TMA4160 Cryptography

This is intended to be a summary for the course TMA4160 lectured in the fall of 2013 at NTNU, Trondheim, Norway. Important results are listed as well as ideas, connections and counter examples.

Contents

Introduction

The objective of cryptography is to ensure that two people can communicate over an insecure channel in a way that an interceptor can not understand what is said. The message being communicated is called the plaintext and the encrypted plaintext using a predetermined key is known as the ciphertext. The definition of a cryptosystem follows:

Definition. A cryptosystem is a five-tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where:

(1) $\mathcal{P}$ is a finite set of possible plaintexts.
(2) $\mathcal{C}$ is a finite set of possible ciphertexts.
(3) $\mathcal{K}$ is a finite set of possible keys called the keyspace.
(4) For each $\mathcal{K}$ there is an encryption rule $e_K \in \mathcal{E}$ and a corresponding decryption rule $d_K \in \mathcal{D}$. Each $e_K : \mathcal{P} \to \mathcal{C}$ and $d_K : \mathcal{C} \to \mathcal{P}$ satisfies $d_K(e_K(x)) = x$ for any $x \in \mathcal{P}$.

From this we note that e_K is an injective function, or else decryption would not be unique. In the case that $\mathcal{P} = \mathcal{C}$ the encryption function is a permutation.

Cryptosystem (Shift cipher)**.** $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$. *For* $0 \leq K \leq 25$ *define for* $(x, y) \in \mathbb{Z}_{26}$*:*

$$e_K(x) = x + K \mod 26$$

$$d_K(y) = y - K \mod 26$$

This is a particularily insecure system and is easy to crack. In average one needs 13 attemps by using brute force (exhaustive key search). We make two considerations:

1. The encryption and decryption functions should be efficiently computable.
2. An opponent seeing the the ciphertext should not be able to determine the key, nor the plaintext.

To make an exhaustive key search attempt infeasible one could increase the size of the keyspace. This is however in itself not enough to guarantee security.

Cryptosystem (Substitution cipher)**.** $\mathcal{P} = \mathcal{C} = \mathbb{Z}_{26}$. $\mathcal{K}$ *consists of all possible permutations of the elements in* $\mathbb{Z}_{26}$*. For each permutation* $\pi \in \mathcal{K}$ *define:*

$$e_\pi(x) = \pi(x)$$

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

where π^{-1} *is the inverse permutation of* π*.*

The number of keys in this case is 26! which is very, very large. There are however other more efficient methods to crack this cipher.

One can consider the shift cipher as a special case of the substitution cipher (using only 26 permutations). Another special case of the substitution cipher is the affine cipher:

Cryptosystem (Affine cipher)**.** $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$. *Let:*

$$\mathcal{K} = \{(a, b) \in \mathbb{Z}_{26} \times \mathbb{Z}_{26} : \gcd(a, 26) = 1\}$$

For $K = (a, b) \in \mathcal{K}$ define for $(x, y) \in \mathbb{Z}_{26}$:

$$e_K(x) = ax + b \mod 26$$

$$d_K(y) = a^{-1}(y - b) \mod 26$$

We need to have the greatest multiple requirement to make sure that the encryption function is injective. In other words, the equation $ax \equiv b \mod m$ has a unique solution if and only if $\gcd(a, m) = 1$.

The previous systems were monoalphabetic, in that each character is mapped to the same character everytime. We now present a polyalphabetic system, namely the Vigeniere cipher.

Cryptosystem (Vigeniere cipher)**.** *Let m be a positive integer and set $\mathcal{P} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_{26})^m$. For $K = (k_1, k_2, \cdots, k_m)$ define:*

$$e_K(x_1, x_2, \cdots, x_m) = (x_1 + k_1, x_2 + k_2, \cdots, x_m + k_m) \mod 26$$

$$d_K(y_1, y_2, \cdots, y_m) = (y_1 - k_1, y_2 - k_2, \cdots, y_m - k_m) \mod 26$$

Since 26^m may be a very large number, exhaustive key search is infeasible.

We now present a polyalphabetic version of the affine cipher. Instead of using one linear congruence we use a system of linear congruences represented by a matrix. Using some linear algebra we see that if the system is to be well-defined the matrix A representing the system of equations must be invertible. In the context of $\mathbb{Z}_{26}$ this can be stated as: A is invertible if and only if $\gcd(\det(A), 26) = 1$.

Cryptosystem (Hill cipher)**.** *Let $m \geq 2$ be an integer, set $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_{26})^m$, and let:*

$$\mathcal{K} = \{m \times m \text{ invertible matrices over } \mathbb{Z}_{26}\}$$

For a $K \in \mathcal{K}$:

$$e_K(x) = xK$$

$$d_K(y) = yK^{-1}$$

All of the systems above are substitution ciphers. The next does not substitute any characters, but rearranges them using a permutation.

Cryptosystem (Permutation cipher)**.** *Let m be a positive integer, set $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_{26})^m$, and let $\mathcal{K}$ be all the permutations of the set $\{1, 2, \cdots, m\}$. For a key π, define:*

$$e_K(x_1, x_2, \cdots, x_m) = (x_{\pi(1)}, x_{\pi(2)}, \cdots, x_{\pi(m)})$$

$$d_K(y_1, y_2, \cdots, y_m) = (y_{\pi^{-1}(1)}, y_{\pi^{-1}(2)}, \cdots, y_{\pi^{-1}(m)})$$

One can show that this is a special case of the Hill cipher with the permutations stored in an $m \times m$ matrix, called a permutation matrix.

All systems above are block ciphers. That is, the ciphertext is generated by a fixed key on successive blocks of plaintext. Another way to approach this is to use stream ciphers. The stream cipher uses a keystream $z = z_1 z_2 \cdots$ to encrypt a given plaintext $x = x_1 x_2 \cdots$ in the following way: $y = e_{z_1}(x_1) e_{z_2}(x_2) \cdots$.

Definition. A synchronous stream cipher is a tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{L}, \mathcal{E}, \mathcal{D})$ together with a function g, such that:

(1) $\mathcal{P}$ is a finite set of possible plaintexts.

(2) $\mathcal{C}$ is a finite set of possible ciphertexts.
(3) $\mathcal{K}$ is a finite set of possible keys called the keyspace.
(4) $\mathcal{L}$ is a finite set called the keystream alphabet
(5) g is the keystream generator which takes a key K as input and generates an infinite string $z_1 z_2 \cdots$ called the keystream where $z_i \in \mathcal{L}$ for all $i \geq 1$.
(6) For each $z \in \mathcal{L}$ there is an encryption rule $e_z \in \mathcal{E}$ and a corresponding decryption rule $d_z \in \mathcal{D}$. Each $e_z : \mathcal{P} \to \mathcal{C}$ and $d_z : \mathcal{C} \to \mathcal{P}$ satisfies $d_z(e_z(x)) = x$ for any $x \in \mathcal{P}$.

For example it can be seen that the Vigeniere cipher is a synchronous stream cipher with a keystream (with a period!) generated from the key.

If we instead of the alphabet now consideres a binary alphabet one can think of $\mod 2$ as the exclusive or-operation. One way to generate a keystream is by using a linear recurrence of degree m. Let $(k_1, \cdots, k_m) \in \mathbb{Z}_2$ and set $z_i = k_i$, and generate:

$$z_{i+m} = \sum_{j=0}^{m-1} c_j z_{i+j} \mod 2, \quad i \geq 1$$

where the c_j's are specified binary constants. In this case the key K consists of the $2m$ values c_j and k_j. One wishes to choose the c_j in such a way that one gets a periodic keystream of period $2^m - 1$; That is a very long period. This is desirable as a short keystream may leave the system vulnurable.

As an example of a non-synchronous keystream cipher we have the autokey cipher. The keyspace is very small which makes the cipher insecure.

Cryptosystem (Autokey cipher)**.** *Let $\mathcal{P} = \mathcal{C} = \mathcal{K} = \mathcal{L} = \mathbb{Z}_{26}$, let $z_1 = K$ and define $z_i = x_{i-1}$ for all $i \geq 2$. For $0 \leq z \leq 25$, define:*

$$e_z(x) = (x + z) \mod 26$$
$$d_z(y) = (y - z) \mod 26$$

Cryptanalysis

Kerckhoff's principle states that the opponent knows what kind of system is being used. One would not want to base the security on the opponent not knowing as this will not provide very good security. There are several attack models on cryptosystems. The most common are:

1. **Ciphertext only attack**
   The opponent possesses a string of ciphertext, $\mathbf{y}$.
2. **Known plaintext attack**
   The opponent possesses a string of plaintext, $\mathbf{x}$ and the corresponing ciphertext $\mathbf{y}$.
3. **Chosen plaintext attack**
   The opponent has obtained temporary access to the encryption machinery. Hence he can choose a plaintext string $\mathbf{x}$ and construct the corresponding ciphertext $\mathbf{y}$.
4. **Chosen ciphertext attack**
   The opponent has obtained temporary access to the decryption machinery. Hence he can choose a ciphertext string $\mathbf{y}$ and construct the corresponding plaintext $\mathbf{x}$.

In each case the object of the opponent (adversary) is to obtain the secret key. In this way he can decrypt any ciphertext he wishes.

We consider first the weakest attack, namely the ciphertext only attack. We assume that the plaintext is written in English and that there are no punctuation or spaces (makes the analysis easier). Many attacks on systems uses statistical properties of the English language. For example there exists tables of the statistically most used letters, and furthermore usual digrams and trigrams.

Affine cipher: To crack the affine cipher we emply frequency analysis and trial and error. For example by counting the two most frequent letters in the ciphertext and pairing it with the most statistical frequent letters. Now we try these two guesses and solving the two linear congruences occuring from this. With a solution one needs to check if it matches the gcd-condition. If we obtain a legal pair we check if this results in a readable decrypted text. If not, try again.

Substitution cipher: To crack this cipher we again use frequency analysis and match up the most frequent letter pairs. Now continue this process by looking at digrams and/or trigrams. At this stage it is essentially a matter of "being clever". Continuing this process and guessing at the decryption/encryption pairs one might suddenly realise that some parts of the text is readable and then guess the rest from that.

Vigeniere cipher: This is the first cryptanalysis that requires some slightly hard work. We first want to find the keyword length m. There are a couple of techniques for doing this. Among them is the Kasiski test: The idea is that two identical segments of plaintext is engrypted to the same ciphertext if they are δ positions apart where $\delta \equiv 0 \mod m$. In practice we search for several such distances $\delta_1, \delta_2, \cdots$. Then $m|\delta_i$ and therefore $m|\gcd(\delta_1, \delta_2, \cdots)$. One could pair this method with the index of coincidence method.

Definition. Suppose $x = x_1x_2\cdots x_n$ is a string of n alphabetic characters. The index of coincidence of x, denoted $I_c(x)$, is defined to be the probability that two random elements of x are identical.

Denote the frequencies of the alphabetic characters in $\mathbf{x}$ by $f_0, f_1, \cdots, f_{25}$ respectively in alphabetic order. One can choose two elements of $\mathbf{x}$ in $\binom{n}{2}$ ways. For each i there are $\binom{f_i}{2}$ ways of choosing both element to be i. Hence:

$$I_c(\mathbf{x}) = \frac{\sum_{i=0}^{25} \binom{f_i}{2}}{\binom{n}{2}} = \frac{\sum_{i=0}^{25} f_i(f_i-1)}{n(n-1)}$$

By denoting the statistical frequencies of English language by $p_0, p_1, \cdots, p_{25}$, we would expect that a string of English language has:

$$I_c(\mathbf{x}) = \sum_{i=0}^{25} p_i^2 = 0.065$$

This estimate holds for any monoalphabetic cipher.

