
512	576	1024

Table 5: Various settings for Keccak functions for $\ell = 6$, $b = 25 \times 2^{\ell} = 1600$

    * Permute the state with the $R = \iota \circ \chi \circ \pi \circ \phi \circ \theta$ operation
  - Squeezing Phase:
    * Take the first $r = 576$ bits
    * If more bits are required, apply the permutation to the state again (no x-or)
    * take the first $r = 576$ bits and append
    * Repeat until have at least as many bits as needed
  - Take the first $M = 512$ bits of the result as the hash
- Large capacity protects against collision and preimage attacks; SHA-3: 1600 bit state
- Illustration in Figure 11
- Pseudocode below

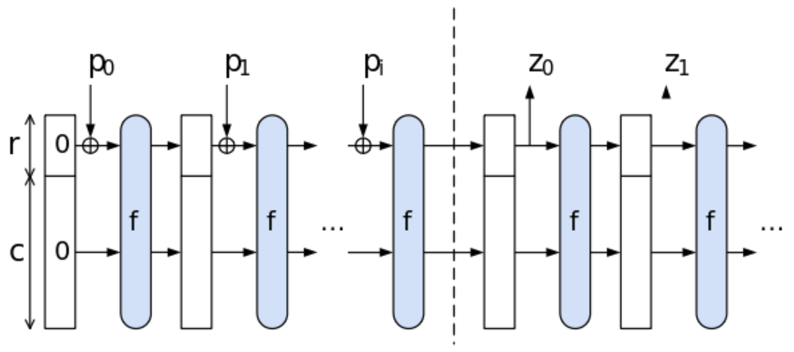


Figure 11: Sponge Function Construction

```
INPUT : Message $M$
1  $P \leftarrow M \,|\, 10^*1$ such that $|P|$ is a multiple of 576
2  $s \leftarrow 0^{1600}$
3  $n \leftarrow$ number of blocks in $P$
4  FOR $i = 0, \ldots, n$ DO
5      $s \leftarrow s \oplus (P_i \,|\, 0^{1024})$
6      $s \leftarrow f(s)$
7  END
8  $Z \leftarrow \lfloor s \rfloor_{576}$
9  WHILE $|Z| < 512$ DO
       //Squeezing isn't necessary for 512 digest
10     $s \leftarrow f(s)$
11     $Z \leftarrow Z \,|\, \lfloor s \rfloor_{576}$
12 END
OUTPUT: $\lfloor Z \rfloor_{512}$
```

10.0.2. Operation

- 1600-bit state is represented as a $5 \times 5 \times 64$ three dimensional bit array
- State is oriented with an (x, y, z) layout
- $s[0, \ldots, 1599]$ can be converted to state a with :

$$a[x][y][z] = s[64(5y + x) + z]$$

- State: column major form (x, y, z) formatting: be *careful*: very "odd" indexing of rows/columns (see FIPS draft)
- See Figure 12

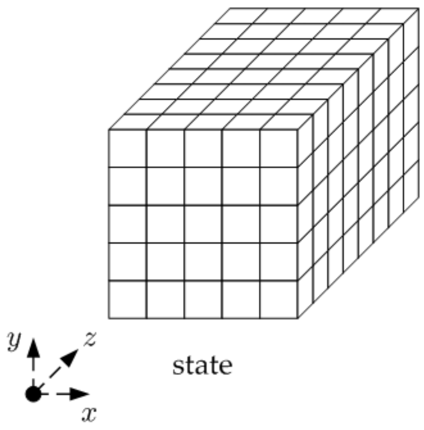


Figure 12: Keccak State Visualization

- Permutation is applied 24 times: rounds $i_r = 0, \ldots, 23$
- Each permutation is 5 operations:

$$R = \iota \circ \chi \circ \pi \circ \rho \circ \theta$$

- Note that composition is right-to-left
- Each column change is modulo 5, 64

Theta:

$$\theta : a[x][y][z] \leftarrow a[x][y][z] + \sum_{j=0}^{4} a[x-1][j][z] + \sum_{j=0}^{4} a[x+1][j][z-1]$$

- For all x, y, z
- Parity of adjacent/diagonal columns
- Addition is always exclusive-or

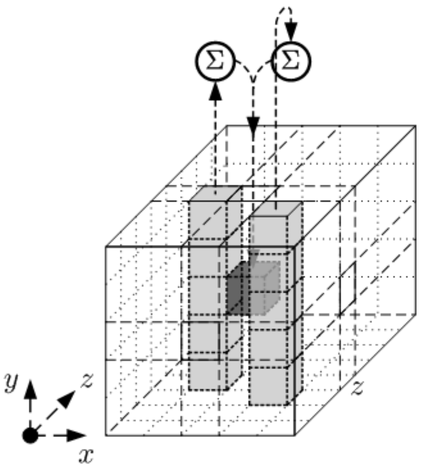


Figure 13: Keccak Theta Operation Visualization

Rho:

$$\rho : a[x][y][z] \leftarrow a[x][y][z - (t+1)(t+2)/2]$$

- Represents a cyclic rotation of the 25 lanes by a different constant for each
- The offset represents triangular numbers:

$$\binom{t+2}{2} = \frac{(t+1)(t+2)}{2}$$

for $0 \leq t \leq 23$ (for $t = -1$, $(x, y) = (0, 0)$ and there is 0 offset)
- Indices are determined by computing a matrix power with respect to t:

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix}^t \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$$

	$x = 3$	$x = 4$	$x = 0$	$x = 1$	$x = 2$
$y = 2$	25	39	3	10	43
$y = 1$	55	20	36	44	6
$y = 0$	28	27	0	1	62
$y = 4$	56	14	18	2	61
$y = 3$	21	8	41	45	15

Table 6: Rho offsets per the reference

- With entries modulo-5
- In tabular form, see Table 6, code form below

```
int rhoOffsets = {
  {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}}
```

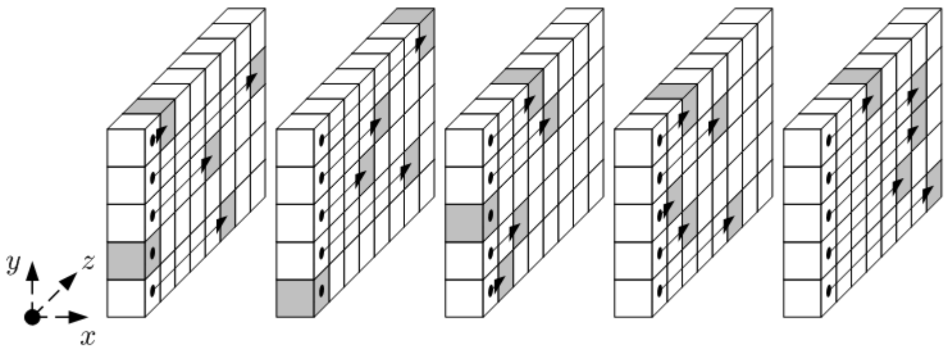


Figure 14: Keccak Rho Operation Visualization

Pi

$$\pi : a[x][y] \leftarrow a[x'][y']$$

- Various rotations of entire "lanes" computed with the same matrix:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x' \\ y' \end{pmatrix}$$

- Simplification:

$$x = y'$$
$$y = 2x' + 3y'$$

- Example: $(x', y') = (2, 3)$, then

$$(x, y) = (y', 2x' + 3y') = (3, 13) = (3, 3)$$

- So the lane at $(2,3)$ moves to $(3,3)$ (the bottom right-most lane moves to the bottom left-most lane)

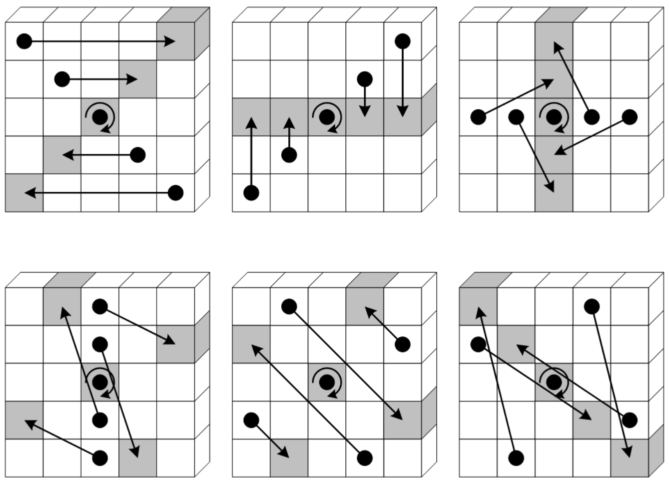


Figure 15: Keccak Pi Operation Visualization

Chi

$$\chi : a[x] \leftarrow a[x] + (a[x+1]+1)a[x+2]$$

- Only non-linear function in the permutation
- Alternatively:

$$a[x] \leftarrow a[x] \oplus (\neg a[x+1] \wedge a[x+2])$$

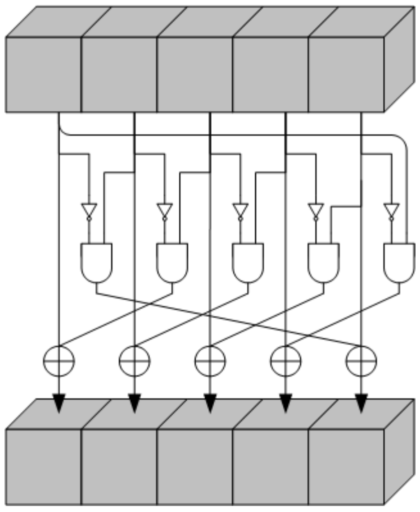


Figure 16: Keccak Chi Operation Visualization

Iota

$$\iota : a[0][0] \leftarrow a[0][0] + RC[r]$$

- *RC* is a round constant, 24 different ones
- Generated by a Linear-Feedback Shift Register
- Each constant is a sparse 64-bit word and is applied to $a[0][0]$ only

```
0x0000000000000001
0x0000000000008082
0x800000000000808A,
0x8000000080008000,
0x000000000000808B,
0x0000000080000001,
0x8000000080008081,
0x8000000000008009,
0x000000000000008A,
0x0000000000000088,
0x0000000080008009,
0x000000008000000A,
0x000000008000808B,
0x800000000000008B,
0x8000000000008089,
0x8000000000008003,
0x8000000000008002,
0x8000000000000080,
0x000000000000800A,
0x800000008000000A,
0x8000000080008081,
0x8000000000008080,
0x0000000080000001,
0x8000000080008008
```

11. Message Authentication Codes

- A secure hash can be used to create a Hash-based Message Authentication Code (HMAC)
- RFC 2104, FIPS 198
- HMAC-MD5, HMAC-SHA1 used in TLS/IPsec
- A secret key K is generated (and shared between two parties or one party if used for storage)
- Then MAC is:

$$HMAC(K, m) = H\big((K \oplus opad) \,\big|\, H((K \oplus ipad) \,\big|\, m)\big)$$

- opad/ipad is outer and inner padding:

$$opad = \texttt{0x5c5c5c} \ldots \texttt{5c5c}$$

$$ipad = \texttt{0x363636}\ldots\texttt{3636}$$

- Outer application of the hash function masks intermediate result (protects against length-extension attack which is possible if a simple prepending of the key is done)
- Keccak is sponge-based and can be directly (prepending a key to the message) used to generate a MAC
- Application: HMACs provide authentication and integrity over an open channel (HMAC need not be protected as long as key is not compromised) without the need for secrecy
- Example: Application Keys (a shared secret) for Open Web Service APIs–service is interested in authentic requests tied to a particular user so they can throttle, track, and charge; but the information itself is not sensitive (Twitter, Google Maps, etc.)
- Another application: as a pepper to your salt and hash (must then be kept secret)

CBC-Based MACs:

- Recall Code Block Chaining mode:

$$y_0 = \mathsf{IV}$$

$$y_i = e_K(y_{i-1} \oplus x_i)$$

- Using any endomorphic cipher system, we can generate a MAC
- Confer Figure

```
  INPUT  : $x = x_1 \mid x_2 \mid \cdots \mid x_n, K$
1 $y_0 = \mathsf{IV} = 00\cdots00$
2 FOR $i = 1, \ldots, n$ DO
3   $y_i \leftarrow e_K(y_{i-1} \oplus x_i)$
4 END
  OUTPUT: $y_n$
```

12. Authentication & Password Management

- Privileged services require authentication: proof of identity
- Username/Passwords are most common, but not the only way:
  - Physical: RFID cards, crypto USB devices, etc.
  - Biometric: fingerprints, retina scans, DNA
  - Electronic: phone numbers (2-factor authentication), email, etc.
- Password "strength", rules and policies: longer, more complex is better (higher entropy)

- Application password storage (services, databases, etc.)
  - Something, somewhere must be stored in order to *verify* claims of identity
  - Should never store passwords plaintext
  - Hashed passwords are best
  - User provides password
  - Server hashes the claimed password and compares to the stored hash
  - If the same, access granted
- Authentication vs Authorization; RBAC (Role-based Access Control)
- Authentication lifecycle: per request? Per Session?

12.1. Doing it Right

- Once a hash has been compromised, it is a matter of time until a pre image is found
- Want to slow attacks: want to use *inefficient* hashes (or force hashes to be slow)
- `http://security.stackexchange.com/questions/211/how-to-securely-hash-passwords/31846#31846`

Bcrypt

- Uses the Blowfish cipher (64 bit blocks, variable key size $[32, 448]$, 16 round Feistel function)
- Slow with rekeying (requires 4kb key schedule generation)
- Iterations must be a power of 2
- Used in OpenBSD
- Forces more memory accesses (rather than CPU-only)
- Input limited to 51 bytes (characters)
- Output is fixed to 192 bits
- Move to 2-fish (or 3-fish) recommended
- Some links: `http://codahale.com/how-to-safely-store-a-password/`

Password-Based Key Derivation Function 2 (PBKDF2)

- RFC 2898 `http://tools.ietf.org/html/rfc2898` (2000)
- Built-in salt (recommended at least 64 bits)
- Variable number of iterations (1000 minimum recommended in 2000)
- Number of iterations can be increased over time as computers become faster
- Allows a variable output length
- Allows "drop-in" pseudorandom function (Hash, HMAC, block cipher, etc.)
- Example: WPA2 (wireless security) uses PBKDF2 (HMAC-SHA1, passphrase, ssid (network name), 4096, 256)

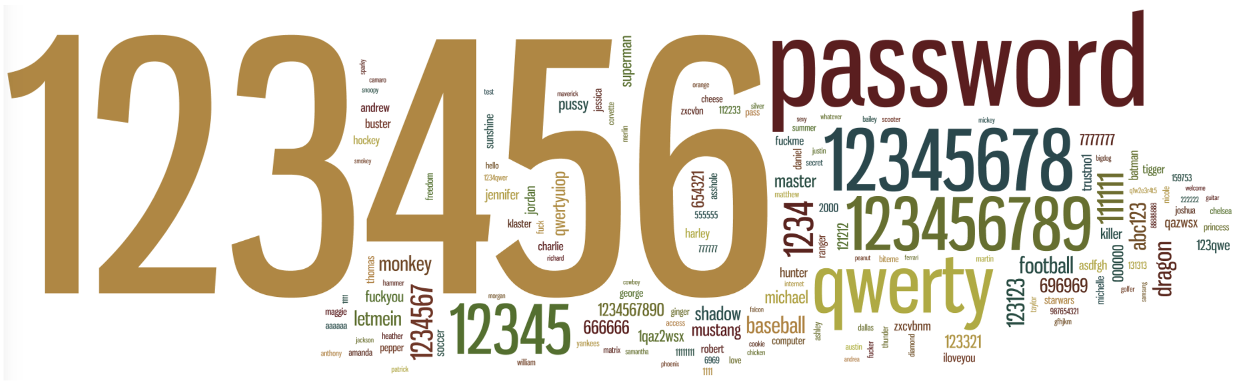


Figure 17: 1000 most common passwords, originally posted at http://www.reddit.com/r/dataisbeautiful/comments/2vdhb6/a_word_cloud_of_the_top_1000_most_common/

- Small Java implementation and integration with JBoss: http://www.rtner.de/software/PBKDF2.html

Other algorithms/tools:

- scrypt http://en.wikipedia.org/wiki/Scrypt

Salt & Peppers

- A *salt* s is an extra string appended (or in some way added) to a password then the stored hash is $h(p + s)$
- Server is responsible for generating and maintaining (user is unaware)
- Salt must be stored
- Salt need not be secret (not a key)
- Salt should be unique to each user
- A *pepper* is a secret key used in addition to a password and salt
- Both are intended to thwart the use of *rainbow tables*
- 10 million passwords from various sites: https://xato.net/passwords/ten-million-passwords [2]

12.2. Centralized Authentication Services & Protocols

- Single-Sign On (SSO) authorities: let someone else do it!
- Outsourcing to another as IdP (Identity Provider)
- Basic outline:
  - User sends request for privileged resources to server S
  - Server responds with a redirect to sign-on server
  - User provides credentials, sign-on server authenticates

  - Sign-on server redirects user back to S along with a *session* key
  - Server S verifies session key with sign-on server
  - Each subsequent request/response the session key is transmitted/checked until expires or invalidated
  - Note: sign out with server S does not necessarily affect session key (which sign-on server manages)
- NSTIC (National Strategy for Trusted Identities in Cyberspace), part of the NIST
  - "Individuals and organizations utilize secure, efficient, easy-to-use, and interoperable identity solutions to access online services in a manner that promotes confidence, privacy, choice, and innovation."
  - `http://www.nist.gov/nstic/`
  - `http://searchenginewatch.com/article/2232626/Google-as-Your-Identity-Provide`
- Google login using OAuth2: `https://developers.google.com/accounts/docs/OAuth2Login`
- Central Authentication Service
  - Home: `https://wiki.jasig.org/display/CAS/Home`
  - UNL info: `http://idm.unl.edu/using-cas-unl`
  - Java client: `https://wiki.jasig.org/display/CASC/CAS+Client+for+Java+3.1`
- Shibboleth (Hebrew word used to differentiate groups)
- LDAP (Lightweight Directory Access Protocol) (IETF)
- Kerberos (network authentication protocol) (MIT, 1993)
- Active Directory (Microsoft, Windows domain network only, uses LDAP and Kerberos)
- SAML (Security Assertion Markup Language) – XML standards for exchanging authentication and authorization data (2001)
- OpenID (open, decentralized protocol among cooperating partners–AOL, Flickr, MS, WordPress, Yahoo!, GitHub, etc.)
- OAuth
  - 2006, Twitter
  - 1.0 (RFC 5849), 2.0 (RFC 6749, 2012); completely based on SSL
  - Highly criticized (framework, not protocol), but widely adopted
  - Not always interoperable
- Vs. in-application authentication (extra level of vulnerability)
- The state-of-the-art: see Figure 18

