Foundations of Cryptography

Summary of the course DD2448 taught at
KTH Royal Institute of Technology by Douglas Wikström

Phillip Gajland

Spring 2019

Lecture 1 - Introduction & Symmetric Cryptosystems

General

- Alice encrypts a message m using key k and encryption algorithm E such that $c = E_k(m)$. Bob decrypts the ciphertext c using the same key k and decryption algorithm E^{-1} such that $m = E_k^{-1}(c)$.

- Mathematically, a cryptosystem can be defined as a tuple $(\mathcal{G}en, \mathcal{P}, E, E^{-1})$ where:

  - $\mathcal{G}en$ is a key generation algorithm for keys in the key space $\mathcal{K}$.

  - $\mathcal{P}$ is the set of plaintexts.

  - E is a deterministic encryption algorithm.

  - E^{-1} is a deterministic decryption algorithm.

  such that $E_k^{-1}(E_k(m)) = m$ for every message $m \in \mathcal{P}$ and $k \in \mathcal{K}$

- The set $\mathcal{C} = E_k(m) \mid m \in \mathcal{P} \wedge k \in \mathcal{K}$ is called the set of ciphertexts.

  (Pronounced: $E_k(m)$ *such that* m *is in* $\mathcal{P}$ *and* k *is in* $\mathcal{K}$*.* I.e. all combinations of keys k and messages m.

Caesar Cipher

- In an alphabet containing 26 letters, the key k is such that $k \in \mathbb{Z}_{26}$.
- The plaintext $m = (m_1, ..., m_n) \in \mathbb{Z}_{26}^n$ gives ciphertext $c = (c_1, ..., c_n)$.
- Encryption is given by $c_i = m_i + k \mod 26$.
- Decryption is given by $m_i = c_i - k \mod 26$.
- The key space $\mathcal{K}$ is too small, making it susceptible to brute force attacks.
- A frequency analysis can be done by maximising the inner product $T(E^{-1}(C)) \cdot F$ where $T(s) \cdot F$ denotes the frequency table of string s and the English language respectively.

Lecture 2 - More Symmetric Cryptosystems

Affine Cipher

- The key k is given by a random pair (a, b), where $a \in \mathbb{Z}_{26}$ is relatively prime to 26, and $b \in \mathbb{Z}_{26}$.
- The plaintext $m = (m_1, ..., m_n) \in \mathbb{Z}_{26}^n$ gives ciphertext $c = (c_1, ..., c_n)$.
- Encryption is given by $c_i = am_i + b \mod 26$.
- Decryption is given by $m_i = (c_i - b)a^{-1} \mod 26$.
- *Relative primality of a and 26 implies that $(a^{-1} \mod 26)$ exists.*

Substitution Cipher

- Both the Caesar cipher and affine cipher are examples of substitution ciphers.
- The key is a random permutation $\sigma \in \mathcal{S}$ of the symbols in the alphabet, for some subset $\mathcal{S}$ of all permutations.
- The plaintext $m = (m_1, ..., m_n) \in \mathbb{Z}_{26}^n$ gives ciphertext $c = (c_1, ..., c_n)$.
- Encryption is given by $c_i = \sigma(m_i)$.
- Decryption is given by $m_i = \sigma^{-1}(c_i)$.

Generic Attacks on Substitution Ciphers

- A **digram** is an ordered pair of symbols.
- A **trigram** is an ordered triple of symbols.
- It is useful to compute frequency tables for the most frequent digrams and trigrams, and not only the frequencies for individual symbols.

  1. Compute symbol / digram / trigram frequency tables for the candidate language and the ciphertext.
  2. Try to match symbols / digrams / trigrams with similar frequencies.
  3. Try to recognise words to confirm guesses (using dictionary or Google).
  4. Repeat until the plaintext can be guessed.

- This is hard when several symbols have similar frequencies - a large amount of cipher text is needed.

Vigenère Cipher

- The key is given by $k = (k_0, ..., k_{l-1})$, where $k_i \in \mathbb{Z}_{26}$ is random.
- The plaintext $m = (m_1, ..., m_n) \in \mathbb{Z}_{26}^n$ gives ciphertext $c = (c_1, ..., c_n)$.
- Encryption is given by $c_i = m_i + k_{i \bmod l} \mod 26$.
- Decryption is given by $m_i = c_i - k_{i \bmod l} \mod 26$.
- *This gives a more uniform frequency table.*

Attack on Vigenère Cipher

- Each probability distribution $p_1, ..., p_n$ on n symbols may be viewed as a point $p = (p1, ..., p_n)$ on a $n-1$ dimensional hyperplane in $\mathbb{R}^n$ orthogonal to the vector $\overline{1} = (1, ..., 1)$.
- Such a point $p = (p_1, ..., p_n)$ is at a distance $\sqrt{F(p)}$ from the origin, where $F(p) = \sum_{i=1}^{n} p_i^2$.
- It is clear that p is closest to the origin, when p is the uniform distribution, i.e., when $F(p)$ is minimised.
- $F(p)$ is invariant under permutation of the underlying symbols. Use tools to check if a set of symbols is the result of some substitution cipher.

  1. For $l = 1, 2, 3, ...$ we form
     $$\begin{pmatrix} C_0 \\ C_1 \\ \vdots \\ C_{l-1} \end{pmatrix} = \begin{pmatrix} c_0 & c_l & c_{2l} & \cdots \\ c_1 & c_{l+1} & c_{2l+1} & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ c_{l-1} & c_{2l-1} & c_{3l-1} & \cdots \end{pmatrix}$$
     and compute $f_l = \frac{1}{l} \sum_{i=0}^{l-1} F(C_i)$.
  2. The local maximum with smallest l is probably the right length.
  3. Then attack each C_i separately to recover k_i, using the attack against the Caesar cipher.

Hill Cipher

- The key is given by $k = A$, where a is an invertible $l \times l$-matrix over $\mathbb{Z}_{26}$.
- The plaintext $m = (m_1, ..., m_n) \in \mathbb{Z}_{26}^n$ gives ciphertext $c = (c_1, ..., c_n)$.
- Encryption is given by $(c_{i+0}, ..., c_{i+l-1}) = (m_{i+0}, ..., m_{i+l-1})A$.
- Decryption is given by $(c_{i+0}, ..., c_{i+l-1}) = (m_{i+0}, ..., m_{i+l-1})A^{-1}$.

  for $i = 1, l+1, 2l+1, ...$
- The Hill cipher is easy to break using a **known plaintext attack**.

Permutation Cipher

- The permutation cipher is a special case of the Hill cipher.
- The key is given by a random permutation $\pi \in \mathcal{S}$ for some subset $\mathcal{S}$ of the set of permutation of $\{0, 1, 2, ..., l-1\}$.
- The plaintext $m = (m_1, ..., m_n) \in \mathbb{Z}_{26}^n$ gives ciphertext $c = (c_1, ..., c_n)$.
- Encryption is given by $c_i = m_{\lfloor i/l \rfloor + \pi(i \mod l)}$.
- Decryption is given by $m_i = c_{\lfloor i/l \rfloor + \pi^{-1}(i \mod l)}$.

Summary of Simple Ciphers

- Caesar cipher and affine cipher: $m_i \mapsto am_i + b$.
- Substitution cipher (generalise Caesar / affine): $m_i \mapsto \sigma(m_i)$.
- Vigenère cipher (more uniform frequency table): $m_i \mapsto m_i + k_{i \mod l}$.
- Hill cipher (invertible linear map): $(m_1, ..., m_l) \mapsto (m_1, ..., ..., m_l)A$.
- Transposition cipher (permutation): $(m_1, ..., m_l) \mapsto (m_{\pi(1)}, ..., m_{\pi(l)})$ equivalent to: $(m_1, ..., m_l) \mapsto (m_1, ..., m_l)M_\pi$.

Good Block Ciphers

- Simple ciphers are bad, but what makes a good block cipher?
- For every key a block-cipher with plaintext / ciphertext space $\{0, 1\}^n$ gives a permutation of $\{0, 1\}^n$.
  - What would be a good cipher?
- A good cipher is one where each key gives a **randomly chosen permutation** of $\{0, 1\}^n$.
  - Why is this not possible?

- The representation of a single typical function $\{0,1\}^n \rightarrow \{0,1\}^n$ requires roughly $n2^n$ bits ($147 \times 10^{6 \cdot 3}$ for $n = 64$).
    - What should we look for instead?
- **Idea:** Compose smaller weak ciphers into a large one. Mix the components thoroughly. Claude Shannon (1948) introduces two terms:
    - **Diffusion:** "In the method of diffusion the statistical structure of M which leads to its redundancy is dissipated into long range statistics..."
    - **Confusion:** "The method of confusion is to make the relation between the simple statistics of E and the simple description of K a very complex and involved one."

Lecture 3 - Substitution-Permutation Networks & AES

Substitution-Permutation Networks

- Block-size: We use a block-size of $n = l \times m$ bits.
- Key Schedule: Round r uses its own round key K_r derived from the key K using a key schedule.
- Each Round the following is invoked:
    1. Round Key: xor with the round key.
    2. Substitution: l substitution boxes each acting on one m-bit word (m-bit S-Boxes).
    3. Permutation: A permutation π_i acting on $\{1, ..., n\}$ to reorder the n bits.

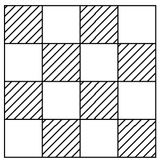
A Simple Block Cipher

- $|P| = |C| = 16$
- 4 rounds
- $|K| = 32$
- r^{th} round key K_r consists of the $4r^{\text{th}}$ to the $(4r+16)^{\text{th}}$ bits of key K.
- 4-bit S-Boxes
- S-Boxes the same ($S \neq S^{-1}$)
- $Y = S(X)$
- Can be described using 4 boolean functions.

Advanced Encryption Standard (AES)

- Chosen in worldwide public competition 1997-2000. Probably no backdoors. Increased confidence!
- Winning proposal named "Rijndael", by Rijmen and Daemen.
- Family of 128-bit ciphers: {Key bits, Rounds} - {128, 10}, {192, 12}, {256, 14}.
- The first key-recovery attacks on full AES found by Bogdanov, Khovratovich, and Rechberger was published in 2011 and is faster than brute force by a factor of about 4.
- The algebraics of AES have made some people *uneasy*, but they have been uneasy for years now...
  - AddRoundKey: xor with round key.
  - SubBytes: Substitution of bytes.
  - ShiftRows: Permutation of bytes.
  - MixXolumns: Linear map.

- The 128 bit state is interpreted as a 4×4 matrix of bytes.



- Something like a mix between substitution, permutation, affine version of Hill cipher. In each round!
- SubBytes is a field inversion in $\mathbb{F}_{2^8}$ plus affine map in $\mathbb{F}_2^8$.
- ShiftRows is ac cyclic shift of bytes with offsets: 0, 1, 2, and 3.
- MixColumns is an invertible linear map over $\mathbb{F}_{2^8}$ (with irreducible polynomial $x^8 + x^4 + x^3 + x + 1$) with good diffusion.
- Decryption uses the following transforms:
    - AddRoundKey
    - InvSubBytes
    - InvShiftRows
    - InvMixColumns

Feistel Networks

- Identical rounds are iterated, but with different round keys.
- The input to the i^{th} round is divided in a left and right part, denoted L^{i-1} and R^{i-1}.
- f is a function for which it is somewhat hard to find pre-images, but f is **not invertible**!
- One round is defined by:
  $L^i = R^{i-1}$
  $R^i = L^{i-1} \oplus f(R^{i-1}, K^i)$
  where K^i is the i^{th} round key.
- The inverse Feistel round is given by:
  $L^{i-1} = R^i \oplus f(L^i, K^i)$
  $R^{i-1} = L^i$
  I.e. reverse direction and swap left and right.

