TTM 4135 notes

Joakim Lier

Contents

1 Number theory **6**
1.1 Basic number theory . . . 6
1.1.1 Basic properties of factors . . . 6
1.1.2 Division algorithm . . . 6
1.2 GCD . . . 6
1.2.1 Euclidean algorithm & extended euclidean algorithm(EEA) . . . 6
1.3 Modular arithmetic . . . 7
1.3.1 Residue class . . . 7
1.3.2 Notation: a mod n . . . 7
1.4 Groups . . . 7
1.4.1 Cyclic groups . . . 7
1.5 Computing inverses modulo n . . . 8
1.6 Modular inverses using Euclidean algorithm . . . 8
1.6.1 An actual example of modular inverses. . . . 8
1.6.2 Another example . . . 10
1.7 the set of residues $\mathbb{Z}_p^*$ . . . 10
1.8 Finding a generator for $\mathbb{Z}_p^*$ . . . 10
1.9 Example of determining if a number is a generator for $\mathbb{Z}_p^*$ . . . 11
1.10 Fields . . . 11
1.10.1 Finite fields GF(p) . . . 11

2 classical encryption **12**
2.1 Terminology . . . 12

3 Block ciphers **12**
3.1 Notation . . . 12
3.2 Criteria for block cipher design . . . 13
3.3 Product cipher . . . 13
3.4 Iterated cipher . . . 14
3.4.1 Encryption in iterated ciphers . . . 14
3.4.2 Decryption in iterated ciphers . . . 14
3.5 Feistel ciphers . . . 14
3.6 Substitution-permutation network . . . 14
3.7 DES . . . 15
3.8 AES . . . 15
3.8.1 Key schedule . . . 16
3.8.2 Security in AES . . . 16

4 Block cipher modes of operation **16**
4.1 Randomized encryption . . . 16
4.2 Efficiency . . . 16
4.3 Padding . . . 16
4.4 Confidentiality modes . . . 17
4.4.1 Electronic Codebook mode (ECB) . . . 17
4.4.2 Cipher block chaining mode (CBC) . . . 17
4.4.3 Counter mode (CTR) . . . 17
4.5 Message integrity . . . 18
4.5.1 Message Authentication Code (MAC) . . . 18
4.5.2 Basic CBC-MAC (CMAC) . . . 18
4.5.3 standardized CBC-MAC . . . 19

5 Pseudo random numbers **19**
5.1 Random numbers . . . 19
5.2 True random number generators . . . 20
5.3 Pseudo randon number generators . . . 20
5.3.1 security of a DRBG . . . 20
5.4 CTR_DRBG . . . 20
5.4.1 update function in CTR_DRBG . . . 20

6 Stream ciphers **21**
6.1 Synchronous stream ciphers . . . 21
6.1.1 Binary synchronous stream cipher . . . 21
6.2 Shannon's definition of perfect secrecy . . . 21
6.2.1 One time pad using roman alphabet example . . . 21
6.3 One time pad properties . . . 22
6.4 Visual cryptography . . . 22
6.5 Conclusion . . . 22

7 More number theory for public key cryptography. **23**
7.1 Chinese remainder theorem . . . 23
7.1.1 Exam question example using chinese remainder theorem: . . . 23
7.2 Euler function ϕ . . . 23
7.2.1 properties of the euler function $\phi(n)$ . . . 23
7.3 Fermat's theorem . . . 24
7.4 Euler's theorem . . . 24
7.5 Discrete logarithm problem . . . 24

8 RSA **24**
8.1 Keys . . . 24
8.2 Encryption . . . 25
8.3 Decryption . . . 25
8.4 Example (stolen from slides) . . . 25
8.5 preprocessing/padding messages . . . 26
8.5.1 PKCS number 1 . . . 26
8.5.2 Optimal Asymmetric Encryption Padding (OAEP) . . . 26
8.6 Attacks against RSA . . . 27

9 Diffie Hellmann key exchange **27**
9.1 Motivation . . . 27
9.2 Basic protocol . . . 28
9.3 Authenticated diffie hellmann . . . 28
9.4 static and ephemeral diffie-hellmann . . . 28

