TTM 4135 notes

Joakim Lier

Number theory

This part is mostly just stolen from the slides. Not much of interest here except for modular inverses, groups and fields.

Basic number theory

$\mathbb{Z}$ denotes the set of integers
a divides b if there exists a k in $\mathbb{Z}$ such that $a * k = b$
$a * k = 3 * 2 = 6 = b$
An integer is prime if the only positive divisors are 1 and p checking primality for a number n can be done by trial division up to $sqrt(n)$.

Basic properties of factors

1. if a divides b and a divides c then a divides b+c
2. if p is a prime and p divides ab, then p divides a or b

example:
6|18 and 6|24 $\rightarrow$ 6|42 |

Division algorithm

> for a and b in Z, a $>$ b, there exists q and r in Z such that $a = bq + r$ where $0 \leq r < b$.

GCD

The value d is the GCD of a and b if all hold: 1. D divides a and b 2. if c divides a and b then c divides d (the greatest) 3. d $>$ 0, by definition of integers

> a and b are relatively prime if $gcd(a, b) = 1$

Euclidean algorithm & extended euclidean algorithm(EEA)

Euclidean algorithm is for finding gcd. See slides 2 for pseudo-code if you need it EEA finds integers x and y in $a * x + b * y = d$, we're interested in the case where a and b are co-prime (x and y = 1)

Modular arithmetic

> b is the residue of a modulo n if a - b = kn for some integer k.

$a \equiv b(\mod n) \leftrightarrow a - b = kn$

Given $a \equiv b(\mod n)$ and $c \equiv d(\mod n)$ then

1. $a + c \equiv b + d(\mod n)$
2. $ac \equiv bd(\mod n)$
3. $ka \equiv kb(\mod n)$

note:
This means we can always reduce the inputs modulo n before performing additions or multiplications

Residue class

> Definition: The set $r_0, r_1, ...r_{n-1}$ is called a complete set of residues modulo n if, for every integer a, $a \equiv r_i(\mod n)$ for exactly one r_i

We usually denote this set as the complete set of residues and denote it $\mathbb{Z}$

Notation: a mod n

we write a mod n to denote the value a' in the complete set of residues with $a\prime \equiv a(\mod n)$
$a = k * n + a\prime \; 0 \leq a\prime < a$

Groups

> a group is a set, G, with a binary operation $\cdot$ satisfying the following properties:

1. Closure: $a \cdot b \in G, \forall a, b \in G$
2. identity: there exists an element 1, so that $a \cdot 1 = 1 \cdot a = a, \; \forall a \in G$
3. inverse: for all a, there exists an element b so that $a \cdot b = 1, \; \forall a \in G$
4. associative: $(a \cdot b) \cdot c = a \cdot (b \cdot c), \forall a, b, c \in G$ (Doesn't matter where you put the parentheses)

In this course we will only consider commutative groups, which are also commutative:

5. $\forall a, b \in G, a \cdot b = b \cdot a$ (order of operands doesn't matter)

Cyclic groups

- The order of a group, G, often written |G|, is the number of elements in G

- we write g^k to denote repeated aplication of g using the group operation.
  - the order of an element g, written |g|, is the smallest integer k with $g^k = 1$
- a group element g is a generator for G if $|g| = |G|$
- a group is cyclic if it has a generator

Computing inverses modulo n

> the inverse of a , if it exists„ is a value x such that $ax \equiv 1(\mod n)$ and is written $a^{-1} mod n$

in cryptosystems, we often need to find inverses so we can decrypt, or undo, certain operations

> Theorem: Let $0 < a < n$. Then a has an inverse modulo n iff $gcd(a, n) = 1$. (a and n are co-prime)

Modular inverses using Euclidean algorithm

to find the inverse of a we can use the Euclidean algorithm, which is very efficient. since gcd(a,n) = 1, we can find ax + ny = 1 for integers x and y by Euclidean algorithm.

An actual example of modular inverses.

Since there are really bad resources for this:
From exam 2018
$8^{-1} \mod 21$
Set up the equation:

$21 = 8(factor) + remainder$
$21 = 8(2) + 5$
shift numbers one to the left
$8 = 5(factor) + remainder$
$8 = 5(1) + 3$
keep shifting till the remainder is 1
$5 = 3(1) + 2$
$3 = 2(1) + 1$

Now, for each line exchange the equation so that the remainder is alone on its side
Labelling each equation in parentheses.

$21 + 8(-2) = 5$ (eq 4)
$8 + 5(-1) = 3$ (eq 3)

$5 + 3(-1) = 2$ (eq 2)
$3 + 2(-1) = 1$ (eq 1)

Look at equation (1). You see it uses the number 2, which is defined in equation (2). Substitute equation (2) in (1):

$3 + (5 + 3(-1))(-1) = 1 \mod 21$
$3 + (5(-1) + 3) = 1 \mod 21$
$3(2) + 5(-1) = 1 \mod 21$

Now we see the number 3, which can be substituted using equation (3):

$(8 + 5(-1))(2) + 5(-1) = 1 \mod 21$
$8(2) + 5(-3) = 1 \mod 21$

Do the same for the number 5, using equation (4):
$8(2) + (21 + 8(-2))(-3) = 1 \mod 21$
$8(2) + 21(-3) + 8(6) = 1 \mod 21$
$8(8) + 21(-3) = 1 \mod 21$

-3 is not representable in mod 21, but its absolute value is smaller than our modulus of 21. Substitute -3 with 21-3 = 18.