Data Encryption Standard (DES)

- Developed at IBM in 1975, or perhaps at NSA; not publicly known.
- 16-round Feistel network.
- Key schedule derives permuted bits for each round key from a 56-bit key. Supposedly not 64-bit due to parity bits.
- DES's f-Function is given by: $f(R^{i-1}, K^i)$

Security of DES

- Brute Force: Try all $2^5 6$ keys. Done in practice with special chip by Electronic Frontier Foundation, 1998. Possibly much earlier by NSA and others.
- Differential Cryptanalysis: 2^{47} chosen plaintexts, Biham and Shamir, 1991. Known earlier by IBM and NSA. DES is surprisingly resistant!
- Linear Cryptanalysis: 2^{43} known plaintexts, Matsui, 1993. Probably **not** known by IBM and NSA!
- Since the key space for DES is too small, one way to increase it is to use DES twice, so called "double DES". $2DES_{k_1,k_2}(x) = DES_{k_2}(DES_{k_1}(x))$.
- However, this is **not** more secure than normal DES!
- Meet-in-the-middle attack:
  - Get hold of a plaintext-ciphertext pair (m, c).
  - Compute $X = \{x \mid k_1 \in \mathcal{K}_{DES} \wedge x = E_{k_1}(m)\}$.
  - For $k_2 \in \mathcal{K}_{DES}$ check if $E_{k_2}^{-1}(c) = E_{k_1}(m)$ for some k_1 using the table X. If so, then (k_1, k_2) is a good candidate.
  - Repeat with (m', c'), starting from the set of candidate keys to identify the correct key.
- Tripple DES: $3DES_{k_1,k_2,k_3}(x) = DES_{k_3}(DES_{k_2}(DES_{k_1}(x)))$.
- Seemingly 112 bit "effective" key size.
- 3 times as slow as DES. DES is slow in software, and this is even worse. One of the motivation for AES.
- Triple DES is sill considered to be secure.

Modes of Operation

- 5 modes of operation:
    - Electronic codebook mode (ECB mode).
    - Cipher feedback mode (CFB mode).
    - Cipher block chaining mode (CBC mode).
    - Output feedback mode (OFB mode).
    - Counter mode (CTR mode).
- **Electronic codebook mode** - encrypt each block independently: $c_i = E_k(m_i)$.
- Identical plaintext blocks give identical ciphertext blocks.
- **Cipher feedback mode** - xor plaintext block with previous ciphertext block **after** encryption:
  $c_0 =$ initialisation vector
  $c_i = m_i \oplus E_k(c_{i-1})$.
- Sequential encryption and parallel decryption.
- Self-synchronising and unidirectional.
- **Cipher block chaining mode** - xor plaintext block with previous ciphertext block **after** encryption:
  $c_0 =$ initialisation vector
  $c_i = E_k(c_{i-1} \oplus m_i)$.
- Sequential encryption and parallel decryption.
- Self-synchronising.
- **Output feedback mode** - generate stream, xor plaintexts with stream (emulate "one-time pad"):
  $s_0 =$ initialisation vector
  $s_i = E_k(s_{i-1})$
  $c_i = s_i \oplus m_i$.
- Sequential.

- Synchronous.
- Allows batch processing.
- Malleable!
- **Counter mode** - generate stream, xor plaintexts with stream (emulate ”one-time pad”):
  $s_0 =$ initialisation vector
  $s_i = E_k(s_0||i)$
  $c_i = s_i \oplus m_i$.
- Parallel.
- Synchronous.
- allows batch processing.
- Malleable!

Lecture 4 - Cryptanalysis of the Simple Permutation Network

- Find an expression of the following form with a high probability of occurrence.

$$P_{i_1} \oplus \cdots \oplus P_{i_p} \oplus C_{j_1} \oplus \cdots \oplus C_{j_c} = K_{l_1,s_1} \oplus \cdots \oplus K_{l_k,s_k}$$

- Each random plaintext / ciphertext pair gives an estimate of

$$K_{l_1,s_1} \oplus \cdots \oplus K_{l_k,s_k}$$

- Collect many pairs and make a better estimate based on the majority vote.
- How do we come up with the desired expression?
- How do we compute the required number of samples?

Bias

- The bias $\epsilon(X)$ of a binary random variable X is defined by
$$\epsilon(X) = Pr[X = 0] - \frac{1}{2}$$
$\approx 1/\epsilon^2(X)$ samples are required to estimate X.

Linear Approximation of S-Box

- Let X and Y be the input and output of an S-box, i.e. $\boldsymbol{Y = S(X)}$.
- We consider the bias of linear combinations of the form
$$a \cdot X \oplus b \cdot Y = \left(\bigoplus_i a_i X_i \right) \oplus \left(\bigoplus_i b_i Y_i \right)$$
- Example: $X_2 \oplus X_3 = Y_1 \oplus Y_3 \oplus Y_4$. The expression holds in 12 out of the 16 cases. Hence, it has a bias of (12-8)/16 = 4/16 = 1/4.
- Let $N_L(a, b)$ be the number of zero-outcomes of a $a \cdot X \oplus b \cdot Y$.
- The bias is then
$$\epsilon(a \cdot X \oplus b \cdot Y = \frac{N_L(a, b) - 8}{16},$$
since there are four bits in X, and Y is determined by X.
- This gives a linear approximation for one round.
- How do we come up with a linear approximation for more rounds?

Piling-Up Lemma

- Let $X_1, ..., X_t$ be independent binary random variables and let $\epsilon_i = \epsilon(X_i)$. Then
$$\epsilon \left(\bigoplus_i X_i \right) = 2^{t-1} \prod_i \epsilon_i.$$
- Proof: Case $t = 2$:
$$\begin{aligned} Pr[X_1 \oplus X_2 = 0] &= Pr[X_1 = 0 \wedge X_1 = 0) \vee (X_1 = 1 \wedge X_1 = 1)] \\ &= (\frac{1}{2} + \epsilon_1)(\frac{1}{2} + \epsilon_2) + (\frac{1}{2} - \epsilon_1)(\frac{1}{2} - \epsilon_2) \\ &= \frac{1}{2} + 2\epsilon_1\epsilon_2. \end{aligned}$$
By induction $Pr[X_1 \oplus \cdots \oplus X_t = 0] = \frac{1}{2} + 2^{t-1} \prod_i \epsilon_i$

Attacking a Linear Trail

- Four linear approximations with $|\epsilon_i| = 1/4$

$$\begin{aligned} S_{12} &: X_1 \oplus X_3 \oplus X_4 = Y_2 \\ S_{22} &: X_2 = Y_2 \oplus Y_4 \\ S_{32} &: X_2 = Y_2 \oplus Y_4 \\ S_{24} &: X_2 = Y_2 \oplus Y_4 \end{aligned}$$

  Combine them to get:

$$U_{4,6} \oplus U_{4,8} \oplus U_{4,14} \oplus U_{4,16} \oplus P_5 \oplus P_7 \oplus P_8 = \bigoplus K_{i,j}$$

  with bias $|\epsilon| = 2^{4-1}(\frac{1}{4})^4 = 2^{-5}$

- Our expression (with bias 2^{-5}) links plaintext bits to input bits to the 4th round.

- Partially undo the last round by guessing the last key. Only 2 S-Boxes are involved, i.e., $2^8 = 256$ guesses.

- For a correct guess, the question holds with bias 2^{-5}. For a wrong guess, it holds with a bias zero (harmless lie).

- Required pairs $2^{10} \approx 1000$. Attack complexity $2^{18} \ll 2^{32}$ operations.

Linear Cryptanalysis Summary

- Linear Cryptanalysis is a **known plaintext attack**.
  - Find linear approximation of S-Boxes.
  - Compute bias of each approximation.
  - Find linear trails.
  - Compute bias of linear trails.
  - Compute data and time complexity.
  - Estimate key bits from many plaintext-ciphertext pairs.

Ideal Block Cipher

- A function $\epsilon(n)$ is negligible if for every constant $c > 0$, there exists a constant n_0, such that
$$\epsilon(n) < \frac{1}{n^c}$$
for all $n \geq n_0$.
- Motivation: Events happening with negligible probability can not be exploited by polynomial time algorithms! (they "never" happen!)
- Caveat! Theoretic notion. Interpret with care in practice.
- A function is pseudo-random if no efficient adversary can distinguish between the function and a random function.
- A family of functions $F : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ is pseudo-random if for all polynomial time oracle adversaries A
$$\left| \Pr_K \left[A^{F_K(\cdot)} = 1 \right] - \Pr_{R:\{0,1\}^n \rightarrow \{0,1\}^n} \left[A^{R(\cdot)} = 1 \right] \right|$$
is negligible.
- A permutation and its inverse are pseudo-random if no efficient adversary can distinguish between the permutation and its inverse, and a random permutation and its inverse.
- A family of permutations $P : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ is pseudo-random if for all polynomial time oracle adversaries A
$$\left| \Pr_K \left[A^{P_K(\cdot), P_K^{-1}(\cdot)} = 1 \right] - \Pr_{\Pi \in \mathcal{S}_{2^n}} \left[A^{\Pi(\cdot), \Pi^{-1}(\cdot)} = 1 \right] \right|$$
is negligible, where $\mathcal{S}_{2^n}$ is the set of permutations of $\{0,1\}^n$.

Idealised Four-Round Feistel Network

- Feistel round (H for "Horst Feistel").
$$H_{F_K}(L, R) = (R, L \oplus F(R, K))$$
- Theorem: (Luby and Rackoff) If F is a pseudo-random family of functions, then
$$H_{F_{k_1}, F_{k_2}, F_{k_3}, F_{k_4}}(x) = H_{F_{k_4}}(H_{F_{k_3}}(H_{F_{k_2}}(H_{F_{k_1}}(x))))$$
(and its inverse) is a pseudo-random family of permutations.
- Why do we need four rounds?

Perfect Secrecy

- When is a cipher perfectly secure?
- How should we formalise this?
- A cryptosystem has perfect secrecy if guessing the plaintext is equally hard to do regardless of whether or not the ciphertext is given.
- A cryptosystem has perfect secrecy if

  $$Pr[M = m \mid C = c] = Pr[M = m]$$

  for every $m \in \mathcal{M}$ and $c \in \mathcal{C}$, where M and C are random variables taking values over $\mathcal{M}$ and $\mathcal{C}$.
- Game Based Definition: Exp_A^b, where A is a strategy:
  - $k \leftarrow_R \mathcal{K}$
  - $(m_0, m_1) \leftarrow A$
  - $c = E_k(m_b)$
  - $d \leftarrow A(c)$, with $d \in \{0, 1\}$
  - Output d.
- A cryptossystem has perfect secrecy if for every computationally unbounded strategy A,

  $$Pr[Exp_A^0 = 1] = Pr[Exp_A^1 = 1].$$

One-Time Pad (OTP)

- The key is given by a random tuple $k = (b_0, ..., b_{n-1}) \in \mathbb{Z}_2^n$.
- The plaintext $m = (m_0, ..., m_{n-1}) \in \mathbb{Z}_2^n$ gives ciphertext $c = (c_0, ..., c_{n-1})$.
- Encryption is given by $c_i = m_i \oplus b_i$.
- Decryption is given by $m_i = c_i \oplus b_i$.