10 Elgamal cryptosystem **28**
10.1 Why does it work? . . . 29
10.2 Security of Elgamal . . . 29

11 Elliptic curves **29**

12 Identity-based cryptography **30**

13 Hash functions **30**
13.1 The birthday paradox . . . 30
13.2 Iterated hash functions . . . 31
13.3 Merkle-Damgård construction . . . 31
13.3.1 Properties of Merkle-Damgård construction . . . 31
13.4 Standardized hash functions . . . 32
13.4.1 MDx family of hashes . . . 32
13.4.2 SHA-0 and SHA-1 . . . 32
13.4.3 SHA-2 family . . . 32
13.4.4 SHA-3 . . . 33
13.5 HMAC . . . 33

14 Authenticated encryption **34**
14.1 Combining encryption and MAC . . . 34
14.2 Modes for Authenticated encryption . . . 34
14.2.1 Counter with CBC-MAC mode (CCM) . . . 35

14.2.2 Galois Counter mode (GCM) 35

15 Digital signatures **35**
15.1 Elements of a digital signature scheme 36
15.1.1 signature generation algorithm 36
15.1.2 signature verification algorithm 36
15.2 Security goals 36
15.3 RSA signatures 37
15.4 Elgamal signature scheme in $\mathbb{Z}_p^*$ 37
15.5 Digital signature algorithm (DSA) 38
15.6 Elliptic curve DSA (ECDSA) 39

16 Certificates and PKI **39**
16.1 Public key infrastructure (PKI) 39
16.2 Digital certificates 39

17 Key establishment **40**

18 TLS **40**
18.1 Record protocol 40
18.1.1 Cryptographic algorithms in TLS 41
18.2 Handshake protocol 42
18.2.1 Phase 1 - “hello” 42
18.2.2 Phase 2 - Server 42
18.2.3 Phase 3 - Client 42
18.2.4 Phase 4 - start of communications, summary of handshake 43
18.2.5 TLS Ciphersuites 43
18.3 Ephemeral Diffie-Hellmann handshake 43
18.4 RSA handshake 43
18.5 Other variants 44
18.6 Generating session keys 44
18.7 Alert protocol 44
18.8 Forward secrecy 44
18.9 Attacks on TLS 45
18.9.1 BEAST 45
18.9.2 CRIME and BREACH 45
18.9.3 POODLE 45
18.9.4 Heartbleed 45
18.9.5 Man-in-the-middle attacks 46

18.10TLS 1.3 46

1 Number theory

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

1.1 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 $sqrt(n)$.

1.1.1 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

1.1.2 Division algorithm

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

1.2 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 $gcd(a, b) = 1$

1.2.1 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)

1.3 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

1.3.1 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 $\mathbb{Z}$

1.3.2 Notation: a mod n

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

1.4 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)

1.4.1 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 application 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

1.5 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)

1.6 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.

1.6.1 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.

1.6.2 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 doesn't exist in mod 12. 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.

1.7 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

1.8 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 $(3-1)*(5-1) = 8$.

1.9 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.
- $g^{\frac{6}{2}} = 1$
- This means that $g = 4$ is not a generator for $\mathbb{Z}_7^*$

1.10 Fields

> a field is a set, F, with two binary operations + and ·, satisfying:

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

1.10.1 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 and a addition 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 represented 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.

2 classical encryption

2.1 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

> TODO

3 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.

3.1 Notation

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

- P_i plaintext block i
- C_i ciphertext block i

3.2 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 exhibit a so-called avalanche effect. Both a `key avalanche` and a `plaintext avalanche` is wanted, according to Shannon's properties mentioned above.

A key avalanche is where a small change in the key results in a big change in ciphertext. This relates to Shannon's 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 Shannon's 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.

3.3 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.

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

3.4 Iterated cipher

A special class of product ciphers are called iterated ciphers. The encryption process in an iterated cipher is divided 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`.

3.4.1 Encryption in iterated ciphers

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

3.4.2 Decryption in iterated ciphers

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

3.5 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 $C = W_r = (L_r, R_r)$.

3.6 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)

3.7 DES

TODO

3.8 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

3.8.1 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).

3.8.2 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.

4 Block cipher modes of operation

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.

4.1 Randomized encryption

We can see patterns if the schemes aren't 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.

4.2 Efficiency

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

4.3 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.

4.4 Confidentiality modes

4.4.1 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 parallel encryption/decryption though.
- Errors propagate within blocks.
- No initialization vector IV.

4.4.2 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 parallel decryption, no encryption.
- IV must be random

4.4.3 Counter mode (CTR)

A counter and nonce 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 parallel encryption and decryption
- Variable, nonce, which must be unique

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

4.5 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 encryption is used for confidentiality.