Given a ciphertext $\mathbf{y}$ constructed from a Vigeniere cipher we split it up into m substrings $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_m$ in the way such that $\mathbf{y}_i = y_i y_{m+i} y_{2m+1} \cdots$. If m is the keyword length one would expect that $I_c(\mathbf{y}_i)$ is roughly equal to 0.065. On the other hand on a completely random string we would have: $I_c = 26\frac{1}{26^2} = 0.038$.

Assume that the keyword length m is found, we still need to find the key. Assume that $f_0, f_1, \cdots, f_{25}$ is the frequencies in the string $\mathbf{y}_i$. Also the length of the string is given by $n' = n/m$. Then the probability distribution of the 26 letters in $\mathbf{y}_i$ is given by $f_0/n', f_1/n', \cdots, f_{25}/n'$. Now since the substring $\mathbf{y}_i$ is just a shift cipher one would expect that the shifted distribution above with the correct value would have an index of coincidence at about 0.065. I.e. If

$$M_g = \sum_{i=0}^{25} \frac{p_i f_{i+g}}{n'}$$

So if $g = k_i$, then $M_g \approx 0.065$ and k_i is most likely a key value.

Hill cipher: In this case we assume a known plaintext attack. Let us first assume that the value of m is known and suppose that the adversary has m distinct plaintext-ciphertext pairs, say $(x_{1,j}, x_{2,j}, \cdots, x_{m,j})$ and $(y_{1,j}, y_{2,j}, \cdots, y_{m,j})$ for $1 \leq j \leq m$ such that $y_j = e_K(x_j)$. By defining two matrices $X = (x_{i,j})$ and $Y = (y_{i,j})$ we get the equation $Y = XK$ where the matrix K is the $m \times m$ unknown key. If the matrix X is invertible it is easy for the adversary to find K. If not he must try another set of plaintext-ciphertext pairs.

LSFR stream cipher: MEH

Some concepts of security:

1. **Computational security**
This notion concerns the computational aspect of breaking a cryptosystem. One might for instance say that a cryptosystem if computationally secure if the best algorithm for solving it uses at least N operations where N is a predetermined large number. No known cryptosystem is secure under this notion. One can talk about computational security against a specific attack, however this does not rule out security against another attack.
2. **Provable security**
Another way to define security is by reduction. In other words showing that breaking the cryptosystem is at least as hard as solving a well-studied thought to difficult to solve problem. This is not an absolute proof of security, but relates the idea of security to some other problem. It is similar to showing that a problem is NP-complete.
3. **Unconditional security**
This is a strong form of security. It assumes that even if the adversary has access to infinite computing power, he can not break the system.

We now discuss the concept of perfect secrecy. Assume that the cryptosystem $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ is specified and that a particular key $K \in \mathcal{K}$ is used for only one encryption. We assume that there is a probability distribution on the plaintext space. Thus the plaintext element defines a random variable $\mathbf{x}$. The a priori probabilty that the plaintext x occurs is then given by $\Pr[\mathbf{x} = x]$. We also assume that the key is chosen by using some fixed probability distribution (can be uniform!). This implies a new random variable $\mathbf{K}$ and the probability of choosing the key K is $\Pr[\mathbf{K} = K]$. The key is chosen before the plaintext so we make the reasonable assumption that the two random variables are independent.

These two distributions induce a third distribution on $\mathcal{C}$. Thus a third random variable $\mathbf{y}$ is considered. By defining for a key $K \in \mathcal{K}$ the set $C(K) = \{e_K(x) : x \in \mathcal{P}\}$ of possible ciphertexts given the key K, one can compute for any $y \in \mathcal{C}$ the probability of choosing y:

$$\Pr[\mathbf{y} = y] = \sum_{\{K : y \in C(K)\}} \Pr[\mathbf{K} = K] \Pr[\mathbf{x} = d_K(y)]$$

Also for any $y \in \mathcal{C}, x \in \mathcal{P}$:

$$\Pr[\mathbf{y} = y | \mathbf{x} = x] = \sum_{\{K : x = d_K(y)\}} \Pr[\mathbf{K} = K]$$

which is the probability that y is the ciphertext given that x is the plaintext. By using Bayes' theorem we have:

$$\Pr[\mathbf{x} = x | \mathbf{y} = y] = \frac{\Pr[\mathbf{x} = x] \sum_{\{K : x = d_K(y)\}} \Pr[\mathbf{K} = K]}{\sum_{\{K : y \in C(K)\}} \Pr[\mathbf{K} = K] \Pr[\mathbf{x} = d_K(y)]}$$

Informally perfect secrecy means that the adversary can not obtain any information about the plaintext from observing the ciphertext. In a more formal sense:

Definition. A cryptosystem has perfect secrecy if $\Pr[x|y] = \Pr[x]$ for all $x \in \mathcal{P}, y \in \mathcal{C}$. I.e. the a posteriori probability that the plaintext is x, given that the ciperhtext y is observed, is identical to the a priori probability that the plaintext is x.

It can be shown that the Shift Cipher obtains perfect secrecy. This is intuitively clear as given any $y \in \mathbb{Z}_{26}$ the plaintext can be any element of $\mathbb{Z}_{26}$. This is partly due to the uniform distribution on the spaces.

Let us assume that $\Pr[x] > 0$ (or else perfect secrecy is trivially satisfied) and furthermore assume that $\Pr[y] > 0$ (if not, omit the element from the space). For $\Pr[x] > 0$ Bayes' theorem implies that the condition $\Pr[x|y] = \Pr[x]$ is equivalent to $\Pr[y|x] = \Pr[y]$ for any $y \in \mathcal{C}$.

Fix $x \in \mathcal{P}$, for each $y \in \mathcal{C}$ we have $\Pr[y|x] = \Pr[y] > 0$. Hence for each $y \in \mathcal{C}$ there must be at least one key K such that $e_K(x) = y$. This implies that $|\mathcal{K}| \geq |\mathcal{C}|$, and in any cryptosystem we must have $|\mathcal{C}| \leq |\mathcal{P}|$ to obtain injectivity. Now for a special case of this:

Theorem (Theorem 2.4)**.** *Suppose $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$ is a cryptosystem where $|\mathcal{K}| = |\mathcal{C}| = |\mathcal{P}|$. Then the cryptosystem provides perfect secrecy if and only if every key is used with equal probability $1/|\mathcal{K}|$, and for every $x \in \mathcal{P}, y \in \mathcal{C}$ there is a unique key K such that $e_K(x) = y$.*

Proof. By above $|\mathcal{K}| \geq |\mathcal{C}|$, but we assume equality, hence $|\mathcal{K}| = |\mathcal{C}| = |\{e_K(x) : K \in \mathcal{K}\}|$ which implies that there does not exist two different keys for a given ciphertext by the injectivity of the encryption function. By Bayes' theorem and the perfect secrecy condition it follows that the key space is uniformly distributed. □

As another example of a cryptosystem with perfect secrecy we present the one-time pad.

Cryptosystem (One-time pad)**.** *Let $n \geq 1$ be an integer, and take $\mathcal{P} = \mathcal{C} = \mathcal{K} = (\mathbb{Z}_2)^n$. For $K \in (\mathbb{Z}_2)^n$ define:*

$$e_K(x) = x \oplus K$$
$$d_K(y) = y \oplus K$$

It is easy to see using Theorem 2.4 that the system has perfect secrecy. It is unconditionally secure, but there are several disadvantages. For example the key is at least as big as the message you want to send. The problem occurs because you need to use a new key for every message to not compromise the security of the system.

Introduction

Most modern-day block ciphers are product ciphers. Often they use a sequence of permutation and substitution operations and they are usually iterated. Therefore the ciphers needs a round function and a key schedule. The encryption of the plaintext will be processed through Nr similar rounds.

Let K be a random binary key of some specified length. We use the key to construct Nr round keys: $K^1, K^2, \cdots, K^{Nr}$ by using a fixed, public algorithm. The list of these keys form the key schedule. The round function takes as input both a round key (K^r) and the current state (denoted w^{r-1}). The next state is then defined as $w^r = g(w^{r-1}, K^r)$ and the initial state w^0 is just the plaintext x. The ciphertext, y, is defined to be the state after all Nr rounds have been performed. Schematically this is carried out like this:

$$\begin{aligned} w^0 &\leftarrow x \\ w^1 &\leftarrow g(w^0, K^1) \\ w^2 &\leftarrow g(w^1, K^2) \\ &\vdots \quad \vdots \\ w^{Nr-1} &\leftarrow g(w^{Nr-2}, K^{Nr-1}) \\ w^{Nr} &\leftarrow g(w^{Nr-1}, K^{Nr}) \\ y &\leftarrow w^{Nr} \end{aligned}$$

Obviously also here the function g needs to be injective in the first argument in order for decryption to be possible. I.e. there exists a function g^{-1} such that $g^{-1}(g(w, y), y) = w$ for all w, y. The decryption process is carried out by reversing the above scheme.

Substitution-Permutation networks and AES

See book.

Modes of operation

Modes of operation are different ways to apply a block cipher (specifically developed for DES). There are six modes that are approved, or under consideration as standards for AES.

Electronic codebook mode (ECB):
This is the naive use of a block cipher: Given a sequence $x_1x_2\cdots$ of plaintext blocks, each block is encrypted with a key K resulting in the ciphertext $y_1y_2\cdots$. An obvious weakness is that identical plaintext blocks are mapped to identical ciphertext blocks. If the plaintext has "low entropy" this implies that not much information is hidden.

Cipher block chaining mode (CBC):
In CBC each ciphertext block y_i is x-ored with the next plaintext block x_{i+1} before being encrypted with the key K. We start with an initialization vector of same length as a plaintext block, denoted IV, and define $y_0 = IV$, then:

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

This mode ensures that a change in a plaintext block means that all subsequent ciphertext blocks are changed as well (obtaining so-called diffusion). This means that CBC-mode is useful for authentication (see Chapter 4). Decryption is given by:

$$x_i = d_K(y_i) \oplus y_{i-1}, \quad i \geq 1$$

Output feedback mode (OFB):
In OFB, a keystream is generated and then x-ored with the plaintext (i.e. a synchronous stream cipher). The keystream is produces by repeteadly encrypting an initialization vector IV. Define $z_0 = IV$ and then compute the keystream $z_1z_2\cdots$ by:

$$z_i = e_K(z_{i-1}), \quad i \geq 1$$

The plaintext $x_1x_2\cdots$ is then encrypted by:

$$y_i = x_i \oplus z_i$$

Decryption is simple:

$$x_i = y_i \oplus z_i$$

Note that the encryption function is used both in encrypting and decrypting in this mode.

Cipher feedback mode (CFB):
CFB also generates a keystream for use in a synchronous stream cipher. Again we start with an initialization vector $y_0 = IV$ and produce a keystream element z_i by encrypting the previous ciphertext block:

$$z_i = e_K(y_{i-1}), \quad i \geq 1$$

We use the same formulas as in OFB mode to encrypt and decrypt.

Counter mode:
Counter mode is similar to OFB, but differs in how the keystream is generated. Let the length of the plaintext block be m. In counter mode we choose a counter denoted ctr which is a bitstring of length m. Then we construct a sequence of bitstrings of length m denoted

$T_1, T_2, \cdots$ as follows:

$$T_i = ctr + i - 1 \mod 2^m, \quad i \geq 1$$

Encryption of the plaintext $x_1 x_2 \cdots$ is done by:

$$y_i = x_i \oplus e_K(T_i), \quad i \geq 1$$

The keystream is obtained by encrypting the sequence of counters using the key K. As with OFB one can construct the keystream independently of the plaintext, but the counter mode can be made far more effective by using parallellism since it does not need to iteratively compute a sequence of encryptions.