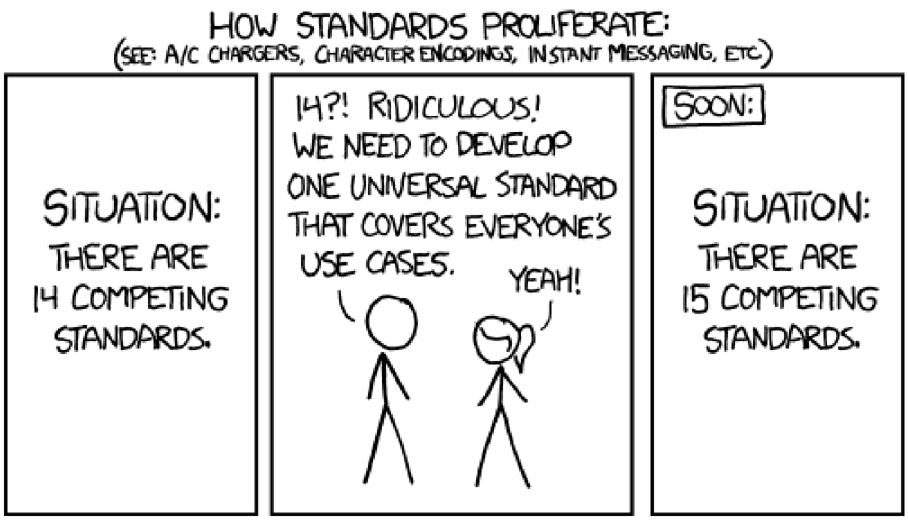

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

Figure 18: An accurate depiction of the SSO landscape

12.2.1. Breaking Passwords

- Dictionary Attacks
- Brute force attacks
- Rainbow tables
  - Common passwords, phrases, and variations are stored
  - Hashes are precomputed and sorted
  - A binary search can be performed on the hash for a reverse look-up
  - Salts and peppers thwart rainbow tables because for each salt, a new table has to be generated; if pepper is not compromised, a rainbow table cannot be generated at all
- `http://null-byte.wonderhowto.com/how-to/hack-like-pro-crack-user-passwords-linux`
- `http://arstechnica.com/security/2012/08/passwords-under-assault/`
- Tools: `http://md5decoder.org/`
- Tool: John the Ripper, `http://www.openwall.com/john/`

Part IV. Public Key Cryptography

Introduction/Motivation:

- Block ciphers provide security but require *secure prior communication* to exchange the key
- If the key is compromised, a new key must be exchanged
- Strangers cannot communicate privately
- Public key cryptography solves this problem by allowing secure communication without prior secure communication

Basics:

- A public-key crypto system consists of a key with two parts: a *public* key and a *private* key
- Anyone can encrypt a message using the public key
- (Ideally) decryption can only be performed using the private key
- If both parties setup their own system, two-way communication is possible
- Full communication can be done using the public-key system, but usually the public key system is used to exchange symmetric keys to a block cipher (AES)
- It is possible, but should be *infeasible* to derive the private key from the public key
- First publicly proposed by Diffie-Hellman (and Merkle) (1976) (British Clifford Cocks found it in 1973 but it was classified until 1997)

13. RSA

Introduction & History

- Invented by Ronald Rivest, Adi Shamir, and Leonard Adleman (1977)
- ANSI X9.31, PKCS#1 (Public Key Cryptography Standards, RSA Labs)
- RSA Factoring Challenge (1991 – 2007), `http://en.wikipedia.org/wiki/RSA_numbers` Largest break so far: RSA-768 (2009, took 2 years))

13.1. The System

- Let p, q be close primes (same number of bits)
- Let $n = pq$
- Let $P = C = \mathbb{Z}_n$
- Compute Euler's Totient: $\phi(n) = (p-1)(q-1)$ (since n is a product of two primes)

- Choose a such that $\gcd(a, \phi(n)) = 1$ (thus a has an inverse)
- Compute $b = a^{-1} \bmod \phi(n)$
- Define a key: $K = (n, p, q, a, b)$
- Encryption of $x \in \mathbb{Z}_n$ is done by:

$$e_K(x) = x^a \bmod n$$

- Decryption of $y \in \mathbb{Z}_n$ is done by:

$$d_K(y) = y^b \bmod n$$

- (n, a) is the *public key*
- (p, q, b) is the *private key*

13.1.1. Necessary Tools

- A way to represent and manipulate large numbers (use a library)
- A way to generate (and verify) large primes (that conform to certain expectations to prevent easy factorization)
- A way to compute multiplicative inverses (Extended Euclidean Algorithm)
- A way to efficiently compute (modular) exponentiation

13.1.2. Repeated Squaring/Fast Binary Exponentiation

- Say we want to compute

$$\alpha^n \bmod m$$

- We *could* simply compute

$$\underbrace{\alpha \cdot \alpha \cdot \cdots \cdot \alpha}_{n \text{ times}}$$

- Requires $O(n)$ (*exponential*) operations
- Consider:

$$n = 1705566317424998551470821443128983228242358667740233493$$

  - $\log_{10}(n) \approx 54.23$, so about 55 digits
  - $\log_2(n) \approx 180.15$, so 181 bits
  - Straight-forward multiplication: $n - 1$ multiplications
  - At 1 trillion multiplications per second: 5.4046×10^{34} years
  - *54 Decillion* years (sun only has about 5 billion, 5×10^9 years left)
  - Good luck

- We can do better: perform a *repeated squaring* of the base,

$$\alpha, \alpha^2, \alpha^4, \alpha^8, \ldots$$

requiring $\log(n)$ operations instead.
- Formally, we note that

$$\begin{aligned} \alpha^n &= \alpha^{b_k 2^k + b_{k-1} 2^{k-1} + \cdots + b_1 2 + b_0} \\ &= \alpha^{b_k 2^k} \times \alpha^{b_{k-1} 2^{k-1}} \times \cdots \times \alpha^{2b_1} \times \alpha^{b_0} \end{aligned}$$

- So we can compute α^n by evaluating each term as

$$\alpha^{b_i 2^i} = \begin{cases} \alpha^{2^i} & \text{if } b_i = 1 \\ 1 & \text{if } b_i = 0 \end{cases}$$

- We can save computation because we can simply square previous values:

$$\alpha^{2^i} = (\alpha^{2^{i-1}})^2$$

```
INPUT  : Integers $\alpha, m$ and $n = (b_k b_{k-1} \ldots b_1 b_0)$ in binary.
OUTPUT: $\alpha^n \bmod m$
term = $\alpha$
IF ($b_0 = 1$) THEN
    product = $\alpha$
END
ELSE
    product = 1
END
FOR $i = 1 \ldots k$ DO
    term = term $\times$ term mod $m$
    IF ($b_i = 1$) THEN
        product = product $\times$ term mod $m$
    END
END
output product
```

Example: Compute

$$12^{26} \bmod 17$$

Thus

$$12^{26} \bmod 17 = 9$$

1	1	0	1	0	$= (26)_2$
4	3	2	1	-	i
1	16	13	8	12	term
9	9	8	8	1	product

13.1.3. Small example

Consider the message: `Hello World!` which can be encoded in our printable character map as follows:

`38-65-72-72-75-1-53-75-78-72-64-2`

Choose two primes:

$$p = 53, q = 59$$

Compute:

$$n = 3127, \phi(n) = 3016$$

We can choose any a relatively prime to $3,016 = 2^3 \cdot 13 \cdot 29$, say:

$$a = 17$$

Then

$$b = a^{-1} = 2129$$

Encryption:

$$e_K(x) = x^{17} \bmod 3127$$

For $x = \texttt{38}$:

$$e_K(\texttt{38}) = 38^{17} \bmod 3127 = 1368$$

Observe: $17_{10} = 10001_2$ (for easy exponentiation)

In all:

$$e_K(x) = \texttt{1368-1571-986-986-2558-1-848-2558-197-986-2160-2865}$$

Decryption example: consider the message

`2588-1600-3101-369-3101-161-2865`

Observe: $2129_{10} = 0b100001010001$

$$d_K(\texttt{2588}) = 2588^{2129} \bmod 3127$$

In all:

`49-68-61-86-61-73-2`

Which is decoded as: `Shazam!`

13.2. Foundations

Theorem 6 (Chinese Remainder Theorem)**.** Named for Sun Tzu (not the same "Art of War" Sun Tzu; 3rd-5th century).

「韓信點兵」、「孫子定理」、「鬼谷算」、「隔墻算」、「剪管術」、「秦王暗點兵」、「物不知數」

Figure 19: Chinese Name Variations on the CRT

Let $n_1, n_2, \ldots, n_k$ be pairwise coprime positive integers. Then for any sequence $a_1, a_2, \ldots, a_k$ there exists a unique integer x solving the system of congruences:

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

Example:

$$\begin{aligned} x &\equiv 2 \pmod 4 \\ x &\equiv 1 \pmod 5 \\ x &\equiv 6 \pmod 7 \\ x &\equiv 3 \pmod 9 \end{aligned}$$

Process:

1. Compute $m = n_1 n_2 \cdots n_k$.
2. For each $i = 1, 2, \ldots, k$ compute
$$M_i = \frac{m}{n_i}$$
3. For each $i = 1, 2, \ldots, k$ compute the inverse, y_i of $M_i \pmod{n_i}$ (these are *guaranteed* to exist)
4. The solution is the sum
$$x = \sum_{i=1}^{n} a_i M_i y_i \bmod m$$

First, $m = 4 \cdot 5 \cdot 7 \cdot 9 = 1260$ and

$$\begin{aligned} M_1 &= \tfrac{1260}{4} = 315 \\ M_2 &= \tfrac{1260}{5} = 252 \\ M_3 &= \tfrac{1260}{7} = 180 \\ M_4 &= \tfrac{1260}{9} = 140 \end{aligned}$$

The inverses of each of these is $y_1 = 3, y_2 = 3, y_3 = 3$ and $y_4 = 2$. Therefore, the unique solution is

$$\begin{aligned} x &= a_1 M_1 y_1 + a_2 M_2 y_2 + a_3 M_3 y_3 + a_4 M_4 y_4 \\ &= 2 \cdot 315 \cdot 3 + 1 \cdot 252 \cdot 3 + 6 \cdot 180 \cdot 3 + 3 \cdot 140 \cdot 2 \\ &= 6726 \bmod 1260 = 426 \end{aligned}$$

13.2.1. Fermat's Little Theorem

- We prove that RSA is correct using Fermat's Little Theorem
- Correct in the sense that $d_K(e_K(x)) = x$
- Due to Pierre de Fermat (1640)
- A variety of proofs exists

Theorem 7 (Fermat's Little Theorem)**.** Let p be a prime such that p does not divide $a \in \mathbb{Z}$, then

$$a^{p-1} \equiv 1 \pmod{p}$$

A simple proof:

- Write out the sequence (of $p-1$ numbers);

$$a, 2a, 3a, \ldots, (p-1)a$$

- Reducing each number by modulo p is just (some) permutation of the sequence

$$1, 2, 3, \ldots, (p-1)$$

- Now take the product of each sequence:

$$a \cdot 2a \cdot 3a \cdots (p-1)a = a^{p-1}(p-1)!$$

  and

$$1 \cdot 2 \cdot 3 \cdots (p-1) = (p-1)!$$

- Example: $p = 7, a = 10$, the sequence

$$10, 20, 30, 40, 50, 60$$

  modulo y is

$$3, 6, 2, 5, 1, 4$$

  which is a permutation of $1, \ldots, 6$

- Since the second is simply a reduction modulo p of the first, they should be congruent to each other:

$$a^{p-1}(p-1)! \equiv (p-1)! \pmod{p}$$

- Simplifying we get

$$a^{p-1} \equiv 1 \pmod{p}$$

Proof using Fermat's little theorem:

- Let p, q be primes with $n = pq$ and $\phi(n) = (p-1)(q-1)$
- Let a, b be inverses of each other modulo $\phi(n)$

- We want to show that for all $x \in \mathbb{Z}_n$,

$$x^{a^b} = x^{ab} \pmod{n} = x$$

- Observe that

$$ab \equiv 1 \pmod{\phi(n)} = ab \equiv 1 \pmod{(p-1)(q-1)}$$

- So by definition of modulo, we can write

$$ab - 1 = k(p-1)(q-1) \qquad (*)$$

  for some $k \in \mathbb{Z}$
- Want to show that x^{ab} and x are congruent modulo n, but as a result of the Chinese Remainder Theorem, it suffices to show that

$$x^{ab} \equiv x \pmod{p}$$

  and

$$x^{ab} \equiv x \pmod{q}$$

- Case 1: $x \equiv 0 \pmod{p}$
  - That means that x is a multiple of p: $x = kp$
  - Thus

$$x^\ell = (kp)^\ell = (k^\ell p^{\ell-1})p \equiv 0 \pmod{p}$$

  - In particular, for $\ell = ab$, we have the result:

$$x^{ab} \equiv 0 \pmod{p}$$

  - For q likewise
- Case 2: $x \not\equiv 0 \pmod{p}$ (i.e. p does not divide x, observe:

$$\begin{array}{rcll} x^{ab} & = & x^{ab-1}x & \\ & = & x^{k(p-1)(q-1)}x & \text{by } (*) \text{ above} \\ & = & x^{p-1^{k(q-1)}}x & \\ & = & 1^{k(q-1)}x \pmod{p} & \text{by Fermat's Little Theorem} \\ & = & x \pmod{p} & \end{array}$$

- The application of Fermat's Little Theorem is valid since p is a prime and p does not divide x (the assumption that started this case)

13.2.2. Euler's Theorem

Theorem 8 (Euler's Theorem)**.** Let n and a be integers such that $\gcd(a, n) = 1$ (i.e. they are coprime). Then

$$a^{\phi(n)} \equiv 1 \pmod{n}$$

- Due to Leonhard Euler (1736)
- Proof sketch: Lagrange's theorem states that the order of any subgroup of a finite group also divides the order of the entire group (here, $\phi(n)$); then the powers of a form a subgroup of order k; k must divide $\phi(n)$ (i.e. $\phi(n) = mk$), which can then be used to show:

  $$a^{\phi(n)} = a^{mk} = a^{k^m} = 1^m \equiv 1 \pmod{n}$$

  Where $a^k \equiv 1 \pmod{n}$ as all powers of a in that subgroup are coprime to n

Proof using Euler's Theorem (also in Stinson):

- Same setup
- Again we note that

  $$ab = 1 + k(p-1)(q-1) = 1 + k\phi(n)$$

- Assume that $\gcd(x, n) = 1$, then

  $$\begin{array}{rcll} x^{ab} & = & x^{1+k\phi(n)}x & \\ & = & x(x^{\phi(n)})^k & \\ & = & x(1)^k & \text{by Euler's Theorem} \\ & = & x \pmod{n} & \end{array}$$

- If $\gcd(x, n) \neq 1$ then we need a different argument
- If that's the case, then $x \equiv 0 \pmod{p}$ or $x \equiv 0 \pmod{q}$
- In either case, we can apply Case 1 from the argument above (Fermat-based proof)

13.3. Primality Testing

Problem 1 (Primality Testing)**.**
Given: An integer $x \in \mathbb{Z}$
Output: True if x is prime, false otherwise

- A *decision* problem
- Does not require us to *factor* x, just determine if its prime or not
- It can be shown that primality testing reduces to factoring
- Variety of probabilistic, randomized, and deterministic algorithms that rely on some testing some equivalent condition.

Common Tool: Jacobi Symbols

- Generalization of Legendre Symbols
- For any $a \in \mathbb{Z}$ and any odd positive integer n,

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1}\left(\frac{a}{p_2}\right)^{\alpha_2}\cdots\left(\frac{a}{p_k}\right)^{\alpha_k}$$

  where $n = p_1^{\alpha_1}p_2^{\alpha_2}\cdots p_k^{\alpha_k}$
- Each term with a prime is the Legendre Symbol:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a \equiv 0 \pmod{p} \\ +1 & \text{if } a \not\equiv 0 \pmod{p} \text{ and for some } x, a \equiv x^2 \pmod{p} \\ -1 & \text{if there is no such } x \end{cases}$$

- Examples: $\left(\frac{3}{9}\right) = 0$, $\left(\frac{3}{5}\right) = -1$, $\left(\frac{4}{7}\right) = +1$ for $x = 2$
- Though defined in terms of a prime factorization, it is efficient to compute in $O(\log a \log n)$ time by applying several known rules and reducing the top/bottom in a manner similar to Euclid's Algorithm
- The second criterion defines what is called a *quadratic residue*: q is a quadratic residue modulo n if there exists an x such that

$$x^2 \equiv q \pmod{n}$$

Types of Algorithms:

- Monte Carlo algorithms are randomized algorithms that run in polynomial time but may produce incorrect results with some (small) probability
- One-sided error: if all yes answers are always correct, all no answers are correct with probability $\geq 1/2$ ($\mathsf{RP}, \mathsf{coRP}$)
- Las Vegas algorithms are randomized algorithms that always produce correct answers (yes and no) however, their running time may, with some probability, be exponential. They run in *expected* polynomial time ($\mathsf{ZPP} = \mathsf{RP} \cap \mathsf{coRP}$)

13.3.1. Fermat's Test

- Corollary to Fermat's Little Theorem: if p is prime, then for all $a, 1 \leq a < p$,

$$a^{p-1} \equiv 1 \pmod{p}$$

- Contrapositive: if, for some a, $a^{p-1} \not\equiv 1 \pmod{p}$, then p is composite
- Pick a random $a \in [1, p-1]$, compute a^{p-1}
  - If the congruence does not hold, then p is composite (a "Fermat Witness")
  - If the congruence does hold, p may still be composite (a Fermat Pseudoprime– $\forall a a^{p-1} \not\equiv 1 \pmod{p}$ but p is still composite) or may probably be prime

- Cannot feasibly test all a (exponential)
- Even if we test all a, there are some numbers (Carmichael numbers, Fermat Pseudoprimes) that will fail the test for all a
- Smallest Fermat Pseudoprime: $561 = 3 \cdot 11 \cdot 17$; for all a (such that $\gcd(a, 561) = 1$, $a^{560} \equiv 1 \pmod{561}$)
- No nice analysis of the distribution of Fermat Pseudoprimes; even ignoring those, no nice analysis on the probability of failure