4.5.1 Message Authentication Code (MAC)

A mechanism for ensuring message integrity. On input secret key, K , and an arbitrary length message M, a MAC algorithm outputs 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.

4.5.2 Basic CBC-MAC (CMAC)

we've only discussed the properties of a MAC, not how the tag is created. CMAC is one way of creating a tag, using a block cipher. This is unforgeable as long as the message length is fixed.

Let M be the message consisting of n blocks. To compute CBC-MAC(M,k), do:

Algorithm 1: CBC-MAC

Input : Message M, initialization vector IV, Key K

Output: CMAC tag

for $t \leftarrow 1$ **to** n **do**
  $C_t = E(M_t \oplus Ct - 1, K)$, where $C_0 = IV$
end
return $T = C_t$

Note that unlike the CBC-mode, the IV has to be fixed and public. A random IV is not secure in this application. E is defined as the encryption for CBC-mode, see sec. 4.4.2.

4.5.3 standardized CBC-MAC

A secure version of CMAC is standardized with some changes from the basic version:

- The original key, K, is used to derive two new keys, K_1 and K_2.
- One is used in the basic algorithm, the other is XORed into the final message block. Pad if necessary.
- The IV is set to all zeroes.
- The MAC tag is the T_{len} most significant bits of the output.

Choice of T_{len} depends on the degree of security needed. The standard states that 64 bits lets most applications resist guessing attacks. More generally, the standard states that the tag should be at least $log_2 \frac{I}{R}$ bits long, where I is how many invalid messages that can be detected before you change the key and R is the accepted risk that a false message is accepted.

5 Pseudo random numbers

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

5.1 Random numbers

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

Generators of random numbers are divided 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

5.2 True random number generators

NIST has provided a framework for design 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.

5.3 Pseudo randon number generators

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

5.3.1 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 measured 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 resistance
    - An attacker who knows the current state of the DRBG should not be able to distinguish between later states and the current.

5.4 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 in normal CTR mode, but rather uses the seed value.

5.4.1 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 two blocks using the old key to make a new one.

6 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.

6.1 Synchronous stream ciphers

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.

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

6.2 Shannon's 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_j is observed. The cipher achieves perfect secrecy if for all messages and ciphertexts that $Pr(M_i|C_j) == Pr(M_i)$

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.

6.2.1 One time pad using roman alphabet example

Plaintext: HELLO

Keystream: EZABD

Ciphertext: LDLMR

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

6.3 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.

6.4 Visual cryptography

> An application 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 doesn't 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 overlaid. Each black pixel is black in the overlay, each white pixel is half-white in the overlay.

6.5 Conclusion

TRNGS can be constructed from physical 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

7 More number theory for public key cryptography.

7.1 Chinese remainder theorem

> let *d1*, ... ,*dr*, be pairwise relatively prime and n = d1,d2, ... , dr, given any integers c_i there exists a unique integer x with $0 \leq x < n$ such that
>
> $x \equiv c_1 (\mod d_1)$
>
> $x \equiv c_2 (\mod d_2)$
>
> $x \equiv c_3 (\mod d_3)$
>
> ...
>
> $x \equiv c_r (\mod d_r)$

7.1.1 Exam question example using chinese remainder theorem:

Given a set of equations on the form $x_i \equiv y_i (\mod z_i)$
Like stated above, the Chinese remainder theorem can only be used if the set $Z = (z_1, ..., z_n)$ are pairwise coprime.

Example from exam 2018:

> Which of the following pairs of equations can be solved using the chinese remainder theorem?
> (a) $x \equiv 3 (\mod 6)$ and $x \equiv 4 (\mod 8)$
> (b) $x \equiv 3 (\mod 6)$ and $x \equiv 4 (\mod 10)$
> (c) $x \equiv 3 (\mod 7)$ and $x \equiv 4 (\mod 12)$
> (d) $x \equiv 3 (\mod 7)$ and $x \equiv 4 (\mod 14)$

Looking at the z values (the moduluses), we see that only 7 and 12 are coprime (answer c). the other options are divisible by 2 (6 and 8, 6 and 10) or a factor of each other (7 and 14).