Counter with cipher-block chaining mode (CCM):

This is a combination of the counter mode for encryption and the CBC for authentication.

Hash functions and data integrity

Hash functions can be used to provide integrity to data. One might think of a hash function as a way to construct a fingerprint of some data set. So if the data is changed, so is the fingerprint. This fingerprint is known as a *message digest*, which usually is a fairly short binary string; 160 bits is a common choice. Let h be a hash function and x some data, then $y = h(x)$ is the message digest. Now suppose that y is stored in a safe place, but x is not. If x is changed to x' one would hope that $h(x') \neq y$. If this is the case one have a way to detect change in the data. This can with some modification (using a secret key) also be used to make sure that a message between two people is not altered by using so-called *Message Authentication Codes* (MACs for short).

Definition. A *hash family* is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ where the following holds:

(1) $\mathcal{X}$ is a set of possible messages.
(2) $\mathcal{Y}$ is a set of possible message digests or authentication tags.
(3) $\mathcal{K}$, the keyspace, is a finite set of possible keys.
(4) For each $K \in \mathcal{K}$ there is a hash function $h_k \in \mathcal{H}$ where $h_k : \mathcal{X} \to \mathcal{Y}$.

In this definition $\mathcal{X}$ can be finite or infinite while $\mathcal{Y}$ always is finite. If $\mathcal{X}$ is finite, the hash function is called a *compression function*. In this situasion we always assume that $|\mathcal{X}| \geq |\mathcal{Y}|$ and often that $|\mathcal{X}| \geq 2|\mathcal{Y}|$.

A pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$ is called a *valid pair* under K if $h_K(x) = y$. Most of the chapter concerns how one might prevent an adversary from constructing valid pairs.

Let $\mathcal{F}^{\mathcal{X},\mathcal{Y}}$ be the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$ and suppose that $|\mathcal{X}| = N, |\mathcal{Y}| = M$. Then $|\mathcal{F}^{\mathcal{X},\mathcal{Y}}| = M^N$ and any hash family $\mathcal{F} \subset \mathcal{F}^{\mathcal{X},\mathcal{Y}}$ is denoted an $(N, M) - hashfamily$.

An *unkeyed hash function* is a function $h : \mathcal{X} \to \mathcal{Y}$. One can think of an unkeyed hash function as a hash-family with only one key.

Security of hash functions

Suppose that $h : \mathcal{X} \to \mathcal{Y}$ is an unkeyed hash function. Let $x \in \mathcal{X}$ and $y = h(x)$. One would in most cases very much like that the only way to obtain a valid pair (x, y) is by plugging an element in $\mathcal{X}$ and evaluating h at this point. There are three main problems concerned with hash functions that we would like to be "hard to solve".

The preimage problem:
Given: *A hash function* $h : \mathcal{X} \to \mathcal{Y}$ *and an element* $y \in \mathcal{Y}$.
Find: $x \in \mathcal{X}$ *such that* $h(x) = y$.

Given a (possible) message digest y this problems asks if $x \in \mathcal{X}$ can be found such that $h(x) = y$. If this problem can be solved a valid pair is obtained. A hash function for which this problem cannot be efficiently solved is often said to be *one-way or preimage-resistant*.

The second preimage problem:
Given: *A hash function* $h : \mathcal{X} \to \mathcal{Y}$ *and an element* $x \in \mathcal{X}$.
Find: $x' \in \mathcal{X}$ *such that* $x \neq x'$ *and* $h(x') = h(x)$.

Given a message x this problem asks if another message x' can be found such that both messages have the same message digest $h(x') = h(x)$. If this can be done $(x', h(x))$ is a valid pair. A hash function for which this problem cannot be efficiently solved is often said to be *second preimage resistant*.

The collision problem:
Given: *A hash function* $h : \mathcal{X} \to \mathcal{Y}$.
Find: $x, x' \in \mathcal{X}$ *such that* $x \neq x'$ *and* $h(x') = h(x)$.

This problem asks if two different messages x, x' can be found such that $h(x') = h(x)$. A solution to this problems gives two valid pairs. A hash function for which this problem cannot be solved efficiently is often said to be *collision resistant*.

One would like for the only efficient way to obtain $h(x)$ is to actually evaluate h at x. This should remain true even though many values of the function is known. By having these requirements one can see that a linear congruence will not be a good hash function as it is too predictable. We therefore introduce the *random oracle model*. In this model a hash function h is chosen at random from a $\mathcal{F}^{\mathcal{X},\mathcal{Y}}$ and we are only allowed oracle access to the function h. That is we have to "ask" the oracle for each value of $h(x)$. There is obviously not possible to do this in real life, but we hope to the design a hash function that will behave in a similar manner.

Theorem (4.1)**.** *Suppose that* $h \in \mathcal{F}^{\mathcal{X},\mathcal{Y}}$ *is chosen randomly and let* $\mathcal{X}_0 \subset \mathcal{X}$. *Suppose that* $h(x)$ *has been determined by querying an oracle if and only if* $x \in \mathcal{X}_0$. *Then* $\Pr[h(x) = y] = 1/M$ *for all* $x \in \mathcal{X} \setminus \mathcal{X}_0$ *and all* $y \in \mathcal{Y}$.

This is essentially just saying that no matter how many values $h(x)$ that have been computed in the random oracle model, the ones remaining outside that set are still uniformly distributed in $\mathcal{Y}$. In a way it is an independence property. Note that the probability is in fact conditional on h.

We now consider the complexity of the problems above. We will use randomized algorithms for this; They can make random choices while executing. A *Las Vegas algorithm* is a randomized algorithm which may terminate without an answer, but if the algorithm returns an answer it must be correct.

Suppose that $0 \leq \epsilon < 1$ is a real number. A randomized algorithm has worst-case success probability ϵ if for every problem instance the algorithm returns a correct answer with probability at least ϵ. It has an average-case success probability ϵ if the probability that the algorithm returns a correct answer averaged over all the problem instances of a specific size is ϵ.

We say that a Las Vegas algorithm is a (ϵ, Q)-algorithm if the average-case success probability is ϵ and the number of oracle queries is at most Q. The success probability is the average over all hash functions and all possible random choices of x, y if they are specified in the problem instance. We present trivial algorithms for each of the problem above:

FIND-PREIMAGE(h, y, Q):
choose any $\mathcal{X}_0 \subset \mathcal{X}$, $|\mathcal{X}_0| = Q$
for each $x \in \mathcal{X}_0$
  if $h(x) = y$
    return (x)
return ('failure')

Theorem (4.2)**.** *For any $\mathcal{X}_0 \subset \mathcal{X}$ with $|\mathcal{X}_0| = Q$ the average-case success probability of* FIND-PREIMAGE *is $\epsilon = 1 - (1 - 1/M)^Q$.*

Proof. Fix $y \in \mathcal{Y}$ and let $\mathcal{X}_0 = \{x_1, \cdots, x_Q\}$. For $1 \leq i \leq Q$ let E_i denote the event that $h(x_i) = y$. Then all E_i are independent since we are considering a random oracle model, and furthermore that $\Pr[E_i] = 1/M$ for all i. Therefore:

$$\Pr\left[\bigcap_{1 \leq i \leq Q} E_i\right] = 1 - \Pr\left[\overline{\bigcap_{1 \leq i \leq Q} E_i}\right] = 1 - \Pr\left[\bigcup_{1 \leq i \leq Q} \overline{E_i}\right] = 1 - \left(1 - \frac{1}{M}\right)^Q$$

Since this is a constant number independent of y it remains the same when averaging over all $y \in \mathcal{Y}$. □

When Q is small compared to M the success probability is approximately Q/M. The next algorithm is very similar and is analyzed similarly.

FIND-SECOND-PREIMAGE(h, x, Q):
$y \leftarrow h(x)$
choose $\mathcal{X}_0 \subset \mathcal{X} \setminus \{x\}$, $|\mathcal{X}_0| = Q - 1$
for each $x_0 \in \mathcal{X}_0$
  if $h(x_0) = y$
    return (x)
return ('failure')

Theorem (4.3)**.** *For any $\mathcal{X}_0 \subset \mathcal{X} \setminus \{x\}$ with $|\mathcal{X}_0| = Q - 1$ the success probability of* FIND-SECOND-PREIMAGE *is $\epsilon = 1 - (1 - 1/M)^{Q-1}$.*

FIND-COLLISION(h, Q):
choose $\mathcal{X}_0 \subset \mathcal{X}$, $|\mathcal{X}_0| = Q$
for each $x \in \mathcal{X}_0$
  do $y_x \leftarrow h(x)$
if $y_x = y_{x'}$ for some $x' \neq x$
  then return (x,x')
else return ('failure')

We use the birthday paradox to analyse this algorithm. This paradox states that in a group of 23 randomly chosen persons the probaility that at least two will share birthday is at least 1/2. It is of course not really a paradox, however it may be counter-intuitive. It is easy to see how this is relevant in the context of collisions in hash functions. If in FIND-COLLISION $Q = 23$ and $M = 365$ then we would have a success probability of at least 1/2. One can think of the algorithm as throwing Q balls into M bins and checking if some bins contains at least two balls.

Theorem (4.4)**.** *For any $\mathcal{X}_0 \subset \mathcal{X}$ with $|\mathcal{X}_0| = Q$ the success probability of* FIND-COLLISION *is*

$$\epsilon = 1 - \left(\frac{M-1}{M}\right)\left(\frac{M-2}{M}\right)\cdots\left(\frac{M-Q+1}{M}\right)$$

Proof. Let $\mathcal{X}_0 = \{x_1, \cdots, x_Q\}$ and for $1 \leq i \leq Q$ denote by E_i the event: $h(x_i) \notin \{h(x_1), \cdots, h(x_{i-1})\}$. Obviously $\Pr[E_1] = 1$ and by induction:

$$\Pr\left[E_i | \bigcup_{k=1}^{i-1} E_k\right] = \frac{M-i+1}{M}, \quad 2 \leq i \leq Q$$

So by independence the result follows. □

This theorem shows that the probability of finding no collisions is:

$$\prod_{i=1}^{Q-1}\left(1 - \frac{i}{M}\right)$$

If x is small then we have the estimate $1 - x \approx e^{-x}$ from the two first terms of the series expansion of e^{-x}. This implies that we get:

$$\prod_{i=1}^{Q-1}\left(1 - \frac{i}{M}\right) = \prod_{i=1}^{Q-1} e^{-i/M} = e^{-\sum_{i=1}^{Q-1} i/M} = e^{-\frac{Q(Q-1)}{2M}}$$

Therefore the probability of finding at least one collision ϵ is approximated by $\epsilon \approx 1 - e^{-\frac{Q(Q-1)}{2M}}$. This implies:

$$1 - \epsilon \approx e^{-\frac{Q(Q-1)}{2M}} \Leftrightarrow Q^2 - Q \approx 2M \ln\left(\frac{1}{1-\epsilon}\right)$$

By ignoring $-Q$ (for large Q the squared term dominates) and setting $\epsilon = 0.5$ we get $Q \approx 1.17\sqrt{M}$. So by hashing over $\sqrt{M}$ random elements of $\mathcal{X}$ yields a collision with a probability of 50%. Therefore the algorithm is a $(1/2, O(\sqrt{M}))$-algorithm. By using these estimates one can say that a message digest of 128 bits is the minimum acceptable length (birthday attack takes 2^{64} hashes). 160 bits or larger is recommended.