Bayes' Theorem and OTP's Perfect Secrecy

- If A and B are events and $Pr[B] > 0$, then
$$Pr[A \mid B] = \frac{Pr[A]Pr[B \mid A]}{Pr[B]}$$
- Probabilistic Argument. Bayes implies that:
$$\begin{aligned} Pr[M = m \mid C = c] &= \frac{Pr[M = m]Pr[C = c \mid M = m]}{Pr[C = c]} \\ &= Pr[M = m]\frac{2^{-n}}{2^{-n}} \\ &= Pr[M = m]. \end{aligned}$$
- Simulation Argument: The ciphertext is uniformly and independently distributed form the plaintext. We can simulate it on our own!
- Bad News! "For every cipher with perfect secrecy, the key requires at least as much space to represent as the plaintext."
  - Dangerous in practice to rely on no reuse of, e.g., file containing randomness!

Lecture 5 - Hash Functions & Random Oracles

Universal Hash Functions

- An ensemble $f = \{f_\alpha\}$ of hash functions $f_\alpha : X \to Y$ is (strongly) 2-universal if for every $x, x' \in X$ and $y, y' \in Y$ with $x \neq x'$ and a random α
$$Pr[f_\alpha(x) = y \wedge f_\alpha(x') = y'] = \frac{1}{|Y|^2}.$$
I.e., for any fixed $x' \neq x$, the outputs $f_\alpha(x)$ and $f_\alpha(x')$ are uniformly and independently distributed when α is chosen randomly.

  In particular x and x' are both mapped to the same value with probability $\frac{1}{|Y|}$.
- Example: The function $f : \mathbb{Z}_p \to \mathbb{Z}_p$ for prime p defined by
$$f(z) = az + b \mod p$$
is strongly 2-universal
- Proof: Let $x, x', y, y' \in \mathbb{Z}_p$ with $x \neq x'$. Then
$$\begin{pmatrix} x & 1 \\ x' & 1 \end{pmatrix}\begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} y \\ y' \end{pmatrix}$$
has a unique solution. Random (a, b) satisfies this solution with probability $\frac{1}{p^2}$.
- Universal hash functions are **not** one-way or collision resistant!

Hash Functions

- A hash function maps arbitrary long bit strings into strings of fixed length.
- The output of a hash function should be ”unpredictable”
- The following properties should be met by a hash function:
  - Finding a pre-image of an output should be hard.
  - Finding two inputs giving the same output should be hard.
  - The output of the function should be ”random”.
- Let $f : \{0,1\}^* \rightarrow \{0,1\}$ be a polynomial time commutable function.
- We can derive an ensemble $\{f_n\}_{n\in\mathbb{N}}$, with
  $$f_n : \{0,1\}^n \rightarrow \{0,1\}^*$$
  by setting $f_n(x) = f(x)$.
- Note that we may recover f form the ensemble by $f(x) = f_{|x|}(x)$.
- When convenient we give definitions for a function, but it can be turned into a definition for an ensemble.
- Consider $F = \{f_n\}_{n\in\mathbb{N}}$, where f_n is itself an ensemble $\{f_{n,\alpha_n}\}_{\alpha_n\in\{0,1\}^n}$, with
  $$f_{n,\alpha_n} : \{0,1\}^{l(n)} \rightarrow \{0,1\}^{l'(n)}$$
  for some length polynomials $l(n)$ and $l'(n)$.
- Here n is the security parameter and α_n is a ”key” that is chosen randomly.
- We may also view F as an ensemble $\{f_\alpha\}$, where $f_\alpha = \{f_{n,\alpha_n}\}_{n\in\mathbb{N}}$ and $\alpha = \{\alpha_n\}_{n\in\mathbb{N}}$.
- These conventions allow us to talk about what in everyday language is a ”function” f in several convenient ways.
- FROM NOW ON WE CAN FORGET THE ABOVE AND ASSUME EVERYTHING WORKS....

One-Wayness

- Definition: A function $f : \{0,1\}^* \to \{0,1\}^*$ is said to be one-way if for every polynomial time algorithm A and a random x
$$Pr[A(f(x)) = x' \wedge f(x') = f(x)] < \epsilon(n)$$
for a negligible function ϵ.
- Normally f is computable in polynomial time in its input size.
- Definition: A function $h : \{0,1\}^* \to \{0,1\}^*$ is said to be second pre-image resistant if for every polynomial time algorithm A and a random x
$$Pr[A(x) = x' \wedge x' \neq x \wedge f(x') = f(x)] < \epsilon(n)$$
for a negligible function ϵ.
- Note that A is given not only the output of f, but also the input x, but it must find a second pre-image.
- Definition: Let $f = \{f_\alpha\}_\alpha$ be an ensemble of functions. the "function" f is said to be collision resistant if for every polynomial time algorithm A and randomly chosen α
$$Pr[A(\alpha) = (x, x') \wedge x \neq x' \wedge f_\alpha(x') = f_\alpha(x)] < \epsilon(n)$$
for a negligible function ϵ.
- An algorithm that gets a small "advice string" for each security parameter can easily hardcode a collision for a fixed function f, which explains the random index α.

Relations for Compressing Hash Functions

- If a function is not second pre-image resistant, then it is not collision-resistant.
  - Pick random x.
  - Request second pre-image $x' \neq x$ with $f(x') = f(x)$.
  - Output x' and x.
- If a function is not one-way, then it is not second pre-image resistant.
  - Given a random x, compute $y = f(x)$.
  - Request pre-image x' of y.
  - Repeat until $x' \neq x$, and output x'.

Random Oracles

- A random oracle is simply a randomly chosen function with appropriate domain and range.
- A random oracle is the perfect hash function. Every input is mapped independently and uniformly in the range.
- Let us consider how a random oracle behaves with respect to our notions of security of hash functions.

Pre-Image of Random Oracle

- We assume with little loss that an adversary always ”knows” if it has found a pre-image, i.e., it queries the random oracle on its output.
- Theorem: Let $H : X \to Y$ be a randomly chosen function and let $x \in X$ be randomly chosen. Then for every algorithm A making q oracle queries

$$Pr[A^{H(\cdot)}(H(x)) = x' \wedge H(x) = H(x')] \leq 1 - \left(1 - \frac{1}{|Y|}\right)^q .$$

- Proof: Each query x' satisfies $H(x') \neq H(x)$ independently with probability $1 = \frac{1}{|Y|}$.

Second Pre-Image of Random Oracle

- We assume with loss that an adversary always ”knows” if it has found a second pre-image, i.e., it quries the random oracle on the input and its output.
- Theorem: Let $H : X \to Y$ be a randomly chosen function and let $x \in X$ be randomly chosen. Then for every such algorithm A making q oracle queries

$$Pr[A^{H(\cdot)}(x) = x' \wedge x \neq x' \wedge H(x) = H(x')] \leq 1 - \left(1 - \frac{1}{|Y|}\right)^{q-1} .$$

- Proof: Same as pre-image case, except we must waste one query on the input value to get the target in Y.

Collision Resistance of Random Oracles

- We assume with little loss that and adversary always "knows" if it has found a collision, i.e. it queries the random oracle on its outputs.
- Theorem: Let $H : X \to Y$ be a randomly chosen function and let $x \in X$ be randomly chosen. Then for every such algorithm A making q oracle queries

$$Pr[A^{H(\cdot)} = (x, x') \wedge x \neq x' \wedge H(x) = H(x')] \leq 1 - \prod_{i=1}^{q-1} \left(1 - \frac{i}{|Y|}\right)$$

$$Pr[A^{H(\cdot)} = (x, x') \wedge x \neq x' \wedge H(x) = H(x')] \leq \frac{q(q-1)}{2|Y|}.$$

- Proof: $1 - \frac{i-1}{|Y|}$ bounds the probability that the i^{th} query does not give a collision for any of the $i - 1$ previous queries, conditioned on no previous collisions.

Lecture 6 - Hash Functions and MACs

Iterated Hash Functions (Merkle-Damgård)

- Suppose that we are given a collision resistant hash function

$$f : \{0,1\}^{n+t} \to \{0,1\}^n.$$

- How can we construct a collision resistant hash function

$$f : \{0,1\}^* \to \{0,1\}^n$$

  mapping any length inputs?

- Construction:
  - Let $x = (x_1, ..., x_k)$ with $|x_i| = t$ and $0 < |x_k| \leq t$.
  - Let x_{k+1} be the total number of bits in x.
  - Pad x_k with zeros until it has length t.

  - $y_0 = 0^n, y_i = f(y_{i-1}, x_i)$ for $i = 1, ..., k+1$.
  - Output y_{k+1}

- Here the total number of bits is bounded by $2^t - 1$, but this can be relaxed.
- Suppose A finds collisions in Merkle-Damgård.
  - If the number of bits differ in a collision, then we can derive a collision from the last invocation of f.
  - If not, then we move backwards until we get a collision. Since both inputs have the same length, we are guaranteed to find a collision.

Standardised Hash Functions

- Despite that theory says it is impossible, in practice people simply live with **fixed** hash functions and use them as if they are randomly chosen functions.
- **SHA**
  - Secure Hash Algorithm (SHA-0,1, and the SHA-2 family) are hash functions standardised by NIST to be used in, e.g., signature schemes and random number generation.
  - SHA-0 was weak and withdraws by NIST. SHA-1 was withdrawn 2010. The SHA-2 family is based on similar ideas but seems safe so far.
  - All are iterated hash functions, starting from a basic compression function.
- **SHA-3**
  - NIST ran an open competition for the next hash function, named SHA-3. Several groups of famous researchers submitted proposals.
  - Call for SHA-3 explicitly asked for "different" hash functions.
  - The competition ended on October 2, 2012, and the hash function Keccak was selected as the winner.
  - It was constructed by Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche.

Message Authentication Codes (MACs)

- Message Authentication Codes (MACs) are used to ensure integrity and authentication of messages.
- Scenario:
  - Alice and Bob share a common key k.
  - Alice computes an authentication tag $\alpha = MAC_k(m)$ and sends (m, α) to Bob.
  - Bob receives (m', α') form Alice, but before accepting m' as coming from Alice, Bob checks that $MAC_k(m') = \alpha'$.

Security of a MAC

- A message authentication code MAC is secure if for a random key k and every polynomial time algorithm A,

  $$Pr[A^{MAC_k(\cdot)} = (m, \alpha) \wedge MAC_k(m) = \alpha \wedge \forall i : m \neq m_i]$$

  is negligible, where m_i is the i^{th} query to the oracle $MAC_k(\cdot)$.

Random Oracle As MAC

- Suppose that $H : \{0,1\}^* \to \{0,1\}^n$ is a random oracle.
- Then we can construct a MAC as $MAC_k(m) = H(k, m)$.
- Could we plug in an iterated hash function in place of the random oracle?

HMAC

- Let $H : \{0,1\}^* \to \{0,1\}^n$ be a "cryptographic hashfunction", e.g. SHA-256.
- $HMAC_{k_1,k_2}(x) = H\big(k_2 || H(k_1 || x)\big)$
- This is provably secure under the assumption that
  - $H(k_1 || \cdot)$ is unknown-key collision resistant, and
  - $H(k_2 || \cdot)$ is a secure MAC for fixed-size messages.

Lecture 7 - MACs and Information Theory

MACs

CBC-MAC

- Let E be a secure block-cipher, and $x = (x_1, ..., x_t)$ an input. The MAC-key is simply the block-cipher key.
  - $y_0 = 000...0$
  - For $i = 1, ..., t$, $y_i = E_k(y_{i-1} \oplus x_i)$
  - Return y_t.
- Is this secure?