7.2 Euler function ϕ

> For a positive integer n, the euler function $\phi(n)$ denotes the number of positive integers less than n and relatively 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 of positive integers less than n and relatively prime to n form the reduced residue class $\mathbb{Z}_n^*$
$\mathbb{Z}_n^* = 1, 3, 7, 9$

7.2.1 properties of the euler function $\phi(n)$

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 $\phi(n) = \prod p_i^{e_i-1}(p_i - 1)$ (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 $24 = 2^3 * 3$)

7.3 Fermat's theorem

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

7.4 Euler's 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 Euler's theorem

7.5 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.

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

8 RSA

8.1 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 today's 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 $2^{2^4}+1$) 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.

8.2 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 $E(M, K_E) = M^e \mod n$

8.3 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 don't know math.
It achieves up to 4 times speed up sequentially, or 8 times if it is ran in parallel. 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.

8.4 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.

8.5 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.

8.5.1 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.

8.5.2 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)

8.6 Attacks against RSA

A properly set up RSA scheme has pretty good security. Many proposed attacks on RSA are avoided by using standardised padding schemes.

The best known attack against RSA is factorization of n, which is a hard problem. The attack is prevented by choosing a large enough n.

Another candidate is to find the private key from the public key. It is supposedly not feasible, and at least as hard as factorizing n.

Other possibilities are the use of quantum computers, which do not exist yet, and timing attacks, which have countermeasures.

It is an open problem whether it is possible to crack RSA without factorizing n.

In other words, the biggest flaw of RSA isn't the algorithm itself. Key generation flaws due to poor random number generation is often the culprit.

9 Diffie Hellmann key exchange

9.1 Motivation

Discrete log based ciphers 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 $\mathbb{Z}_p^*$

9.2 Basic protocol

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

a and b must satisfy $0 < a, b < p - 1$.

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 solve the discrete logarithm problem, which is hard.

9.3 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 each other, 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.

9.4 static and ephemeral 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 $Y_a = g^{x_a} \mod p$. If each party has a long term key, they can simply look up each others keys and possibly skip the initial handshake.

10 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 encryption is $K_D = x$ with $y = g^x \mod p$

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

10.1 Why does it work?

The sender knows the ephemeral private key k. The receiver knows the static private key x. Both sender and recipient 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.

10.2 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.

11 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 cryptography will increase at higher security levels.

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

12 Identity-based cryptography

TODO
See lecture 9

13 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.

13.1 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.

13.2 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.

13.3 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 string.

The Merkle-Damgård construction 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 $h_0 = IV$

13.3.1 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 construction is used in standard and former standard hash functions (MD5, SHA-1, SHA-2)

13.4 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.

13.4.1 MDx family of hashes

Old and insecure by today's 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 birthday paradox and how many bits were recommended (sec. 13.1).

13.4.2 SHA-0 and SHA-1

Based off of MDx hashes (which makes it a Merkle-Damgård construction), 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.

13.4.3 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

	Hash size	Block Size	Security Match
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.

13.4.4 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.

13.5 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)

14 Authenticated encryption

14.1 Combining encryption and MAC

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.

14.2 Modes for Authenticated encryption

There are two types of input data:

- Payload data: both encrypted and authenticated
- Associated data: only authenticated

Both modes of operation use the CTR-mode to add confidentiality, but add integrity in different ways. They also allow some data to be only authenticated, not encrypted, providing `authenticated encryption with associated data` (AEAD). This property is important in newer versions of TLS.

14.2.1 Counter with CBC-MAC mode (CCM)

This mode offers confidentiality for the payload, and authentication for both payload and associated data.

The nonce, N, payload, P and associated data, A needs to be formatted to produce a set of blocks. The format is complex and different implementations exist in different standards. By blindly following the slides, we see that the lengths of N and P are included in the first block. If A is non-zero, it is formatted from the second block up to its length.

After formatting, we compute a T_{len} bits long tag for these blocks. Using normal counter mode, we compute m blocks, where $m = \lceil \frac{P_{len}}{128} \rceil$

Output $C = (P \oplus MSB_{plen}(S))||(T \oplus MSB_{tlen}(S_0))$,
where $S = S_1, ..., S_m$,
and $MSB_n(S)$ returns the n most significants bits from S.

14.2.2 Galois Counter mode (GCM)

A problem with CCM is that the formatting of N,A and P requires the length of both A and P. This prevents it from being used in a streaming application. GCM overcomes this limitation. It combines CTR mode on a block cipher, E (AES is a good and common choice), with a hash function called `GHASH`.

Skipping implementational details.