We have seen that in the random oracle model it is easier to solve **Collision** than any of the other two problems. Suppose that ORACLE-2ND-PREIMAGE is a (ϵ, Q)-algorithm solving **Second Preimage** for a particular fixed hash function h. By using the following algorithm one can reduce **Collision** to **Second Preimage** for arbitrary hash functions:

COLLISION-TO-SECOND-PREIMAGE(h):
external ORACLE-2ND-PREIMAGE
choose $x \in \mathcal{X}$ uniformly at random
if ORACLE-2ND-PREIMAGE$(h, x) = x'$

then return (x,x')
else return ('failure')

What this means is that if one can solve the problem **Second Preimage** one gets a solution of **Collision** "for free" using the algorithm above. Note that this does not make any assumption on h. Therefore one can say that collision resistance implies second preimage resistance.

Can **Collision** be reduced to **Preimage**? In some special situations this is the case. Here we will assume that both $\mathcal{X}, \mathcal{Y}$ are finite sets with $|\mathcal{X}| \geq 2|\mathcal{Y}|$ and that ORACLE-PREIMAGE is a $(1, Q)$-algorithm for **Preimage**. We present an algorithm for this reduction:

COLLISION-TO-PREIMAGE(h):
external ORACLE-PREIMAGE
choose $x \in \mathcal{X}$ uniformly at random
$y \leftarrow h(x)$
if (ORACLE-PREIMAGE$(h, x) = x'$) **and** $(x' \neq x)$
then return (x,x')
else return ('failure')

Theorem (4.5)**.** *Suppose $h : \mathcal{X} \to \mathcal{Y}$ is a hash function where $|\mathcal{X}|$ and $|\mathcal{Y}|$ are finite and $|\mathcal{X}| \geq 2|\mathcal{Y}|$. Suppose* ORACLE-PREIMAGE *is a $(1, Q)$-algorithm for **Preimage** for the fixed hash function h. Then* COLLISION-TO-PREIMAGE *is a $(1/2, Q + 1)$-algorithm for **Collision** for the fixed hash function h.*

Proof. □

Iterated hash functions

Say **compress** is a compression function. We now study a technique to extend the function to a hash function with an infinite domain. Such a hash function is called an *iterated hash function*. We assume that the input and output are bitstrings. So let $\mathbf{compress} : \{0,1\}^{m+t} \to \{0,1\}^m$ we will then construct:

$$h : \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \to \{0,1\}^l$$

The evaluation of h consists of the following three steps:

Preprocessing step

Given an input string x with $|x| \geq m + t + 1$, construct y using a public fixed algorithm such that $|y| \equiv 0 \mod t$ and denote:

$$y = y_1 \,\|\, y_2 \,\|\, \cdots \,\|\, y_r$$

where $|y_i| = t$ for $1 \leq i \leq r$.

Processing step

Let IV be a public initial bitstring of length m. Then compute:

$$\begin{aligned} z_0 &\leftarrow \mathrm{IV} \\ z_1 &\leftarrow \mathbf{compress}(z_0 \,\|\, y_1) \\ z_2 &\leftarrow \mathbf{compress}(z_1 \,\|\, y_2) \\ &\vdots \quad \vdots \\ z_r &\leftarrow \mathbf{compress}(z_{r-1} \,\|\, y_r) \end{aligned}$$

Output transformation (optional)

Let $g : \{0,1\}^m \to \{0,1\}^l$ be a public function. Define $h(x) = g(z_r)$.

A common way to preprocess is to use a padding function **pad** to construct the string $y = x \,\|\, \mathbf{pad}(x)$. A typical way to design such a function is to use $|x|$ and append enough zero bits to make the string y of desired length. It is important that the mapping $x \mapsto y$ is injective. If not it would be possible to find x, x' so that $y = y'$ which would imply that the hash function is not collision resistant. For the same reason $|y| = rt \geq |x|$.One way to construct a iterated hash function is to use the Merkle-Damgård construction.

Suppose again that $\mathbf{compress} : \{0,1\}^{m+t} \to \{0,1\}^m$ where $t \geq 1$, we will use this function to construct a collision resistant hash function

$$h : \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \to \{0,1\}^m$$

Suppose that $t \geq 2$ and that $|x| = n \geq m + t + 1$. Write $x = x_1 \,\|\, x_2 \,\|\, \cdots \,\|\, x_k$, where $|x_i| = t - 1$ for $1 \leq i \leq k - 1$ and $|x_k| = t - 1 - d$ for a $0 \leq d \leq t - 2$.

MERKLE-DAMGÅRD(x):
external Compress: $\{0,1\}^{m+t} \to \{0,1\}^m$, where $t \geq 2$
$n \leftarrow |x|$
$k \leftarrow \lceil n/(t-1) \rceil$
$d \leftarrow k(t-1) - n$
for $i = 1$ **to** $k-1$
  do $y_i \leftarrow x_i$
$y_k \leftarrow x_k \,\|\, 0^d$
$y_{k+1} \leftarrow$ the binary representation of d
$z_1 \leftarrow 0^{m+1} \,\|\, y_1$
$g_1 \leftarrow \textbf{Compress}(z_1)$
for $i = 1$ **to** k
  do $\begin{cases} z_{i+1} \leftarrow g_i \,\|\, 1 \,\|\, y_{i+1} \\ g_{i+1} \leftarrow \textbf{compress}(z_{i+1}) \end{cases}$
$h(x) \leftarrow g_{k+1}$
return$(h(x))$

Theorem (4.6)**.** *Suppose* ***Compress****:* $\{0,1\}^{m+t} \to \{0,1\}^m$ *is a collision resistant compression function where* $t \geq 2$*. Then the function:*

$$h : \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \to \{0,1\}^m$$

constructed in the algorithm MERKLE-DAMGÅRD *is a collision resistant hash function.*

Proof. Proof by cases: $|x| \not\equiv |x'| \mod t-1$ and $|x| \equiv |x'| \mod t-1 \Rightarrow |x| = |x'|$ or $|x| \neq |x'|$. □

If $t = 1$ we get another algorithm. Suppose $|x| = n \geq m+2$ and define the function f as follows $f(0) = 0, f(1) = 01$. Then:

MERKLE-DAMGÅRD2(x):
external Compress: $\{0,1\}^{m+1} \to \{0,1\}^m$
$n \leftarrow |x|$
$y \leftarrow 11 \,\|\, f(x_1) \,\|\, f(x_2) \,\|\, \cdots \,\|\, f(x_n)$
denote $y = y_1 \,\|\, y_2 \,\|\, \cdots \,\|\, y_k$ where $y_i \in \{0,1\}$, $1 \leq i \leq k$
$g_1 \leftarrow \textbf{compress}(0^m \,\|\, y_1)$
for $i = 1$ **to** $k-1$
  do $g_{i+1} \leftarrow \textbf{compress}(g_i \,\|\, y_{i+1})$
return(g_k))

The mapping $x \mapsto y = y(x)$ is injective. Furthermore no encoding is a postfix of another encoding. I.e. there does not exist $x \neq x'$ and a z such that $y(x) = z \,\|\, y(x')$. This follows because the string y begins with two 1's and nowhere else in the string is a 1 followed by another 1.

Theorem (4.7)**.** *Suppose* ***Compress****:* $\{0,1\}^{m+1} \to \{0,1\}^m$ *is a collision resistant compression function. Then the function:*

$$h : \bigcup_{i=m+2}^{\infty} \{0,1\}^i \to \{0,1\}^m$$

constructed in the algorithm MERKLE-DAMGÅRD2 *is a collision resistant hash function.*

Proof. Suppose that we can find $x \neq x'$ such that $h(x) = h(x')$. Denote $y(x) = y_1 y_2 \cdots y_k$ and $y(x') = y'_1 y'_2 \cdots y'_l$.
Case 1: $k = l$
By working backwards either we get a collision for **compress** or $y = y'$. In either case we get a contradiction.
Case 2: $k \neq l$
Assume wlog that $l > k$. Assuming there is no collision for **compress** we have the following set of equalities:

$$\begin{aligned} y_k &= y'_l \\ y_{k-1} &= y'_{l-1} \\ \vdots &\quad \vdots \\ y_1 &= y'_{l-k+1} \end{aligned}$$

But this contradicts the postfix-free property and therefore we obtain a contradiction. □

Theorem (4.8)**.** *Suppose* ***Compress****:* $\{0,1\}^{m+t} \to \{0,1\}^m$ *is a collision resistant compression function where* $t \geq 1$*. Then there exists a collision resistant hash function:*

$$h : \bigcup_{i=m+t+1}^{\infty} \{0,1\}^i \to \{0,1\}^m$$

The number of times ***compress*** *is computed in the evaluation of* h *is at most:*

$$1 + \left\lceil \frac{n}{t-1} \right\rceil \text{ if } t \geq 2$$
$$2n + 2 \text{ if } t = 1$$

where $|x| = n$*.*

Now we describe the Secure Hash Algorithm (SHA-1), which is an iterated hash function with a 160-bit message digest. It is built from word-oriented operation on bitstrings where a word is a 32-bit (eight hexadecimal digits). The operations used are:

$X \wedge Y$	*bitwise "and" of* X *and* Y
$X \vee Y$	*bitwise "or" of* X *and* Y
$X \oplus Y$	*bitwise "xor" of* X *and* Y
$\neg X$	*bitwise complement of* X
$X + Y$	*integer addition modulo* 2^{32}
$\mathbf{ROTL}^s(X)$	*circular left shift of* X *by* s *positions* $(0 \leq s \leq 31)$

The padding scheme used in SHA-1 requires that $|x| \leq 2^{64} - 1$, which is to say that the binary representation of $|x|$ has at most 64 bits. If it it less than 64 bits it is padded with zeros on the left to be exactly 64 bits.

SHA-1-PAD(x):
comment: $|x| \leq 2^{64} - 1$
$d \leftarrow (447 - |x|) \mod 512$
$l \leftarrow$ the binary representation of $|x|$, where $|l| = 64$
$y \leftarrow x \,\|\, 1 \,\|\, 0^d \,\|\, l$

The resulting string y has length congruent with 512. So we split it up into n blocks of 512 bits: $y = M_1 \,\|\, \cdots \,\|\, M_n$. Define:

$$\mathbf{f}_t(B,C,D) = \begin{cases} (B \wedge C) \vee ((\neg B) \wedge D) & \text{if } 0 \leq t \leq 19 \\ B \oplus C \oplus D & \text{if } 20 \leq t \leq 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) & \text{if } 40 \leq t \leq 59 \\ B \oplus C \oplus D & \text{if } 60 \leq t \leq 79 \end{cases}$$

where the B, C, D are words and the output is one word. Now define the constants:

$$K_t = \begin{cases} \texttt{5A827999} & \text{if } 0 \leq t \leq 19 \\ \texttt{6ED9EBA1} & \text{if } 20 \leq t \leq 39 \\ \texttt{8F1BBCDC} & \text{if } 40 \leq t \leq 59 \\ \texttt{CA62C1D6} & \text{if } 60 \leq t \leq 79 \end{cases}$$

The algorithm itself can be found on page 138 in the book. It is an iterated hash function

Message Authentication Codes

MACs are keyed hash functions satisfying certain security properties. One way is to incorporate a key into an unkeyed hash function by including it in the message to be hashed. This is a bit risky, however. Suppose for example that we construct a keyed hash functions h_K from a hash function h by setting IV $= K$, where K is an m-bit key. Assume for simplicity that there is no preprocessing or output transformation. Then h is constructed from **Compress**: $\{0,1\}^{m+t} \to \{0,1\}^m$ where we require that the input x has length that is a multiple of t.