$8(8) + 21(18) = 1 \mod 21$

we have 21(18), which is 18 times the modulus. Anything multiplied with the modulus is 0:

$8(8) = 1 \mod 21$

This is our solution. Modular inverses should satisfy $XX^{-1} = 1 \mod n$, and we see that $8 * 8 = 1 \mod 21$.

Another example

This example is implicitly required in an RSA exercise from the 2018 exam. We're required to calculate e, given n and d. The formula for e is $e = d^{-1} \mod (\phi(n))$. For completeness, $\phi(n) = \phi(21)$ prime factors of 21 are 3 and 7.
$\phi(21) = (3-1)(7-1) = 12$

Following the same steps as the above example:
$e = 5^{-1} \mod (12)$
$12 = 5(factor) + remainder$
$12 = 5(2) + 2$
$5 = 2(2) + 1$

$5 + 2(-2) = 1$ (1)
$12 + 5(-2) = 2$ (2)

substituting (2) in (1)

$5 + (12 + 5(-2))(-2) = 1 \mod 12$
$5 + 12(-2) + 5(4) = 1 \mod 12$
$5(5) + 12(-2) = 1 \mod 12$
$5(5) + 12(10) = 1 \mod 12$ again, -2 doesnt exist in mod12. 12-2 = 10 works since 2 is smaller than 10. $5(5) = 1 \mod 12$

The answer is 5. We see that $5(5) \mod 12$ does indeed equal 1.

the set of residues $\mathbb{Z}_p^*$

a complete set of resides modulo any prime p with the 0 removed forms a group under multiplication denoted $\mathbb{Z}_p^*$. It has some interesting properties:

- The order of $\mathbb{Z}_p^*$ is $p-1$
- it is also cyclic.
- it has many generators

Finding a generator for $\mathbb{Z}_p^*$

A generator of $\mathbb{Z}_p^*$ is an element of order $p-1$. To find a generator, we can choose a value and test it like so:

- compute all the distinct prime factors of $p-1$, denoted $f_1, f_2 ... f_r$
- g is a generator as long as $g^{\frac{(p-1)}{f_i}} \neq 1 \mod p$, for $i = 1, 2, .., r$

Should you be tasked to find the order of a generator for $\mathbb{Z}_n^*$ where n is not prime you need to factorize n into its prime factors pq and then use the rule
$|g| = (p-1)(q-1)$
to find the order.

For example: (from exam 2018)

> a generator for $\mathbb{Z}_{15}^*$ has order:
> (a) 1
> (b) 3
> (c) 8
> (d) 14

15 consists of the prime factors 3 and 5. The order of the generator must be $(3-1)*(5-1) = 8$.

Example of determining if a number is a generator for $\mathbb{Z}_p^*$

- $p = 7$
- $\mathbb{Z}_7^*$ has a generator $g = 4$ if the test holds.
- 4 has just one prime factor, 2.
- $g^{\frac{6}{2}} \neq 1$
- This means that $g = 4$ is not a generator for $\mathbb{Z}_7^*$

Fields

> a field is a set, F, with two binary operations + and $\cdot$, satisfying:

1. F is commutative group uner the + operation, with identity element denoted 0
2. F $\setminus$ {0} is a commutative group under the dot operation
3. distributive, $\forall (a, b, c) \in F$

Finite fields GF(p)

For secure communications, we only care about fields with a finite number of elements.
a famous theorem says that > Finite fields exist of size pˆn for any prime jesus christ hun dama går fort frem. see slides from lecture 2

- often written Z_p,instead of GF(p)
- multiplication anda ddition are done modulo p
- Multiplicative group is exactly Z_p^*
- used in digital signature schemes

For finite fields of order 2^n can use polynomial arithmetic:
$00101101 = x^5 + x^3 + x^2 + 1$
the field is representeed by use of a primitive polynomial m(x). addition and multiplication is defined by polynomial addition and multiplication modulo m(x). Division is done efficiently by hardware using shifts.

classical encryption

Terminology

We usually divide cryptology into

- Cryptography - designing the systems
- cryptanalysis - breaking them.

We usually study them together, as steganography - the study of concealing information

Block ciphers

> Block ciphers are the main bulk encryption algorithms used in commercial applications.

`Block cipers` are `symmetric key ciphers`, where the plain text is divided into blocks. Each block is encrypted/decrypted using the same key. A block is of a fixed size, often between 64 and 256 bits. Block ciphers are used in certain ways, called `modes of operations`. Each mode has different properties that make them desirable/undesirable in certain applications.

Notation

- the message is n blocks in length
- P Plaintext
- C Ciphertxtt
- K Key
- E Encryption function
- D Decryption function
- W_i Block i
- P_i plaintext block i

- C_i ciphertext block i

Criteria for block cipher design

Claude Shannon discussed two important properties of encryption:

1. Confusion
   - Making the relation between the key and ciphertext as complex as possible
2. Diffusion
   - Dissipate the statistical properties of the plaintext in the ciphertext (letter frequencies etc.)