13.3.2. Solovay-Strassen

- Robert Solovay, Volker Strassen (1978)
- Randomized, Monte-Carlo primality test
- Based on Euler's Criterion: Let p be an odd prime and a coprime to p, then
$$a^{\frac{p-1}{2}} \equiv \begin{cases} 1 \pmod{p} & \text{if there is an integer } x \text{ such that } a \equiv x^2 \pmod{p} \\ -1 \pmod{p} & \text{if there is no such integer } x \end{cases}$$
- Restated:
$$a^{\frac{p-1}{2}} \equiv \left(\frac{a}{p}\right) \pmod{p}$$
- If n is prime, then the above holds for all values of a relatively prime to n
- If the congruence does not hold for any a then n is composite (but we don't get any info about its factors)
- Problem: for any particular a, it can still hold, but n may still be composite (called an "Euler Liar")
- Suggests a randomized algorithm: try random values of a to find a composite witness ("Euler Witness")
- Additional Fact: For every composite n, at least half of all a are Euler witnesses (less than half are Euler Liars)
- Thus the algorithm is an RP algorithm: if it outputs composite, n is composite (with 100% probability) ; if it outputs probably prime, it is prime with probability $\geq 1/2$ (it is composite with probability $< \frac{1}{2}$)
- Error probability can be decreased arbitrarily by running the algorithm k times: if it ever outputs composite, it is composite; if it always outputs probably prime, then its error is at most:
$$\frac{1}{2} \cdot \frac{1}{2} \cdots \frac{1}{2} = \left(\frac{1}{2}\right)^k = 2^{-k}$$
- In general, the numbers with almost 50% Euler liars are rare. On average, the expected number of liars for n is actually
$$\exp\left(-(1 + o(1))\frac{\log n \, \log\log\log n}{\log\log n}\right)$$

- Diminishes for larger n: for $n \approx 2^{1024}$:

$$\approx 1.85365 \times 10^{-148}$$

```
INPUT   : $n \in \mathbb{Z}$ such that $n \neq p^k$ for some prime $p$
//prime powers can be easily detected deterministically
1 $a \leftarrow$ random integer in $[2, n-1]$
2 $x \leftarrow \left(\frac{a}{n}\right)$
3 IF $a^{(n-1)/2} \not\equiv x \ (mod\ n)$ THEN
4     OUTPUT composite
5 ELSE
6     OUTPUT probably prime
7 END
```

Algorithm 6: Solovay-Strassen Primality Testing

Example:

- Let $n = 231 = 3 \cdot 7 \cdot 11$
- Consider $a = 15$:
$$\left(\frac{15}{231}\right) = 0$$
So we know that Euler's test doesn't apply, *but* this is even better: we have a factor of n:
$$\gcd(15, 231) = 3$$
- Consider $a = 64$:
$$\left(\frac{64}{231}\right) = 1$$
and
$$64^{115} \bmod 231 = 1$$
The algorithm would output "probably prime" (which is wrong), thus 64 is an Euler Liar
- Consider $a = 20$:
$$\left(\frac{20}{231}\right) = 1$$
and
$$20^{115} \bmod 231 = 188$$
The algorithm would output "composite"
- In fact, there are only 9 Euler Liars: $41, 62, 64, 83, 148, 167, 169, 190, 230$

13.3.3. Miller-Rabin

- Original formulation due to Gary Miller (1976), probabilistic version by Michael Rabin (1980)
- Another variation on Fermat's Theorem: Let $n-1 = 2^s \cdot d$, then either

  $$a^d \equiv 1 \pmod{n}$$

  or

  $$a^{2^r \cdot d} \equiv -1 \pmod{n}$$

  for some $0 \leq r \leq s-1$
- Contrapositive: if, for some a, $2 \leq a < n-1$ and $\forall r 0 \leq r \leq s-1$ (with $n-1 = 2^s \cdot d$),

  $$a^d \not\equiv 1 \pmod{n} \text{ and } a^{2^r d} \not\equiv -1 \pmod{n}$$

  then n is composite
- May still be the case that for all a, the congruences hold, but it is still composite: (same Euler Liars)
- Improvement in that false positives: liars are much rarer w.r.t. this congruence
- It can be shown that the probability of failure is at most 1/4
- Thus, repeated applications result in error less than 4^{-k}

```
   INPUT  : $n > 1$
1  $2^s \cdot d \leftarrow n-1$
2  $a \leftarrow$ random value in $[2, n-1]$
3  $b \leftarrow a^d \pmod{n}$
4  IF $b \equiv 1 \ (mod\ n)$ THEN
5      output *probably prime*
6  END
7  FOR $i = 0, \ldots, s-1$ DO
8      IF $b \equiv -1 \ (mod\ n)$ THEN
9          output *probably prime*
10     ELSE
11         $b \leftarrow b^2 \pmod{n}$
12     END
13 END
14 output *composite*
```

Algorithm 7: Miller-Rabin Probabilistic Primality Test

Proof of Correctness (Stinson, 3rd Ed, p 186–187)

- By way of contradiction, suppose that the algorithm provides a false positive (answers composite when n is prime)

a	$a^{(n-1)/2} \pmod n$ $a^{115} \pmod{231}$	$\left(\frac{a}{n}\right)$ $\left(\frac{a}{231}\right)$	$a^{(n-1)/2} \overset{?}{\equiv} \left(\frac{a}{n}\right) \pmod n$ $a^{115} \overset{?}{\equiv} \left(\frac{a}{231}\right) \pmod{231}$	Result
2	$2^{115} \pmod{231} = 65$	$\left(\frac{2}{231}\right) = 1$	$2^{115} \overset{?}{\equiv} \left(\frac{2}{231}\right) \pmod{231}$ $65 \not\equiv 1 \pmod{231}$	composite (definite)
3	$3^{115} \pmod{231} = 45$	$\left(\frac{3}{231}\right) = 0$	$3^{115} \overset{?}{\equiv} \left(\frac{3}{231}\right) \pmod{231}$ $45 \not\equiv 0 \pmod{231}$	composite (definite)
4	$4^{115} \pmod{231} = 67$	$\left(\frac{4}{231}\right) = 1$	$4^{115} \overset{?}{\equiv} \left(\frac{4}{231}\right) \pmod{231}$ $67 \not\equiv 1 \pmod{231}$	composite (definite)
5	$5^{115} \pmod{231} = 89$	$\left(\frac{5}{231}\right) = 1$	$5^{115} \overset{?}{\equiv} \left(\frac{5}{231}\right) \pmod{231}$ $89 \not\equiv 1 \pmod{231}$	composite (definite)
6	$6^{115} \pmod{231} = 153$	$\left(\frac{6}{231}\right) = 0$	$6^{115} \overset{?}{\equiv} \left(\frac{6}{231}\right) \pmod{231}$ $153 \not\equiv 0 \pmod{231}$	composite (definite)
7	$7^{115} \pmod{231} = 175$	$\left(\frac{7}{231}\right) = 0$	$7^{115} \overset{?}{\equiv} \left(\frac{7}{231}\right) \pmod{231}$ $175 \not\equiv 0 \pmod{231}$	composite (definite)
8	$8^{115} \pmod{231} = 197$	$\left(\frac{8}{231}\right) = 1$	$8^{115} \overset{?}{\equiv} \left(\frac{8}{231}\right) \pmod{231}$ $197 \not\equiv 1 \pmod{231}$	composite (definite)
9	$9^{115} \pmod{231} = 177$	$\left(\frac{9}{231}\right) = 0$	$9^{115} \overset{?}{\equiv} \left(\frac{9}{231}\right) \pmod{231}$ $177 \not\equiv 0 \pmod{231}$	composite (definite)
10	$10^{115} \pmod{231} = 10$	$\left(\frac{10}{231}\right) = 1$	$10^{115} \overset{?}{\equiv} \left(\frac{10}{231}\right) \pmod{231}$ $10 \not\equiv 1 \pmod{231}$	composite (definite)
11	$11^{115} \pmod{231} = 11$	$\left(\frac{11}{231}\right) = 0$	$11^{115} \overset{?}{\equiv} \left(\frac{11}{231}\right) \pmod{231}$ $11 \not\equiv 0 \pmod{231}$	composite (definite)
12	$12^{115} \pmod{231} = 12$	$\left(\frac{12}{231}\right) = 0$	$12^{115} \overset{?}{\equiv} \left(\frac{12}{231}\right) \pmod{231}$ $12 \not\equiv 0 \pmod{231}$	composite (definite)
13	$13^{115} \pmod{231} = 76$	$\left(\frac{13}{231}\right) = 1$	$13^{115} \overset{?}{\equiv} \left(\frac{13}{231}\right) \pmod{231}$ $76 \not\equiv 1 \pmod{231}$	composite (definite)
14	$14^{115} \pmod{231} = 56$	$\left(\frac{14}{231}\right) = 0$	$14^{115} \overset{?}{\equiv} \left(\frac{14}{231}\right) \pmod{231}$ $56 \not\equiv 0 \pmod{231}$	composite (definite)
15	$15^{115} \pmod{231} = 78$	$\left(\frac{15}{231}\right) = 0$	$15^{115} \overset{?}{\equiv} \left(\frac{15}{231}\right) \pmod{231}$ $78 \not\equiv 0 \pmod{231}$	composite (definite)
16	$16^{115} \pmod{231} = 100$	$\left(\frac{16}{231}\right) = 1$	$16^{115} \overset{?}{\equiv} \left(\frac{16}{231}\right) \pmod{231}$ $100 \not\equiv 1 \pmod{231}$	composite (definite)
17	$17^{115} \pmod{231} = 164$	$\left(\frac{17}{231}\right) = -1$	$17^{115} \overset{?}{\equiv} \left(\frac{17}{231}\right) \pmod{231}$ $164 \not\equiv -1 \pmod{231}$	composite (definite)
18	$18^{115} \pmod{231} = 186$	$\left(\frac{18}{231}\right) = 0$	$18^{115} \overset{?}{\equiv} \left(\frac{18}{231}\right) \pmod{231}$ $186 \not\equiv 0 \pmod{231}$	composite (definite)
19	$19^{115} \pmod{231} = 208$	$\left(\frac{19}{231}\right) = 1$	$19^{115} \overset{?}{\equiv} \left(\frac{19}{231}\right) \pmod{231}$ $208 \not\equiv 1 \pmod{231}$	composite (definite)
20	$20^{115} \pmod{231} = 188$	$\left(\frac{20}{231}\right) = 1$	$20^{115} \overset{?}{\equiv} \left(\frac{20}{231}\right) \pmod{231}$ $188 \not\equiv 1 \pmod{231}$	composite (definite)
21	$21^{115} \pmod{231} = 21$	$\left(\frac{21}{231}\right) = 0$	$21^{115} \overset{?}{\equiv} \left(\frac{21}{231}\right) \pmod{231}$ $21 \not\equiv 0 \pmod{231}$	composite (definite)
22	$22^{115} \pmod{231} = 22$	$\left(\frac{22}{231}\right) = 0$	$22^{115} \overset{?}{\equiv} \left(\frac{22}{231}\right) \pmod{231}$ $22 \not\equiv 0 \pmod{231}$	composite (definite)
23	$23^{115} \pmod{231} = 23$	$\left(\frac{23}{231}\right) = -1$	$23^{115} \overset{?}{\equiv} \left(\frac{23}{231}\right) \pmod{231}$ $23 \not\equiv -1 \pmod{231}$	composite (definite)
24	$24^{115} \pmod{231} = 87$	$\left(\frac{24}{231}\right) = 0$	$24^{115} \overset{?}{\equiv} \left(\frac{24}{231}\right) \pmod{231}$ $87 \not\equiv 0 \pmod{231}$	composite (definite)
25	$25^{115} \pmod{231} = 67$	$\left(\frac{25}{231}\right) = 1$	$25^{115} \overset{?}{\equiv} \left(\frac{25}{231}\right) \pmod{231}$ $67 \not\equiv 1 \pmod{231}$	composite (definite)

Table 7: Partial Solovay-Strassen Computation for $n = 231$, $a = 1, \ldots, 25$

- It must be that $a^d \not\equiv 1 \pmod{n}$ (line 4) $(*)$
- Line 7 loop tests values
$$a^d, a^{2d}, a^{2^2 d}, \ldots, a^{2^{s-1} d}$$
- None of them were satisfied, so
$$a^{2^i d} \not\equiv -1 \pmod{n}$$
for $0 \leq i \leq s-1$
- n is prime by assumption; applying Fermat's Little Theorem tells us ($2^s \cdot d = n-1$):
$$a^{2^s \cdot d} \equiv 1 \pmod{n}$$
- Note that
$$a^{2^s \cdot d} = \left(a^{2^{s-1} \cdot d}\right)^2$$
and so is a square root modulo 1
- The only possible roots of $x^2 \equiv 1 \pmod{n}$ are 1 and -1
- But from above, we know that
$$a^{2^{s-1} \cdot d} \not\equiv -1 \pmod{n}$$
so it must be the other case, and
$$a^{2^{s-1} \cdot d} \equiv 1 \pmod{n}$$
- Apply the same argument to $a^{2^{s-2} \cdot d}, a^{2^{s-3} \cdot d}$, etc. until
$$a^d \equiv 1 \pmod{n}$$
which contradicts with $(*)$ (line 4)

Variation: Deterministic Miller Test

- Original formulation indicated that only a small number of candidates, a needed to be tested, approximately
$$O(\log^2 n)$$
- Based on the Generalized Riemann Hypothesis (1859)
  - Distribution of primes is known to be about
$$\frac{n}{\ln n}$$
  - RH measures the *deviation* of this distribution
  - One of the \$1 million Millennium Problems

```
INPUT : $n > 1$
$2^s \cdot d \leftarrow n - 1$
FOR $a = 2, \ldots, \min\left\{n-1, \lfloor 2\ln^2 n \rfloor\right\}$ DO
    $isWitness \leftarrow true$
    FOR $r = 0, \ldots, s-1$ DO
        IF $\neg\big(a^d \not\equiv 1\ (mod\ n) \wedge a^{2^r \cdot d} \not\equiv -1\ (mod\ n)\big)$ THEN
            $isWitness \leftarrow false$
            break
        END
    END
    IF $isWitness$ THEN
        output composite
    END
END
output prime
```

Algorithm 8: Deterministic, Conditional Miller Primality Test

13.3.4. Lucas-Lehmer

- Lucas-Lehmer test only applies to Mersenne Primes: primes of the form $M_p = 2^p - 1$ for prime p (only 48 known as of 2014)
- Lucas test is more general, but requires that factors of $n-1$ are known (okay if $n-1$ has a lot of small factors using trial division)
- Test: let $n > 1$; if there exists $a, 1 < a < n$ such that

  $$a^{n-1} \equiv 1 \pmod{n}$$

  and for every prime factor q of $n-1$,

  $$a^{(n-1)/q} \not\equiv 1 \pmod{n}$$

  then n is prime. If no such a exists, then n is composite
- Randomized version: you select a at random
- Used (in some versions), along with Miller-Rabin in Java's `BigInteger` class (`isProbablePrime`)

13.3.5. Baillie-PSW

- Basic idea: apply a Miller-Rabin test (if composite, stop);
- Perform a Lucas test on n with particular parameters: let D be the first value in the sequence $5, -7, 9, -11, \ldots$ with Jacobi Symbol $\left(\frac{D}{n}\right)$ is -1; set $P = 1, Q = (1-D)/4$

- Lucas Test: given P, Q, set $D = P^2 - 4Q$; define Lucas sequences $U_k(P,Q), V_k(P,Q)$
- Lucas sequences are those that satisfy

$$x_n = Px_{n-1} - Qx_{n-2}$$

  A generalization of Fibonacci sequence
- Specifically: $U_0(P,Q) = 9, U_1(P,Q) = 1$,

$$U_n(P,Q) = P \cdot U_{n-1}(P,Q) - Q \cdot U_{n-2}(P,Q)$$

  and $V_0(P,Q) = 2, V_1(P,Q) = P$ and

$$V_n(P,Q) = P \cdot V_{n-1}(P,Q) - Q \cdot V_{n-2}(P,Q)$$

- Define $\delta(n) = n - \left(\frac{D}{n}\right)$
- If $U_{\delta(n)} \not\equiv 0 \pmod{n}$, then n is definitely composite
- If the congruence holds, n is probably prime
- Verified for all $n < 2^{64}$, other very strong evidence that Fermat and Lucas pseudo-primes are different (intersection is the empty set)
- Used, in some variation in many big number packages

13.3.6. AKS

- Agrawal-Kayal-Saxena (2001)
- First deterministic, polynomial algorithm (that did not rely on any unproven conjectures)
- Based on the following fact: n is prime if and only if

$$(x-a)^n \equiv (x^n - a) \pmod{n}$$

  for all a coprime to n ($\gcd(a,n) = 1$, x is a free variable; congruence is determined by the polynomial expansion)
- Note: in contrast this congruence is an equivalence (but still a generalization of Fermat's Little Theorem)
- Still an exponential number of a to check, so AKS uses the equivalent:

$$(x-a)^n \equiv (x^n - a) \pmod{(n, x^r - 1)}$$

$$(x-a)^n - (x^n - a) = nf + (x^r - 1)g$$

- f, g are some polynomials, $r = O(\log n)$ is all that is necessary
- Same problem as Fermat's test: some composites satisfy this; AKS proved that for a small r and small set A of integers it suffices to check the congruence for all A

- Not used (necessarily) in practice as the algorithm has running time $O(\log^3 n)$ to $O(\log^{12} n)$ depending on formulation and (unproven) conjectures
- Below, $\text{ord}_r(n)$ is the multiplicative order of n modulo r; the smallest k such that

$$n^k \equiv 1 \pmod r$$

  (see http://cs-haven.blogspot.com/2012/02/efficiently-computing-order-of-element.html for a nice writeup on how to efficiently compute k)
- Note: totient is not polynomially bounded, but r is, so $\sqrt{\phi r}$ is polynomial

```
INPUT : Integer $n > 1$
IF $n = a^b$ (a perfect power) THEN
    output composite
END
$r \leftarrow$ smallest integer such that $\text{ord}(n) > \log^2 n$
IF $1 < \gcd(a, n) < n$ for some $a \leq r$ THEN
    output composite
END
IF $n \leq r$ THEN
    output prime
END
FOR $a = 1, \ldots, \lfloor \sqrt{\phi(r)} \cdot \log n \rfloor$ DO
    IF $(x + a)^n \neq x^n + a \; (mod \; (x^r - 1, n))$ THEN
        output composite
    END
END
output prime;
```

Algorithm 9: AKS Primality Test

13.4. Factoring Attacks

Problem 2 (Factorization).
Given: $x \in \mathbb{Z}$
Ouput: A nontrivial y of x (or 1 if x is prime)

- Primality testing is a decision problem
- Factorization is a functional problem: we want an actual factor, not a yes or no
- Suffices to find one factor as we can repeat the process
- Factorization is at least as hard has primality testing (since if we can factor, we can determine if it is prime or not)

13.4.1. Trial Division & Sieve of Eratosthenes

```
   INPUT  : A positive integer $n \geq 2$
   OUTPUT: A prime factor $p$ of $n$
 1 FOREACH prime number $p$, $2 \leq p \leq \sqrt{n}$ DO
 2     IF $p \mid n$ THEN
 3         output $p$
 4     END
 5 END
 6 output 1
```

Algorithm 10: Trial Division

- Related: Sieve of Eratosthenes (which finds *all* primes up to some bound n)
- Try dividing by each prime (or just all odd numbers) until a factor is found
- Only need to try up to $\sqrt{n}$ (see below)
- But exponential: $O(\sqrt{n}) = O(\sqrt{2^N})$, where $N = \log n$, the true input size
- Full sieve actually finds *all* primes $2, \ldots, \sqrt{n}$

Lemma 1. If n is a composite integer, then n has a prime divisor $x \leq \sqrt{n}$.