Now given any message x and its MAC $h_K(x)$ let x' be any message of length t and consider the message $x \,\|\, x'$. Then we have

$$h_K(x \,\|\, x') = \mathbf{compress}(h_K(x) \,\|\, x')$$

This can be seen by the assumptions we made (no preprocessing implies $y_r = x_r$ and no output transform implies $z_r = h_K(x)$) and following the process of creating iterated hash functions. Now since both $h_K(x)$ and x' is known an opponent can compute a valid MAC for the string $x \,\|\, x'$ without knowing the secret key K. A similar attack can be performed even though the message is padded.

The objective of an opponent is to produce a valid pair under an unknown key K. We allow the opponent to request (up to) Q valid MACs on messages of his own choosing. We conveniently assume that the values are acquired by querying a black box or oracle. Thus the adversary obtains a list of Q valid pairs $(x_1, y_1), (x_2, y_2), \cdots, (x_Q, y_Q)$. For the attack to be successful we demand that the attacker can produce a pair (x, y) such that $x \notin \{x_1, x_2, \cdots, x_Q\}$. This pair is said to be a forgery. If the probability that the adversary outputs a forgery is ϵ then the adversary is said to be an (ϵ, Q)-forger for the given MAC. We will in the following sections take ϵ to be the worst-case probability. I.e. the adversary can produce a forgery with probability ϵ regardless of the key. With this terminology we see that the above attack is a $(1, 1)$-forger.

A nested MAC builds a MAC algorithm from the composition of two (keyed) hash families. Let $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{G})$ and $(\mathcal{Y}, \mathcal{Z}, \mathcal{L}, \mathcal{H})$ be two hash families. Let $\mathcal{M} = \mathcal{K} \times \mathcal{L}$, the composition of these families are $(\mathcal{X}, \mathcal{Z}, \mathcal{M}, \mathcal{G} \circ \mathcal{H})$, where $\mathcal{G} \circ \mathcal{H} = \{g \circ h : g \in \mathcal{G}, h \in \mathcal{H}\}$ and $(g \circ h)_{(K,L)}(x) = h_L(g_K(x))$ for all $x \in \mathcal{X}$. In this it is assumed that $\mathcal{Y}, \mathcal{Z}$ are finite with $|\mathcal{Y}| \geq |\mathcal{Z}|$. $\mathcal{X}$ can be finite or infinite, if it is finite we have $|\mathcal{Y}| < |\mathcal{X}|$. A nested MAC can be shown to be secure if:

1. $(\mathcal{Y}, \mathcal{Z}, \mathcal{L}, \mathcal{H})$ *is secure as a MAC, given a fixed (unknown) key.*
2. $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{G})$ *is collision-resistant, given a fixed (unknown) key.*

Essentially we are building a secure "big MAC" from the composition of a secure "little MAC". We will consider three possible attacks:

A forger for the nested MAC (a "big MAC attack")
A pair of keys (K, L) is chosen and kept secret. The adversary is allows to choose values for x and query a big MAC oracle for values of $h_L(g_K(x))$. Then the adversary tries to find a pair (x', z) such that $z = h_L(g_K(x'))$ where x' was not one of the previous queries.

A forger for the little MAC (a "little MAC attack")
A key L is chosen and kept secret. The adversary is allowed to choose values for y and query a little MAC oracle for values of $h_L(y)$. He then tries to output a pair (y', z) such that $z = h_L(y')$ where y' was not one of the previous queries.

A collision-finder for the hash family, when the key is secret (an "unknown-key

collision attack")
A key K is chosen and kept secret. The adversary is allowed to choose values for x and query a hash oracle for values of $g_K(x)$. He then tries to output a pair (x',x") such that $x' \neq x''$ and $g_K(x') = g_K(x'')$.

Cryptosystem (CBC-MAC(x, K))**.**
Denote $x = x_1 \| \cdots \| x_n$
$IV \leftarrow 00 \cdots 0 \in \{0\}^t$
$y_0 \leftarrow IV$
for $i = 1$ ***to*** n
  do $y_i \leftarrow e_K(y_{i-1} \oplus x_i)$
return(y_n)

Best attack: Birthday attack gives $(1/2, O(2^{t/2}))$-forger. Secure if the encryption is secure. Also CCM (authenticated encryption) which is CBC + ctr and produces MAC as part of encryption.

Introduction to public-key cryptography

In the systems studied so far the choice of a key K gives rise to both an encryption and a decryption rule. The disadvantage of this is that exposure to any of these rules leaves the system insecure. Such systems are known as symmetric-key cryptosystems. Another disadvantage is that the two parts have to agree on a key, requiring previous communication on a secure channel. This might obviously be difficult. The idea behind a public-key cryptosystem is that it might be infeasible to compute the d_K given e_K. If so the encryption rule e_K is a public key and the decryption rule d_K is called a private key. As an analogy consider that Alice puts an object into a box and locks it with a combination lock left by Bob. Bob is the only person who can open it since only he knows the combination. These systems were proposed by Diffie and Hellman in 1976.

Such systems can never provide unconditional security! Given enough time an adversary can always brute force the system. Therefore we only study the computational security of public-key systems. One can think of these systems as a trapdoor one-way function. What this means is that e_K should be easy to compute, but the inverse should be hard to compute for anyone not having the private key. I.e. the person having the private key possesses a trapdoor. There are no proven injective one-way functions, but there are functions believed to be one-way (RSA encryption function).

Number theory

The Euclidean algorithm

Given two positive integers the Euclidean algorithm computes their greatest common divisor. Set $a = r_0, b = r_1$, then:

$$\begin{aligned} r_0 &= q_1 r_1 + r_2, & 0 < r_2 < r_1 \\ r_1 &= q_2 r_2 + r_3, & 0 < r_3 < r_2 \\ &\vdots \quad \vdots \\ r_{m-2} &= q_{m-1} r_{m-1} + r_m, & 0 < r_m < r_{m-1} \\ r_{m-1} &= q_m r_m \end{aligned}$$

The final r_m divides both a, b by repeteadly working backwards. By working the other way one can show that any integer dividing both a, b must divide r_m. This implies that $r_m = \gcd(a, b)$. By calling the algorithm with n and b, b has an inverse modulo n if $r_m = 1$. By working the other way we can find s, t such that $\gcd(a, b) = as + bt$. This provides a mean to compute the inverse. One could also do this by keeping track of the sequences t_i, s_i given by:

$$s_0 = 1, s_1 = 0, \cdots, s_{i+1} = s_{i-1} - q_i s_i$$
$$t_0 = 0, t_1 = 1, \cdots, t_{i+1} = t_{i-1} - q_i t_i$$

Then by induction $\gcd(a, b) = r_m = as_m + bt_m$.

The Chinese remainder theorem

The Chinese remainder theorem is a method to solve a certain system of congruences.

Theorem. *Suppose $m_1, \cdots, m_r$ are pairwise relatively prime positive integers, and suppose $a_1, \cdots, a_r$ are integers. Then the system of r congruences $x \equiv a_i \mod m_i$, $1 \leq i \leq r$ has a unique solution modulo $M = \prod_{1 \leq i \leq r} m_i$ which is given by:*

$$x = \sum_{i=1}^{r} a_i M_i y_i \mod M$$

where $M_i = M/m_i$ and $y_i = M_i^{-1}$ for $1 \leq i \leq r$.

Other useful facts

Let G be a finite multiplicative group. the order of an element $g \in G$ is the smallest positive integer m such that $g^m = 1$.

Theorem (Lagrange)**.** *Let $|G| = n$ and $g \in G$. Then the order of g divides n.*

Since $|\mathbb{Z}_n^*| = \varphi(n)$:

Theorem. *If $b \in \mathbb{Z}_n^*$, then $b^{\varphi(n)} \equiv 1 \mod n$.*

Theorem (Fermat)**.** *Suppose p is prime and $b \in \mathbb{Z}_p$. Then $b^p \equiv b \mod p$.*

Proof. If $b \equiv 0 \mod p$ the result is obviously true. If not, $\varphi(p) = p - 1$ and by the previous theorem $b^{p-1} \equiv 1 \mod p$ and the result follows. □

Theorem. *If p is prime, then $\mathbb{Z}_p^*$ is a cyclic group.*

An element α having order $p-1 \mod p$ is called a primitive element modulo p. α is a primitive element modulo p if and only if $\{\alpha^i : 0 \leq i \leq p-2\} = \mathbb{Z}_p^*$. In this case any element $\beta \in \mathbb{Z}_p^*$ can be written as α^i in a unique way for some $0 \leq i \leq p-2$. The order of β is then $(p-1)/\gcd(p-1,i)$, so β is primitive if and only if $\gcd(p-1,i) = 1$. Therefore the number of primitive elements in $\mathbb{Z}_p^*$ is $\varphi(p-1)$.
If p is large it is cumbersome to compute $p-1$ powers of an element α. However:

Theorem. *Suppose that $p > 2$ is prime and $\alpha \in \mathbb{Z}_p^*$. Then α is a primitive element modulo p if and only if $\alpha^{(p-1)/q} \not\equiv 1 \mod p$ for all primes $q|(p-1)$.*

Proof. If α is primitive then α has order $p-1$ and the result follows.
Assume that $\alpha \in \mathbb{Z}_p^*$ is not primitive modulo p. Let $d = |\alpha|$, then $d|(p-1)$ by Lagrange and $d < (p-1)$ by assumption. Hence $1 < (p-1)/d = n$ so let q be a prime dividing n, but then $d|(p-1)/q$. Now since $\alpha^d \equiv 1 \mod p$ and $d|(p-1)/q$ we get that $\alpha^{(p-1)/q} \equiv 1 \mod p$. □

The RSA cryptosystem

Let p, q be distinct odd primes and set $n = pq$, then $\varphi(n) = (p-1)(q-1)$. We define the RSA cryptosystem:

Cryptosystem (RSA cryptosystem)**.** *Let $n = pq$ where p and q are primes. Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$ and define:*

$$\mathcal{K} = \{(n, p, q, a, b) : ab \equiv 1 \mod \varphi(n)\}$$

For $K = (n, p, q, a, b)$ define:

$$e_K(x) = x^b \mod n$$
$$d_K(y) = y^a \mod n$$

for $x, y \in \mathbb{Z}_n$. The values n and b comprise the public key and the values p, q, a form the private key.

Since $ab \equiv 1 \mod \varphi(n)$ we have $ab = t\varphi(n) + 1$ for some $t \in \mathbb{Z}$. Suppose that $x \in \mathbb{Z}_n^*$, then:

$$(x^b)^a \equiv x^{t\varphi(n)+1} \equiv x(x^{\varphi(n)})^t \equiv x \mod n$$

If $x \in \mathbb{Z}_n \setminus \mathbb{Z}_n^*$ then either $p|x$ or $q|x$, so assume wlog the former. Then $(x^b)^a \equiv 0 \mod p$ and by Fermat's theorem $(x^b)^a \equiv x \mod q$. We now use the Chinese remainder theorem to conclude that $(x^b)^a \equiv x \mod n$.

The security of the system is based on the belief that $e_K = x^b \mod n$ is a one-way function. The trapdoor in this case is the factorization of n which means that Bob easily can compute $\varphi(n)$ and then compute the decryption exponent a by using $ab \equiv 1 \mod \varphi(n)$.