The receiver receives the ciphertext, C, the nonce, N, the tag, T, and the authenticated data, A. This is all the receiver needs to recompute T and check whether it matches the received T. If it does, the receiver can decrypt C just like in CTR-mode.

This is used in TLS 1.2.

15 Digital signatures

MACs provide data integrity and authentication (is the data tampered with? Is the person you're interacting with who he/she claims?). Generating a MAC tag requires the message, as well as the key. Digital signature provides all the same properties, as well as a few additions. It

is a technique that uses `public key` cryptography. Digital signatures provide non-repudiation, which is a legal concept. That means you can say your key was lost to a hacker and whatever malicious actions that key was involved with was not your work.

15.1 Elements of a digital signature scheme

1. Key generation
   - Outputs two keys, `signature generation key`, K_s, and `signature verification key`, K_v.
2. Signature generation
   - outputs a signature, s, given a message and a signing key.
3. Signature verification
   - outputs a boolean, given a message, a verification key and a signature.

15.1.1 signature generation algorithm

Only the owner of the signing key (signature generation key) should be able to generate a valid signature for any message.

Algorithm 2: Signature generation
Input : Message m, signing key K_s
Output: Signature s

15.1.2 signature verification algorithm

Anyone should be able to use the signature verification key to verify a signature.

Algorithm 3: Signature verification
Input : Message m, Verification Key K_v, Signature s
Output: boolean

The verifying function should always return true for matching signing/verification keys (correctness property).
It should be infeasible for anyone without K_s to construct m and s such that the verification returns true.

15.2 Security goals

digital signatures may be broken in different ways

- Key recovery:
  - Recovering the private key from the public key and some known signatures

- selective forgery:
  - The attacker chooses a message and attempts to obtain a signature on that message
- existential forgery:
  - The attacker attempts to forge a signature on any message not previously signed.

Modern digital signatures are only considered secure if they can resist existential forgery under a `chosen plaintext attack`.

15.3 RSA signatures

One way of generating digital signature keys is by using RSA. Just like in the encryption scheme, RSA signatures rely on the difficulty of factorizing primes.

The public verification key becomes (e,n), where $n = pq$. The signing key is d. RSA signatures also require a hash function, h, which should be a fixed and publicly known parameter of the scheme. The choice hash function is important, as some allow you to prove your security. A full domain hash function (can output all values between 1 to n) or `PSS` are good choices.

Signature generation takes the message m, the modulus n and the private exponent d as input. The signature s is computed as $h(m)^d \mod n$.

Signature verification takes the signature and the public key e as input. If $s^e \mod n = h\prime$ the signature is legit.

15.4 Elgamal signature scheme in $\mathbb{Z}_p^*$

Signature scheme based on the discrete logarithm problem. It consists of the following keys, given p, a large prime with generator g:

1. Private signing key, x, where $0 < x < p - 1$.
2. Public key, $y = g^x \mod p$, where y, p and g are public knowledge.

Signature generation:

1. Select random k such that $gcd(k, p - 1) = 1$ and compute
   $r = g^k \mod p$
2. Solve equation $s = k^{-1}(m - xr) \mod (p - 1)$
3. return (m, r, s)

Signature verification:

1. verify that $g^m \equiv y^r r^s(\mod p)$

RSA signature generation is fast, which is why it sees much use.

15.5 Digital signature algorithm (DSA)

Based on the Elgamal signature scheme, but with simpler calculations and shorter signatures. This is due to restricting calculations to a smaller group or to an elliptic curve group.

It is also designed to use with a SHA hash function. Avoids some attacks Elgamal signatures may be vulnerable to.

The prime, p, is chosen such that p - 1 has a prime divisor, q. The sizes of p and q are denoted L and P, respectively. Note how the size of q is much smaller than p. The generator, g, is replaced with the value $h^{\frac{p-1}{q}} \mod p$, where h is a generator of $\mathbb{Z}_p^*$. This implies that g has order q, which in turn means that all exponents in the algorithm can be reduced modulo q before exponentation – saving precious computation power.

Table 3: valid combinations of L and N in DSA. Some NIST publication does not approve the first choice of parameters.