Proof:

- Let n be a composite integer.
- By definition, n has a prime divisor a with $1 < a < n$, thus $n = ab$.
- Its easy to see that either $a \leq \sqrt{n}$ or $b \leq \sqrt{n}$. Otherwise, if on the contrary, $a > \sqrt{n}$ *and* $b > \sqrt{n}$, then
$$ab > \sqrt{n}\sqrt{n} = n$$
- Finally, either a or b is prime divisor or has a factor that is a prime divisor by the Fundamental Theorem of Arithmetic, thus n has a prime divisor $x \leq \sqrt{n}$.

13.4.2. Fermat's Factorization (and strong primes)

- Let n be an odd composite
- Any odd number can be represented as the difference of squares: $n = a^2 - b^2 = (a-b)(a+b)$
- Proof: Let $n = 2k + 1$, then
$$2k + 1 = (k+1)^2 - k^2$$
- Thus, n has factors: $a - b$, $a + b$
- Trick is to find perfect squares a, b; ignoring trivial factors $(1, n)$

Iteration	a	b^2	b
–	78	7	2.64
1	79	164	12.80
2	80	323	17.97
3	81	484	22.00

Table 8: Fermat's Factorization for $n = 6,077$

- Basis of Fermat's method (and the basis for many other methods)
- Square roots can be calculated efficiently using Newton's Method
- Complexity isn't better (can actually be worse):

$$O(n - \sqrt{n} - c/2)$$

- But can be heuristically better (when combined with trial division)
- Improvements can be made to give:

$$O(n^{1/3})$$

```
INPUT   : $n > 2$
1 $a \leftarrow \lceil \sqrt{n} \rceil$
2 $b^2 \leftarrow a^2 - n$
3 WHILE $b$ is not a square DO
4     $a \leftarrow a + 1$
5     $b^2 \leftarrow a^2 - n$
6 END
7 Output $a - \sqrt{b^2}$
```

Algorithm 11: Fermat's Factorization

- Any composite will have a difference of square representation, so its correct
- Still be exponential
- Used in conjunction with trial division

Example: $n = 6,077$ ($\lceil \sqrt{6077} \rceil = 78$)

Thus, the factors are

$$(a - b) = (81 - 22) = 59$$

and

$$(a + b) = (81 + 22) = 103$$

13.4.3. Pollard $p - 1$

- John Pollard (1974)

- Specialized algorithm that operates on "unsafe" primes: primes p such that $p-1$ is *B-powersmooth*
- n is B-smooth if all of its prime factors p are less than B
- n is B-powersmooth if all its prime powers, p^k (in its factorization) are less than B,

$$p^k \leq B$$

- Example: $113,400 = 2^3 \cdot 3^4 \cdot 5^2 \cdot 7^1$ is 7-smooth, but not 7-powersmooth since $2^3 > 7$. It *is* 81-powersmooth
- Safe primes: primes p for RSA must satisfy: $p-1 = 2q$ where q is a Sophie Germain prime (q is Sophie Germain if $2q+1$ is prime)

```
   INPUT : $n > 1$, a basis bound $B$
1  $a \leftarrow 2$
2  FOR $j = 2, \ldots, B$ DO
3      $a \leftarrow a^j \bmod n$
4  END
   //$a = a^{2^{3^{\cdots B}}} = 2^{B!} \bmod n$
5  $d \leftarrow \gcd(a-1, n)$
6  IF $1 < d < n$ THEN
7      output $d$
8  ELSE
9      output "idunno"
10 END
```

Algorithm 12: Pollard $p-1$ Factorization

Justification

First, consider the following corollary to Fermat's Little Theorem.

Corollary 2. Let p be prime with a coprime to p. Then

$$a^{k(p-1)} \equiv 1 \pmod{p}$$

Proof. Let a, p be as specified, then by Fermat's Little Theorem, we have

$$\begin{array}{llll} a^{p-1} \equiv 1 \pmod{p} & \Rightarrow & a^{p-1} - 1 = \ell p & \text{definition of congruence, } \ell \in \mathbb{Z} \\ & \Rightarrow & a^{(p-1)} = \ell p + 1 & \\ & \Rightarrow & (a^{(p-1)})^k = (\ell p + 1)^k & \text{raise both sides to } k \\ & \Rightarrow & a^{k(p-1)} = \ell p q(x) + 1 & (*) \\ & \Rightarrow & a^{k(p-1)} - 1 = \ell p q(x) & \\ & \Rightarrow & a^{k(p-1)} \equiv 1 \pmod{p} & \end{array}$$

The equality in $(*)$ above follows from the fact that $(\ell p+1)^k$ will be a degree-k polynomial; each term will be divisible by ℓq except the last one $+1$. $\square$

- As in the comment, at the end of the loop, $a = 2^{B!} \bmod n$, thus
$$a \equiv 2^{B!} \pmod{n}$$
- Assume p is a prime divisor of n, that is $p|n$
- If $p|n$, then
$$a \equiv 2^{B!} \pmod{p} \quad (1)$$
Aside justification: consider the following
$$\begin{aligned} x \equiv y \pmod{pq} &\iff x - y = kpq \\ &\iff x - y = (kq)p \\ &\iff x - y = k'p \\ &\iff x \equiv y \pmod{p} \end{aligned}$$
- Suppose that $(p-1)|B!$ (and so $B! = k(p-1)$), then (1) implies that
$$a \equiv 2^{k(p-1)} \pmod{p}$$
- By Lemma 2 we also have that
$$2^{k(p-1)} \equiv 1 \pmod{p} \quad (2)$$
- By (1) and (2) and transitivity of congruences, we have that
$$a \equiv 1 \pmod{p}$$
- By definition of congruences, that means that
$$a - 1 = kp$$
That is, $p|(a-1)$
- We now have that $p|n$ and $p|(a-1)$, thus it must divide their GCD:
$$p|\gcd(a-1, n)$$
(Aside justification: if $p|a$ and $p|b$, then p must divide $sa + tb = \gcd(a, b)$)
- Finally, $\gcd(a-1, n)$ gives a non-trivial factor under the assumptions above

Analysis

- Running time is
$$O(B \log B \log^2 n + \log^3 n)$$
- Thus if B is polynomial ($B \in O(\log^k n)$), then the algorithm is polynomial
- For smaller B, the higher the probability of failure (starting with large B may also be bad if you get n as a factor; you must try a smaller B)
- As $B \to \sqrt{n}$, guaranteed to succeed

13.4.4. Pollard-Rho

- John Pollard 1975, Monte Carlo
- Basic intuition: randomly trying factors has a low $\frac{2}{n}$ probability of success. Trying pairs whose difference has a common factor with n (using the GCD) has a much higher $\frac{p+q-2}{n}$ probability of success:

$$p, 2p, 3p, \ldots, (q-1)p, \quad q, 2q, 3q, \ldots, (p-1)q$$

- We want to find two elements in this sequence, x, y since $x - y = kp$, we can then compute

$$\gcd(|x-y|, n) = d$$

  gives a nontrivial factor
- To find two such numbers, we can define a pseudorandom sequence,

$$x_0, x_1, x_2, \ldots$$

  using a polynomial $g(x)$
- The idea is that if $x_i \in [0, n-1]$, then it will eventually hit some $k_1 p$ and some $k_2 p$ giving us what we need
- Probabilistic argument: by the birthday argument, a sequence $x_1, x_2, \ldots, x_t$ with $x_i \in [0, n-1]$ will repeat with probability $> \frac{1}{2}$ if $t > \sqrt{p}$
- However: we cannot keep all x_i in memory, $O(t) = O(\sqrt{p}) = O(n^{1/4})$, exponential
- Floyd's cycle detection algorithm allow us to only hold 2 values in memory
- Intuition with Floyd: two runners, A, B on a circular track: we want to know when we've completed a lap; so B runs twice as fast as A, when they meet again, they have completed at least 1 cycle
- For a sequence defined with $g(x)$, we want

$$x_1, x_2, x_3, \ldots \quad \text{and} \quad x_2, x_4, x_6, \ldots$$

- Analysis: an expected running time of $O(\sqrt{p})$; if $p \approx \sqrt{n}$, then it has a running time of $O(n^{1/4})$ with a probability of failure as $\leq \frac{p}{n} < \frac{\sqrt{n}}{n} = \frac{1}{\sqrt{n}}$ which is small for large n
- A very nice tutorial: http://www.cs.colorado.edu/~srirams/classes/doku.php/pollard_rho_tutorial

```
Define $g(x) = x^2 + 1$
$x, y, d \leftarrow 2, 2, 1$
//Any value of $x, y > 1$ is acceptable
WHILE $d = 1$ DO
    $x \leftarrow g(x) \bmod n$
    $y \leftarrow g(g(y)) \bmod n$
    $d \leftarrow \gcd(|x - y|, n)$
END
IF $d = n$ THEN
    output "idunno"
ELSE
    output $d$
END
```

Algorithm 13: Pollard Rho Factorization Algorithm

Iteration	x	y	$\gcd(\lvert x-y \rvert, n)$
0	2	2	1
1	5	26	1
2	26	12276	1
3	677	24657	1
4	12276	6415	1
5	29508	26337	151

Table 9: Execution of Pollard Rho for $n = 31861 = 151 \cdot 211$

Compare: only 5 iterations versus trial division of about 89.

13.4.5. Dixon's Method

- John Dixon (1981)
- Ideas built upon Fermat and Kraitchik (1920s)
- Basic Idea: any product ab can be written as the difference of squares:

$$n = ab = \frac{1}{4}\left[(a+b)^2 - (a-b)^2\right]$$

- Example: $8051 = 8100 - 49 = 90^2 - 7^2$
- Easy to find factors: $p = (a+b), q = (a-b)$ $(8051 = 83 \cdot 97)$
- Kraitchik's improvement: look for differences of squares not equal to n, but that are multiples of n

- That is:
$$x^2 - y^2 = kn \quad \Rightarrow \quad x^2 \equiv y^2 \pmod{n} (1)$$
- Solutions may be uninteresting: $x \equiv \pm y \pmod{n}$
- Solutions may be interesting: $x \not\equiv \pm y \pmod{n}$
- Equivalently:
$$x^2 - y^2 = kn \Leftrightarrow n | (x^2 - y^2) \quad \text{from (1)}$$
but
$$n \not| (x - y) \quad \text{and} \quad n \not| (x + y)$$
Thus the factors of n are split between $(x - y)$ and $(x + y)$, so the GCD reveals a factor
- Once we have an interesting solution,
$$\gcd(x - y, n) | n$$
(that is the GCD is a non-trivial factor of n)
- At least half of all solutions are interesting

Illustrative Example:

- Starting near $\sqrt{n}$, we search squares of some sequence (ex: $Q(x) = x^2 - n$ or some continued fraction (Lehmer and Powers, 1931)
- Ex: 2041: $\lceil\sqrt{2041}\rceil = 46$, so compute $Q(46), Q(47), Q(48), \ldots = 75, 168, 263, 360, 459, 560, \ldots$
- If any are squares, then done, but none are
- *BUT*: some have some very easy common factors:
$$\begin{aligned} Q(46) = 75 &= 3 \cdot 5^2 \\ Q(47) = 168 &= 2^3 \cdot 3 \cdot 7 \\ Q(49) = 360 &= 2^3 \cdot 3^2 \cdot 5 \\ Q(51) = 560 &= 2^4 \cdot 5 \cdot 7 \end{aligned}$$
- Taking the product of all 4 gives:
$$2^{10} \cdot 3^4 \cdot 5^4 \cdot 7^2 = \left(2^5 \cdot 3^2 \cdot 5^2 \cdot 7\right)^2 = (50,400)^2 = y^2$$
a square!
- The numbers in the sequence that gave us these values:
$$x = (46 \cdot 47 \cdot 49 \cdot 51) \quad \text{so} \quad x^2 = (46 \cdot 47 \cdot 49 \cdot 51)^2$$
- Further,
$$x \equiv 311 \pmod{2041}$$
and
$$y \equiv 1416 \pmod{2041}$$
but
$$311 \not\equiv \pm 1416 \pmod{2041}$$

- Thus, a non-trivial solution and

$$\gcd(1416 - 311, 2041) = 13$$

  is a factor of $2,041 = 13 \cdot 157$

Dixon's Method

- Establish a prime base $B = \{-1, 2, 3, 5, \ldots, p\}$ (a larger one will mean there are more potential squares, but be less efficient with factoring over the base)
- Probe for potential squares by examining $x = \lfloor\sqrt{ni}\rfloor$ and $\lceil\sqrt{ni}\rceil$ for $i = 1, 2, \ldots$
- Note: any sequence can be considered, but there is some heuristic evidence this is good; further, you can stop at a certain point or keep going until a square is found
- For each potential square, $x^2 \bmod n$, factor it over B (select the version that has the lowest absolute value so that it is more likely to factor over B); if it doesn't fully factor, ignore it, move on
- Keep a collection of factored vectors and find a *basis* over $\mathbb{Z}_2^{|B|}$ (its actually enough to find a linear combination whose sum gives the zero vector)
- That is, find a collection whose sum has exponents that are all zero modulo 2 (because we want a square
- Once such a basis is found, use the original x and the congruence to compute

$$x^2 \equiv y^2 \pmod{n}$$

- Compute $\gcd(x \pm y, n)$ to get a factor (may be trivial, if so keep trying different basis)
- Basis can be found using Gaussian Elimination in general
- Ideal size of base: $|B| \in O(\sqrt{\log n \log\log n})$ giving

$$O(\exp(2\sqrt{2}\sqrt{\log n \log\log n})$$

Example

- Let $n = 87463$, $B = \{-1, 2, 3, 5, 7, 11, 13\}$
- We'll compute floor/ceilings of $\sqrt{ni}$ for $i = 1, \ldots, 10$
- See full computation in Figure 20
- Most are uninteresting (prime or not factorable over B)
- Two nice factorizations

$$-245 = -1^1 \cdot 5^1 \cdot 7^2 \quad -405 = -1^1 \cdot 3^4 \cdot 5^1$$

- Their product is a perfect square:

$$(512 \cdot 935)^2 \equiv (3^2 \cdot 5 \cdot 7)^2 \pmod{87463}$$

```
295: 295^2 = 87025 = -438 (prime or failure)
296: 296^2 = 87616 =  153 (prime or failure)
418: 418^2 = 174724 = -202 (prime or failure)
419: 419^2 = 175561 =  635 (prime or failure)
512: 512^2 = 262144 = -245 mod n = (-1^1 x 2^0 x 3^0 x 5^1 x 7^2
   x 11^0 x 13^0 ) = (1, 0, 0, 1, 0, 0, 0)
513: 513^2 = 263169 =  780 mod n = (-1^0 x 2^2 x 3^1 x 5^1 x 7^0
   x 11^0 x 13^1 ) = (0, 0, 1, 1, 0, 0, 1)
591: 591^2 = 349281 = -571 (prime or failure)
592: 592^2 = 350464 =  612 (prime or failure)
661: 661^2 = 436921 = -394 (prime or failure)
662: 662^2 = 438244 =  929 (prime or failure)
724: 724^2 = 524176 = -602 (prime or failure)
725: 725^2 = 525625 =  847 mod n = (-1^0 x 2^0 x 3^0 x 5^0 x 7^1
   x 11^2 x 13^0 ) = (0, 0, 0, 0, 1, 0, 0)
782: 782^2 = 611524 = -717 (prime or failure)
783: 783^2 = 613089 =  848 (prime or failure)
836: 836^2 = 698896 = -808 (prime or failure)
837: 837^2 = 700569 =  865 (prime or failure)
887: 887^2 = 786769 = -398 (prime or failure)
888: 888^2 = 788544 = 1377 (prime or failure)
935: 935^2 = 874225 = -405 mod n = (-1^1 x 2^0 x 3^4 x 5^1 x 7^0
   x 11^0 x 13^0 ) = (1, 0, 0, 1, 0, 0, 0)
936: 936^2 = 876096 = 1466 (prime or failure)
```

Figure 20: Dixon's run for $n = 87,463 = 149 \cdot 587$

- Reducing:
$$(41405)^2 \equiv (315)^2 \pmod{87463}$$
- And so $\gcd(41405 + 315, 87463) = 149$
- And $87463 = 149 \cdot 587$

13.4.6. Quadratic Sieve

- Carl Pomerance (1981)
- Same basic idea as Dixon
- However, vector creation phase is optimized by "Sieving"
- Define $f(x) = x^2 - n$ and use the sequence
$$f(x + kp) = f(x) + 2xkp + (kp)^2 \equiv f(x) \pmod{p}$$
which gives a sequence each of which is divisible by p (Sieve similar to Eratosthenese, each value is a quadratic)

- An improvement to
$$O(\exp((1+o(1))\sqrt{\log n \log\log n}))$$

13.4.7. Number Field Sieve

- Lenstra, Lenstra, Manasse, Pollard (1993)
- Involves congruences and factorizations of polynomials over a number field ($\mathbb{Q}$) instead of a factor base
- Searches for numbers that are subexponential in n, and thus more likely to be smooth
- Complexity:
$$O\big(\exp(c((\log n)^{1/3} \cdot (\log\log n)^{2/3}))\big)$$
with $1.5263 \leq c \leq 1.922$ depending on the nature of the integer being factored
- Not asymptotically better, but heuristically better for larger numbers
- Fastest general purpose algorithm to date
- msieve (GPL implementation) `http://sourceforge.net/projects/msieve/`
- A nice overview of the history of sieves: `http://www.ams.org/notices/199612/pomerance.pdf`

13.4.8. Elliptic Curve Factorization

- Lenstra (1987)
- Improvement of Pollard's $p-1$ algorithm
- Uses a moderately large integer, $B!$
- Instead of searching for smooth squares, it searches for factors over a Elliptical Curves:
$$y^2 = x^3 + ax + b \pmod{n}$$
- Complexity:
$$O(\exp((\sqrt{2}+o(1))\sqrt{\log p \log\log p}))$$

13.4.9. Shor's Algorithm: Quantum Factorization

- Peter Shor (1994)
- Quantum Algorithm: operates on qubits (bits in a superposition $[0,1]$, but when measured collapse to $0,1$; Complexity class: BQP
- Qubits can "store" an exponential number of states (as n qubits can be in a superposition of 2^n possible states)
- Quantum operations perturb a system so that with higher probability, the observed final state will be solution

- Number of quantum gates required grows as $O(\log^3 n)$
- Long way off to developing such computers on such a large scale
- Algorithm works by reducing to order-finding: given a, n (a random), find smallest r such that
$$a^{x+r} \pmod{n} = a^x \pmod{n}$$
- There is a randomized reduction from factorization to order finding (Miller 1976, Miller's algorithm for factorization)
- Places the input into $2n^2$ qubits and applies the Quantum Fourier Transform that improves that probability that the measured state will collapse to the order

13.5. Non-Factoring Attacks

Observation: there is no known reduction from factorization to breaking RSA. That is: RSA is not necessarily as hard as factorization. Thus, an attack on RSA need not depend on Factorization.