RSA PARAMETER GENERATION:

1. Generate two large primes p, q such that $p \neq q$.
2. $n \leftarrow pq$ and $\varphi(n) \leftarrow (p-1)(q-1)$.
3. Choose a random integer $1 < b < \varphi(n)$ such that $\gcd(b, \varphi(n)) = 1$.
4. $a \leftarrow b^{-1} \mod \varphi(n)$.
5. The publik key is (n, b) and the private key is (p, q, a).

An obvious attack on the system is trying to factorize n. If this can be done it is easy to break the system as this exposes the private key. This means that in order for the system to be secure, it is necessary that $n = pq$ is sufficiently large so that it is computationally infeasible to factor it. It is recommended that both p and q are 512-bits primes making n a 1024-bits modulus.

We need a fast way to compute modular exponents. Assume we have $x^c \mod n$, where we can write $c = \sum_{i=0}^{l-1} c_i 2^i$, i.e. the binary representation of the exponent. In this case we have the following algorithm drastically shortening the computation time:

SQUARE-AND-MULTIPLY:(x, c, n)

$z \leftarrow 1$
for $i = l - 1$ **downto** 0
do
- $z \leftarrow z^2 \mod n$
- **if** $c_i = 1$
  - **then** $z \leftarrow (z \cdot x) \mod n$

return(z)

Using this algorithm one can both decrypt and encrypt with the RSA cryptosystem in time $O((\log n)^3)$. We now discuss how we can find suitable primes.

Primality testing

How many random integers of a specified size do we need to check before we find a prime? Let $\pi(N)$ denote the number of primes less than or equal to N. The Prime number theorem states that $\pi(N) \approx N/\ln N$. So the probability of choosing a prime between 1 and N is approximately $1/\ln N$. For a 512 bit prime we then have a probability of about $1/\ln 2^{512} \approx 1/355$. If we restrict ourself to random odd integer the probability doubles. This shows that the prime generation problem is practical.

A decision problem is a problem that can be answered by "yes" or "no". We will introduce a new type of randomized algorithms:

Definition. A yes-biased Monte Carlo algorithm is a randomized algorithm for a decision problem in which a "yes" answer is always correct, but a "no" may be incorrect. A no-biased Monte Carlo algorithm is defined similarly. A yes-biased Monte Carlo algorithm is said to have error probability ϵ if for any instance where the answer is "yes" the algorithm will output (incorrectly) "no" with probability at most ϵ.

The decision problem is this:

Composites:
Instance: *A positive integer $n \geq 2$.*
Question: *Is n composite?*

Definition. Let p be an odd prime and a an integer. a is a quadratic residue modulo p if $a \not\equiv 0 \mod p$ and the congruence $y^2 \equiv a \mod p$ has a solution $y \in \mathbb{Z}_p^*$. Else a is a quadratic non-residue.

Suppose that p is an odd prime and a is a quadratic residue modulo p, then there is a $y \in \mathbb{Z}_p^*$ such that $y^2 \equiv a \mod p$. Then $(-y)^2 \equiv a \mod p$ also and $y \equiv -y \mod p$ since p is odd. Now $x^2 - a \equiv 0 \mod p$ can be factored as $(x-y)(x+y) \equiv 0 \mod p$ which implies that $p|(x-y)$ or $p|(x+y)$ so $x \equiv \pm y \mod p$. So there are two solutions of $x^2 - a \equiv 0 \mod p$ and they are the negatives of each other modulo p. A new decision problem:

Quadratic residues:
Instance: *An odd prime p, and an integer a.*
Question: *Is a a quadratic residue modulo p?*

Theorem (Euler's criterion)**.** *Let p be an odd prime. Then a is a quadratic residue modulo p if and only if*

$$a^{(p-1)/2} \equiv 1 \mod p$$

Proof. Suppose $a \equiv y^2 \mod p$. Now for any $a \not\equiv 0 \mod p$ we have $a^{p-1} \equiv 1 \mod p$, thus:

$$a^{(p-1)/2} \equiv (y^2)^{(p-1)/2} \equiv y^{p-1} \equiv 1 \mod p$$

Now suppose that $a^{(p-1)/2} \equiv 1 \mod p$. Let b be a primitive element modulo p. Then $a \equiv b^i \mod p$ for some i. Then:

$$1 \equiv a^{(p-1)/2} \equiv (b^i)^{(p-1)/2} \equiv b^{i(p-1)/2} \mod p$$

Now since b has order $p-1$, we must have that $(p-1)|i(p-1)/2 \Rightarrow i$ is even and so the square roots of a are $\pm b^{i/2}$. □

Definition. Suppose p is an odd prime. For any integer a, define the Legendre symbol $\left(\frac{a}{p}\right)$ as:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a \equiv 0 \mod p \\ 1 & \text{if } a \text{ is a quadratic residue modulo } p \\ -1 & \text{if } a \text{ is a quadratic non-residue modulo } p \end{cases}$$

Now by the previous criterion if a is a non-residue:

$$\left(a^{(p-1)/2}\right)^2 \equiv a^{p-1} \equiv 1 \mod p$$

but since a is not a residue we must have $a^{(p-1)/2} \equiv -1 \mod p$. Summarised:

Theorem (5.10)**.** *If p is an odd prime, then:*

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \mod p$$

Definition. Suppose n is an odd positive integer with prime factorization $n = \prod_{i=1}^{k} p_i^{e_i}$. For any integer a we define The Jacobi symbol as:

$$\left(\frac{a}{p}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i}$$

Note that Theorem 5.10 might not hold in the case of composites. If the congruence holds n is said to be an Euler pseudoprime to the base a. Furthermore $\left(\frac{a}{n}\right) = 0$ if and only if $\gcd(a, n) > 1$, so if $1 \leq a \leq n-1$ and $\left(\frac{a}{n}\right) = 0$ then n is composite. It can also be shown that for any odd composite n, n is the Euler pseudoprime to the base a for at most half of the integers $a \in \mathbb{Z}_a^*$.

By the paragraph above the following algorithm is a yes-biased Monte Carlo algorithm for **composites** with error probability at most $1/2$.

SOLOWAY-STRASSEN:(n)
choose a random integer a such that $1 \leq a \leq n-1$
$x \leftarrow \left(\frac{a}{n}\right)$
if $x = 0$
  then return("n is composite")
$y \leftarrow a^{(p-1)/2} \mod n$
if $x \equiv y \mod n$
  then return("n is prime")
else return("n is composite")

Now some nice properties for evaluating the Jacobi symbol.

(1) If n is a positive odd integer and $m_1 \equiv m_2 \mod n$, then

$$\left(\frac{m_1}{n}\right) = \left(\frac{m_2}{n}\right)$$

(2) If n is a positive odd integer, then:

$$\left(\frac{2}{n}\right) = \begin{cases} 1 & \text{if } n \equiv \pm 1 \mod 8 \\ -1 & \text{if } n \equiv \pm 3 \mod 8 \end{cases}$$

(3) If n is a positive odd integer, then:

$$\left(\frac{m_1 m_2}{n}\right) = \left(\frac{m_1}{n}\right)\left(\frac{m_2}{n}\right)$$

in particular, if $m = 2^k t$ and t is odd, then

$$\left(\frac{m}{n}\right) = \left(\frac{2}{n}\right)^k \left(\frac{t}{n}\right)$$

(4) Suppose m and n are positive odd integers. Then:

$$\left(\frac{m}{n}\right) = \begin{cases} -\left(\frac{n}{m}\right) & \text{if } m \equiv n \equiv 3 \mod 4 \\ \left(\frac{n}{m}\right) & \text{otherwise} \end{cases}$$

Suppose that we have run SOLOWAY-STRASSEN m times on an integer n. How sure can we be that n is prime? By some careful application of probability analysis (Baye's theorem) and the Prime number theorem one can show that the probability of n being composite given that the algorithm has answered that it is prime m times in succesion (denoted $\Pr[\mathbf{a}|\mathbf{b}]$) is approximately

$$\frac{\ln n - 2}{\ln n - 2 + 2^{m+1}}$$

In practice around 50 or 100 times reduces the error probability very much.

Another Monte Carlo algorithm for **composites** is presented.

MILLER-RABIN:(n)
write $n - 1 = 2^k m$ where m is odd
choose a random integer a such that $1 \leq a \leq n - 1$
$b \leftarrow a^m \mod n$
if $b \equiv 1 \mod n$
  then return("n is prime")
for $i = 0$ **to** $k - 1$
  do **if** $b \equiv -1 \mod n$
    then return("n is prime")
    else $b \leftarrow b^2 \mod n$
return("n is composite")

Theorem (5.11)**.** *The* MILLER-RABIN *algorithm for* ***composites*** *is a yes-biased Monte Carlo algorithm.*

Proof. Proof by contradiction. Assume n prime and algorithm answer composite. Then $2^{2^i m} \not\equiv -1 \mod n$ for $0 \leq i \leq k-1$ and $a^m \not\equiv 1 \mod n$. Now since n is prime by Fermat:$a^{2^k m} \equiv 1 \mod n$ so $a^{2^{k-1} m}$ is a square root of 1 modulo n, but n is prime so we must have $a^{2^{k-1} m} \equiv \pm 1 \mod n$ which implies that $a^{2^{k-1} m} \equiv 1 \mod n$. By repeating the arguement $a^m \equiv 1$. Contradiction! □

The error probability can be shown to be 1/4.

Square roots modulo n

Suppose n is odd and $\gcd(a,n) = 1$. How many solutions in $\mathbb{Z}_n$ is there to the congruence $y^2 \equiv a \mod n$. The case where n is prime is already covered. But what about odd prime power?

Theorem (5.12)**.** *Suppose that p is an odd prime, e a positive integer and $\gcd(a,p) = 1$. Then the congruence $y^2 \equiv a \mod p^e$ has no solutions if $\left(\frac{a}{p}\right) = -1$ and two solutions (modulo p^e) if $\left(\frac{a}{p}\right) = 1$.*

Note the use of the Legendre symbol. By using the Chinese remainder theorem we can extend this result:

Theorem. *Suppose that $n > 1$ is an odd integer having factorization $n = \prod_{i=1}^{l} p_i^{e_i}$, where the p_i's are distinct primes and the e_i's positive integers. Suppose further that $\gcd(a,n) = 1$. Then the congruence $y^2 \equiv a \mod n$ has 2^l solutions modulo n if $\left(\frac{a}{p_i}\right) = 1$ for all $i \in \{1, \cdots, l\}$ and no solutions otherwise.*

Factoring algorithms

We assume that n is odd. If n is composite it has a prime factor $p \leq \lfloor \sqrt{n} \rfloor$ (seen by squaring it). Therefore trial division (dividing n by every odd integer up to $\lfloor \sqrt{n} \rfloor$ trying to find a factor). If $n < 10^{12}$ this is perfectly reasonable. In most methods we are content with finding any non-trivial factor. If we wanted the complete factorization we would just apply the algorithm to the smaller parts repeatedly.