Universal Hashfunction As MAC

- Theorem: A t-universal hashfunction f_α for a randomly chosen secret α is an **unconditionally secure** MAC, provided that the number of queries is smaller than t.

Information Theory

- Information theory is a mathematical theory of communication.
- Typical questions studied are how to compress, transmit, and store information.
- Information theory is also useful to argue about some cryptographic schemes and protocols.
- Memory Source Over Finite Alphabet: A source produces symbols from an alphabet $\Sigma = \{a_1, ..., a_n\}$. Each generated symbol is independently distributed.
- Binary Channel: A binary channel can (only) send bits.
- Coder/Decoder: Our goal is to come up with a scheme to:
  - Convert a symbol a from the alphabet Σ into a sequence $(b_1, ..., b_l)$ of bits,
  - send the bits over the channel, and
  - decode the sequence into a again at the receiving end.
- Optimisation goal: We want to minimise the **expected** number of bits/symbols we send over the binary channel, i.e., if X is a random variable over Σ and $l(x)$ is the length of the codeword of x then we wish to minimise.

$$E[l(X)] = \sum_{x \in \Sigma} P_X(x) l(x).$$

Examples

- X takes values in $\sigma = \{a, b, c, d\}$ with uniform distribution. How would you encode this?
- X takes values in $\sigma = \{a, b, c\}$, with $P_X(a) = \frac{1}{2}$, $P_X(b) = \frac{1}{4}$, and $P_X(c) = \frac{1}{4}$. How would you encode this?
- It seems we need $l(x) = \log |\Sigma|$. This gives the Hartley measure.
- It seems we need $l(x) = \log \frac{1}{P_X(x)}$ bits to encode x.
- Let us turn this expression into a definition.
- Let X be a random variable taking values in $\mathcal{X}$. Then the entropy of X is
$$H(X) = -\sum_{x \in \Sigma} P_X(x) \log P_X(x).$$
- Examples and intuition are nice, but what we need is a theorem that states that this is **exactly** the right length of an optimal code.

Jensen's Inequality

- Definition: A function $f : \mathcal{X} \to (a, b)$ is **concave** if
$$\lambda \cdot f(x) + (1 - \lambda) f(y) \leq f(\lambda \cdot x + (1 - \lambda) y),$$
for every $x, y \in (a, b)$ and $0 \leq \lambda \leq 1$.
- Theorem: Suppose f is continuous and strictly concave on (a, b), and X is a discrete random variable. Then
$$E[f(X)] \leq f(E[X]),$$
with equality if and only if X is constant.
- Proof idea: Consider two points + induction over number of points.

Kraft's Inequality

- Theorem: There exists a prefix-free code E with codeword lengths l_x, for $x \in \Sigma$ if and only if
$$\sum_{x\in\Sigma} 2^{-l_x} \leq 1.$$

- Proof Sketch: Given a prefix-fee code, we consider the corresponding binary tree with codewords at the leaves. We may "fold" it by replacing two siblings leaves $E(x)$ and $E(y)$ by (xy) with length $l_x - 1$. Repeat.

- Given lengths $l_{X_1} \leq l_{X_1} \leq ... \leq l_{X_n}$ we start with the complete binary tree of depth l_{X_n} and prune it.

Binary Source Coding Theorem

- Theorem: Let E be an optimal code and let $l(x)$ be the length of the codeword of x. Then
$$H(X) \leq E[l(X)] < H(X) + 1.$$

- Proof of Upper Bound: Define $l_x = \lceil -\log P_X(x) \rceil$. Then we have
$$\sum_{x\in\Sigma} 2^{-l_x} \leq \sum_{x\in\Sigma} 2^{\log P_X(x)} = \sum_{x\in\Sigma} P_X(x) = 1$$
Kraft's inequality implies that there is a code with codeword lengths l_x. Then note that $\sum_{x\in\Sigma} P_X(x)\lceil -\log P_X(x)\rceil < H(X) + 1$.

- Proof of Lower Bound:
$$\begin{aligned} E[l(X)] &= \sum_x P_X(x) l_x \\ &= -\sum_x P_X(x) \log 2^{-l_x} \\ &\geq -\sum_x P_X(x) \log P_X(x) \\ &= H(X) \end{aligned}$$

Huffman's Code

```
1: Input: $\{(a_1, p_1), ..., (a_n, p_n)\}$.
2: Output: 0/1-labeled rooted tree.
3: procedure HUFFMAN($\{(a_1, p_1), ..., (a_n, p_n)\}$)
4:     $S \leftarrow \{(a_1, p_1, a_1), ..., (a_n, p_n, a_n)\}$
5:     while $|S| \geq 2$
6:         Find $(b_i, p_i, t_i), (b_j, p_j, t_j) \in S$ with minimal $p_i$ and $p_j$.
7:         $S \leftarrow S \setminus \{(b_i, p_i, t_i), (b_j, p_j, t_j)\}$
8:         $S \leftarrow S \cup \{(b_i || b_j, p_i + p_j, \text{NODE}(t_i, t_j))\}$
9:     return $S$
```

- Theorem: Huffman's code is optimal.
- Proof idea: There exists an optimal code where the tow least likely symbols are neighbours.

Entropy

- Let us turn this expression into a definition.
- Definition: Let X be a random variable taking values in $\mathcal{X}$. Then the **entropy** of X is
$$H(X) = -\sum_{x \in \mathcal{X}} P_X(x) \log P_X(x).$$

Conditional Entropy

- Definition: Let (X, Y) be a random variable taking values in $\mathcal{X} \times \mathcal{Y}$. We define **conditional entropy**
$$\begin{aligned} H(X|y) &= -\sum_x P_{X|Y}(x|y) \log P_{X|Y}(x|y) \quad \text{and} \\ H(X|Y) &= \sum_y P_Y(y) H(X|y) \end{aligned}$$
- Note that $H(X|y)$ is simply the ordinary entropy function of a random variable with probability function $P_{X|Y}(\cdot|y)$.

Properties of Entropy

- Let X be a random variable taking values in $\mathcal{X}$.
- Upper Bound: $H(X) = E[-\log P_X(X)] \leq \log |\mathcal{X}|$.
- Chain Rule and Conditioning:

$$\begin{aligned} H(X,Y) &= -\sum_{x,y} P_{X,Y}(x,y) \log P_{X,Y}(x,y) \\ &= -\sum_{x,y} P_{X,Y}(x,y)\big(\log P_Y(y) + \log P_{X|Y}(x|y)\big) \\ &= -\sum_{y} P_Y(y) \log P_Y(y) - \sum_{x,y} P_{X,Y}(x,y) \log P_{X|Y}(x|y) \\ &= H(Y) + H(X|Y) \leq H(Y) + H(X) \end{aligned}$$

Lecture 8 - Elementary Number Theory

Greatest Common Divisors

- Definition: A common divisor of two integers m and n is an integer d such that $d \mid m$ and $d \mid n$.
- Definition: A greatest common divisor (GCD) of two integers m and n is a common divisor d such that every common divisor d' divides d.
- The GCD is the positive GCD.
- We denote the GCD of m and n by $\gcd(m, n)$.
- Properties:
  - $\gcd(m, n) = \gcd(n, m)$
  - $\gcd(m, n) = \gcd(m - n, n)$ if $m \geq n$
  - $\gcd(m, n) = \gcd(m \mod n, n)$
  - $\gcd(m, n) = 2 \gcd(m/2, n/2)$ if m and n are even.
  - $\gcd(m, n) = \gcd(m/2, n)$ if m is even and n is odd.

Euclidean Algorithm

```
1: procedure EUCLIDEAN($m, n$)
2:     while $n \neq 0$
3:         $t \leftarrow n$
4:         $n \leftarrow m \mod n$
5:         $m \leftarrow t$
6:     return $m$
```

Steins Algorithm (Binary GCD Algorithm)

```
1: procedure STEIN($m, n$)
2:     if $m = 0$ or $n = 0$ return 0
3:     $s \leftarrow 0$
4:     while $m$ and $n$ are even
5:         $m \leftarrow m/2$
6:         $n \leftarrow n/2$
7:         $s \leftarrow s + 1$
8:     while $n$ is even
9:         $n \leftarrow n/2$
10:    while $m \neq 0$
11:        while $m$ is even
12:            $m \leftarrow m/2$
13:        if $m < n$
14:            SWAP(m,n)
15:        $m \leftarrow m - n$
16:        $m \leftarrow m/2$
17:    return $2^s n$
```

Bezout's Lemma

- Lemma: There exists integers a and b such that

$$\gcd(m, n) = am + bn.$$

- Proof: Let $d > \gcd(m, n)$ be the smallest positive integer of the form $d = am + bn$. Write $m = cd + r$ with $0 < r < d$. Then

$$\begin{aligned} d > r &= m - cd \\ &= m - c(am + bn) \\ &= (1 - ca)m + (-cb)n, \end{aligned}$$

  a contradiction! Thus, $r = 0$ and $d \mid m$. Similary, $d \mid n$.

Extended Euclidean Algorithm (Recursive Version)

```
procedure ExtendedEuclidean($m, n$)
    if $m \mod n = 0$
        return $(0, 1)$
    else
        $(x, y) \leftarrow$ ExtendedEuclidean($n, m \mod n$)
        return $(y, x - y\lfloor m/n \rfloor)$
```

- If $(x, y) \leftarrow \textsc{ExtendedEuclidean}(m, n)$ then $\gcd(m, n) = xm + yn$.

Coprimality (Relative Primality)

- Definition: Two integers m and n are coprime if their greatest common divisor is 1.
- Fact: If a and n are coprime, then there exists a b such that $ab = 1 \mod n$.

Chinese Remainder Theorem (CRT)

- Theorem: (Sun Tzu 400 AC) Let $n_1, ..., n_k$ be positive pairwise coprime integers and let $a_1, ..., a_k$ be integers. Then the equation system

$$\begin{aligned} x &= a_1 \mod n_1 \\ x &= a_2 \mod n_2 \\ x &= a_3 \mod n_3 \\ &\vdots \\ x &= a_k \mod n_k \end{aligned}$$

has a unique solution in $\{0, ..., \prod_i n_i - 1\}$.

Constructive Proof of CRT

- Set $N = n_1 \cdot n_2 \cdot ... \cdot n_k$.
- Find r_i and s_i such that $r_i n_i + s_i \frac{N}{n_i} = 1$ (Bezout).
- Note that

$$s_i \frac{N}{n_i} = 1 - r_i n_i = \begin{cases} 1 \mod n_i \\ 0 \mod n_j & \text{if } j \neq i \end{cases}$$

- The solution to the equation system becomes:

$$x = \sum_{i=1}^{k} \left(s_i \frac{N}{n_i} \right) \cdot a_i$$

The Multiplicative Group

- The set $\mathbb{Z}_n^* = \{0 \leq a < n : \gcd(a,n) = 1\}$ forms a group, since:
  - Closure: It is closed under multiplication modulo n.
  - Associativity: For $x, y, z \in \mathbb{Z}_n^*$:
    $$(xy)z = x(yz) \mod n.$$
  - Identity: For every $x \in \mathbb{Z}_n^*$:
    $$1 \cdot x = x \cdot 1 = x.$$
  - Inverse: For every $a \in \mathbb{Z}_n^*$ there exists $b \in \mathbb{Z}_n^*$ such that:
    $$ab = 1 \mod n.$$

Lagrange's Theorem

- Theorem: If H is a subgroup of a finite group G, then $|H|$ divides $|G|$.
- Proof: Define $aH = \{ah : h \in H\}$. This gives an equivalence relation $x \approx y \iff x = yh \wedge h \in H$, and a partition of G.
- The map $\phi_{a,b} : aH \to bH$, defined by $\phi_{a,b}(x) = ba^{-1}x$ is a bijection, so $|aH| = |bH|$ for $a, b \in G$.