Survey of attacks: https://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf

13.5.1. Exposing $\phi(n)$

- Computing $\phi(n)$ (from a, n) is no easier than factoring n
- Important to keep $\phi(n)$ secret: if known, we can easily factor n
- Suppose n and $\phi(n) = (p-1)(q-1)$ are known
- Then
$$\phi(n) = (p-1)(q-1) = pq - p - q + 1$$
substituting $q = n/p$ gives:
$$\phi(n) = n - p - n/p + 1$$
multiply both sides by p and collect terms:
$$np - p^2 - n + p - p\phi(n) = 0$$
$$p^2 - (n - \phi(n) + 1)p + n = 0$$
- We now have a quadratic equation; apply the quadratic formula and solve for p

13.5.2. Exposing The Decryption Exponent

- Exposing a is obviously bad: with only a, n, anyone can decrypt
- If a is compromised, the entire system is compromised: the modulus n *must* be changed

- Why: if we have a, there exists a randomized algorithm that, given a, b, n can factor n
- Basic idea: since $n = pq$, the equation

$$x^2 \equiv 1 \pmod{p}$$

has four square roots: two trivial ones $(1, -1)$ and two non-trivial ones
- If we can find such roots, x_1, x_2, then the factors:

$$p = \gcd(x_1 + 1, n), \quad q = \gcd(x_1 - 1, n)$$

(or the other)
- Knowing a, b allows us to start from a much smaller number than n, namely:

$$ab - 1 = 2^s \cdot r$$

(we start from r, compute $w^r \pmod{n}$ for $r, 2r, 4r, \ldots$)
- See Stinson, section 5.7.2 for full details

13.5.3. Wiener's small a (decryption) attack

- Using a small encryption modulus is common and can make encryption more efficient
- Doing the same for the decryption modulus is dangerous
- In particular, if $d = b$ is small:

$$3d < n^{1/4}, \quad \text{and} \quad q < p < 2q$$

- Then (e, n) (public keys) can be used to approximate t/a using a *continued fraction* and Euclid's algorithm:

$$q_1 + \cfrac{1}{q_2 + \frac{1}{q_3 + \cdots + \frac{1}{q_m}}}$$

- We test all convergences of this continued fraction to get

$$\phi(n) = (ab - 1)/t$$

and apply the first attack (known totient attack)
- Full details, Stinson, Section 5.7.3

13.5.4. Chosen Ciphertext Attack

- Stinson Exercise 5.14
- Oscar intercepts ciphertext y

- Oscar chooses y' such that $yy' \equiv 1 \pmod{n}$
- Fact: $e_K(x_1) \cdot e_K(x_2) \bmod n = e_K(x_1 \cdot x_2 \bmod n)$
- Observe:
$$1 = yy' \bmod n = e_K(x)e_K(x_2) \bmod n = e_K(xx' \bmod n)$$
- We could decrypt both sides and get:
$$d_K(1) = xx' \bmod n$$
- But decryption of 1 is always 1, so
$$1 = xx' \bmod n$$
- If we could find x', then we could decrypt to get x (find its inverse)
- Not necessarily a serious attack (still need to find x'), but it shows that its vulnerable to chosen ciphertext attack

13.5.5. Common Modulus Attack

- Simmons Attack (Stinson exercise 5.16)
- Suppose Alice encrypts x and sends to Bob and Charlie
- Bob, Charlie have same modulus, n, different encryption exponents b_1, b_2
- Oscar intercepts $y_1 = x^{b_1} \pmod{n}, y_2 = x^{b_2} \pmod{n}$
- Oscar computes Algorithm 14

```
1 $c_1 \leftarrow b_1^{-1} \mod b_2$
2 $c_2 \leftarrow (c_1 b_1 - 1)/b_2$
3 $x \leftarrow y_1^{c_1}(y_2^{c_2})^{-1} \mod n$
4 output $x$
```

Algorithm 14: Common Modulus Attack

Justification:

$$\begin{aligned}
y_1^{c_1}(y_2^{c_2})^{-1} \mod n &= \left(x^{b_1}\right)^{c_1} \cdot \left(x^{b_2{}^{c_2}}\right)^{-1} \pmod{n} \\
&= \left(x^{b_1 c_1}\right) \cdot \left(x^{b_2 c_2}\right)^{-1} \pmod{n} \\
&= \left(x^{b_1 c_1}\right) \cdot \left(x^{b_2((c_1 b_1 - 1)/b_2)}\right)^{-1} \pmod{n} \quad \text{definition of } c_2 \\
&= \left(x^{b_1 c_1}\right) \cdot \left(x^{(c_1 b_1 - 1)}\right)^{-1} \pmod{n} \\
&= x^{b_1 c_1 - b_1 c_1 + 1} \pmod{n} \\
&= x \pmod{n}
\end{aligned}$$

13.5.6. Hastad/Coppersmith Attack

- Suppose Alice sends the same message m to k people
- Each person has the same public exponent e and $e \leq k$ but different moduli, $n_1, \ldots, n_k$
- Oscar intercepts $y_1, \ldots, y_k$
- If all n_i, n_j are coprime (if not, then its even *easier* to break–why?), then Oscar has:

$$\begin{aligned} y_1 &= x^e \pmod{n_1} \\ y_2 &= x^e \pmod{n_2} \\ &\vdots \\ y_k &= x^e \pmod{n_k} \end{aligned}$$

- Observe that since $x < n_i$ for all i, we have

$$x^e < n_1 \cdot n_2 \cdots n_k$$

- Applying the Chinese Remainder theorem to

$$\begin{aligned} y_1 &\equiv x^e \pmod{n_1} \\ y_2 &\equiv x^e \pmod{n_2} \\ &\vdots \\ y_k &\equiv x^e \pmod{n_k} \end{aligned}$$

  Gives a solution $y = x^e$
- Computing the e-the root of y gives us x
- To reiterate: this is only possible if the same message is broadcast to k people and e is small ($e < k$)
- Other, related attacks on weak padding schemes (see `http://en.wikipedia.org/wiki/Coppersmith's_Attack`)

13.6. RSA In Practice

13.6.1. Size of Modulus

- Commonly, n is 1024 bits (309 digits, two 155 digit primes)
- Recent attacks: 768 bit (232 digit) broken in 2009 (took 2 years)
- Larger keys are suggested, 1,024–4,096 (1233 digits) are typical
- A modulus of size n requires two primes p, q each of size about $n/2$

13.6.2. Choice of Exponent

- Usually recommended to use $a \in \{3, 5, 17, 257, 65537\}$
- $a = 65537 = 2^{16} + 1 = 0b10000000000000001$
- Only about 18 multiplications needed; fast encryption in practice
- No guesswork: prime, so guaranteed to be coprime
- IETF RFC 4871 (http://www.ietf.org/rfc/rfc4871.txt)

13.6.3. Padding

The "text-book" version of RSA is insecure:

- It is not *semantically secure*: given two possible plaintexts p_1, p_2 and a ciphertext c, Oscar can determine if
$$e_K(p_1) = c$$
or
$$e_K(p_2) = c$$
That is, Oscar can *distinguish* between which plaintexts match which ciphertexts by simply using the public key to encrypt.
- This of very pratical concern:
  - If the message is a yes/no answer to a known question
  - Many protocols have predictable metadata and headers
- Malleability: Some forgery is possible: $c' = c \cdot 2^a \pmod n$ decrypts to $2m \pmod n$ (i.e. a shift)
- RSA is completely deterministic: a user could brute-force encrypt all possible x, $e_k(x)$ until a match was found.
- Especially a concern for small messages
- Suffers from similar problems of ECB mode: same encrypted block will have the same encryption
- Hastad/Coppersmith attack (using the same exponent with different moduli)

Solution: Optimal Asymmetric Encryption Padding (OAEP)

- Padding Scheme due to Bellare and Rogaway (1994)
- Standardized in PKCS#1 v2.2, RFC 3447 (2012)
http://www.emc.com/collateral/white-papers/h11300-pkcs-1v2-2-rsa-cryptography-st
pdf
- Adds randomized padding and a Feistel network prior to encryption
- Solves the problems identified above: protects against chosen plaintext and chosen ciphertext attacks

- More in-depth papers:
http://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf
http://www.ssi.gouv.fr/archive/fr/sciences/fichiers/lcr/bojong00.pdf http://www.di.ens.fr/~pnguyen/pub_BoJoNg00.htm
- MD2, 5 (back-compatible, not recommended) and SHA-1 (default), -224, -512/224/-512/256 supported
- Current variations:
EME-OAEP
EMSA-PSS (Encoding Method for Signatures with Appendix–Probabilistic Signature Scheme)
- Next pages taken from PKCS1v2.2 document (EME-OAEP)

7.1.1 Encryption operation

RSAES-OAEP-Encrypt ((n, e), M, L)

Options:	Hash	hash function (*hLen* denotes the length in octets of the hash function output)
	MGF	mask generation function
Input:	(n, e)	recipient's RSA public key (k denotes the length in octets of the RSA modulus n)
	M	message to be encrypted, an octet string of length *mLen*, where $mLen \le k - 2hLen - 2$
	L	optional label to be associated with the message; the default value for L, if L is not provided, is the empty string
Output:	C	ciphertext, an octet string of length k
Errors:	"message too long"; "label too long"	
Assumption:	RSA public key (n, e) is valid	

Steps:

1. *Length checking*:

   a. If the length of L is greater than the input limitation for the hash function ($2^{61} - 1$ octets for SHA-1), output "label too long" and stop.

   b. If $mLen > k - 2hLen - 2$, output "message too long" and stop.

2. *EME-OAEP encoding* (see Figure 1 below):

   a. If the label L is not provided, let L be the empty string. Let *lHash* = Hash (L), an octet string of length *hLen* (see the note below).

   b. Generate an octet string *PS* consisting of $k - mLen - 2hLen - 2$ zero octets. The length of *PS* may be zero.

   c. Concatenate *lHash*, *PS*, a single octet with hexadecimal value 0x01, and the message M to form a data block *DB* of length $k - hLen - 1$ octets as

   $$DB = lHash \parallel PS \parallel 0x01 \parallel M .$$

   d. Generate a random octet string *seed* of length *hLen*.

   e. Let *dbMask* = MGF (*seed*, $k - hLen - 1$)

f. Let *maskedDB* = *DB* ⊕ *dbMask*.

g. Let *seedMask* = MGF (*maskedDB*, *hLen*).

h. Let *maskedSeed* = *seed* ⊕ *seedMask*.

i. Concatenate a single octet with hexadecimal value 0x00, *maskedSeed*, and *maskedDB* to form an encoded message *EM* of length *k* octets as

$$EM = \text{0x00} \,\|\, maskedSeed \,\|\, maskedDB.$$

3. *RSA encryption*:

a. Convert the encoded message *EM* to an integer message representative *m* (see Section 4.2):

$$m = \text{OS2IP}\ (EM)\ .$$

b. Apply the RSAEP encryption primitive (Section 5.1.1) to the RSA public key (*n*, *e*) and the message representative *m* to produce an integer ciphertext representative *c*:

$$c = \text{RSAEP}\ ((n, e), m)\ .$$

c. Convert the ciphertext representative *c* to a ciphertext *C* of length *k* octets (see Section 4.1):

$$C = \text{I2OSP}\ (c, k)\ .$$

4. Output the ciphertext *C*.

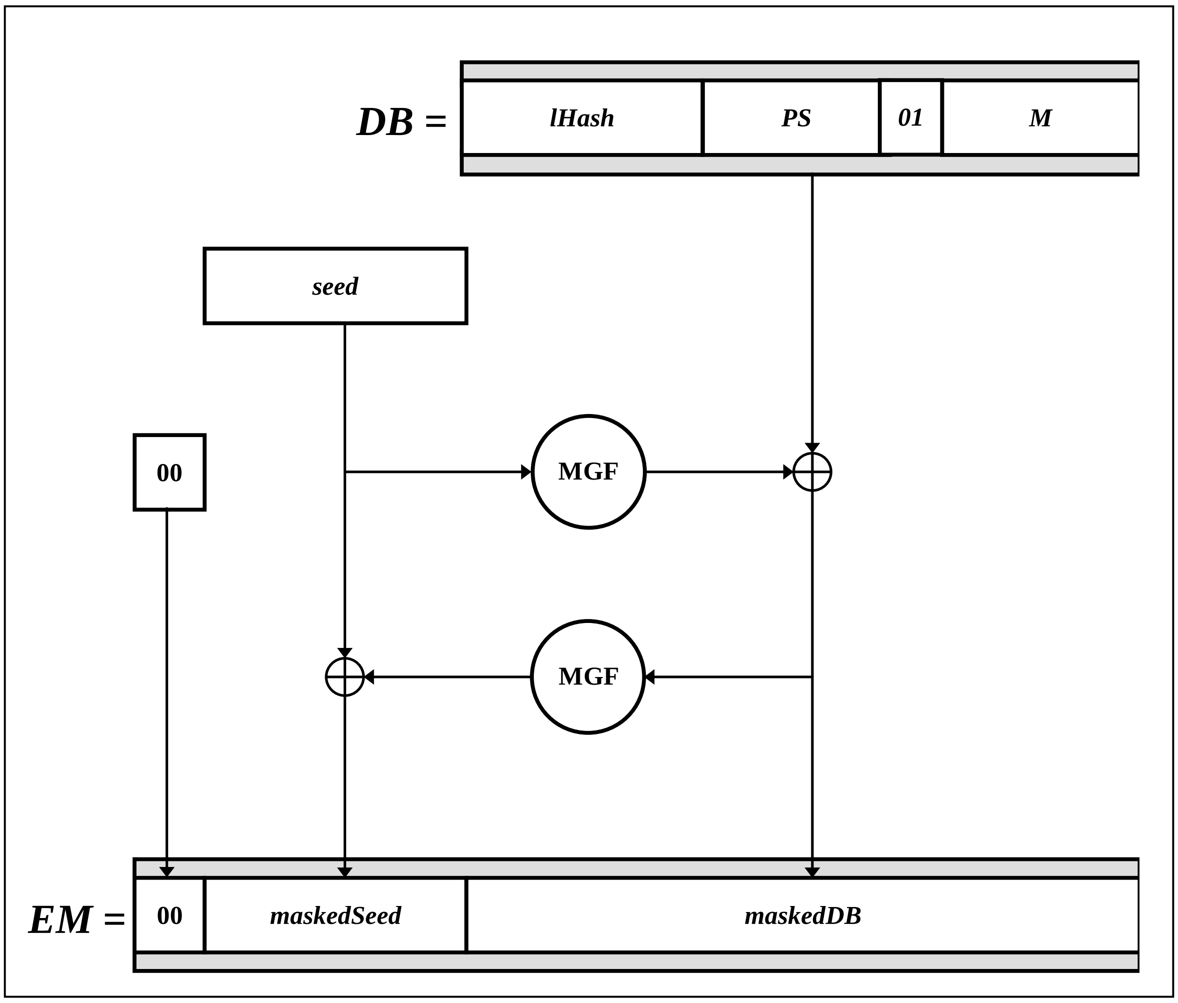


Figure 1: EME-OAEP encoding operation. *lHash* is the hash of the optional label *L*. Decoding operation follows reverse steps to recover *M* and verify *lHash* and *PS*.