We present the first algorithm:

POLLARD $p-1$ FACTORING ALGORITHM:(n, B)
$a \leftarrow 2$
for $j = 2$ **to** B
  do $a \leftarrow a^j \mod n$
if $1 < d < n$
  then return(d)
else return("failure")

Here is how it works. Suppose p is a prime divisor of n and that $q \leq B$ for every $q|(p-1)$, then $(p-1)|B!$. At the end of the for loop we have

$$a \equiv 2^{B!} \mod n$$

now since $p|n$ we must have $a \equiv 2^{B!} \mod p$ and by Fermat we have $2^{p-1} \equiv 1 \mod p$. Since $(p-1)|B$ then $a \equiv 1 \mod p$, so

$$p|(a-1), \text{ and } p|n$$

Then it must be the case that $p|d = \gcd(a-1, n)$. We have now found a non trivial divisor of n, namely d, unless $a = 1$. The choice of B is obviously very important. If one chooses B too small, then the algorithm will likely output failure; Too large, however, and the algorithm is not much faster than trial division. To make the RSA primes "immune" to this factorization one would choose p, q such that both $p-1$ and $q-1$ have "large" prime factors. One way is to use primes of the form $2p+1$ where p are large primes (512 bits).

Now for Pollard Rho. Let p be the smallest prime factor of n and suppose there exists $x, x' \in \mathbb{Z}_n$ such that $x \neq x'$ and $x \equiv x' \mod p$. Then $p \geq \gcd(x - x', n) < n$ and we have a non-trivial divisor of n. We do not need to know the value of p for this to work.
We again turn to the birthday paradox. Suppose we choose a random subset $X \subset \mathbb{Z}_n$ and then compute $\gcd(x - x', n)$ for all distinct $x, x' \in X$. This will be successful if and only if the mapping $x \mapsto x \mod p$ yields at least one collision for $x \in X$. If $|X| \approx 1.17\sqrt{p}$ there is a 50% probability that there is at least one collision and hence we find a non-trivial factor of n. To find the wanted collision $x \mod p = x' \mod p$ we need to compute $\gcd(x - x', n)$. This is because the value of p is not known! Hence we need to compute more than $\binom{|X|}{2}$ gcd's.

The Pollard Rho is a variation of this technique using fewer computations and less memory. Suppose that we have a polynomial f with integer coefficients (e.g. $f(x) = x^2 + 1$)where the map $x \mapsto f(x) \mod p$ behaves "randomly". Let $x_1 \in \mathbb{Z}_n$ and consider the sequence:

$$x_j = f(x_{j-1}) \mod n$$

Now let m be an integer and define $X = \{x_1, \cdots, x_m\}$ and assume, to make our lives slightly easier, that the values are distinct. This will be our "random" set. We want to find two values $x_j, x_i \in X$ such

that $\gcd(x_j - x_i, n) > 1$. In stead of computing all the previous gcd's for every new term we employ a technique.

Suppose that $x_j \equiv x_i \mod p$, since f has integer coefficients we have $f(x_j) \equiv f(x_i) \mod p$, but from the definition of the sequence we then get:

$$x_{i+1} \mod p = (f(x_i) \mod n) \mod p = f(x_i) \mod p$$

since $p|n$. Similarly for x_{j+1}, and hence $x_{i+1} \equiv x_{j+1}$. By repeating this argument we get for any integer $\delta \geq 0$ that:

$$x_i \equiv x_j \mod p \Rightarrow x_{i+\delta} \equiv x_{j+\delta} \mod p$$

By setting $l = j - i$ it follows that $x_{i'} \equiv x_{j'} \mod p$ if $j' > i' \geq i$ and $j' - i' \equiv 0 \mod l$ if $x_i \equiv x_j \mod p$.

Now suppose we form a graph G on vertex set $\mathbb{Z}_p$ where for all $i \geq 1$ we have a directed edge from $x_i \mod p$ to $x_{i+1} \mod p$. There must exist a first pair x_i, x_j with $i < j$ such that $x_i \equiv x_j \mod p$. By the observations above one can see that the graph will have a tail:

$$x_1 \mod p \to x_2 \mod p \to \cdots \to x_i \mod p$$

and an infinite cycle:

$$x_i \mod p \to x_{i+1} \mod p \to \cdots \to x_j \mod p = x_i \mod p$$

Giving the name to the algorithm. We do not need the first such collision as long as we find one. For improvement we just search for collisions with $i, j = 2i$. The index j will therefore jump 2 edges for each iteration while the i only one. Therefore they will meet sooner or later.

POLLARD RHO FACTORING ALGORITHM:(n, x_1)
external f
$x \leftarrow x_1$
$x' \leftarrow f(x) \mod n$
$p \leftarrow \gcd(x - x', n)$
while $p = 1$
do
- **comment** in the ith tieration $x = x_i$ and $x' = x_{2i}$
- $x \leftarrow f(x) \mod n$
- $x' \leftarrow f(x') \mod n$
- $x' \leftarrow f(x') \mod n$
- $p \leftarrow \gcd(x - x', n)$

if $p = n$
then return("failure")
else return(p)

Another idea is this: Suppose that we can find $x \not\equiv \pm y \mod n$ such that $x^2 \equiv y^2 \mod n$, then:

$$n|(x - y)(x + y)$$

But neither $x - y$ nor $x + y$ is divisible by n. Therefore $\gcd(x + y, n)$ (and $\gcd(x - y, n)$) is a non-trivial factor of n. The Dixon's random squares algorithm uses this property. It uses a factor base $\mathcal{B}$ which is a set of the b smallest primes for an appropriate b. That is $\mathcal{B} = \{p_1, \cdots, p_b\}$. Now let c be slightly larger

than b say $c = b + 4$ and suppose we have c congruences:

$$z_j^2 \equiv p_1^{\alpha_{1j}} \cdot p_2^{\alpha_{2j}} \cdots p_b^{\alpha_{bj}} \mod n, \quad 1 \leq j \leq c$$

For each j consider the vector

$$a_j = (\alpha_{1j} \mod 2, \cdots, \alpha_{bj} \mod 2) \in (\mathbb{Z}_2)^b$$

If we can find a subset J of the a_j's that sum modulo 2 to the zero vector, then the product of the corresponding z_j's will use each factor in $\mathcal{B}$ an even number of times. We then get

$$z'^2 \equiv p'^2 \mod n$$

and we can use $\gcd(z' \pm p', n)$ to find a divisor. It is often useful to try z's of the form $j + \lceil \sqrt{kn} \rceil$ for $j = 0, 1, 2, \cdots, k = 1, 2, \cdots$, and also $\lfloor \sqrt{kn} \rfloor$. By including $-1 \in \mathcal{B}$ we can factor $-z^2 \mod n$ when z^2 is close to n and therefore the negative is small and easy to factorize.

Other attacks on RSA

Another attack on RSA might be to compute $\varphi(n)$. However this is no easier than factoring n: If both n and $\varphi(n)$ is known, then:

$$n = pq, \quad \varphi(n) = (p-1)(q-1)$$

Which gives the quadratic equation:

$$p^2 - (n - \varphi(n) + 1)p + n = 0$$

meaning that an attacker easily could find the values of p, q and therefore factor n.

If the decryption exponent a is known, then n can be factored in polynomial time by a randomized (Las Vegas) algorithm. In other words, if Bob reveals the decryption exponent a new modulus must be chosen!

The ElGamal cryptosystem and semantic security

G is finite multiplicative group. For an element $\alpha \in G$ with $|\alpha| = n$ denote:

$$\langle \alpha \rangle = \{\alpha^i : 0 \leq i \leq n-1\}$$

this is a cyclic subgroup of G of order n.

Discrete logarithm:
Instance: *A multiplicative group $(G, \cdot)$, an element $\alpha \in G$ with order n and an element $\beta \in \langle \alpha \rangle$.*
Question: *Find the unique integer a, $0 \leq a \leq n-1$, such that*

$$\alpha^a = \beta$$

This integer is denoted by $\log_\alpha \beta$ and is called the discrete logarithm of β.

The motivation is that this behaves like a one-way trapdoor function (in suitable groups) in that it is (probably) difficult to find the discrete logarithm, but the inverse operation of exponentiation is easy (square-and-multiply).

Cryptosystem (ElGamal public-key cryptosystem in $\mathbb{Z}_p^*$)**.** *Let p be a prime such that the **Discrete Logarithm** problem in $\mathbb{Z}_p^*$ is infeasible and let $\alpha \in \mathbb{Z}_p^*$ be a primitive element. Let $\mathcal{P} = \mathbb{Z}_p^*$, $\mathcal{C} = \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ and define:*

$$\mathcal{K} = \{(p, \alpha, a, \beta) : \beta \equiv \alpha^a \mod p\}$$

The values p, α, β are the public key and a is the private key. For $K = (p, \alpha, a, \beta)$, and for a secret random number $k \in \mathbb{Z}_{p-1}$ define:

$$e_K(x, k) = (\alpha^k \mod p, x\beta^k \mod p)$$

For $y_1, y_2 \in \mathbb{Z}_p^*$ define

$$d_K(y_1, y_2) = y_2(y_1^a)^{-1} \mod p$$

Informally the plaintext x is "masked" by β^k and the value α^k is transmitted as part of the ciphertext. Bob knows a and can therefore compute β^k from α^k. He then inverts β^k and computes x. There is a randomization going on which means that several ciphertexts are encryptions of the same plaintext. One makes sure that the Dlog problem is infeasible by choosing p carefully and letting α be primitive modulo p. p should have about 300 digits and $p-1$ at least one "large" prime factor.

Discrete logarithm problem

$(G, \cdot)$ is a multiplicative group and $\alpha \in G$ has order n. One could brute-force the problem, but this would take a lot of time and memory. One could also precompute α^i and then sort (i, α^i) with respect to the second coordinates and then binary search when given a value β.

SHANK'S ALGORITHM:(G, n, α, β)

1. $m \leftarrow \lceil \sqrt{n} \rceil$
2. **for** $j = 0$ **to** $m - 1$
   do compute α^{mj}
3. Sort the m ordered pairs (j, α^{mj}) with respect to their second coordinates, obtaining a list L_1
4. **for** $i = 0$ **to** $m - 1$
   do compute $\beta\alpha^{-i}$
5. Sort the m ordered pairs $(i, \beta\alpha^{-i})$ with respect to their second coordinates, obtaining a list L_2
6. Find a pair $(j, y) \in L_1$ and a pair $(i, y) \in L_2$
7. $\log_\alpha \beta = (mj + i) \mod n$

Step 6 implies that $\alpha^{mj+i} = \beta$ which is what we want. Conversely one can use the division algorithm to show that $\log_\alpha \beta = mj + i$.

Since $\langle \alpha \rangle$ is cyclic of order n we can treat $\log_\alpha \beta$ as an element of $\mathbb{Z}_n$. There is a Pollard Rho algorithm for discrete logarithms as well. It is the same strategy in this instance as well; We want to find a collision. Let $S_1 \cup S_2 \cup S_3$ be a partition of G into three subsets of roughly equal size. Define $f : \langle \alpha \rangle \times \mathbb{Z}_n \times \mathbb{Z}_n \to \langle \alpha \rangle \times \mathbb{Z}_n \times \mathbb{Z}_n$ as follows:

$$f(x, a, b) = \begin{cases} (\beta x, a, b + 1) & \text{if } x \in S_1 \\ (x^2, 2a, 2b) & \text{if } x \in S_2 \\ (\alpha x, a + 1, b) & \text{if } x \in S_3 \end{cases}$$

Furthermore we require each triple (x, a, b) to satisfy $x = \alpha^a \beta^b$. We begin with an inital triple satisfying this, e.g. $(1, 0, 0)$ (also note that $1 \notin S_2$ or else we would stand still). f preserves this property, so define the sequence:

$$(x_i, a_i, b_i) = \begin{cases} (1, 0, 0) & \text{if } i = 0 \\ f(x_{i-1}, a_{i-1}, b_{i-1}) & \text{if } i \geq 1 \end{cases}$$

By doing the same thing as before we find a value $i \geq 1$ such that $x_{2i} = x_i$, then $\alpha^{a_{2i}} \beta^{b_{2i}} = \alpha^{a_i} \beta^{b_i}$ and by denoting $c = \log_\alpha \beta$ it must be the case that:

$$\alpha^{a_{2i} + cb_{2i}} = \alpha^{a_i + cb_i}$$

and since $|\alpha| = n$ it follows that:

$$a_{2i} + cb_{2i} \equiv a_i b_i \mod n \Leftrightarrow c(b_{2i} - b_i) \equiv a_{2i} - a_i \mod n$$

If $d = \gcd(b_{2i} - b_i, n) = 1$ we can solve uniquely for c. In the case that $d > 1$ one can still solve the congruence for d solutions and check each solution to see which one is correct if d is not too large.