Euler's Phi-Function (Totient Function)

- Definition: Euler's Phi-function $\phi(n)$ counts the number of integers $0 < a < n$ relatively prime to n.
  - Clearly: $\phi(p) = p - 1$ when p is prime.
  - Similarly: $\phi(p^k) = p^k - p^{k-1}$ when p is prime and $k > 1$.
  - In general $\phi\Big(\prod_i^{k_i}\Big) = \prod_i \Big(p_i^k - p_i^{k-1}\Big)$.
- How does this follow from CRT?
  - $\mathbb{Z}_n \simeq \prod_i \mathbb{Z}_{p_i^{k_i}}$ (CRT is a bijection)
  - If $a \in \mathbb{Z}_n^*$, then $a \mod p_i^{k_i} \in \mathbb{Z}_{p_i^{k_i}}$ (aligns bijection on subsets)

Fermat's and Euler's Theorems

- Theorem: (Fermat) If $b \in \mathbb{Z}_p^*$ and p is prime, then $b^{p-1} = 1 \mod p$.
- Theorem: (Euler) If $b \in \mathbb{Z}_n^*$, then $b^{\phi(n)} = 1 \mod n$.
- Proof: Note that $|\mathbb{Z}_n^*| = \phi(n)$. b generates a subgroup $\langle b \rangle$ of $\mathbb{Z}_n^*$, so $|\langle b \rangle|$ divides $\phi(n)$ by Lagrange's theorem and $b^{|\langle b \rangle|} = 1 \mod n$.

Multiplicative Group of a Prime Order Field

- Definition: A group G is called cyclic if there exists an element g such that each element in G is of the form g^x for some integer x.
- Theorem: If p is prime, then $\mathbb{Z}_p^*$ is cyclic.
- Every group of pime order is cyclic. Why?
  Keep in mind the difference between:
  - $\mathbb{Z}_p$ with *prime order* as an *additive group*,
  - $\mathbb{Z}_p^*$ with *non-prime order* as a *multiplicative group*.
  - Group G_p of *prime order*.

Lecture 9 - Public-Key Cryptography

Public-key cryptography was discovered:

- By Ellis, Cocks, and Williamson at the Government Communications Headquareters (GCHQ) in the UK in the early 1970s (not public until 1997).
- Independently by Merkle in 1974 (Merkle's puzzles).
- Independently in its discrete-logarithm based for by Diffie and Hellman in 1977, and instantiated in 1978 (key-exchagne).
- Independently in its factoring-based form by Rivest, Shamir and Adlemand in 1977.

- Alice encrypts a message m using Bob's public key pk and encryption algorithm E such that $c = E_{\mathrm{pk}}(m)$. Bob decrypts the ciphertext c using his secret key sk and decryption algorithm D such that $m = E_{\mathrm{sk}}(c)$.
- Definition: Mathematically, a public-key cryptosystem can be defined as a tuple $(\mathcal{G}en, E, D)$ where:
  - $\mathcal{G}en$ is a probabilistic key generation algorithm that outputs key pairs (pk, sk),
  - E is a (possibly probabilistic) encryption algorithm that given a public key pk and a message m in the plaintext space $\mathcal{M}_{\mathrm{pk}}$ outputs a ciphertxt c, and
  - D is a decryption algorithm that given a secret key sk and a ciphertext c outputs a plaintext m,

  such that $D_{\mathrm{sk}}(E_{\mathrm{pk}}(m)) = m$ for every (pk, sk) and $m \in \mathcal{M}_{pk}$.

RSA

- Key Generation:
  - choose $n/2$-bit primes p and q randomly and define $N = pq$.
  - Choose e in $\mathbb{Z}^*_{\phi(N)}$ and compute $d = e^{-1} \mod \phi(N)$.
  - Output the key pair $((N, e), (p, q, d))$, where (N, e) is the public key and (p, q, d) is the secret key.
- Encryption: Encrypt a plaintext $m \in \mathbb{Z}^*_N$ by computing
$$c = m^e \mod N.$$
- Decryption: Decrypt a ciphertext c by computing
$$m = c^d \mod N.$$

Why does it work?

$$\begin{aligned}(m^e \mod N)^d \mod N &= m^{ed} \mod N \\ &= m^{1+t\phi(N)} \mod N \\ &= m^1 \cdot \left(m^{\phi(N)}\right)^t \mod N \\ &= m \cdot 1^t \mod N \\ &= m \mod N\end{aligned}$$

Implementing RSA

- Modular arithmetic
- Greatest common divisor
- Primality test

Modular Arithmetic

- Basic operations on $\mathcal{O}(n)$-bit integers using "text book" implementations.

Operation	Running time
Addition	$\mathcal{O}(n)$
Subtraction	$\mathcal{O}(n)$
Multiplication	$\mathcal{O}(n^2)$
Modular reduction	$\mathcal{O}(n^2)$
Greatest common divisor	$\mathcal{O}(n^2)$

- Optimal algorithms for multiplication and modular reduction are much faster.
- What about modular exponentiation?

Square-and-Multiply

```
procedure SquareAndMultiply($x, e, N$)
    $z \leftarrow 1$
    $i$ = index of most signifiant one
    while $i \geq 0$
        $z \leftarrow z \cdot z \mod N$
        if $e_i = 1$
            $z \leftarrow z \cdot x \mod N$
        $i \leftarrow i - 1$
    return $z$
```

- Although basically the same, the most efficient algorithms for exponentiation are faster.
- Computing $g^{x_1}, ..., g^{x_k}$ can be done much faster!
- Computing $\prod_{i\in[k]} g^{x_i}$ can be done much faster!
- Computing $g_1^x, ..., g_k^x$ can be done somewhat faster!
- What about side-channel attacks?

Prime Number Theorem

- The primes are relatively dense.
- Theorem: Let $\pi(m)$ denote the number of primes $0 < p \leq m$. Then

$$\lim_{m\to\infty} \frac{\pi(m)}{\frac{m}{\ln m}} = 1.$$

- To generate a random prime, we repeatedly pick a random integer m and check if it is prime. It should be prime with probability $1/\ln m$ in a sufficiently large interval.

Legendre Symbol

- Definition: Given an odd integer $b \geq 3$, an integer a is called a quadratic residue modulo b if there existst and integer x such that $a = x^2 \mod b$.
- Definition: The Legendre Sybol of an integer a modulo an odd prime p is define by
$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a = 0 \\ 1 & \text{if } a \text{ is a quadratic residue modulo } p \\ -1 & \text{if } a \text{ is a quadratic non-residue modulo } p \end{cases}$$
- Theorem: If p is an odd prime, then
$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \mod p$$
- Proof:
  - If $a = y^2 \mod p$, then $a^{(p-1)/2} = y^{p-1} = 1 \mod p$.
  - If $a^{(p-1)/2} = 1 \mod p$ and b generates $\mathbb{Z}_p^*$, then $a^{(p-1)/2} = b^{x(p-1)/2} = 1 \mod p$ for some x. Since b is a generator, $(p-1) \mid x(p-1)/2$ and x must be even.
  - If a is a non-residue, then $a^{(p-1)/2} \neq 1 \mod p$, but $\left(a^{(p-1)/2}\right)^2 = 1 \mod p$, so $a^{(p-1)/2} = -1 \mod p$.

Jacobi Symbol

- Definition: The Jacobi Symbol of an integer a modulo an odd integer $b = \prod_i p_i^{e_i}$, with p_i prime, is defined by
$$\left(\frac{a}{b}\right) = \prod_i \left(\frac{a}{p_i}\right)^{e_i}.$$
- Note that we can have $\left(\frac{a}{b}\right) = 1$ even when a is a non-residue modulo b.

- Basic Properties:

$$\left(\frac{a}{b}\right) = \left(\frac{a \mod b}{b}\right)$$
$$\left(\frac{ac}{b}\right) = \left(\frac{a}{b}\right)\left(\frac{a}{b}\right).$$

- Law of Quadratic Reciprocity: If a and b are odd integers, then

$$\left(\frac{a}{b}\right) = (-1)^{\frac{(a-1)(b-1)}{4}}\left(\frac{b}{a}\right).$$

- Supplementary Laws: If b is an odd integer, then

$$\left(\frac{-1}{b}\right) = (-1)^{\frac{b-1}{2}} \text{ and } \left(\frac{2}{b}\right) = (-1)^{\frac{b^2-1}{8}}.$$

Computing the Jacobi Symbol

The following assumes that $a \geq 0$ and that $b \geq 3$ is odd.

```
1: procedure JACOBI($a, b$)
2:     if $a < 2$
3:         return $a$
4:     $s \leftarrow 1$
5:     while $a$ is even
6:         $s \leftarrow s \cdot (-1)^{\frac{1}{8}(b^2-1)}$
7:         $a \leftarrow a/2$
8:     if $a < b$
9:         SWAP($a, b$)
10:        $s \leftarrow s \cdot (-1)^{\frac{1}{4}(a-b)(b-1)}$
11:    return $s \cdot$ JACOBI($a \mod b, b$)
```

Solovay-Strassen Primality Test

The following assumes that $n \geq 3$.

```
1: procedure SOLOVAYSTRASSEN($n, r$)
2:     for $i = 1$ to $r$
3:         Choose $0 < a < n$ randomly.
4:         if $\left(\frac{a}{n}\right) = 0$ or $\left(\frac{a}{n}\right) \neq a^{(n-1)/2} \mod n$
```

```
5:            return composite
6:      return probably prime
```

- Analysis: If n is prime, then $0 \neq \left(\frac{a}{n}\right) = a^{(n-1)/2} \mod n$ for all $0 < a < n$, so we never claim that a prime is composite.
- If $\left(\frac{a}{n}\right) = 0$, then $\left(\frac{a}{p}\right) = 0$ for some prime factor p of n. Thus, $p \mid a$ and n is composite, so we never wrongly return from within the loop.
- At most half of all elements a in $\mathbb{Z}_n^*$ have the property that

$$\left(\frac{a}{n}\right) = a^{(n-1)/2} \mod n.$$

More On Primality Tests

- The Miller-Rabin test is faster.
- Testing many primes can be done faster than testing each separately
- Those are *probabilistic* primality tests, but there is a *deterministic* test, so primes are in P.

Security of RSA

Factoring

- The obvious way to break RSA is to factor the public modulus N and recover the prime factors p and q.
  - The number of field sieve factors N in time

$$\mathcal{O}\left(e^{(1.92+o(1))\left((\ln N)^{1/3}+(\ln\ln N)^{2/3}\right)}\right).$$

  - The elliptic curve method factors N in time

$$\mathcal{O}\left(e^{(1+o(1))\sqrt{2\ln p\ln\ln p}}\right).$$

- Note that the latter only depends on the size of p!

Small Encryption Exponents

- Suppose that $e = 3$ is used by all parties as an encryption exponent.
    - Small Message: If m is small, then $m^e < N$. Thus, no reduction takes place, and m can be computed in $\mathbb{Z}$ by taking the e^{th} root.
    - Identical Plaintexts: If a message m is encrypted under moduli N_1, N_2, N_3, and N_4 as c_1, c_2, c_3, and c_4, then CRT implies a $c \in \mathbb{Z}_{N_1 N_2 N_3 N_4}$ such that $c = c_i \mod N_i$ and $c = m^e \mod N_1 N_2 N_3 N_4$ with $m < N_i$.