Shannon proposed to use these techniques repeatedly using the concept of `product cipher`

Good block ciphers exihibit a socalled avalanche effect. Both a `key avalanche` and a `plaintext avalanche` is wanted, according to Shannons properties mentioned above.
A key avalanche is where a small change in the key results in a big change in ciphertext. This relates to Shannons notion of confusion. Try encrypting the same text using a simple substitution cipher and then swap two characters in the key. Observe small changes in ciphertext. Next, try doing the same using a more sophisticated encryption scheme like AES with an online tool. Observe a huge difference after altering key.
A plaintext avalanche is when a small change in plaintext results in a big change in the ciphertext. Ideally we'd like each bit to have a 50% probability to flip. This is related to Shannons notion of diffusion. Try encrypting the same text using a simple substitution cipher and then change one letter in the plaintext. Observe small changes in ciphertext. Next, try doing the same using a more sophisticated encryption scheme like AES with an online tool. Observe a huge difference after altering the plaintext.

Product cipher

> A product cipher is a cryptosystem in which the encryption function is formed by composing several sub-encryption functions

Most block ciphers compose simple functions, each with different keys.

$C = E(P, K) = f_r(...(f_2(f_1(P, K_1), K_2)...), K_r)$

Iterated cipher

A special class of product ciphers are called iterated ciphers. The encryption process in an iterated cipher is divded into r similar `rounds`, and the sub-encryption functions are all the

same function, g, called the `round function`. Each key, K_i, is derived from the `master key`, K. Each key K_i are called `round keys` or `subkeys` and are derived using a process called the `key schedule`.

Encryption in interated ciphers

$W_0 = P$
$W_1 = g(W_0, K_1)$
$W_2 = g(W_1, K_2)$
.....
$W_r = g(W_{r-1}, K_r)$
$C = W_r$

Decryption in interated ciphers

in order to decrypt the messages, an inverse of the round function, g^{-1}, must be available. the inverse must satisfy $g^{-1}(g(W, K_i), K_i) = W$, $\forall K_i, W$

Feistel ciphers

Feistel ciphers are iterated ciphers where the round function swaps two halves of the block and forms a new half on the right side.

Encryption is done in three steps:

1. Split the block into two halves: $W_0 = (L_0, R_0)$
2. For each of the r rounds, do:
   $L_i = R_{i-1}$
   $R_i = L_{i-1} \oplus f(R_{i-1}, K_i)$
   Where f is any function, note that choice of f affects security
3. Ciphertext is $C = W_r = (L_r, R_r)$.

Substitution-permutation network

Substitution-permutation networks (SPNs) are iterated ciphers. They require the block length, n to allow each block to be split into m sub-blocks of length l, so that $n = lm$ (The block length must allow you to split it into m equally long sub-blocks). SPNs define two operations:

1. Substitution, π_s, operates on sub-blocks of size l bits:
   $\pi_s : \{0,1\}^l \rightarrow \{0,1\}^l$
   The permutation π_s is usually called an S-Box(substitution box)
2. Permutation, π_p, swaps the inputs from $\{1, ..., n\}$. This is similar to the transposition cipher.
   $\pi_p : \{1, 2, ..., n\} \rightarrow \{1, 2, ..., n\}$

During the round function g of an SPN, there are three steps:

1. The round key K_i is `XOR`ed with the current block W_i.
2. Each sub-block is substituted by using substitution (pi_s)
3. The whole block W_i is permuted using permutation (pi_p)

DES

> TODO

AES

Data blocks are always 128 bits, while the key length (and number of rounds) may vary. Supports 128, 192 and 256bit master key lengths, each requiring 10, 12 or 14 rounds respectively. This makes it a substitution-permutation network with $n = 128$ and $l = 8$. The structure of the AES cipher is a byte-based substitution-permutation network consisting of:

1. initial round key addition (only AddRoundKey stage)
2. (number of rounds - 1) rounds
3. final round. (no MixColumn stage)

AES represents each block as a 4x4 matrix of bytes (128 bits = 16 bytes, which is the reason for the fixed block size), and performs both finite field operations in $GF(2^8)$ and bit string operations:

1. ByteSub (non-linear `substitution`)
   - Using a predefined lookup table (S-box), substitute each matrix cell.
2. ShiftRow (`Permutation, Diffusion`)
   - Leave top row as is.
   - Second row is shifted by 1 byte (AA,BB,CC,DD $\rightarrow$ BB,CC,DD,AA)
   - third row is shifted by 2 bytes
   - fourth row is shifted by 3 bytes
3. MixColumn (Diffusion)
   - For every column multiply by, in the field, a predetermined matrix.
4. AddRoundKey

- For every column, XOR with corresponding column of round key K_i

Key schedule

The keys are also represented as a 4x4 matrix, similar to the blocks. This requires a 128bit subkey to be used in every round. These subkeys are derived from the master key. You'll need (number of rounds + 1) subkeys in total (since you need an initial subkey for the initial round).