Note. If *L* is the empty string, the corresponding hash value *lHash* has the following hexadecimal representation for different choices of Hash:

```
SHA-1:        (0x) da39a3ee 5e6b4b0d 3255bfef 95601890 afd80709
SHA-224:      (0x) d14a028c 2a3a2bc9 476102bb 288234c4 15a2b01f 828ea62a c5b3e42f
SHA-256:      (0x) e3b0c442 98fc1c14 9afbf4c8 996fb924 27ae41e4 649b934c a495991b 7852b855
SHA-384:      (0x) 38b060a7 51ac9638 4cd9327e b1b1e36a 21fdb711 14be0743 4c0cc7bf 63f6e1da
                   274edebf e76f65fb d51ad2f1 4898b95b
SHA-512:      (0x) cf83e135 7eefb8bd f1542850 d66d8007 d620e405 0b5715dc 83f4a921 d36ce9ce
                   47d0d13c 5d85f2b0 ff8318d2 877eec2f 63b931bd 47417a81 a538327a f927da3e
SHA-512/224: (0x) 6ed0dd02 806fa89e 25de060c 19d3ac86 cabb87d6 a0ddd05c 333b84f4
SHA-512/256: (0x) c672b8d1 ef56ed28 ab87c362 2c511406 9bdd3ad7 b8f97374 98d0c01e cef0967a
```

Encoding (original formulation; newer versions vary):

1. Message m is of length $n - (k_0 + k_1)$ (n is the modulus, k's parameters of the protocol)
2. Message is padded with k_1 zeros so that its length is $n - k_0$
3. Random string r of length k_0 is generated
4. $G(r)$ is an expansion function, $G : \{0,1\}^{k_0} \to \{0,1\}^{n-k_0}$
5. Compute:
$$x = m \,\|\, 0^{k_1} \oplus G(r)$$
6. H is a (compression) function $H : \{0,1\}^{n-k_0} \to \{0,1\}^{k_0}$
7. Compute
$$y = r \oplus H(x)$$
8. Output $x' = x \,\|\, y$ and encrypt it

Decoding:

1. Decrypt $d_K(x') = x \,\|\, y$
2. Recover $r = y \oplus H(x)$
3. Recover $m = x \oplus G(r)$ (ignoring trailing zero pad)

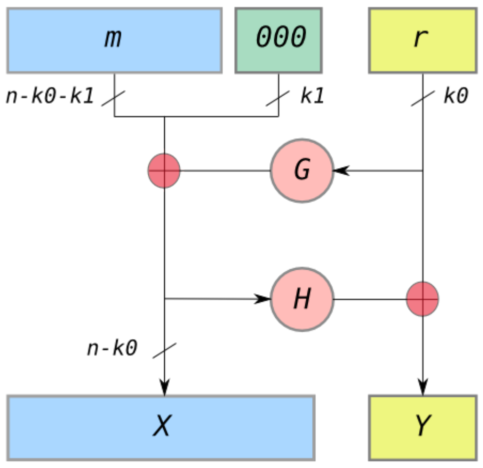


Figure 21: OAEP Padding Scheme

13.6.4. Chinese Remainder Theorem Speed Up

- Decryption can be sped up by splitting the message into 2 parts
- Alice knows factors $n = pq$, here $d = b$ (the private decryption exponent)

- Compute (division and Euclid):

$$d_p = d \pmod{p-1}$$
$$d_q = d \pmod{q-1}$$
$$q^{-1} \pmod{p}$$

- Split the encrypted message y up into two parts:

$$m_1 = y^{d_p} \pmod{p}$$
$$m_2 = y^{d_q} \pmod{q}$$

- Compute:
$$h = q^{-1}(m_1 - m_2) \pmod{p}$$

- Recover the message:
$$m = m_2 + hq$$

- Since d_p, d_q are smaller with smaller moduli, less computations in the exponentiation

Justification:

$$\begin{array}{rcll} m_1 & = & y^{d_p} \pmod{p} & \\ & = & y^{d \pmod{p-1}} \pmod{p} & \\ & = & y^d \pmod{p} & \text{by Fermat's Little Theorem} \end{array}$$

m_2 likewise: that is, splitting it up into two parts is valid.
The recombination step is justified by application of the CRT: our claim is that:

$$m \equiv m_1 \pmod{p}$$
$$m \equiv m_2 \pmod{q}$$

which would imply (by CRT) that

$$m = y^d \pmod{pq} = y^d \pmod{n}$$

For the first:

$$\begin{array}{rcl} m = m_2 + hq \pmod{p} & = & \big(m_2 + qq^{-1}(m_1 - m_2) \pmod{p}\big) + \pmod{p} \\ & = & \big(m_2 + (m_1 - m_2) \pmod{p}\big) + \pmod{p} \\ & = & \big(m_1 \pmod{p}\big) + \pmod{p} \\ & = & m_1 \pmod{p} \end{array}$$

For the second:

$$\begin{array}{rcl} m = m_2 + hq \pmod{q} & = & (m_2 \pmod{q}) + (hq \pmod{q}) \\ & = & (m_2 \pmod{q}) + 0 \\ & = & m_2 \pmod{q} \end{array}$$

Example:

- Let $(p, q, n, \phi(n), a, b) = (7, 13, 91, 72, 11, 59)$
- Consider $y = 50$ (note, $50^{59} \pmod{91} = 71$)
- Then

$$d_p = 59 \pmod{6} = 5$$
$$d_q = 59 \pmod{12} = 11$$
$$q^{-1} \pmod{p} = 6$$

- Message split:

$$m_1 = 50^5 \pmod{7} = 1$$
$$m_2 = 50^{11} \pmod{13} = 6$$

- Now:

$$h = 6(1 - 6) \pmod{7} = 5$$

- Finally:

$$m = m_2 + hq = 6 + 5 \cdot 13 = 71$$

- Contrast: with exponent of 59: about 11 multiplications; with $5, 11$: $5 + 7$ (BUT: they can be reused for each one!): At least a 25% reduction in multiplications

13.6.5. Misc

- Not used for encryption; used for key-exchange
- Using RSA as a Digital Signature (hash then encrypt)
- TLS setup/rant: `http://wingolog.org/archives/2014/10/17/ffs-ssl`
- Key Management: centralized vs decentralized
- Certificate Authorities (CA)
- More later

14. Discrete Logarithm

14.1. Problem

14.1.1. Preliminaries

- Let G be a *cyclic* group (it contains an element g of order n)
- The order of a group is its size, $|G|$, the order of an element $a \in G$ is the smallest integer k such that $a^k = 1$

- An element $g \in G$ with order $n = |G|$ is a *generator* of G (also called a *primitive element*) since
$$G = \langle g \rangle = \{g^i | 0 \leq i \leq n-1\}$$
That is, it *generates* the group.
- Example: set of all residues modulo p that are relatively prime to p (order is $p-1$):
$$\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$$
Recall that this is a group under multiplication, we don't need 0 as we don't need an additive identity.
- Examples for $p = 7, p = 17$ in Tables 10 and 11

Element	Order	Cyclic Subgroup
0	∞	$\langle 0 \rangle = \{\}$
1	1	$\langle 1 \rangle = \{1\}$
2	3	$\langle 2 \rangle = \{1, 2, 4\}$
3	6	$\langle 3 \rangle = \{1, 2, 3, 4, 5, 6\}$
4	3	$\langle 4 \rangle = \{1, 2, 4\}$
5	6	$\langle 5 \rangle = \{1, 2, 3, 4, 5, 6\}$
6	2	$\langle 6 \rangle = \{1, 6\}$

Table 10: Elements and orders of the group $\mathbb{Z}_7$

Element	Order	Cyclic Subgroup
0	∞	$\langle 0 \rangle = \{\}$
1	1	$\langle 1 \rangle = \{1\}$
2	8	$\langle 2 \rangle = \{1, 2, 4, 8, 9, 13, 15, 16\}$
3	16	$\langle 3 \rangle = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$
4	4	$\langle 4 \rangle = \{1, 4, 13, 16\}$
5	16	$\langle 5 \rangle = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$
6	16	$\langle 6 \rangle = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$
7	16	$\langle 7 \rangle = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$
8	8	$\langle 8 \rangle = \{1, 2, 4, 8, 9, 13, 15, 16\}$
9	8	$\langle 9 \rangle = \{1, 2, 4, 8, 9, 13, 15, 16\}$
10	16	$\langle 10 \rangle = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$
11	16	$\langle 11 \rangle = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$
12	16	$\langle 12 \rangle = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$
13	4	$\langle 13 \rangle = \{1, 4, 13, 16\}$
14	16	$\langle 14 \rangle = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$
15	8	$\langle 15 \rangle = \{1, 2, 4, 8, 9, 13, 15, 16\}$
16	2	$\langle 16 \rangle = \{1, 16\}$

Table 11: Elements and orders of the group $\mathbb{Z}_{17}$

- Note: the number of generators (primitive elements) in $\mathbb{Z}_p$ is

$$\phi(\phi(p)) = \phi(p-1)$$

- For $p = 7, p = 17$,

$$\phi(6) = \phi(2 \cdot 3) = (2-1)(3-1) = 2$$
$$\phi(16) = \phi(2^4) = 2^3(2-1) = 8$$

just as above.

14.1.2. Definition

Problem 3 (Discrete Logarithm)**.**
Given: A multiplicative group G, a generator $g \in G$ (g has order n), an element $b \in \langle g \rangle$
Output: The unique integer $a, 0 \le a \le n-1$ such that

$$g^a = b$$

Equivalently,

$$a = \log_g(b)$$

14.1.3. Examples

Let $p = 17$, $g = 5$; what is the discrete log of 13? That is, what a satisfies

$$5^a \bmod 17 = 13$$

Or,

$$a = \log_5(13)$$

Trial multiplication:

$$5^1 \bmod 17 = 5$$
$$5^2 \bmod 17 = 8$$
$$5^3 \bmod 17 = 6$$
$$5^4 \bmod 17 = 13$$

So

$$\log_5(13) = 4$$

All discrete logs of $g = 5$:

$$\begin{aligned}
5^6 &= 2\\
5^{13} &= 3\\
5^{12} &= 4\\
5^1 &= 5\\
5^3 &= 6\\
5^{15} &= 7\\
5^2 &= 8\\
5^{10} &= 9\\
5^7 &= 10\\
5^{11} &= 11\\
5^9 &= 12\\
5^4 &= 13\\
5^5 &= 14\\
5^{14} &= 15\\
5^8 &= 16
\end{aligned}$$

14.2. ElGamal System

- Taher Elgamal (1985)
- Used in GPG
- Public key, used for key exchange ("hybrid cryptosystem")
- Similar problems to RSA (semantic security, chosen ciphertext, etc.), usually some form of padding is used

Let p be prime such that the Discrete Logarithm problem in $\mathbb{Z}_p^*$ is infeasible; let $g \in \mathbb{Z}_p^*$ be a generator. Define a key:

$$(p, g, a, b)$$

where

$$b \equiv g^a \pmod{p}$$

Define an encryption function:

$$e_K(x, r) = (y_1, y_2)$$

where r is a random element $r \in \mathbb{Z}_p^*$ (randomly chosen for any encryption or reencryption of x) and

$$y_1 = g^r \bmod p$$

$$y_2 = x \cdot b^r \bmod p$$

Define a decryption function:

$$d_k(y_1, y_2) = y_2(y_1^a)^{-1} \bmod p$$

- You *choose* a, then you can easily compute b (but choosing b, then finding a is solving the discrete log problem)

- b^r is a *mask*, introduces nondeterminism
- Pair (y_1, y_2) is transmitted
- Public key: (p, g, b)
- Private key: (a)

Correctness:

$$\begin{array}{rcl} d_k(e_k(x,r)) & = & d_k(y_1, y_2) \\ & = & xb^r \cdot (g^{r^a})^{-1} \bmod p \\ & = & xb^r \cdot ((g^a)^r)^{-1} \bmod p \\ & = & xb^r \cdot (b^r)^{-1} \bmod p \quad (*) \\ & = & x \end{array}$$

Where $(*)$ follows from the fact that $b \equiv g^a \pmod{p}$.

Brute-force breaking: given (p, g, b), we want to find a such that

$$b \equiv g^a \pmod{p}$$

that is,

$$b - g^a = \ell p$$

or

$$b = g^a \bmod p$$

or

$$a = \log_g(b)$$

the discrete log problem!

14.2.1. Examples

Consider the following instance of the ElGamal system:

- $p = 53$
- Note that $\phi(\phi(p)) = \phi(2^2 \cdot 13) = 24$, so there are 24 potential generators
- Choose $g = 3$ (2 is also a generator)
- Choose $a = 12$, then

$$g^a \bmod 53 = 3^{12} \bmod 53 = 10 = b$$

- Thus,

$$(p, g, a, b) = (53, 3, 12, 10)$$

Let's encrypt $x = 15$, choose $r = 20$, then

$$\begin{array}{rcl} e_k(x,r) & = & e_k(15, 20) \\ & = & (g^r \bmod p, x \cdot b^r \bmod p) \\ & = & (3^{20} \bmod 53, 15 \cdot 10^{20} \bmod 53) \\ & = & (49, 36) \end{array}$$

Let's decrypt the same:

$$\begin{aligned} d_k(y_1, y_2) &= d_k(49, 36) \\ &= y_2\,(y_1^a)^{-1} \bmod p \\ &= 36\,(49^{12})^{-1} \bmod 53 \\ &= 36\,(13)^{-1} \bmod 53 \\ &= 36 \cdot 49 \bmod 53 \\ &= 15 \end{aligned}$$

14.3. Algorithms

- Discrete Log is closely related to factorization. Many algorithms for one can be adapted to the other.
- Choice of group is essential (some simple groups it is easy to compute)
- Current records: 596-bit (180 digit) safe prime broken using Number Field Sieve, 400,000 core hours (2014)

TODO: how does Discrete Log reduce to order finding?

- Brute Force: simple trial multiplication, $O(n)$
- Shanks Algorithm (aka Baby-step Giant-step)
  - Time-space trade off
  - Rewrite:
    $$g^a = b \Rightarrow b\left(g^{-m}\right)^i = g^j$$
    where $a = im + j$, $m = \rceil\sqrt{n}\lceil$
  - Precompute g^j for several values of j and stores them
  - Tests values of i, more precomputed values cuts down on number of tests but requires more precomputation/storage
  - Complexity: $O(\sqrt{(}n))$
- Pollard-Rho
  - Analog of Pollard-Rho for Discrete Log
  - Searches for w, x, y, z such that
    $$g^w b^x = g^y b^z$$
    then
    $$(z - x)a = (w - y) \bmod (n)$$
    provides the solution to a
  - Uses Floyd's cycle finding algorithm which will find w, x, y, z in expected $\sqrt{\pi n/2}$ steps with the sequence
    $$s_i = g^{w_i} a^{x_i}$$

  - Sequence can be generated by partitioning G into S_1, S_2, S_3, if $x_i \in S_1$, double w, x; if $x_i \in S_2$, increment w; if $x_i \in S_3$ increment x
  - Complexity: $O(\sqrt{n})$
- Pholig-Hellman
  - Special purpose algorithm for groups whose order is *smooth* (small prime factors)
  - Similar ideas to Pollard $p - 1$
  - Complexity:
$$O\left(\sum_i e_i\left(\log n + \sqrt{p_i}\right)\right)$$
    where $n = \prod_i p_i^{e_i}$; for smooth numbers, this is small; for strong primes it devolves into $O(\sqrt{n})$
- Index Calculus Algorithm
  - Similar ideas to Dixon: attempts to find DL over a factor base
  - Not applicable to Elliptic Curves
  - Complexity:
$$L_n[1/2, \sqrt{2} + o(1)] = O(e^{(\sqrt{2}+o(1))\sqrt{\ln n \ln \ln n}})$$
- Number Field Sieve
  - Adapted by Gordon (1993)
  - Complexity:
$$L_n[1/3, 2.0800\ldots] = O(e^{(2.08)(\ln n)^{1/3}(\ln \ln n)^{2/3}})$$
  - Second parameter subsequently lowered to 1.9229
- Shor's Quantum algorithm also works for Discrete Logs
  - Shor's algorithm actually performs *order finding* to factor. Both Factorization and Discrete Log can be solved with order finding, thus it solves both.

15. Elliptic Curve Cryptography

- First proposed in 1985 by Neil Koblitz, Victor Miller
- Gaining popularity over RSA-based encryption, key identification, and signatures since 2005 (first NIST approved curves)
- Plagued by potential patent issues (Blackberry) and general distrust of NIST, NSA
- Advantages: same security can be achieved with much smaller primes (fewer bits)
- Ex: 256-bit ECC is equivalent to 3072-bit RSA key

15.1. Elliptic Curves

An elliptic curve is defined by a Weierstrass equation:

$$y^2 = x^3 + ax + b$$

that is *non-singular*. That is, if and only if the discriminant

$$-16(4a^3 + 27b^2) \neq 0$$

Moreover, the curve has 1 part if the discriminant is negative, 2 parts if positive.

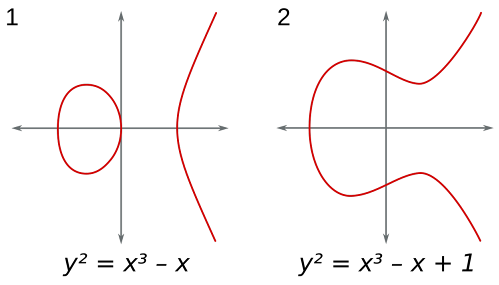


Figure 22: Elliptic Curves with 2, 1 components

- A non-singular EC defines a group operation (addition) over the set of all points on the EC
- Let $E \subseteq \mathbb{R} \times \mathbb{R}$ be the set of all points on a non-singular elliptic curve, let $P = (x_1, y_1), Q = (x_2, y_2) \in E$
- E along with a special *point at infinity* O (the *identity* element) defines group elements
- The group operation, $P + Q = (x_1, y_1) + (x_2, y_2)$ is defined as follows (derivation omitted).
  1. If $x_1 \neq x_2$, then $P + Q = (x_3, y_3)$ where

$$\begin{array}{rcl} x_3 & = & \lambda^2 - x_1 - x_2 \\ y_3 & = & \lambda(x_1 - x_3) - y_1 \\ \lambda & = & \frac{y_2 - y_1}{x_2 - x_1} \end{array}$$

  2. If $x_1 = x_2$ and $y_1 = -y_2$, then

$$P + Q = O$$

  3. If $x_1 = x_2$ and $y_1 = y_2$, then same as case 1 except

$$\lambda = \frac{3{x_1}^2 + a}{2y_1}$$

- Graphical interpretation (Figure 23):
    1. Case 1: Two points define a third on the EC; the sum $P + Q$ is equal to $-R$ (the point of intersection reflected about the x-axis
    2. Case 2: Vertical lines map to the identity element (the third point is the point at infinity
    3. Case 3: $P + P$ defines a tangent line, so we map to the other point intersected by the tangent
- All group axioms hold (most are trivial to prove, associativity is a bit difficult)

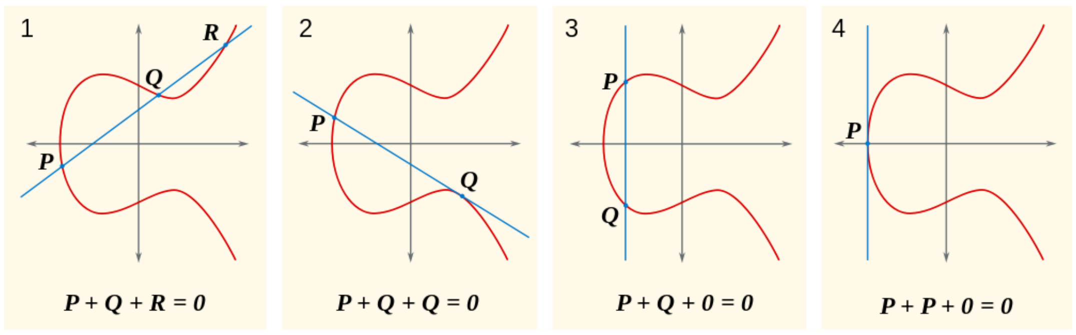


Figure 23: Group operation defined on an Elliptic Curve

15.1.1. Elliptic Curves Over $\mathbb{Z}_p$

Definition 8 (Elliptic Curve Over $\mathbb{Z}_p$)**.** Let $p > 3$ be prime, then the elliptical curve $y^2 = x^3 + ax + b$ over $\mathbb{Z}_p$ is the set of points $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$ satisfying the congruence

$$y^2 \equiv x^3 + ax + b \pmod{p}$$

with $a, b \in \mathbb{Z}_p$ such that $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. We also include the special *point at infinity* O.