Additional Caveats

- Identical Moduli: If a message m is encrypted as c_1 and c_2 using distinct encryption exponents e_1 and e_2 with $\gcd(e_1, e_2) = 1$, and a modulus N, then we can find a, b such that $ae_1 + be_2 = 1$ and $m = c_1^a c_2^b \mod N$.
- Reiter-Franklin Attack: If e is small enough then encryptions of m and $f(m)$ for a polynomial $f \in \mathbb{Z}_N[x]$ allows efficient computation of m.
- Wiener's Attack: If $3d < N^{1/4}$ and $q < p < 2q$, then N can be factored in polynomial time with good probability.

Factoring From Order of Multiplicative Group

- Given N and $\phi(N)$, we can find p and q by solving

$$\begin{aligned} N &= pq \\ \phi(N) &= (p-1)(q-1) \end{aligned}$$

Lecture 10 - CPA Security, ROM-RSA, Rabin and Diffie-Hellman

Factoring from Encryption & Decryption Exponents

- If $N = pq$ with p and q prime, then the CRT implies that
$$x^2 = 1 \mod N$$
has four distinct solutions in $\mathbb{Z}_N^*$, and two of these are non-trivial, i.e., distinct from ± 1.

- If x is a non-trivial root, then
$$(x-1)(x+1) = tN$$
but $N \nmid (x-1), (x+1)$, so
$$\gcd(x-1, N) > 1 \text{ and } \gcd(x+1, N) > 1.$$

- The encryption & decryption exponents satisfy
$$ed = 1 \mod \phi(N),$$
so if we have $ed - 1 = 2^s r$ with r odd, then
$$\begin{aligned}(p-1) &= 2^{s_p} r_p \text{ which divides } 2^s r \text{ and}\\ (q-1) &= 2^{s_q} r_q \text{ which divides } 2^s r.\end{aligned}$$

- If $v \in \mathbb{Z}_N^*$ is random, then $w = v^r$ is random in the subgroup of elements with order 2^i for some $0 \leq i \leq \max\{s_p, s_q\}$.

- Suppose $s_p \geq s_q$. Then for some $0 < i < s_p$,
$$w^{2^i} = \pm 1 \mod q$$
and
$$w^{2^i} \mod p$$
is uniformly distributed in $\{1, -1\}$.

- Conclusion: $w^{2^i}(\mod N)$ is a non-trivial root of 1 with probability 1/2, which allows us to factor N.

CPA Security

- RSA clearly provides some kind of "security", but it is clear that we need to be more careful with what we ask for.
- Intuitively, we want to leak no **information** of the encrypted plaintext.
- Intuitively, we want to leak no **knowledge** of the encrypted plaintext.
- In other words, no function of the plaintext can efficiently be guessed notably better from its ciphertext than without it.
- $\textsc{Exp}^{b}_{\mathcal{CS},A}$(CPA Security Experiment)
  - Generate Public Key: (pk, sk) $\leftarrow$ Gen(1^n).
  - Adversarial Choice of Messages: $(m_0, m_1, s) \leftarrow A(\mathrm{pk})$.
  - Guess Message: Return the first output of $A(E_{\mathrm{pk}}(m_b), s)$.
- Definition: A cryptosystem $\mathcal{CS} = (\mathcal{G}en, E, D)$ is said to be CPA secure if for every polynomial time algorithm A

  $$|Pr[\mathrm{Exp}^{0}_{\mathcal{CS},A} = 1] - Pr[\mathrm{Exp}^{1}_{\mathcal{CS},A} = 1]|$$

  is negligible.
- Every CPA secure cryptosystem must be probabilistic!
- Theorem: Suppose $\mathcal{CS} = (\mathcal{G}en, E, D)$ is a CPA secure cryptosystem.
  Then the related cryptosystem where a $t(n)-$list of messages, with $t(n)$ polynomial, is encrypted by repeated independent encryption of each component using the same public key is also CPA secure.
- CPA security is useful!

ROM-RSA

- Definition: The RSA assumption states that if:
  - $N = pq$ factors into two randomly chosen primes p and q of the same bit-size,
  - e is in $\mathbb{Z}^*_{\phi(N)}$,
  - m is randomly chosen in $\mathbb{Z}^*_N$,

  then for every polynomial time algorithm A

  $$Pr[A(N, e, m^e \mod N) = m]$$

  is negligible.

CPA Secure ROM-RSA

- Suppose that $f : \{0,1\}^n \to \{0,1\}^n$ is a randomly chosen function (a random oracle).
  - Key Generation: Choose a random RSA key pair $((N, e), (p, q, d))$, with $\log_2 N = n$.
  - Encryption: Encrypt a plaintext $m \in \{0,1\}^n$ by choosing $r \in \mathbb{Z}_N^*$ randomly and computing
    $$(u, v) = (r^e \mod N, f(r) \oplus m).$$
  - Decryption: Decrypt a ciphertext (u, v) by
    $$m = v \oplus f(u^d).$$
- We increase the ciphertext size by a factor of two.
- Our analysis is in the random oracle model, which is unsound!
- Solutions:
  - Using a "optimal" padding the first problem can be reduced. See standard OAEP+.
  - Using a scheme with much lower rate, the second problem can be removed.

Rabin

- Key Generation:
  - Choose $n-$bit primes p and q such that $p, q = 3 \mod 4$ randomly and define $N = pq$.
  - Output the key pair $(N, (p, q))$, where N is the public key and (p, q) is the secret key.
- Encryption: Encrypt a plaintext m by computing
  $$c = m^2 \mod N.$$

- Decryption: Decrypt a ciphertext c by computing

$$m = \sqrt{c} \mod N.$$

- There are four roots, so which one should be used?

- Suppose y is a quadratic residue modulo p.

$$\begin{aligned}\left(\pm y^{(p+1)/4}\right)^2 &= y^{(p+1)/2} \mod p\\ &= y^{(p-1)/2}y \mod p\\ &= \left(\frac{y}{p}\right)y\\ &= y \mod p\end{aligned}$$

- In Rabin's cryptosystem:
  - Find roots for $y_p = y \mod p$ and $y_q = y \mod q$.
  - Combine roots to get the four roots modulo N. Choose the "right" root and output the plaintext.

Security of Rabin's Cryptosystem

- Theorem: Breaking Rabin's cryptosystem is equivalent to factoring.

- Idea:
  - Choose random element r.
  - Hand $r^2 \mod N$ to adversary.
  - Consider outputs r' from the adversary such that $(r')^2 = r^2 \mod N$. then $r' \neq \pm r \mod N$, with probability $1/2$, in which cased $\gcd(r' - r, N)$ gives a factor of N.

A Goldwasser-Micali Variant of Rabin

- Theorem [CG98]: If factoring is hard and r is a random quadratic residue modulo N, then for every polynomial time algorithm A

  $$Pr[A(N, r^2 \mod N) = \text{lsb}(r)]$$

  is negligible.

  - Encryption: Encrypt a plaintext $m \in \{0,1\}$ by choosing a random quadratic residue r modulo N and computing

    $$(u,v) = r^2 \mod N, \text{lsb}(r) \oplus m).$$

  - Decryption: Decrypt a ciphertext (u,v) by

    $$m = v \oplus \text{lsb}(\sqrt{u}) \text{ where } \sqrt{u} \text{ is a qudratic residue.}$$

Diffie-Hellman

- Diffie and Hellman asked themselves: How can two parties efficiently agree on a secret key using only public communication?
- Construction: Let G be a cyclic group of order q with generator g.
  - Alice picks $a \in \mathbb{Z}_q$ randomly, computes $y_a = g^a$ and hands y_a to Bob.
  - Bob picks $b \in \mathbb{Z}_q$ randomly, computes $y_b = g^b$ and hands y_b to Alice.
  - Alice computes $k = y_b^a$.
  - Bob computes $k = y_a^b$.
  - The joint secret key is k.
- Problems:
  - Susceptible to man-in-the-middle attack without authentication.
  - How do we map a random element $k \in G$ to a random symmetric key in $\{0,1\}^n$?

The El Gamal Cryptosystem

- Definition: Let G be a cyclic group of order q with generator g.
  - The key generation algorithm chooses a random element $x \in \mathbb{Z}_q$ as the private key and defines the public key as
  $$y = g^x.$$
  - The encryption algorithm takes a message $m \in G$ and the public key y, chooses $r \in \mathbb{Z}_q$, and outputs the pair
  $$(u, v) = E_y(m, r) = (g^r, y^r m).$$
  - The decryption algorithm takes a ciphertext (u, v) and the secret key and outputs
  $$m = D_x(u, v) = vu^{-x}.$$
- El Gamal is essentially Diffie-Hellman + OTP.
- Homomorhpic property (with public key y)
  $$E_y(m_0, r_0)E_y(m_1, r_1) = E_y(m_0 m_1, r_0 + r_1).$$
  This property is very important in the construction of cryptographic protocols!

Lecture 11 - Number Theory continued

Discrete Logarithm

- Definition: Let G be a cyclic group of order q and let g be a generator G. The discrete logarithm of $y \in G$ in the basis g (written $\log_g y$) is defined as the unique $x \in \{0, 1, ..., q-1\}$ such that
  $$y = g^x.$$
  Compare with a "normal" logarithm! ($\ln y = x$ iff $y = e^x$).
- Example: 7 is a generator of $\mathbb{Z}_{12}$ additively, since gcd(7,12) = 1. What is $\log_7 3$? ($9 \cdot 7 = 63 = 3 \mod 12$, so $\log_7 3 = 9$)
- Example: 7 is a generator of $\mathbb{Z}_{13}^*$. What is $\log_7 9$? ($7^4 = 9 \mod 13$, so $\log_7 9 = 4$)

Discrete Logarithm Assumption

- Let G_{q_n} be a cyclic group of prime order q_n such that $\lfloor \log_2 q_n \rfloor = n$ for $n = 2, 3, 4, ...$, and denote the family $\{G_{q_n}\}_{n \in \mathbb{N}}$ by G.
- Definition: The Discrete Logarithm (DL) Assumption in G states that if generators g_n and y_n of G_{q_n} are randomly chosen, then for every polynomial time algorithm A
$$Pr[A(g_n, y_n) = \log_{g_n} y_n]$$
is negligible.
- We usually remove the indices from our notation!
$$Pr[A(g, y) = \log_g y]$$

Diffie-Hellman Assumption

- Definition: Let g be a generator of G. The Diffie-Hellman (DH) Assumption in G states that if $a, b \in \mathbb{Z}_q$ are randomly chosen, then for every polynomial time algorithm A
$$Pr[A(g^a, g^b) = g^{ab}]$$
is negligible.

Decision Diffie-Hellman Assumption

- Definition: Let g be a generator of G. The Decision Diffie-Hellman (DDH) Assumption in G states that if $a, b, c \in \mathbb{Z}_q$ are randomly chosen, then for every polynomial time algorithm A
$$|Pr[A(g^a, g^b, g^{ab}) = 1] - Pr[A(g^a, g^b, g^c) = 1]|$$
is negligible.
- Relating DL Assumptions:
    - Computing discrete logarithms is at least as hard as computing a Diffie-Hellman element g^{ab} from g^a and g^b.
    - Computing a Diffie-Hellman element $g^a b$ from g^a and g^b is at least as hard as distinguishing a Diffie-Hellman triple (g^a, g^b, g^{ab}) from a random triple (g^a, g^b, g^c).
    - In most groups where the DL assumption is conjectured, DH and DDH assumptions are conjectured as well.
    - There exists special elliptic curves where DDH problem is easy, but DH assumption is conjectured.