Security in AES

No severely dangerous attacks are known yet. If you reduce the number of rounds, security decreases. If an attacker gets hold of cipher text encrypted with a key that has a special relation to the master key, a related key attack is possible. What is a related key attack? this course doesn't know.

Block cipher modes of operation

Motivation

Block ciphers encrypt blocks, but many of them are encrypted sequentially. This is generally insecure. Using different standardised *modes of operation* with different levels of security and efficiency. This can also be used for authentication and integrity.

Randomized encryption

We can see patterns if the schemes arent random. Typically this is achieved using an initialization vector IV, which may need to be either random or unique. One can also use a state variable that changes.

Efficiency

There are several features of the modes that affect its efficiency. These do not affect seccurity, but we would like to encrypt our data before the millennia is over. Features like possibility of parrallell processing etc.

Padding

some modes require the plantext to consist of only whole blocks. If the plaintext is not a length that is divisible by block length you will need to pad the plaintext to get the desired length.

Confidentiality modes

Electronic Codebook mode (ECB)

This is dumb because you just take each block, and apply E or D to it using the same key every time.

- Not randomised.
- Padding required.
- We can do parallell encryption/decryption though.
- Errors propagate within blocks.
- No initialization vector IV.

Cipher block chaining mode (CBC)

CBC "chains" the blocks together.

Encryption: $C_t = E(P_t \oplus C_{t-1}, k) where C_0 = IV$

Decryption: $C_t = D(C_t, k) \oplus C_t - 1, where C_0 = IV$

- randomised.
- Padding required.
- Errors propagate within blocks and to specific bits of next blocks
- We can do parallell decryption, no encryption.
- IV must be random

Counter mode (CTR)

A counter and nounce is used. They are initialized by a randomly chosen value N. T_t is the concatenation between the nonce and block number t, N||t. $O_t = E(T_t, k)$

This is XORed with the plaintext block.

Encryption: $C_t = O_t \oplus P_t$

Decryption: $P_t = O_t \oplus C_t$

A one bit change in the ciphertext produces a one bit error in the plaintext at the same location

- randomised.
- Padding not required.
- Errors occur in specific bits of the current block
- both parallell encryption and decryption
- Variable, nonce, which must be unique

Good for accessing specific plaintext blocks without decrypting the whole stream.

Message integrity

How to ensure that the message is not altered in the transmission? We treat message integrity and message authentication as the same thing. This includes preventing an adversary from fucking with your blocks. Message integrity can be provided whether or not encrpytion is used for confidentiality.

Message Authentication Code (MAC)

A mechanism for ensuring message integrity. On input secret key, K , and an arbitrary length message M, a MAC algorithm otuts a short fixed-length string, T, known as the tag.

$T = MAC(M, K)$

Two entities A and B share a common key K, and A wants to send message, M, to B.

- A computes the tag
- A sends the message M and also the tag.
- B recomputes the tag on the received message and checks if the new tag and received tag are equal.

This provides sender authentication to the message, since only A and B knows the key, and are thus the only ones that can produce the tag, T. If A and B makes two different tags, B must conclude that shit happened and the message is fiddled with. If the tags are the same, all is good.

This basic property is called `unforgeability`. It is not feasible to produce a message, M, and a tag, T, such that $T = MAC(M, K)$ without knowing the key, K. This includes the scenario where the attacker has access to a `forging oracle`, which means that the attacker can insert any message to a function and get the corresponding tag out. (chosen plaintext attacks)

It is not feasible for an attacker to produce a valid forgery.

Pseudo random numbers and stream ciphers

> Random values are important, and are the building blocks of stream ciphers.

Random numbers

Defining randomness is hard. We would like to get bitstrings that just as random as any other.

Generators of random numbers are divded into categories:

- True random number generators (TRNG)

  - Physical processes which outputs each valid string independently with equal probability
- Pseudo random number generator (PRNG)
  - A deterministic algorithm made to approximate a true random number generator

True random number generators

NIST has provided a framework for esign and validation of TRNGs, called entropy sources. These entropy sources includes a physical source of noise, a sampling process and post processing. The output is any requested number of bits. The standard specified by NIST also includes statistical tests for validating the entropy sources.

Pseudo randon number generators

NIST recommends specific PRNG algorithms named Deterministic random bit genrators (DRBG), based on hash functions, HMAC and block ciphers in counter mode (see lec 3). They often use a TRNG to seed the state.

security of a DRBG

The security is defined in terms of the ability of an attacker to distinguish between a PRNG and TRNG. This is measuree by two properties:

1. Backtracking resistance
   - An attacker who knows the current state of the DRBG cannot distinguish between earlier outputs.
   - If you see one output, you cannot make sense of, or guess, earlier outputs?
2. forward prediction resistnace
   - An attacker who kows the current state of the DRBG should not be able to distinguish between later states and the current.

CTR_DRBG

Uses a block cipher in CTR mode (see lec. 3), such as AES with 128bit keys. The DRBG is initialized with a seed which matches the length of the key and block summed. This seed defines a key, k , which is used in AES. No nonce and CTR value is used, like innormal CTR mode, but rather uses the seed value.

update function in CTR_DRBG

Each request to the DRBG generates of up 2^{19} bits. State must be updated after each request, which is handled by the update function. (K,ctr) must be updated by generating twoblocks using the old key to make a new one.