L	N	to use with
1024 bits	160 bits	SHA-1
2048 bits	224 bits	SHA-224
2048 bits	256 bits	SHA-256
3072 bits	256 bits	SHA-256

Keys:

1. Private signing key, x, where $0 < x < q$.
2. Public key, $y = g^x \mod p$, where y, p and g are public knowledge.

Signature generation:

1. Select random k such that $0 < k < q$ and compute $r = g^k \mod q$
2. Solve equation $s = k^{-1}(H(m) - xr) \mod (q)$, where H is a SHA-family hash function that outputs N bits
3. return (m, r, s)

The returned signature is of size 2N bits.

Signature verification:

1. check that $0 < r < q$ and $0 < sq$

2. compute $w = s^{-1} \mod q$,
   $u_1 = H(m)w \mod q$
   $u_2 = rw \mod q$
3. verify that $(g^{u_1}y^{-u_2} \mod p) \mod q = r$

15.6 Elliptic curve DSA (ECDSA)

Elliptic curve parameters are chosen from a list of NIST approved curves.
Signature generation and verification is the same as in DSA with a few exceptions:

1. q becomes the order of the elliptic curve group.
2. multiplication mod p is replaced by the elliptic curve group operation $\cdot$.
3. after each operation on a group element, only the x-coordinate is kept.

Signatures generated from ECDSA is generally not shorter than DSA with the same security level. ECDSA signature sizes can vary form 326 to 1142 bits (while DSA signatures are often 448 or 512 bits).

ECDSA have shorter public keys.

16 Certificates and PKI

16.1 Public key infrastructure (PKI)

> A public key infrastructure is the key management environment for public key information of a public key cryptographic system

Key management includes generation of cryptographic keys as well as distribution, storage and destruction of these. Many entities spanning several disciplines may be involved, but our focus is technical.

16.2 Digital certificates

Certificates (or certs, for short) are what binds a public key to its owner. Without a cert, you can't **REALLY** be certain that the person on the other side is who he or she claims. This is achieved by having each cert signed by someone trusted by the certificate verifier, the certification authority (CA).

CAs create, issue and revoke certs for subscribers to that CA and other CAs. CAs have a certification practice statement (CPS) covering several issues like legal and privacy issues and checks they perform before a cert is issued.

The X.509 standard is the most widely used standard of digital certificates. Important fields included in this standard includes:

- Version number
- Serial number (set by CA)
- signature algorithm identifier (which algorithm is used for signatures)
- Issuer (name of CA)
- Subject (name of receiver)
- public key information
- validity period
- digital signature of the certificate (signed by CA)

Certificates are verified by checking that the CA signature is valid and that any conditions set in the cert are correct.

17 Key establishment

> TODO
> Skipping to TLS, as the summary lecture hinted to its importance.

18 TLS

Is one of the most widely used security protocol of today.

It is the successor of SSL, which is now advised against, and is still being developed - the newest version, TLS 1.3, was released in 2018. TLS is based on the use of a PKI and is often used to establish secure sessions with web servers, among other things. The design goal is to secure reliable end-to-end services over TCP, using TCP (although UDP-variants exist).

Even though TLS 1.3 is released, the most supported version is 1.2.

TLS consists of three higher level protocols:

1. TLS Handshake protocol to set up secure sessions
2. TLS alert protocol to signal events such as failures
3. TLS change cipher spec protocol to change the cryptographic algorithms in use

As well as the TLS record protocol which provides basic services to the higher level protocols.

18.1 Record protocol

The record protocol aims to provide two services for TLS connections, that we have discussed earlier in the course:

1. Message confidentiality:
    - message contents cannot be read in transit
2. message integrity:
    - detect tampering with messages

These services are provided by a symmetric encryption scheme and a MAC. Later TLS-versions (1.2+) also offer these services by using authenticated encryption modes (CCM and GCM, see sec. 14.2). Necessary keys are set up by the handshake protocol.

The record protocol format contains a header describing the type and length of the content, as well as a specification of which TLS version in use. Legal types include:

- change-cipher-spec
- alert
- handshake
- application-data

After the header comes an encrypted part containing the message, optionally compressed (compression removed in version 1.3, due to CRIME 2012 attack), and a MAC field in case authenticated encryption is not used.

The record protocol splits each application layer message in blocks of 2^{14} bytes or less.