Security of El Gamal

- Finding the secret key is equivalent to DL problem.
- Finding the plaintext from the ciphertext and the public key is equivalent to DH problem.
- The CPA security of El Gamal is equivalent to DDH problem.

Brute Force and Shank's

- Let G be a cyclic group of order q and g a generator. We wish to compute $\log_g y$.
  - Brute Force: $\mathcal{O}(q)$
  - Shanks: Time and Space $\mathcal{O}(\sqrt{q})$.
    - Set $z = g^m$ (think of m as $m = \sqrt{q}$).
    - Compute z^i for $0 \leq i \leq q/m$.
    - Find $0 \leq j \leq m$ and $0 \leq i \leq q/m$ such that $yg^j = z^i$ and output $x = mi - j$.

Birthday Paradox

- Lemma: Let $q_0, ..., q_k$ be randomly chosen in a set S. Then
  - the probability that $q_i = q_j$ for some $i \neq j$ is approximately $1 - e^{-\frac{k^2}{2s}}$, where $s = |S|$, and
  - with $k \approx \sqrt{-2s \ln(1-\delta)}$ we have a collision-probability of δ.
- Proof:

$$\left(\frac{s-1}{s}\right) \cdot \left(\frac{s-2}{s}\right) \cdot ... \cdot \left(\frac{s-k}{s}\right) \approx \prod_{i=1}^{k} e^{-\frac{i}{s}} \approx e^{-\frac{k^2}{2s}}$$

Pollard-ρ

- Partition G into S_1, S_2, and S_3 "randomly".
  - Generate "random" sequence $\alpha_0, \alpha_1, \alpha_2 ...$
$$\alpha_0 = g$$
$$\alpha_i = \begin{cases} \alpha_{i-1} g & \text{if } \alpha_{i-1} \in S_1 \\ \alpha_{i-1}^2 & \text{if } \alpha_{i-1} \in S_2 \\ \alpha_{i-1} y & \text{if } \alpha_{i-1} \in S_3 \end{cases}$$
  - Each $\alpha_i = g^{a_i} y^{b_i}$, where $a_i, b_i \in \mathbb{Z}_q$ are known!
  - If $\alpha_i = \alpha_j$ and $(a_i, b_i) \neq (a_j, b_j)$ then $y = g^{(a_i - a_j)(b_j - b_i)^{-1}}$.
  - If $\alpha_i = \alpha_j$, then $\alpha_{i+1} = \alpha_{j+1}$.
  - The sequence $(a_0, b_0), (a_1, b_1), ...$ is "essentially random".
  - The Birthday bound implies that the (heuristic) expected running time is $\mathcal{O}(\sqrt{q})$.
  - We use "double runners" to reduce memory.

Index Calculus

- Let $\mathcal{B} = \{p_1, ..., p_B\}$ be a set of small prime integers.
- Compute $a_i = \log_g p_i$ for all $p_i \in \mathcal{B}$.
  - Choose $s_j \in \mathbb{Z}_q$ randomly and attempt to factor $g^{s_j} = \prod_i p_i^{e_{j,i}}$ as an integer.
  - If g^{s_j} factored in $\mathcal{B}$ and $e_j = (e_{j,1}, ..., e_{j,B})$ is linearly independent of $e_i, ..., e_{j-1}$, then $j \leftarrow j + 1$.
  - If $j < B$, then go to (1).
- Let $\mathcal{B} = \{p_1, ..., p_B\}$ be a set of small prime integers.
- Compute $a_i = \log_g p_i$ for all $p_i \in \mathcal{B}$.
  - Choose $s \in \mathbb{Z}_q$ randomly.
  - Attempt to factor $y g^s = \prod_i p_i^{e_i}$ as an integer.
  - If a factorisation is found, then output $(\sum_i a_i e_i - s) \mod q$.
- Why doesn't this work for any cyclic group?

Example Groups

- $\mathbb{Z}_n$ additively? Bad for crypto!
- Large prime order subgroup of $\mathbb{Z}_p^*$ with p prime. In particulate $p = 2q + 1$ with q prime.
- Large prime order subgroup of $GF_{p^k}^*$.
- "Carefully chosen" elliptic curve group.

Lecture 12 - Elliptic Curves & Signature Schemes

- We have argued that discrete logarithm problems are hard in large subgroups of $\mathbb{Z}_p^*$ and $\mathbb{F}_q^*$.
- Based on discrete logarithm problems (DL, DH, DDH) we can construct public key cryptosystems, key exchange protocols, and signature schemes.
- An elliptic curve is another candidate of a group where discrete logarithm problems are hard.
- Motivation for studying elliptic curves:
    - What if it turns out that solving discrete logarithms in $\mathbb{Z}_p^*$ is easy? Elliptic curves give an alternative.
    - The best known DL-algorithms in an elliptic curve group with prime order q are generic algorithms, i.e. the have running time $\mathcal{O}(\sqrt{q})$.
    - Arguably we can use shorter keys. This is very important in some practical applications.
- Definition: A plane cubic curve E (in Weierstrass form) over a field $\mathbb{F}$ is given by a polynomial
$$y^2 = x^3 + ax + b$$
with $a, b \in \mathbb{F}$. The set of points (x, y) that satisfy this equation $\mathbb{F}$ is written $E(\mathbb{F})$.
- Every plane cubic curve over a field of characteristic $\neq 2, 3$ can be written in the above form without changing any properties we care about.
- We also write
$$g(x, y) = x^3 + ax + b - y^2 \text{ or}$$
$$y^2 = f(x)$$
where $f(x) = x^3 + ax + b$.

Singular Points

- Definition: A point $(u, v) \in E(\mathbb{E})$, with $\mathbb{E}$ an extension field of $\mathbb{F}$, is singular if

$$\frac{\partial g(x,y)}{\partial x}(u,v) = \frac{\partial g(x,y)}{\partial y}(u,v) = 0.$$

- Definition: A plane cubic curve is smooth if $E(\overline{\mathbb{F}})$ contains no singular points. ($\overline{\mathbb{F}}$ is the algebraic closure of $\mathbb{F}$.)

- Note that

$$\begin{aligned} &\frac{\partial g(x,y)}{\partial x}(x,y) = f'(x) = 3x^2 + a \text{ and} \\ &\frac{\partial g(x,y)}{\partial y}(x,y) = -2y \end{aligned}$$

- Thus, any singular point $(u, v) \in E(\mathbb{F})$ must have:
  - $v = 0$,
  - $f(u) = 0$, and $f'(u) = 0$.

- Then $f(x) = (x - u)h(x)$ and $f'(x) = h(x) + (x - u)h'(x)$, so (u, v) is singular if $v = 0$ and u is a double-root of f.

Discriminant

- In general a "discriminant" can be used to check if a polynomial has a double root.

- Definition: The discriminant $\Delta(E)$ of a plane curve $y^2 = x^3 + ax + b$ is given by $-4a^3 - 27b^2$.

- Lemma: The polynomial $f(x)$ does not have a double root iff $\Delta(E) \neq 0$, in which case the curve is called smooth.

Line Defined By Two Points On Curve

- Let $l(x)$ be a line that intersects the curve in (u_1, v_1) and (u_2, v_2). Then
$$l(x) = k(x - u_1) + v_1$$
where
$$k = \begin{cases} \frac{v_2 - v_1}{u_2 - u_1} & \text{if } (u_1, v_1) \neq (u_2, v_2) \\ \frac{3u_1^2 + a}{2v_1} & \text{otherwise} \end{cases}$$

- We are cheating a little here in that we assume that we don't have $u_1 = u_2$ and $v_1 \neq v_2$ or $v_1 = v_2 = 0$. In both such cases we get a line parallel with $x = 0$ that we deal with in a special way.

Finding the Third Point

- The intersection points between $l(x)$ and the curve are given by the zeros of
$$t(x) = g(l(x), x) = f(x) - l(x)^2$$
which is a cubic polynomial with known roots u_1 and u_2.

- To find the third intersection point (u_3, v_3) we note that
$$t(x) = (x - u_1)(x - u_2)(x - u_3) = x^3 - (u_1 + u_2 + u_3)x^2 + r(x)$$
where $r(x)$ is linear. Thus, we can find u_3 from t's coefficients!

- Given any two points A and B on the curve that defines a line, we can find a third intersection point C with the curve (even if $A = B$).

- The only exception is if our line $l(x)$ is parallel with the y-axis.

- To "fix" this exception we add a point at infinity O, roughly at $(0, \infty)$ (the projective plane). Intuition: the side of a long straight road seem to intersect infinitely far away.

- We define the sum of A and B by $(x, -y)$, where (x, y) is the third intersection point of the line defined by A and B with the curve.

- We define the inverse of (x, y) by $(x, -y)$.

- The main technical difficulty in proving that this gives a group is to prove the associative law. This can be done with Bezout's theorem (not the one covered in class), or by (tedious) elementary algebraic manipulation.

Elliptic Curves

- There are many elliptic curves with special properties.
- There are many ways to represent the same curve and to implement curves as well as representing and implementing the underlying field.
- More requirements than smoothness must be satisfied for a curve to be suitable for cryptographic use.
- Fortunately, there are standardised curves.
  (I would need a very strong reason not to use these curves and I would be extremely careful, consulting researchers specialising in elliptic curve cryptography.)

Signature Schemes

Digital Signature

- A digital signature is the public-key equivalent of a MAC; the receiver verifies the integrity and authenticity of a message.
- Does a digital signature replace a real handwritten one?

Textbook RSA Signature

- Generate RSA keys $((N, e), (p, q, d))$.
- To sign a message $m \in \mathbb{Z}_N$, compute $\sigma = m^d \mod N$.
- To verify a signature σ of a message m verify that $\sigma^e = m \mod N$.
- Are Textbook RSA Signatures any good?
- If σ is a signature of m, then $\sigma^2 \mod N$ is a signature of $m^2 \mod N$.
- If σ_1 and σ_2 are signatures of m_1 and m_2, then $\sigma_1\sigma_2 \mod N$ is a signature of $m_1m_2 \mod N$.
- We can also pick a signature σ and compute the message it is a signature of by $m = \sigma^e \mod N$.
- We must be more careful!

Signature Scheme

- Gen generates a key pair (pk, sk).
- Sig takes a secret key sk and a message m and computes signature σ.
- Vf takes a public key pk, a message m, and a candidate signature σ, verifies the candidate signature, and outputs a single-bit verdict.

Existential Unforgeability

- Definition: A signature scheme (Gen, Sig, Vf) is secure against existential forgeries if for every polynomial time algorithm A and a random key pair (pk, sk) $\leftarrow \text{Gen}(1^n)$,
$$Pr[A^{\text{Sig}_{\text{sk}}(\cdot)}(\text{pk}) = (m, \sigma) \wedge \text{Vf}_{\text{pk}}(m, \sigma) = 1 \wedge \ \forall i : m \neq m_i]$$
is negligible where m_i is the i^{th} query to $\text{Sig}_{\text{sk}}(\cdot)$.

Provably Secure Signature Schemes

- Provably secure signature schemes exist if one-way functions exist (in plain model without ROM), but the construction is more involved and typically less efficient.
- Provably secure signature schemes are rarely used in practice!
- Standards used in practices: RSA Full Domain Hash, DSA, EC-DSA. The latter two may be viewed as variants of Schnorr signatures.