Stream ciphers

> Stream ciphers are characterise by the generation of a keystream using a short key and an init value as input

Each element of the stream is used successively to encrypt one or more plaintext characters.

Symmetric. Given the same key value, both sender and receiver can encrypt and decrypt the same.

Synchornous stream cipherrs

Simplest kind of stream ciphers, as the keystream is generated independently of the plaintext. Both sender and receiver nee the same keystream, and synchronise their position in it. In a way, one can look at the vigenere cipher as a periodic synchronous stream cipher where each shift is defined by a key letter.

Binary synchronous stream cipher

For each time interval, t, each of the following are defined:

- A binary sequence, $s(t)$, called the keystream
- a binary plaintext $p(t)$
- a binary ciphertext $c(t)$

encryption: $c(t) = p(t) \oplus s(t)$

decryption: $p(t) = c(t) \oplus s(t)$

Shannons definition of perfect secrecy

> to define perfect secrecy, consider a cipher with message set M and ciphertext set C. Then $PR(M_i|C_j)$ is the probability that the message M_i was encrypted given that ciphertext C_i is observed. The cipher achieves perfect secrecy if for all messages and ciphertexts that $Pr(M_i|C_i) == Pr(M_i)$

If we cannot tell whether the message is encrypted with one key or another, and everything is just complete bullshittery guessing - it is a perfect secret.

One time pad using roman alphabet example

Plaintext: HELLO

Keystream: EZABD

ciphertext: LDLMR

Since the probability of each charcter in he keystream is equally plausible, the 5-letter ciphertext can equally possibly be every 5-letter string.

One time pad properties

Any cipher with perfect secrecy must have as many keys as there are messages. In a sense, it is the only unbreakable cipher. But it suffers from key management problems and actually getting completely random keys. Key generations, transportation, sync, destruction and problematic since the keys are possibly very large.

Visual cryptography

> An appication of the one time pad is visual cryptography which splits an image into two shares Decryption works by overlaying the two shared images

Works by splitting the pixel in a random way, just like splitting a bit in the one time pad. Each split doesnt reveal any info about the image, again alike the one time pad.

encrypting an image:

- generate the one time pad, P, (random string of bits), with length equal to the number of pixels in the image
- Generate an image share, S_1, by replacing each bit in the random bitstring by using some sub pixel patterns.
- Generate the other image share, S_2, with pixels as follows:
  - The same as S_1 for all white pixels of the image
  - The opposite for all black pixels.

To reveal the hidden images, the two shares are overlaye. Each black pixel is black in the overlay, each white pixel is half-white in the overlay.

Conclusion

TRNGS can be constructed from phsical devices and used as seeds. PRNGs can be constructed from other primitives like block ciphers. TRNGs can be use to make unbreakable encryption via one time pad. PRNGs can be use as practical synchronous stream ciphers

More number theory for public key cryptography.

Chinese remainder theorem

> let $d1 \dots dr$, be pairwise relatively prime and n = d1d2...dr, given any integers c_i there exists a unique integer x with $0 \leq x < n$ such that
>
> $x \equiv c_1(\mod d_1)$
>
> $x \equiv c_2(\mod d_2)$
>
> $x \equiv c_3(\mod d_3)$
>
> $\dots$
>
> $x \equiv c_r(\mod d_r)$

$x \equiv 5(\mod 6)$ $x \equiv 33(\mod 35)$

set $d_1 = 6$, and $d_2 = 35$, then $n = d_1 \cdot d_2 = 210$

see slides from lecture 7 for an example. I can't write this.

Euler function ϕ

> For a positive integer n, the euler function $\phi(n)$ denotes the number of positive integers less than n and relatvely prime to n.

for example: $\phi(10) = 4$ since 1,3,7,9 are each relatively prime to 10 and less than 10. The set ofp positive integerrs less than n and relatively prime to n form the reduced residue class $\mathbb{Z}_n^*$
$\mathbb{Z}_n^* = 1, 3, 7, 9$

properties of the euler functon $\phi(n)$

1. $\phi(p) = p - 1$ for p prime
2. $\phi(pq) = (p-1)(q-1)$ for p and q distinct primes
3. let $n = p_1^{e_1} ... p_t^{e_t}$ where p_i are distinct primes. Then $\phi(n) = \prod p_i^{e_1 - 1}(p_i - 1)$ (generalization of point 2)

Example:

$\phi(15) = \phi(5) * \phi(3) = (5-1) * (3-1) = 4 * 2 = 8$

$\phi(24) = 2^2(2-1)3^0(3-1) = 8$

(where $24 = 2^3 * 3$)

Fermats theorem

> let p be a prime, then $a^{p-1} \mod p = 1$, for all integers a with $1 < a \leq p - 1$.

Eulers theorem

> $a^{\phi(n)} \mod n = 1$, if $gcd(a, n) = 1$.

When p is prime then $\phi(p) = p - 1$, so fermat's theorem is a special case of eulers theorem

Discrete logarithm problem

> let g be a generator for $\mathbb{Z}_p^*$ for a prime p. The discrete log problem is:
> given y in $\mathbb{Z}_p^*$ find x with $y = g^x \mod p$.