18.1.1 Cryptographic algorithms in TLS

MAC:

The HMAC scheme is used to provide integrity in TLS, with an agreed-upon hash functions - SHA2 is only allowed in TLS 1.2+ - MD5 and SHA-1 not allowed in TLS 1.3

Encryption:

An agreed-upon block cipher in CBC-mode or stream cipher is used. Most commonly AES.

```
- RC4 and 3DES is supported in TLS 1.2
- Both removed from TLS 1.3 (in order to avoid a handful of attacks)
```

Authenticated encryption:

Using authenticated encryption is also allowed from TLS 1.2, replacing the MAC and encryption algorithm. In TLS 1.3, the only allowed configuration is AES in CCM or GCM modes. Authenticated additional data is the header and implicit record sequence number.

18.2 Handshake protocol

All above mentioned "agreed-upon" algorithms, as well as which version of TLS to use, is decided in the handshake. The handshake also establishes a shared session key to use in the record protocol, authenticates the server and finally completes session establishment. Some times it also authenticates the client, but it is not always required.

There are several variants:

- RSA
- Diffie-Hellmann
- pre-shared key
- mutual authentication or server-only authentication

TLS 1.3 simplifies this process (whew!).

The handshake is split into four phases:

- phase 1: initiate the connection and establish security capabilities (determine algorithms, versions etc)
- phases 2 and 2: Key exchange. How this is done depends on the outcome of phase 1.
- phase 4: finalize the setup of the secure connection.

18.2.1 Phase 1 - "hello"

Client and server negotiate TLS version, cipher suite and nonces to be used in compression and key exchange. This consists of two messages:

First a client "hello" which states the highest version of TLS available as well as which ciphersuites are available. Also sends client nonce, N_c.

Afterwards, the server responds with a "hello" that returns the servers' selection of version and ciphersuite from the list sent by the client. Also sends server nonce, N_s, to client.

18.2.2 Phase 2 - Server

Server sends its certificate (obtained by a CA) to client, as well as its input to the key exchange algorithm (server key exchange). If the negotiated scheme includes client authorization, request the clients' certificate.

18.2.3 Phase 3 - Client

Client reponds by sending its input to the key exchange algorithm (client key exchange). If requested by the server, also send own certificate. Verify server's cert with CAs public key. This

public key is assumed to be available, as its often shipped with your web browser etc.

18.2.4 Phase 4 - start of communications, summary of handshake

Finalizing the handshake, both the server and client switches to the ciphersuite negotiated earlier. Both also sends a checksum of the previous messages so both can verify it is correct.

18.2.5 TLS Ciphersuites

TLS ciphersuites specify which algorithms to use, both for key establishment as well as the later authenticated encryption and key generation. There are a literal fuck ton of suites to choose from, and many are bad. TLS 1.3 have removed a bunch, and requires all ciphersuites to be `AEAD` (see sec. 14.2)

An example of a ciphersuite is TLS_RSA_WITH_3DES_EDE_CBC_SHA. This unholy abomination means:

- The key exchange will use RSA for encryption
- 3DES in CBC mode will be used for encryption of secure communications
- SHA-1 will be used in HMAC

The first part denotes the handshake algorithm, the second part denotes (authenticated)encryption scheme and the third part denotes the means of achieving integrity (key deriving or hash function) ?

18.3 Ephemeral Diffie-Hellmann handshake

This is one of several variants of the TLS handshake protocol.

The server key exchange sends the generator, group parameters and the server ephemeral value to be used in Diffie-Hellmann. All of these values are signed by the server.

The client key exchange sends the client ephemeral Diffie Hellmann value, which might be signed depending on whether client certification is used or not.

Pre-master secret, *pms*, is the shared Diffie-Hellmann secret.

18.4 RSA handshake

The server key exchange is not a thing in RSA handshake.

The client key exchange sends the pre-master secret, *pms*, by selecting a random value for it and encrypting it with the server's public key. The server then retrieves *pms* by decrypting it using its own private key.

18.5 Other variants

(Static) Diffie-Hellmann can be used with certified keys. Should the client not have a cert (which often is the case browsing the internet), then the client uses an ephemeral Diffie-Hellmann key.

Another variant is the anonymous Diffie-Hellmann. Here the ephemeral keys are not signed, which means they are only protected against passive eavesdropping.

18.6 Generating session keys

To generate session keys, a master secret, ms, is needed. This is defined using the pre-master secret, pms, as:
$ms = PRF(pms, \text{"master secret"}, N_c||N_s)$

All "keying material", k, are generated from ms by using:
$k = PRF(ms, \text{"key expansion"}, N_s||N_c)$

Session keys are partitioned from k in each direction (write key || read key). Depending on ciphersuite the keying material, k, can be an encryption key, a MAC key, an IV etc.