Lecture 13 - Trapdoor One-Way Permutations & PKIs

Based on Trapdoor One-Way Permutations

- Let $f = \{f_\alpha\}$ be an ensemble of permutations (bijections).
  - Gen generates a random key pair α = (pk, sk). Less formaly Gen generates a pair $(f_\alpha, f_\alpha^{-1})$.
  - Eval takes pk and x as input and efficiently evaluates $f_\alpha(x)$.
  - Invert takes sk and y as input and efficiently evaluates the inverse $f_\alpha^{-1}(y)$.
- One-way if $\text{Eval}_{\text{pk}}(\cdot)$ is one-way for a random pk.
- RSA is a trap-door permutation over $\mathbb{Z}_N^*$.

Full Domain Hash Signature in ROM

- Let $f = \{f_\alpha\}$ be a trapdoor permutation (family) and let $H : \{0,1\}^* \to \{0,1\}^n$ be a random oracle.
    - Gen samples a pair $(f_\alpha, f_\alpha^{-1})$.
    - Sig takes f_α^{-1} and a message m as input and outputs $f_\alpha^{-1}\big(H(m)\big)$.
    - Vf takes f_α, a message m, and a candidate signature σ as input, and outputs 1 if $f_\alpha(\sigma) = H(m)$ and 0 otherwise.

Based on Proofs of Knowledge

Proof of Knowledge of Exponent

- In an identification scheme one party convinces another that it holds some special token.
    - Let G_q be a group of prime order q with generator g.
    - Let $x \in \mathbb{Z}_q$ and define $y = g^x$.
    - Can we prove knowledge of x without disclosing anything about x?

Schnorr's Protocol

- The prover chooses $r \in \mathbb{Z}_q$ randomly and hands $\alpha = g^r$ to the verifier.
- The verifier chooses $c \in \mathbb{Z}_q$ randomly and hands it to the prover.
- The prover computes $d = cx + r \mod q$ and hands d to the verifier.
- The verifier accepts if $y^c\alpha = g^d$.
- Suppose that a machine convinces us in the protocol with probability δ. Does it mean that is knows x such that $y = g^x$?
- Run the machine to get α.

- Complete the interaction twice using the same α, once for a challenge c and once for a challenge c', where $c, c' \in \mathbb{Z}_q$ are chosen randomly.
- Repeat the previous two steps until the resulting interactions (α, c, d) and (α, c', d') are accepting and $c \neq c'$.
- Note that:
$$y^{c-c'} = \frac{y^c}{y^{c'}} = \frac{y^c\alpha}{y^{c'}\alpha} = \frac{g^d}{g^{d'}} = g^{d-d'}$$
which gives the logarithm $x = (d - d')(c - c')^{-1} \mod q$ such that $y = g^x$.
- Anybody can sample $c, d \in \mathbb{Z}_q$ randomly and compute $\alpha = g^d/y^c$.
- The resulting tuple (α, c, d) has exactly the same distribution as the transcript of an interaction!
- Such protocols are called (honest verifier) zero-knowledge proofs of knowledge.

Schnorr's Signature Scheme in ROM

- Let $H : \{0,1\}^* \to \mathbb{Z}_q$ be a random oracle.
  - Gen chooses $x \in \mathbb{Z}_q$ randomly, computes $y = g^x$ and outputs $(\mathrm{pk}, \mathrm{sk}) = (y, x)$.
  - Sig does the following on input x and m:
    1. it chooses $r \in \mathbb{Z}_q$ randomly and computes $\alpha = g^r$,
    2. it computes $c = H(y, \alpha, m)$,
    3. it computes $d = cx + r \mod q$ and outputs (α, d).
  - Vf takes the public key y, a message m and a candidate signature (α, d), and accepts iff $y^{H(y,\alpha,m)}\alpha = g^d$.

PKIs

- We have constructed public-key crypotsystems and signature schemes.
- Only the holder of the secret key can decrypt ciphertexts and sign messages.
- How do we know who holds the secret key corresponding to a public key?

Signing Public Keys of Others

- Suppose that Alice computes a signature $\sigma_{A,B} = \mathrm{Sig}_{\mathrm{sk}_A}(\mathrm{pk}_B, Bob)$ of bob's public key pk_B and his identity and hands it to Bob.
- Suppose that Eve holds Alice's public key pk_A.
- Then anybody can hand $(\mathrm{pk}_B,\ \sigma_{A,B})$ directly to Eve, and Eve will be convinced that pk_B is Bob's key (assuming she trusts Alice).

Certificate

- A certificate is a signature of a public key along with some information on how the key may be used, e.g., it may allow the holder to issue certificates.
- A certificate is valid for a given setting if the signature is valid and the usage information in the certificate matches that of the setting.
- Some parties mus be trusted to issue certificates. These parties are called Certificate Authorities (CA).

Certificate Chains

- A CA may be "distribute" using in certificate chains.
  - Suppose that Bob holds valid certificates

    $$\sigma_{0,1}, \sigma_{1,2}, \ldots, \sigma_{n-1,n}$$

    where $\sigma_{i-1,i}$ is a certificate of pk_{P_i} by P_{i-1}.
  - Who does Bob trust?

Pseudo-random Generators

- Everything we have done so far requires randomness!
- Can we ”generate” random strings?
- We could flip actual coins. This would be extremely impractical and slow (and boring unless you are Rain man).
- We could generate ”physical” randomness using hardware, e.g., measuring radioactive decay
  - Slow or expensive.
  - Hard to verify and trust.
  - Biased output.
- We could use a deterministic algorithm that outputs a ”random looking string”, but would that be secure?

Pseudo-Random Generator

- A pseudo-random generator requires a short random string and deterministically expands this to a longer ”random looking” string.
- This looks promising:
  - Fast and cheap?
  - Practical since it can be implemented in software or hardware?
  - What is ”random looking”?

Lecture 14 - Pseudo-Random Generator

- Definition: An efficient algorithm PRG is a pseudo-random generator (PRG) if there exists a polynomial $p(n) > n$ such that for every polynomial time adversary A, if a seed $S \in \{0,1\}^n$ and a random string $u \in \{0,1\}^{p(n)}$ are chosen randomly, then
$$|Pr[A(\text{PRG}(s)) = 1] - Pr[A(u) = 1]|$$
is negligible.
- informally, A can not distinguish PRG(s) from a truly random string in $\{0,1\}^{p(n)}$.
- Before we consider how to construct a PRG we consider what the definition gives us:
  - Suppose that there exists a PRG that extends its output by a single bit.
  - This would not be very useful to us, e.g., to generate a random prime we need many random bits.
  - Can we use the given PRG to construct another PRG which extends its output more?
- Construction: Let PRG be a pseudo-random generator. We let PRG_t be the algorithm that takes $s_{-1} \in \{0,1\}^n$ as input, computes $s_0, ..., s_{t-1}$ and $b_0, ..., bt-1$ as
$$(s_i, b_i) = \text{PRG}(s_{i-1})$$
and outputs $(b_0, ..., bt - 1$.
- Theorem: Let $p(n)$ be a polynomial and PRG a pseudo-random generator. Then $\text{PRG}_{p(n)}$ is a pseudo-random generator that on input $s \in \{0,1\}^n$ outputs a string in $\{0,1\}^{p(n)}$.
- We can go on "forever"!

Random String From Random Oracle

- Theorem: If $F : \{0,1\}^n \rightarrow \{0,1\}^m$ is a random function, then $(F(0), F(1), F(2), ..., F(t-1))$ is a tm-bit string.
- Can we do this using a pseudo-random function?
- Can we replace the random function by SHA-2?

Pseudo-Random Function

- Recall the definition of a pseudo-random function.
- Definition: A family of functions $F : \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ is pseudo-random if for all polynomial time oracle adversaries A

$$\left| \Pr_{K}\left[A^{F_K(\cdot)} = 1\right] - \Pr_{R:\{0,1\}^n \rightarrow \{0,1\}^n}\left[A^{R(\cdot)} = 1\right] \right|$$

  is negligible.

Pseudo-Random Generator From Pseudo-Random Function

- Theorem: Let $\{F_K\}_{K\in\{0,1\}^k}$ be a pseudo-random function for a random choice of K. then the PRG defined by:

$$\text{PRG}(s) = (F_s(0), F_s(1), F_s(2), ..., F_s(t))$$

  is a pseudo-random generator.
- Construction: Let PRG: $\{0,1\}^k \rightarrow \{0,1\}^{2k}$ be a pseudo-random generator, and define a family of functions $F = \{F_K\}_{K\in\{0,1\}^k}$ as follows.
  - Let $x_{[i]} = (x_0, ..., x_i)$
  - On key K and input x, F_K computes its output as follows:
    1. computes $(r_0^0, r_1^0) = \text{PRG}(K)$.
    2. Computes
    $$(r^i_{x_{[i-1]}||0}, r^i_{x_{[i-1]}||1}) = \text{PRG}(r^{i-1}_{x_{[i-1]}})$$
    for $i = 1, ..., n-1$.
    3. Outputs $r_{x_{[n-1]}}$.

One-Way Permutation

- Definition: A family $F = \{f_n\}$ of permutations $f_n\{0,1\}^n \rightarrow \{0,1\}^n$ is said to be one-way if for a random $x \in \{0,1\}^n$ and every polynomial time algorithm A

$$Pr[A(f_n(x)) = x] < \epsilon(n)$$

  for a negligible function $\epsilon(n)$.
- (Note that for permutations we may request the unique preimage.)

Hardcore Bit

- Definition: Let $F = \{f_n\}$ be a family of permutations $f_n : \{0,1\}^n \to \{0,1\}^n$ and $B = \{b_n\}$ a family of "bits" $b_n : \{0,1,\}^n \to \{0,1\}$. Then B is a hardcore bit of F if for random $x \in \{0,1\}^n$ and every polynomial time algorithm A
$$|Pr[A(f_n(x)) = b_n(x)] - 1/2| < \epsilon(n)$$
for a negligible function $\epsilon(n)$.
- Theorem: For every one-way permutation, there is a (possibly different) one-way permutation with a hardcore bit.

PRG from One-Way Permutation

- Construction: Let F be a one-way permutation and define PRG by $\mathrm{PRG}_n(s) = f_n(s)||b_n(s)$ for $x \in \{0,1\}^n$.
- Theorem: $\mathrm{PRG} : \{0,1\}^n \to \{0,1\}^{n+1}$ is a pseudo-random generator.

PRG From Any One-Way Function

- Theorem: There is a construction PRG_f that is a a PRG if f is a one-way function (possibly non-permutation).
- The construction is very involved and is completely impractical. (Johan Håstad)

What is Used in Practice?

- Various standards contain some of the following elements.
  - Fast hardware generator + "algorithmic strengthening".
  - `/dev/random`
    - Entropy gathering deamon with estimate of amount of entropy.
    - FreeBSD: Executes the PRG *Yarrow* (or *Futura*) pseudo-random algorithm.
    - SunOs and Un*xes use similar approaches.
    - Windows has similar devices.
  - Stream cipher, e.g. block-cipher in CFB or CTR mode.
  - Hashfunction with secret prefix and counter (essentially our PRF $\to$ PRG construction).

Infamous Mistakes

- (1995) The original Netscape SSL code used time of the day and process IDs to seed its pseudo random giving way too little entropy in the seed.
- (2008) Debian's OpenSSL commented out a critical part of the code that reduced the entropy of keys drastically!
- (2012) RSA public keys with common factors.

Important Conclusions

- Security bugs are not found by testing!
- With an insecure pseudo-random generator anything on top of it will be insecure.
- Any critical code must be reviewed after every modification, e.g., keep hashes of critical code.