If p is large enough, this is believed to be a hard problem. Usually rsa-length, 2048 bits.

$log_x g^x = x log_g g = x$

RSA

Keys

Keys consist of several numbers. You need two random, distinct primes, p and q, the product of these called the modulus, n, a public exponent, e and a private exponent, d.
After choosing two primes of satisfying size, p and q, multiply these to attain n. The size should be at least 1024 bits by todays standards (according to slides.).

Next up is e. It only needs to satisfy $gcd(e, \phi(n)) = 1$. 3 is the smallest possible value, and is sometimes used. Not recommended as it might introduce security problems. However, 65537 ($2^{16} + 1$, or 2^{2^4}) is a popular choice. Since it is prime (largest known prime on the form $2^{2^n} + 1$, called a fermat prime), it satisfies the equation for every n and does not require any additional checking. As an added bonus, this number has just two set bits in binary (`10000000000000001`). This makes it an easy number to perform arithmetic on.

d is computed as $e^{-1} mod(\phi(n))$.

These values make up for the private and public keys for RSA encryption. n and e is the public key, written as $K_E = (n, e)$. p, q and d is the private key, written as $K_D = d$. The values p and q are not used directly in encryption or decryption.

Note that the equation for e, given n and d is similar to the equation for d:
$d = e^{-1} mod(\phi(n))$.
$e = d^{-1} mod(\phi(n))$.

Might be useful for an exam.

Encryption

The input of the encryption is called M, which is a value that is less than n.
$0 < M < n$
in order for a message to become M, it needs to be preprocessed by encoding letters to numbers, as well as adding randomness.

The ciphertext, C, is computed as $E(M, K_E) = M^e \mod n$

Decryption

The plaintext is retrieved by computing $D(C, K_D) = C^d \mod n = M$.
RSA decryption can be done more efficiently by utilizing the chinese remainder theorem. Good luck to you, I dont know math.
It achieves up to 4 times speed up sequentially, or 8 times if it is ran in parallell. Because of optimizations like this, you generally want to keep p and q instead of discarding them after generating them – even though they aren't strictly needed for encryption or decryption.

Example (stolen from slides)

not using chinese remainder theorem.
let $p = 43$, $q = 59$.
This means $n = pq = 2537$ and $\phi(n) = (p-1)(q-1) = 2436$.
let $e = 5$. We assume whoever wrote the slides have checked whether 5 satisfies the equation for e
This means $d = e^{-1} \mod (\phi(n)) = 5^{-1} \mod (2436) = 1949$. (by calculating the modular inverse, which totally sucks ass).
Assuming an already preprocessed message, M, with the value 50, the encryption process looks like:
$C = M^e \mod (n) = M^5 \mod (2537) = 2488$.

Likewise, decryption looks like this:
$M = C^d \mod (n) = C^{1949} \mod (2537) = 50$.

Note how only publicly known values are used in encryption, while decryption uses the private exponent.

preprocessing/padding messages

Just encoding each letter to a number offers weak security. This can be observed to create an attack dictionary, or even for a `known plaintext attack`. `Håstads attack` is also a thing, which is described later. To prevent this, we preprocess the messages by padding to prepare the messages for encryption. These mechanisms must include redundancy and randomness.

PKCS number 1

Table 1: encryption block format

00	02	PS	00	D

- 00 is a byte
- 02 is a byte
- PS is a string of non-zero, pseudo random bytes. Minimum 8 bytes long.
- D is the data to be encrypted. This is the same length as the modulus, n.

Using this scheme ensure that even short messages result in a big number for encryption.

Optimal Asymmetric Encryption Padding (OAEP)

OAEP is an encoding scheme that is a feistel network. It includes k_0 bits of randomness and k_1 bits of redundancy. It also features the use of two hash functions in the network. This means that small changes in any bit going into the hash functions drastically alters the output. Because of this we say that the scheme features an "all or nothing" security. This means that in order to recover the message, you also need to recover the complete random string included.

Not sure if the algorithm is a big part of the curriculum. So it is left out for now.

Key points:

1. it pads the input to achieve the same as PKCS #1
2. It includes randomness and redundancy
3. because of the hash functions ("all or nothing"), partial messages or other information will not be leaked
4. provides security against chosen ciphertext attacks (and possibly chosen plaintext, not 100% sure)

Diffie Hellmann key exchange

Motivation

Discrete log based cipers are currently the main alternative public key systems, other than rsa.

> Designed to allow two users, alice and bob, to share a secret using only public communications.

Public knowledge includes:

- Large prime, p
- generator g of $\mathbb{Z}_p^*$

Basic protocol

Alice chooses an $a \in \mathbb{Z}_p^*$, sends $K_a = g^a \mod p$ to bob. Next, Bob chooses a $b \in \mathbb{Z}_p^*$ and sends $K_b = g^b \mod p$ to alice.

now both have knowledge of a $Z = (g^b)^a \mod p$. Z can be used to compute a key for various crypto schemes. It is secure, as the only numbers that are being broadcast are $g^a \mod p$ and $g^b \mod p$. To find A and B from this you'd need to factorise the numbers which is hard.