Addition is defined as follows.

$$P + Q = (x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$

- If $x_1 = x_2$ and $y_2 = -y_1$ (Case 2), then $P + Q = O$
- Otherwise

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{aligned}$$

  as before, but

$$\lambda = \begin{cases} (y_2 - y_1)(x_2 - x_1)^{-1} & \text{if } P \neq Q \\ (3{x_1}^2 + a)(2y_1)^{-1} & \text{if } P = Q \end{cases}$$

- Define associativity for the point at infinity:

$$P + O = O + P = P$$

- No nice geometric interpretation: this is a discrete analog of an Elliptic Curve; visually it would be a (symmetric) collection of points in $\mathbb{Z}_p \times \mathbb{Z}_p$

15.2. Example

- Fix $p = 17$, so our Elliptic Curve is defined over $\mathbb{Z}_{17} \times \mathbb{Z}_{17}$
- Fix $a = 3, b = 5$, so our Elliptic Curve is

$$y^2 = x^3 + 3x + 5$$

- For each $x \in \mathbb{Z}_{17}$, compute the RHS and solve

$$y^2 \equiv x^3 + 3x + 5 \pmod{17}$$

- If they are quadratic residues (perfect squares in $\mathbb{Z}_{17}$, then they define two points in E
- Full computation is in Figure 24

x	$x^3 + ax + b$ mod p	quadratic residue?	y
0	5	`false`	$\{\}$
1	9	`true`	$\{3, 14\}$
2	2	`true`	$\{6, 11\}$
3	7	`false`	$\{\}$
4	13	`true`	$\{8, 9\}$
5	9	`true`	$\{3, 14\}$
6	1	`true`	$\{16, 1\}$
7	12	`false`	$\{\}$
8	14	`false`	$\{\}$
9	13	`true`	$\{8, 9\}$
10	15	`true`	$\{7, 10\}$
11	9	`true`	$\{3, 14\}$
12	1	`true`	$\{16, 1\}$
13	14	`false`	$\{\}$
14	3	`false`	$\{\}$
15	8	`true`	$\{5, 12\}$
16	1	`true`	$\{16, 1\}$

Figure 24: Quadratic Residues for the Elliptic Curve $y^2 = x^3 + 3x + 5$ over $\mathbb{Z}_{17}$. The number of points on the curve is 23, which is prime, so the curve defines a cyclic group of prime order. Thus, any element of the curve is a generator.

Thus our Elliptic Curve defines the set of points over $\mathbb{Z}_{17}$:

$$E = \{O\} \cup \{(12, 1), (1, 3), (2, 6), (5, 14), (15, 12), (16, 16), (6, 16), (15, 5),$$

$$(4,9),(9,8),(10,7),(16,1),(2,11),(4,8),(11,3),(1,14),$$
$$(9,9),(6,1),(10,10),(5,3),(11,14),(12,16)\}$$

- Now we need a generator G for this group
- Since the order (size) is prime, *every* non identity element is a generator
- Choose
$$g = (12,1)$$
- Compute $g+g$: since the points are the same,
$$\begin{array}{rcl} \lambda & = & (3{x_1}^2 + a)(2y_1)^{-1} \\ & = & (3 \cdot 12^2 + 3)(2 \cdot 1)^{-1} \\ & = & 10 \cdot 2^{-1} \\ & = & 10 \cdot 9 \\ & = & 5 \end{array}$$
- Now,
$$\begin{array}{rcl} 2g = g+g & = & (12,1) + (12,1) \\ & = & (\lambda^2 - x_1 - x_2, \lambda(x_1 - x_3) - y_1) \\ & = & (5^2 - 12 - 12, 5(12 - x_3) - 1) \\ & = & (1, 5(12-1) - 1) \\ & = & (1,3) \end{array}$$
- Next compute
$$3g = 2g + g = (1,3) + (12,1)$$
Since the points are different,
$$\begin{array}{rcl} \lambda & = & (y_2 - y_1)(x_2 - x_1)^{-1} \\ & = & (1-3)(12-1)^{-1} \\ & = & 15 \cdot 14 \\ & = & 6 \end{array}$$
- Now,
$$\begin{array}{rcl} 3g = 2g+g & = & (1,3) + (12,1) \\ & = & (\lambda^2 - x_1 - x_2, \lambda(x_1 - x_3) - y_1) \\ & = & (6^2 - 1 - 12, 6(1 - x_3) - 3) \\ & = & (6, 6(1-6) - 3) \\ & = & (6,1) \end{array}$$
- Figure 25 contains the full addition table

$$\begin{aligned}
1 \cdot g &= (12, 1)\\
2 \cdot g &= (1, 3)\\
3 \cdot g &= (6, 1)\\
4 \cdot g &= (16, 16)\\
5 \cdot g &= (2, 11)\\
6 \cdot g &= (4, 8)\\
7 \cdot g &= (10, 10)\\
8 \cdot g &= (11, 3)\\
9 \cdot g &= (15, 5)\\
10 \cdot g &= (5, 14)\\
11 \cdot g &= (9, 8)\\
12 \cdot g &= (9, 9)\\
13 \cdot g &= (5, 3)\\
14 \cdot g &= (15, 12)\\
15 \cdot g &= (11, 14)\\
16 \cdot g &= (10, 7)\\
17 \cdot g &= (4, 9)\\
18 \cdot g &= (2, 6)\\
19 \cdot g &= (16, 1)\\
20 \cdot g &= (6, 16)\\
21 \cdot g &= (1, 14)\\
22 \cdot g &= (12, 16)
\end{aligned}$$

Figure 25: Addition table for generator $g = (12, 1)$. Note that the next element would be the point at infinity.

15.3. Example Encryption System

We now apply our example to create an Elgamal Encryption System based on the Elliptic Curve above.

Observe that:

- Multiplication in G is now addition in E
- We omit how to encode/map data to E

Now:

- Fix $p = 17$, $g = (12, 1)$
- Note that the group $G = E$ (the points on the Elliptic Curve), and that $G \cong \mathbb{Z}_{23}$
- However, operations are performed over $\mathbb{Z}_{17}$
- Choose a private key $a = 10$, then

$$b = 10 \cdot g = (5, 14)$$

- Public key:
$$(p = 17, g = (12, 1), b = (5, 14))$$
- The encryption is (for $x \in E$, $0 \leq k \leq 23$):
$$\begin{aligned} e_k(x, k) &= (k \cdot g, x + k \cdot b) \\ &= (k \cdot (12, 1), x + k \cdot (5, 14)) \end{aligned}$$
- The decryption is:
$$\begin{aligned} d_k(y_1, y_2) &= y_2 \, (a \cdot y_1)^{-1} \\ &= y_2 - 10y_1 \end{aligned}$$

Encryption:

- First observe that we cannot encrypt points not on the curve, $(3, 4)$ for example
- Encrypt $x = (9, 8)$, choose $k = 5$
- Observe that $(9, 8) = 11g$ and note that we do this in terms of orders of g, but in practice "repeated doubling" is performed

$$\begin{aligned} e_k(x, k) &= e_k((9, 8), 5) \\ &= (k \cdot g, x + k \cdot b) \\ &= (5 \cdot (12, 1), (9, 8) + 5 \cdot (5, 14)) \\ &= ((2, 11), 11g + 5 \cdot 10g) \\ &= ((2, 11), 61g) \\ &= ((2, 11), 15g) \\ &= ((2, 11), (11, 14)) \end{aligned}$$

Decryption:

- Decrypt the same:
$$y = (y_1, y_2) = ((2, 11), (11, 14))$$
-
$$\begin{aligned} d_k(y) &= d_k((2, 11), (11, 14)) \\ &= y_2 = 10y_1 \\ &= (11, 14) - 10(2, 11) \\ &= 15g - 10 \cdot 5g \\ &= -35g \\ &= 11g \\ &= (9, 8) \end{aligned}$$

15.4. Additional Resources

- http://patricklonga.webs.com/Presentation_CFRG_Selecting_Elliptic_Curves_for_Cryptography.pdf
- More resources: http://www.reddit.com/r/crypto/comments/281t0g/elliptic_curve_encryption_resources/

15.5. Curves in Practice

- NIST FIPS 186-4 (Most current version, July 2014):
  http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
- Computing valid parameters is time consuming, instead *standard curves* have been established by various entities (NIST, ECC Brainpool)

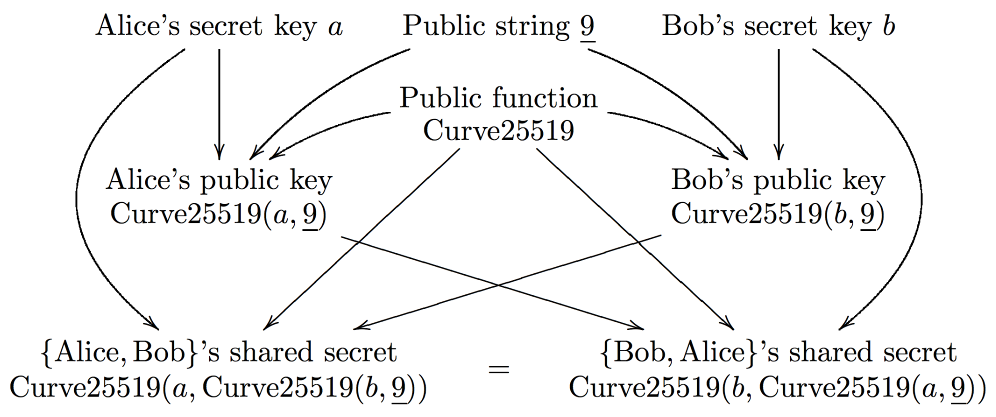
15.6. Curve25519

- Proposed by Daniel Bernstein (2005)
- Alternative to NIST curves
- Several advantages: faster, can be used to sign and exchange keys at the same time
- http://cr.yp.to/ecdh/curve25519-20060209.pdf
- http://cr.yp.to/ecdh.html
- Public domain, only 256-bit secret keys
- Group: $E(\mathbb{F}_{p^2})$ where

  $$p = 2^{255} - 19$$

  $x = 9$ is the base point and E is the Montgomery (which can be converted to Weierstrass) curve

  $$y^2 = x^3 + 486662x^2 + x$$



Alice's secret key a
Public string $\underline{9}$
Bob's secret key b
Public function Curve25519
Alice's public key Curve25519$(a, \underline{9})$
Bob's public key Curve25519$(b, \underline{9})$
$\{$Alice, Bob$\}$'s shared secret Curve25519$(a,$ Curve25519$(b, \underline{9}))$
$=$
$\{$Bob, Alice$\}$'s shared secret Curve25519$(b,$ Curve25519$(a, \underline{9}))$

Figure 26: Overview of key exchange

16. Diffie-Helman Key Exchange

- Public key crypto systems are not used for encryption: too inefficient
- Instead, used to exchange keys for use in symmetric key encryption

- Sometimes called "hybrid crypto system"
- Due to Whitfield Diffie, Martin Hellman (1976)

Basic Outline:

- Suppose Alice, Bob know G, a cyclic group and g a generator.
- Alice knows a secret integer a
- Bob knows a secret integer b
- Alice publishes g^a
- Bob publishes g^b
- Shared secret key:
$$k = (g^a)^b = (g^b)^a = g^{ab}$$
- Oscar cannot determine g^{ab} without solving the discrete log

17. Digital Signatures

- Need a way to ensure that a message is *authentic*
- A user can *digitally sign* a message as well as encrypt it
- Basic Idea:
  - Alice has a message m
  - Alice computes a hash $h(m)$
  - Alice "encrypt" $h(m)$ by using her private key (so actually the decryption algorithm):
$$s = d_k(h(m))$$
  - Transmit (or encrypt and transmit) (m, s)
  - Verification can be done by (decrypting), to get (m, s); then compute
$$e_k(s) = h'$$
    (which can be done since e_k is public); verify that
$$h' = h(m)$$
- Important to sign *then/and* encrypt to protect against forgeries
- Example: GPG
- *Disavowal* protocols exist to destroy a key
- Protocols also exist for *non-repudiation*: a *challenge* protocol can be used to prove that a message came from Alice even if she denies it (probabilistic)
- In practice: keys are managed by Certificate Authorities

- DSA (Digital Signature Algorithm): FIPS 186-4, variation of ElGamal Signature Scheme

Part V. Real-world Systems and Protocols

18. Transport Layer Security

- TLS is the most ubiquitous security layer in the OSI stack
- Normal application protocols usually work on top of TLS to provide secure versions of those protocols
- HTTP (port 80) vs HTTPS (port 443)
- SFTP, SSH, etc. (all port 22)
- Usually use different ports (convention)
- Originally developed by Netscape (1994)
- TLS/HTTPS/SSL (NIST TLS standard: `http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf`)

18.1. OSI & TCP/IP Models

- Open Systems Interconnection Model (OSI, ISO/IEC 7498-1)
- OSI is very general and the various layers can have different interpretations (see Figures)

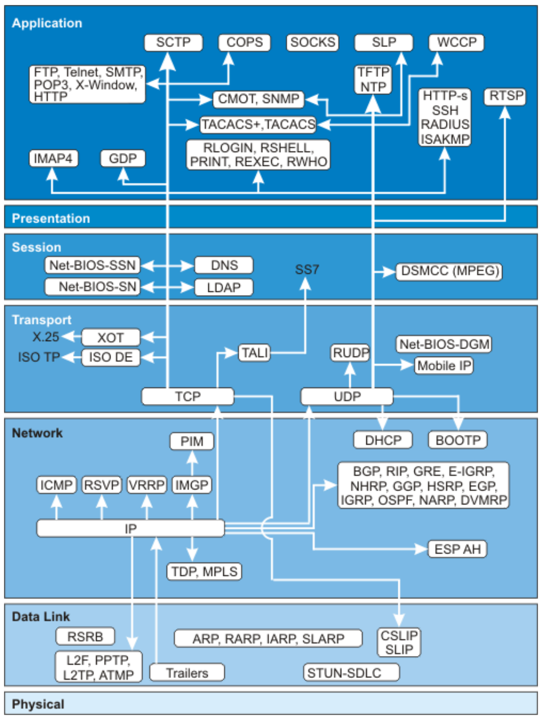


Figure 27: OSI Model – One Interpretation

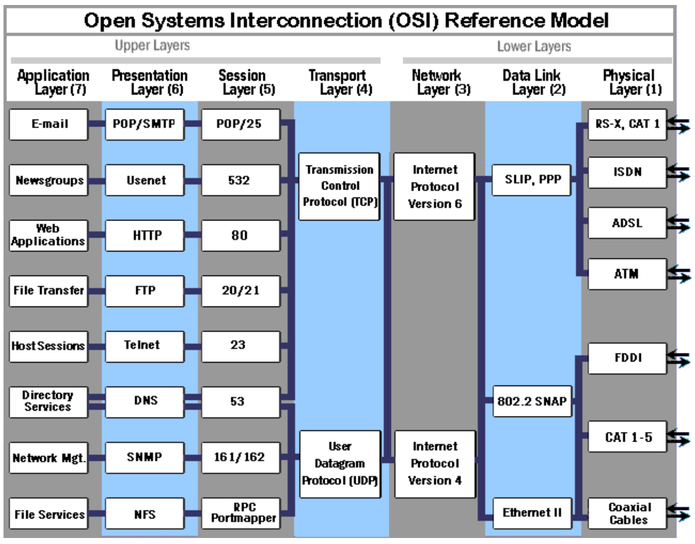


Figure 28: OSI Model – Another Interpretation

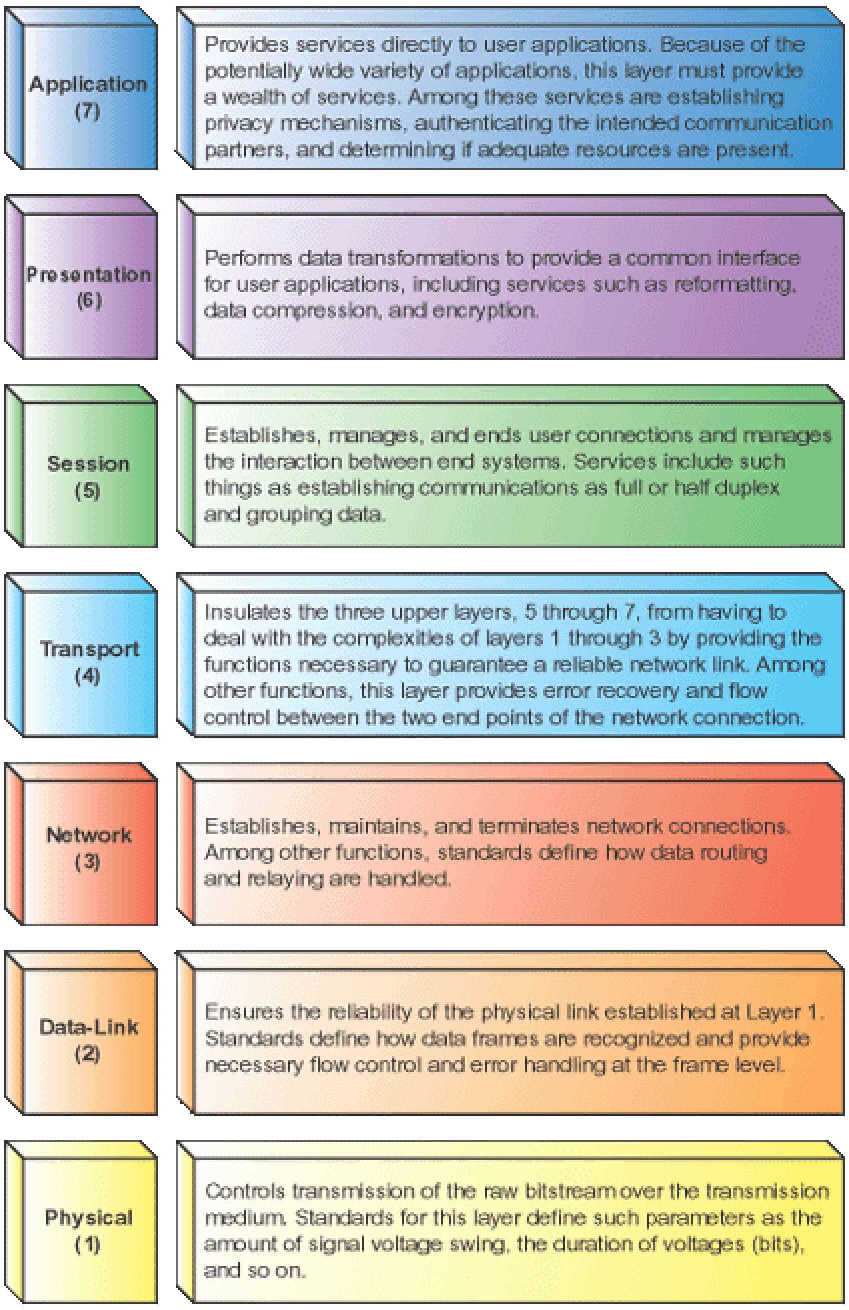

Application (7)
Provides services directly to user applications. Because of the potentially wide variety of applications, this layer must provide a wealth of services. Among these services are establishing privacy mechanisms, authenticating the intended communication partners, and determining if adequate resources are present.
Presentation (6)
Performs data transformations to provide a common interface for user applications, including services such as reformatting, data compression, and encryption.
Session (5)
Establishes, manages, and ends user connections and manages the interaction between end systems. Services include such things as establishing communications as full or half duplex and grouping data.
Transport (4)
Insulates the three upper layers, 5 through 7, from having to deal with the complexities of layers 1 through 3 by providing the functions necessary to guarantee a reliable network link. Among other functions, this layer provides error recovery and flow control between the two end points of the network connection.
Network (3)
Establishes, maintains, and terminates network connections. Among other functions, standards define how data routing and relaying are handled.
Data-Link (2)
Ensures the reliability of the physical link established at Layer 1. Standards define how data frames are recognized and provide necessary flow control and error handling at the frame level.
Physical (1)
Controls transmission of the raw bitstream over the transmission medium. Standards for this layer define such parameters as the amount of signal voltage swing, the duration of voltages (bits), and so on.