POLLARD RHO DLOG ALGORITHM:(G, n, α, β)

procedure $f(x, a, b)$
  if $x \in S_1$
    then $f \leftarrow (\beta x, a, (b + 1 \mod n))$
  else if $x \in S_2$
    then $f \leftarrow (x^2, (2a \mod n), (2b \mod n))$
  else if $x \in S_3$
    then $f \leftarrow (\alpha x, (a + 1 \mod n), b)$
return(f)

main
  define the partition $G = S_1 \cup S_2 \cup S_3$
  $(x, a, b) \leftarrow f(1, 0, 0)$
  $(x', a', b') \leftarrow f(x, a, b)$
while $x \neq x'$

$$\textbf{do} \begin{cases} (x, a, b) \leftarrow f(x, a, b) \\ (x', a', b') \leftarrow f(x', a', b') \\ (x', a', b') \leftarrow f(x', a', b') \end{cases}$$

if $\gcd(b' - b, n) \neq 1$
  return("failure")
else return$((a - a')(b' - b)^{-1} \mod n)$

Suppose that $n = \prod_{i=1}^{k} p_i^{c_i}$ where the p_i's are distinct primes. The value $a = \log_\alpha \beta$ is determined uniquely modulo n. If we can compute $a \mod p_i^{e_i}$ for all i we can compute $a \mod n$ using the Chinese remainder theorem. Suppose that q is prime, $n \equiv 0 \mod q^c$ and $n \not\equiv 0 \mod q^{c+1}$. We will compute $x = a \mod q^c$ where $0 \leq x \leq q^c - 1$. Express x in radix q:

$$x = \sum_{i=0}^{c-1} a_i q^i, \quad 0 \leq a_i \leq q - 1$$

for $0 \leq i \leq c - 1$. Also $a = x + sq^c$ for some integer s, so:

$$a = \sum_{i=0}^{c-1} a_i q^i + sq^c$$

The first step is to compute a_0. Note that:

$$\beta^{n/q} = (\alpha^a)^{n/q} = (\alpha^{\sum_{i=0}^{c-1} a_i q^i + sq^c})^{n/q} = (\alpha^{a_0 + Kq})^{n/q} = \alpha^{a_0 n/q} \alpha^{Kn} = \alpha^{a_0 n/q}$$

Therefore one can compute a_0 for example by computing $\gamma = \alpha^{n/q}, \gamma^2, \cdots$ until $\gamma^i = \beta^{n/q} \Rightarrow a_0 = i$. We continue this process until all a_i's are known. One can then use $\beta_j = \beta\alpha^{-(a_0 + a_1 q + \cdots + a_{j-1} q^{j-1})}$ to compute $\beta_{j+1} = \beta_j \alpha^{-a_j q^j}$. The algorithm calculates the a_i's in $\log_\alpha \beta \mod q^c = \sum_{i=0}^{c-1} a_i q^i$

POHLIG-HELLMAN ALGORITHM:$(G, n, \alpha, \beta, q, c)$

$j \leftarrow 0$
$\beta_j \leftarrow \beta$
while $j \leq c - 1$

$$\textbf{do}\begin{cases}\delta \leftarrow \beta_j^{n/q^{j+1}}\\ \text{find } i \text{ such that } \delta = \alpha^{in/q}\\ a_j \leftarrow i\\ \beta_{j+1} \leftarrow \beta_j \alpha^{-a_j q^j}\\ j \leftarrow j+1\end{cases}$$

$\textbf{return}(a_0, \cdots, a_{c-1})$

We run this algorithm on all factors of n and paste a solution from the Chinese remainder theorem.

Now assume that $G = \mathbb{Z}_p^*$ and that α is primitive. Then we can use the INDEX CALCULUS METHOD. In this algorithm we have a factor base $\mathcal{B} = \{p_1, p_2, \cdots, p_B\}$ consisting of small primes.
Step 1 is to find the logarithm of the factor primes.
Step 2 is to compute the discrete logarithm of the desired element β by using the knowledge of the logarithms of the factor base.
Let $C = B + 10$ for example. In step 1 we will construct C congruences modulo p which have the form:

$$\alpha^{x_j} \equiv p_1^{a_{1j}} p_2^{a_{2j}} \cdots p_B^{a_{Bj}} \mod p, \quad 1 \leq j \leq C$$

Note that this is equivalent to:

$$x_j \equiv a_{1j} \log_\alpha p_1 + \cdots + a_{Bj} \log_\alpha p_B \mod p-1, \quad 1 \leq j \leq C$$

Given these congruences we hope that there is a unique solution in terms of the unknowns $\log_\alpha p_i$. If so, we can compute the logarithms of the elements in the factor base. To generate C such congruences one might try random exponents and using trial division, for example.
Supposing that this is done, we compute the desired logarithm by using a Las Vegas algorithm. Choose a random integer $1 \leq s \leq p-2$ and compute:

$$\gamma = \beta\alpha^s \mod p$$

If we can factor γ over the factor base $\mathcal{B}$ we get a congruence of the form:

$$\beta\alpha^s \equiv p_1^{c_1} p_2^{c_2} \cdots p_B^{c_B} \mod p$$

which gives:

$$\log_\alpha \beta + s \equiv c_1 \log p_1 + \cdots + c_B \log p_B \mod p-1$$

and we can easily solve for $\log_\alpha \beta$.

Finite fields & irreducible polynomials

Elliptic curves

Elliptic curves are describes as solutions to certain equations in two variables. We can motivate the use from the real numbers where it is easy to see the geometrical properties.

Definition. Let $a, b \in \mathbb{R}$ be constants such that $4a^3 + 27b^2 \neq 0$. A non-singular elliptic curve is the set E of solutions $(x, y) \in \mathbb{R} \times \mathbb{R}$ to the equation $y^2 = x^3 + ax + b$ together with a special point $\mathcal{O}$ called the point at infinity.

We want to make E into an abelian group. We make $\mathcal{O}$ the identity element, but we have to define addition between other points $P = (x_1, y_1)$, $Q = (x_2, y_2)$. There are three cases:

(1) $x_1 \neq x_2$
   We define the addition $P + Q$ to be the point R acquired from forming a line L between P and Q intersection E at a third point: R'. R is the reflection of R' through the x-axis.
(2) $x_1 = x_2$ and $y_1 = -y_2$
   Here we define $P + Q = Q + P = \mathcal{O}$; making them inverses.
(3) $x_1 = x_2$ and $y_1 = y_2$
   We define the line L as the tangent of P but do the same as in case 1.

Analytically this results in:

(1) $P + Q = (x_3, y_3)$ where
$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$
$$\lambda = (y_2 - y_1)(x_2 - x_1)^{-1}$$
(2) $P + Q = Q + P = \mathcal{O}$
(3) $P + Q = (x_3, y_3)$ where
$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$
$$\lambda = (3x_1^2 + a)(2y_1)^{-1}$$

In the case of odd primes $p > 3$ the elliptic curves over $\mathbb{Z}_p$ is defined analogously only changing the operations to the corresponding group operations. We do have to use the theory of quadratic residues to decide the solution set E. This can be done through Euler's criterion etc. The known cryptosystems can be transformed into an elliptic curve variant by changing from multiplicative notation to additive.

The number of points on an elliptic curve E over $\mathbb{Z}_p$ is roughly p ($p + 1 - 2\sqrt{p} \leq |E| \leq p + 1 + 2\sqrt{p}$). It is possible to compute this value exactly. Given this value we want to find a cyclic subgroup of E in which the discrete logarithm problem is intractable.

Theorem (6.1)**.** *Let E be an elliptic curve defined over $\mathbb{Z}_p$, where $p > 3$ is prime. Then there exist positive integers n_1, n_2 such that $(E, +) \simeq \mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$. Further $n_2 | n_1$ and $n_2 | (p - 1)$.*

Note that $n_2 = 1$ is allowed. In fact $n_2 = 1$ if and only if E is cyclic. Also if $|E|$ is prime or a product of distinct primes, then the group is cyclic.

By using a special sign of a quadratic residue one can save 50% of the memory at the expense of running addition computations to reconstruct the points.

One can similarly to the square-and-multiply algorithm compute point multiples on an elliptic curve.

Introduction

Definition. A signature scheme is a five-tuple $(\mathcal{P}, \mathcal{A}, \mathcal{K}, \mathcal{S}, \mathcal{V})$ satisfying the following:

(1) $\mathcal{P}$ is a finite set of possible messages.
(2) $\mathcal{A}$ is a finite set of possible signatures.
(3) $\mathcal{K}$, finite set of possible keys.
(4) For each $K \in \mathcal{K}$ there is a signing algorithm $\mathbf{sig}_K \in \mathcal{S}$ and a corresponding verification algorithm $\mathbf{ver}_K \in \mathcal{V}$. Each $\mathbf{sig}_K : \mathcal{P} \to \mathcal{A}$ and $\mathbf{ver}_K : \mathcal{P} \times \mathcal{A} \to \{\text{true, false}\}$ are functions such that the following equation is satisfied for every message $x \in \mathcal{P}$ and for every signature $y \in \mathcal{A}$

$$\mathbf{ver}_K(x, y) = \begin{cases} \text{true}, & \text{if } y = \mathbf{sig}_K(x) \\ \text{false}, & \text{if } y \neq \mathbf{sig}_K(x) \end{cases}$$

A pair (x, y) with $x \in \mathcal{P}, y \in \mathcal{A}$ is called a signed message.

The verification function is public and the signing is private. The should be polynomial-time functions, and given a message x computationally infeasible for anyone other than Alice to compute a signature y such that $\mathbf{ver}_K(x, y) = \text{true}$. Forgeries are defined in the same way as for MACs.

The RSA cryptosystem can be used as a signature scheme:

Cryptosystem (RSA signature scheme)**.** *Let $n = pq$ where p, q distinct primes. Let $\mathcal{P} = \mathcal{A} = \mathbb{Z}_n$ and define:*

$$\mathcal{K} = \{(n, p, q, a, b) : n = pq,\ p, q \text{ primes}, ab \equiv 1 \mod \varphi(n)\}$$

The values n, b are the public key and the values p, q, a are the private key. For $K = (n, p, q, a, b)$ define for $x, y \in \mathbb{Z}_n$:

$$\mathbf{sig}_K(x) = x^a \mod n$$

and

$$\mathbf{ver}_K(x, y) = \textit{true} \Leftrightarrow x \equiv y^b \mod n$$

This is essentially RSA used with $e_K = \mathbf{ver}$ and $d_K = \mathbf{sig}$. This because of the private and public values in a signature scheme.

Security requirements for signature schemes

Schnorr signature scheme

Full domain hash

Diffie-Hellman key agreement