Authenticated diffie hellmann

It is rather easy to set up a man in the middle attack for this scheme. Just have the adversary set up keys with both alice and bob, and just relay the messages using these keys. Alice and Bob shouldn't be any wiser.

Alice thinks she sends message to bob, is in reality sending to malicious attacker - constructs K_{ac}. The attacker does the same with bob and constructs K_{bc}. If alice wants to send something to bob, it goes through the attacker using K_{ac} and is passed onto bob using K_{bc}.

Alice and Bob cannot in reality send messages directly to eachother, it has to go through the man in the middle, which can read everything thanks to the keys.

This is fixed by adding digital signatures.

static and emphemeral diffie-hellmann

The protocol described above uses *ephemeral keys*: Keys which are used once and then discarded. In a static diffie-hellmann scheme you'd let each party choose a long-term private key X_a with corresponding key $Y_a = g^{xA} \mod p$. If each party has a long term key, they can simply look up eachothers keys and possibly skip the initial handshake.

Elgamal cryptosystem

> Turning the diffie-hellmann protocol into a cryptosystem since 1985

Based on one party having ephemeral keys, while the other has a long-term key. The long-term key works like a public key, while the ephemeral keys are private

Key generation:

- Select a prime p and a generator g of $\mathbb{Z}_p^*$
- select a long term private key x where $0 < x < p - 1$ and compute $y = g^x \mod p$
- The public key is (p, g, y)

Encryption:
The public key for encryption is $K_E = (p, g, y)$

1. for any value (message) M, where $0 < M < p$
2. choose k at random, and compute $g^k \mod p$
3. $C = E(M, K_E) = (g^k \mod p, My^k \mod p)$

Decryption:
The private key for ecryption is $K_D = x$ with $y = g^x \mod p$

1. let $C = (C_1, C_2)$
2. $D(C_1, K_D) = C2 \cdot (C_1^x)^{-1} \mod p = M$

Why does it work?

The sender knows the ephemeral private key k. The receiver knows the static private key x. Both sender and recpient can compute the diffie-hellmann value for the two public keys $C_1 = g^k \mod p$ and $y = g^x \mod p$. The value $y^k \mod p = C_1^x \mod p$ is used as a mask for the message m that pushes the value to another value in that group.

Security of Elgamal

The whole system is based on the difficulty of the discrete logarithm problem. If you can solve this problem and find x from $g^x \mod p$, the system is broken. Does not need padding, and does not require unique keys for every user.

Elliptic curves

> Elliptic curves are algebraic structures formed from cubic equations. But hey, we won't be using elliptic curves in the reals. -Cris Carr

For example:
The set of all (x, y) pairs which satisfy the equation $y^2 = x^3 + ax + b \mod p$. This is a curve over the field $\mathbb{Z}_p$. Elliptic curves can be defined over any field.

Adding an identity element makes it possible to define binary operations on these curves. Doing so defines elliptic groups.

The discrete log problem can be defined on elliptic curve groups, with the same definition if we define the elliptic curve group with multiplication. The best known algorithm to solve the elliptic curve discrete logarithm problem are exponential with the length of the parameters. Because of this, most elliptic curve implementations use much smaller keys. If we compare this to RSA, the relative advantage of elliptic curve cryptographywill increase at higher security levels.

We can use elliptic curves in several cryptosystems, like the diffie hellmann key exchange and elgamal.

Identity-based cryptography

TODO
See lecture 9

Hash functions

> A hash function, H, is a public function such that:
> H is simple and fast to compute
> H Takes as input a message, m, of arbitrary length and outputs a message digest H(m) of fixed length

Good hash functions show some properties:

1. Collision resistance
    - It should not be feasible to find two different inputs that produces the same output.
    - It's not possible to construct two values that produce the same hash
2. Second-preimage resistance
    - Given a value, it should be infeasible to find a different value that produces the same hash
    - It's not possible to construct a value that produce the same hash as a given value
3. One-way (preimage resistance)

- Given a hash, it should be infeasible to find an input that produce the same hash.
- You cannot find the input, given the output.

The birthday paradox

If we choose $\sqrt{M}$ values from a set of size M, the probability of getting two identical values (or in this context: hash collision) is about 50%. This is particularly useful in this course to compute how many bits a hash should be in order to be safe.

If a hash function has an output of k bits, and is regarded random to the outside world, then $2^{\frac{k}{2}}$ attempts should yield a collision with a 50% probability. For those, including me, who keeps forgetting math: $\sqrt{2^k} = 2^{\frac{k}{2}}$, so this is all according to the birthday paradox.

Today (the date the slides were made) 2^{128} attempts is considered infeasible, so your hash functions should output at least $2^{\frac{k}{2}} = 2^{128} \rightarrow k = 128 * 2 = 256$ bits in order to be considered collision resistant.

Iterated hash functions

Just like block ciphers, hash functions also need to be able to handle inputs of all sizes and shapes to produce a fixed size output. Iterated hash functions solve this challenge like the iterated block ciphers did, by splitting the input into fixed sized blocks and repeatedly using the function.

Note that iterated hash functions operate on each block sequentially using the same function.