Figure 29: OSI Model – Layer Description

- TCP/IP model is the implementation of this model (simplified)
- Less rigid in enforcement of layers (can jump/ignore layers)
- Security protocols have been established at each layer:
  - Application: app-dependent
  - Transport: TLS
  - Internet: IPSec

– Link: WEP, WPS (wireless), 802.1x

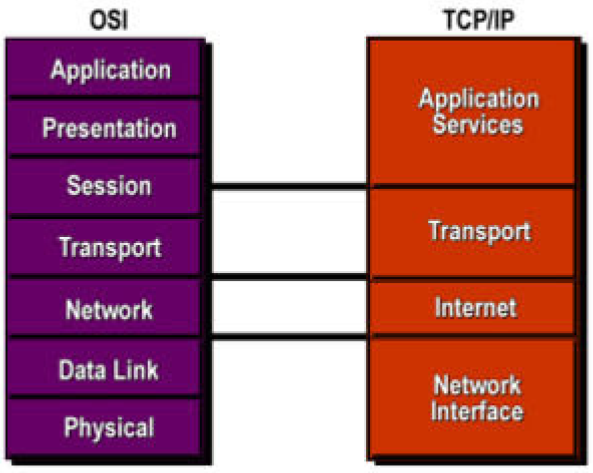
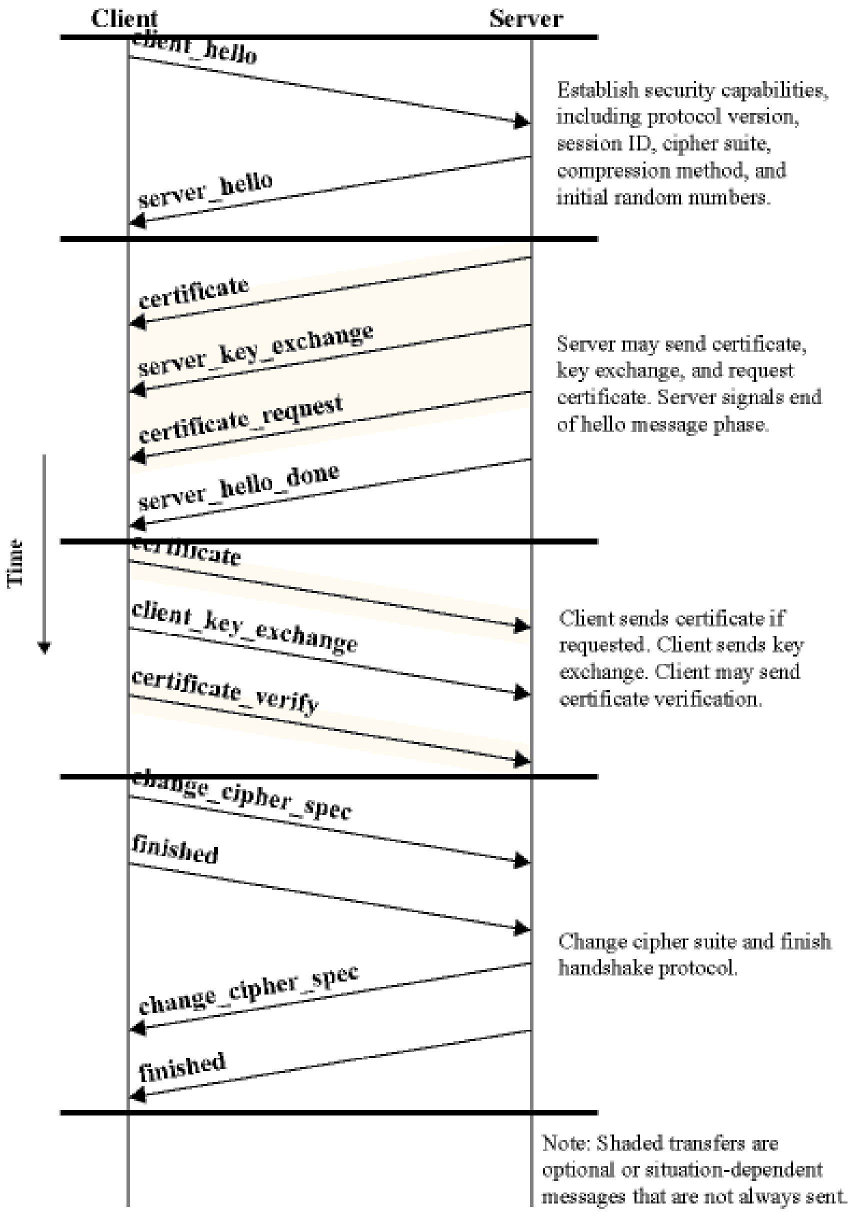


Figure 30: OSI comparison with TCP/IP

18.2. TLS Handshake



Client
Server
client_hello
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.
server_hello
certificate
server_key_exchange
certificate_request
server_hello_done
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Change cipher suite and finish handshake protocol.
Note: Shaded transfers are optional or situation-dependent messages that are not always sent.

Figure 14.6 Handshake Protocol Action

Figure 31: TLS Handshake

A simple connection example follows, illustrating a handshake where the server (but not the client) is authenticated by its certificate: (see http://en.wikipedia.org/wiki/Transport_Layer_Security#TLS_handshake, another summary: http://4orensics.wordpress.com/2011/10/21/ssl-in-a-nutshell/)

1. Negotiation phase:
   - A client sends a `ClientHello` message specifying the highest TLS protocol

version it supports, a random number, a list of suggested CipherSuites and suggested compression methods. If the client is attempting to perform a resumed handshake, it may send a session ID.

  - The server responds with a `ServerHello` message, containing the chosen protocol version, a random number, CipherSuite and compression method from the choices offered by the client. To confirm or allow resumed handshakes the server may send a session ID. The chosen protocol version should be the highest that both the client and server support. For example, if the client supports TLS1.1 and the server supports TLS1.2, TLS1.1 should be selected; SSL 3.0 should not be selected.
  - The server sends its `Certificate` message (depending on the selected cipher suite, this may be omitted by the server).
  - The server sends its `ServerKeyExchange` message (depending on the selected cipher suite, this may be omitted by the server). This message is sent for all DHE and DH_anon ciphersuites.
  - The server sends a ServerHelloDone message, indicating it is done with handshake negotiation.
  - The client responds with a `ClientKeyExchange` message, which may contain a PreMasterSecret, public key, or nothing. (Again, this depends on the selected cipher.) This PreMasterSecret is encrypted using the public key of the server certificate.
  - The client and server then use the random numbers and PreMasterSecret to compute a common secret, called the "master secret". All other key data for this connection is derived from this master secret (and the client- and server-generated random values), which is passed through a carefully designed pseudorandom function.

2. The client now sends a `ChangeCipherSpec` record, essentially telling the server, "Everything I tell you from now on will be authenticated (and encrypted if encryption parameters were present in the server certificate)." The ChangeCipherSpec is itself a record-level protocol with content type of 20.
   - Finally, the client sends an authenticated and encrypted `Finished` message, containing a hash and MAC over the previous handshake messages.
   - The server will attempt to decrypt the client's Finished message and verify the hash and MAC. If the decryption or verification fails, the handshake is considered to have failed and the connection should be torn down.
3. Finally, the server sends a `ChangeCipherSpec`, telling the client,*Everything I tell you from now on will be authenticated (and encrypted, if encryption was negotiated).*
   - The server sends its authenticated and encrypted `Finished` message.
   - The client performs the same decryption and verification.
4. Application phase: at this point, the "handshake" is complete and the application protocol is enabled, with content type of 23. Application messages exchanged be-

tween client and server will also be authenticated and optionally encrypted exactly like in their Finished message. Otherwise, the content type will return 25 and the client will not authenticate.

Misc

- Client-authenticated TLS Handshake (not necessary if client is authenticated at the application level via a password login)
- Resumed TLS Handshake, used in Single Sign-On
- Session IDs & Session Tickets, etc.
- Packet byte details, message types, error codes, etc.
- Good Tutorials using wireshark:
  - http://www.moserware.com/2009/06/first-few-milliseconds-of-https.html
  - http://russell.ballestrini.net/how-to-capture-https-ssl-tls-packets-with-wi
  - http://blogs.msdn.com/b/openspecification/archive/2013/09/11/ms-rdpeudp-glan
    aspx
  - http://en.wikiversity.org/wiki/Wireshark/HTTPS

18.3. TLS algorithm suite

Code	Algorithm	Notes
`TLS_RSA`	RSA	static RSA, being deprecated
`TLS_DH_RSA`	Diffie-Hellman RSA	
`TLS_DHE_RSA`	Ephemeral Diffie-Hellman	Forward Security
`TLS_ECDH`	Elliptic Curve Diffie-Hellman	
`TLS_ECDHE`	Ephemeral Elliptical Curve DH	Forward Security
`TLS_DH_anon`	Anonymous Diffie-Hellman	Anonymous (no authentication)
`TLS_PSK`	Pre-Shared Key	No key exchange: instead the key is presumed to already be shared (previously agreed upon or communicated via a secure channel)

Table 12: Authentication & Key Exchange Algorithms in TLS 1.2

Cipher	Bits	Notes
`AES_CBC`	128/256	AES, Cipher Block Chaining
`AES_GCM`	128/256	AES, Galois Counter Mode
`AES_CCM`	128/256	AES, Counter with CBC-MAC (Message Authentication Code generated using CBC mode)

Table 13: Block Ciphers & Modes in TLS 1.2

Other supported Ciphers:

- Carmellia, a Festel-network based 128 bit block cipher developed by Mitsubishi/NTT (2000)
- SEED – 128 bit nested Feistel block cipher used in Korea (1998)
- ARIA – 128-bit SPN cipher used in Korea (2003)
- CHACHA20/POLY1305 – 256-bit stream ciphers
- Others have been recommended to no longer be used: RC4, Triple DES, etc.

18.4. TLS Versions & Problems

- SSL 1.0 (never released)
- SSL 2.0 (1995)
- SSL 3.0 (1996) – Secure Sockets Layer, by Taher Elgamal
- TLS 1.0 (1999), RFC 2246, broke backward compatibility, renamed
- TLS 1.1 (2006) – RFC 4346
- TLS 1.2 (2008) – Current version, RFC 5246; RFC 6176 (2011) removes SSL 2.0 compatibility; likely soon: removal of SSL 3.0 compatibility (POODLE)
- TLS 1.3 (draft) – Removal of static DH and RSA key exchange (see article below)

Problems:

- Renegotiation attack (2009) - All versions affected; most websites have dropped support for the feature, RFC 5746 fixes; allows an attacker to inject their own requests before the client's
- POODLE (Padding Oracle On Downgraded Legacy Encryption) attack (2014) on SSL 3.0: `https://www.openssl.org/~bodo/ssl-poodle.pdf`
- BEAST (2011) attack on CBC ciphers in SSL 3.0, TLS 1.0
- TLS 1.3 deprecating RSA handshake: `http://threatpost.com/tls-1-3-working-group-has-c` `105916`
- Many more attacks

18.5. Other Issues

Certificate Authorities

- X.509 standard for Public Key Infrastructure (PKI, 1988)
- Public Key management and infrastructure
- Hierarchical system of control: authorities issue trust
- Contrast with decentralized "web-of-trust" as with PGP/GPG: you issue trust
- Certificate authorities provide identification of servers, they do not issue or hold public keys

Perfect Forward Security

- Often, secret session keys (short-term, hours or minutes) are derived from public keys (long-term, years)
- Forward security ensures that if long-term keys are compromised, short-term session keys are still secure
- Compromised: breaking of someone's public RSA key, RSA key being stolen, RSA key being handed over to authorities via court order or illegal, extra-judicial tyranny
- If session data can be sniffed and stored, then future compromises will compromise historic communication
- Instead, compromise of a single key only compromises the data encrypted with that key
- Perfect Forward Security: the public keys used to exchange private keys are only used once per session, and they are not generated by any deterministic (predictable) algorithm
- Its also best to throw them away after use (attacker can capture public key, but entity cannot be forced to give up the private key)
- TLS:
  - Diffie-Hellman Key Exchange PFSs: DHE-RSA, DHE-DSS
  - Elliptic Curve Diffie-Hellman PFSs: ECDHE-RSA, ECDHE-ECDSA
  - Unfortunately, still not widely adopted (only about 13.8% as of Fall 2014)
- Other issues: Off-the-record messaging (opposite of nonrepudiation: deniable encryption for "whistle blowers")
- Perfect Illustration of why this is necessary: During the celebrity nude photo hack (`http://www.wired.co.uk/news/archive/2014-09/01/celebrity-photo-hack-icloud`), Apple was quoted as:

  > "It's entirely possible that the leaks have nothing to do with Apple's encryption of iCloud, which has up until now proven solid and stores photos with 'a minimum of 128-bit AES encryption' according to Apple. Only Apple holds the master key to decrypt iCloud data, which it has to do on occasion at the request of government agencies."

Examples: Chrome

- https://cse.unl.edu - TLS 1.2, AES_128_GCM, DHE_RSA, CA: InCommon Server (UNL Certificate Authority)
- https://www.unl.edu - TLS 1.0, AES_128_CBC, SHA1, DHE_RSA, CA: InCommon Server (note Chrome warning: not all resources were loaded with HTTPS)
- https://blackboard.unl.edu – TLS 1.0 AES_256_CBC, with SHA1 for message authentication and RSA
- https://reddit.com – TLS 1.2, AES_128_GCM, DHE_RSA, CA: Gandi Standard SSL
- https://godaddy.com – TLS 1.2, AES_256_CBC, with SHA1, RSA, CA: Go Daddy Secure CA
- https://github.com – TLS 1.2, AES_128_GCM, ECDHE_RSA, CA: DigiCert SHA2
- https://gmail.com – TLS 1.2, AES_128_GCM, ECDHE_ECDSA (PFS!!!!)

Part VI. Misc

19. Other Standards

- ASN.1
- 509.X
- HTTP Auth
- SPDY
- Why everything should be encrypted: https://citizenlab.org/2014/08/cat-video-and-the-

20. Other Topics

- Off the record messaging (http://en.wikipedia.org/wiki/Off-the-Record_Messaging) – protocol that provides plausible deniability (in contrast to non-repudiation) for the participants.
- Firewalls
- TOR (The Onion Router)
- Phishing
- RansomWare
- Identity Theft

- Social Engineering
- Pen(etration) testing
- Backdoors
- Code injection attacks: XSS attacks, buffer overflows, SQL injections, PHP eval injections
- Cryptocurrencies (bit coin)
- Homomorphic Encryption (lattice based: https://en.wikipedia.org/wiki/Lattice-based_cryptography)
- Differential Cryptanalsysis
- Crypto Coding Standard: https://cryptocoding.net/index.php/Cryptography_Coding_Standard

Physical Security

- Keyloggers
- RFID exploits
- Tamper proofing measures and counter measures (hardware with self-destruct upon tamper)

Tools

- Wireshark
- TrueCrypt
- Metasploit
- Decryption tools: http://www.tools4noobs.com/online_tools/decrypt/
- dsniff (mac ports: sudo port install dsniff +libnet -libnet11)
- inet, netstat
- LSFR
- RC4
  - http://en.wikipedia.org/wiki/RC4
  - Problems/potential attacks: http://blog.cryptographyengineering.com/2013/03/attack-of-week-rc4-is-kind-of-broken-in.html
- Digital Signatures
  - Non-repudiation
  - Merkle Signature (https://en.wikipedia.org/wiki/Merkle_signature_scheme)

21. Resources

- https://bettercrypto.org/

21.1. Java Resources

- bcrypt Java port: http://www.mindrot.org/projects/jBCrypt/
- Bouncy Castle: https://www.bouncycastle.org/

References

[1] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak sponge function family. http://keccak.noekeon.org/, 2015. [Online; accessed 10-February-2015].

[2] Mark Burnett. Today I Am Releasing Ten Million Passwords. https://xato.net/passwords/ten-million-passwords/, February 2015.