The function `PRF` (pseudorandom function) used above is built from HMAC with a specified hash function. Older TLS versions (1.0, 1.1) used MD5 and SHA-1, but newer (1.2+) uses SHA-2.

18.7 Alert protocol

The alert protocol is there to allow signals to be sent between peers. These signals are mostly used to inform the peer about the cause of a protocol failure. Some of these signals are used internally by the protocol and the application protocol does not have to cope with them, and others refer to the application protocol solely. An alert signal includes a level indication which may be either fatal, close_notify or warning (under TLS1.3 all alerts are fatal). Fatal alerts always terminate the current connection, and prevent future re-negotiations using the current session ID.

The alert messages are protected by the record protocol, thus the information that is included does not leak. Improper handling of alert messages can be vulnerable to truncation attacks.

18.8 Forward secrecy

Forward secrecy is a property that describes whether leaking a long-term key fucks you up or not. If a leaked long term key (used for key generation) compromises session keys made before the long-term key was lost, you do not have forward secrecy. If only future session keys, made after the long-term key was leaked, are compromised then you have forward secrecy.

RSA-based handshakes in TLS does not provide forward secrecy.

Diffie-Hellmann handshakes and ciphersuites in TLS does provide forward secrecy.

18.9 Attacks on TLS

TLS 1.3 is the newest and safest version of TLS, which aimed to remove unnecessary/unsafe things and boost performance while retaining backwards compatibility. Sadly, TLS 1.3 isn't universally supported yet so unsafe features are still used (like unsafe ciphers).

18.9.1 BEAST

BEAST (Browser Exploit Against SSL/TLS) exploits non-standard use of IV in CBC mode (see sec. 4.4.2). This attack allows the attacker to retrieve the plaintext.

No longer considered a threat, as TLS 1.1 only allows random IVs and browsers implement migitation strategies against it.

18.9.2 CRIME and BREACH

These attacks target the fact that compression leaks information. CRIME targets optional compression in TLS and BREACH target compression in HTTP. Mitigated by not using compression, in fact TLS1.3 removed the option to have compression.

18.9.3 POODLE

A padding oracle is a way for an attacker to know if a message in a ciphertext was correctly padded. CBC mode encryption may act as a padding oracle due to its error propagation. This can be applied to attacks on TLS, and is mostly mitigated by having a uniform error response so the attacker doesn't know what kind of error occurred.

POODLE (Padding Oracle on Downgraded Legacy Encryption) forces downgrade to SSL3.0 (which is deprecated) during the handshake by stating it as the highest available version and then does a padding oracle attack.

18.9.4 Heartbleed

A bug that could be exploited, which is now fixed. Due to some missing boundary checks in the heartbeat messages that checks if the connection is still active, the server could leak memory that could contain session- and long-term keys.

18.9.5 Man-in-the-middle attacks

These attacks reply on issuing a new certificate and installing a root certificate in the browser. The ad/bloatware/malware company superfish had a scandal in 2015, called the "lenovo incident". Some lenovo computers were pre-installed with superfish software that included an unsafe, universal certificate authority signed by itself which would allow superfish to put ads on encrypted websites.

This backfired the hell out of this world and allowed everyone to eavesdrop and tamper with encrypted traffic from these computers.

18.10 TLS 1.3

Some changes from TLS 1.2 to 1.3 has been already mentioned. For your viewing pleasure, a more readable and complete list is presented:

Things removed:

- Static RSA key exchange
- Diffie-Hellmann key exchange
- session renegotiation
- SSL negotiation
- DSA
- Compression
- non-AEAD ciphersuites (sec. 14.2)
- MD5 hash
- SHA-224 hash
- change cipher spec protocol
- shittons of ciphersuites and encryption algorithms.
  - TLS 1.2 supported 319 suites
  - TLS 1.3 only allows 5
- PRF
  - TLS 1.3 introduced a new way of deriving keys called HKDF (HMAC-based Key Derivation Function)

Things changed/added:

- Handshake is cleaned up and more efficient
- ONLY AEAD ciphersuites
- added new cipher and mac algorithms
- separate key agreement and authentication algorithms in ciphersuites
- Encrypted messages in handshake

- 0-RTT mode (fast handshake given a pre-shared key, no forward secrecy yet. This is unfortunate)
- backwards compatibility (which adds *a lot* of complexity)