Merkle-Damgård construction

> Use a fixed-size compression function applied to multiple blocks of the message

A compression function here is defined as a function that takes two n-bit input strings and produces a single n-bit output stirng.

The Merkle-Damgård constuction chains these together. An IV (similar to the ones used in block ciphers) and the first block of a message, m_1, is input to the compression function. The output is used as the input for the second compression, instead of the IV, in addition to the block, m_2.

Note that this scheme requires padding, as well as encoding of the length of m. In the last chain of the process, the input is not a message block but the padding and encoded length.

In mathematical notation:
compression function, $h(m_l, h_{l-1}) = h_l$

Merkle-Damgård construction:
$H(m) = h(PADDING, h_l)$

$h_l = h(m_l, h_{l-1})$ where $h_0 = IV$

Properties of Merkle-Damgård construction

If the compression function, h, is collision-resistant, then the hash function, H, is collision-resistant.

The construction suffers from some weaknesses as well:

1. Once you find a collision, it is easy to find more (length extension attack)
2. second pre-image attacks are not as hard as you'd think (construct a value that produce the same hash as a given value)
3. Collisions for multiple messages can be found without much more difficulty than collisions for 2 messages.

Still, the Merkle-Damgård constuction is used in standard and former standard hash functions (MD5, SHA-1, SHA-2)

Standardized hash functions

Slides lack many implementational details, so I'm guessing its not important for the course. This section only includes some basic key points.

MDx family of hashes

Old and insecure by todays standards. MD2, 4 and 5 have been used in practice, but are all easily broken.

Is based on the Merkle-Damgård construction. They all output 128 bits. Recall the section about the birthday paradox and how many bits were recommended.

SHA-0 and SHA-1

Based off of MDx hashes (which makes it a Merkle-Damgård constuction), but with added complexity and a bigger output size of 160 bits (weak). Both are broken, but this is quite recent. First attack of SHA-1 was found in 2017.

SHA-2 family

Several versions of SHA-2 exist, hence the term "family of SHA-2 hashes". They are developed in response to attacks on MD5 and SHA-1. Still a Merkle-Damgård construction.

Table 2: summary of SHA-2 family. Taken from lecture 10 slides of spring 2019.

	Hash size	Block Size	Security Match
SHA-224	224 bits	512 bits	2key 3DES
SHA-512/224	224 bits	1024 bits	2key 3DES
SHA-256	256 bits	512 bits	AES-128
SHA-512/256	256 bits	1024 bits	AES-128
SHA-384	384 bits	1024 bits	AES-192
SHA-512	512 bits	1024 bits	AES-256

The SHA-2 family is still a Merkle-Damgård construction so it needs a padding scheme. First off it needs a field for the message length encoding. This field is 64 bits long if the block length is 512 bits, and 128 bits long if the block length is 1024 bits. After the message length field, there is padding. There is always at least one bit of padding. After the first `1` in the padding, enough `0` are added to get a complete block.

Since the padding requires at least 1 bit of pad and either 64 or 128 bits of encoding, this sometimes results in adding a new block.

SHA-3

The MDx and previous SHA hashes were based on the same design, which has encountered unexpected attacks. The SHA-3 hash is the result of a competition (just like AES), held in 2007-2008. This ended up with a new function that was standardized in 2015 and is NOT based on the Merkle-Damgård construction. It uses a sponge construction, whatever that is.

HMAC

A MAC constructed from any iterated cryptographic hash function (like SHA256 etc). HMAC is defined as: $HMAC(M, K) = H((K \oplus opad)||H((K \oplus ipad)||M))$

- M: Message to be authenticated
- K: Key padded with zeros to the blocksize of H
- opad: hardcoded string
- ipad: hardcoded string

HMACs are secure if H is collision resistant or if H is a pseudorandom function. It is designed to resist length extension attacks, even if H is a Merkle-Damgård construction (which are vulnerable to such attacks).

HMAC is often used as a pseudorandom function for deriving keys (since they are deterministic but seem random)

Authenticated encryption

How do you ensure both confidentiality (no one can read your messages) and integrity (you know the message is from a legitimate sender)? A proposed solution is to split your assumed established shared key, K, into two parts - one for encryption and one to obtain a MAC.

There are three possible ways to combine encryption and MACs:

1. Encrypt-and-MAC
   - Encrypt message, apply MAC to message and send the two results
   - $C \leftarrow Enc(M, K_1)$
   - $T \leftarrow MAC(M, K_2)$
   - Send $C||T$
2. MAC-then-encrypt
   - Apply MAC to message to get tag. Then encrypt message concatenated with tag and send the ciphertext.
   - $T \leftarrow MAC(M, K_1)$
   - $C \leftarrow Enc(M||T, K_2)$
   - Send C
3. encrypt-then-MAC
   - encrypt message to get ciphertext. Then apply MAC to ciphertext and send the two results
   - $C \leftarrow Enc(M, K_1)$
   - $T \leftarrow MAC(C, K_2)$
   - Send $C||T$

Encrypt-then-MAC is the safest of the three. (Because the tag cannot possibly leak information about the plaintext?)

Some schemes do, however, provide both confidentiality and integrity with one key.

Galois Counter Mode