



## 2 Cryptography

Cryptography is one of the main building blocks available to the security engineer. In this section we describe the basics of *cryptography*, the design and building of ciphers, and *cryptanalysis*, the analysis and breaking of ciphers, together referred to as *cryptology*.

<p><b>Contents</b></p> <ul style="list-style-type: none"> <li>■ Cryptography goals</li> <li>■ Encryption principles</li> <li>■ Encryption quality</li> <li>■ Public key cryptography</li> </ul> <p>Next week:</p> <ul style="list-style-type: none"> <li>■ Example algorithms             <ul style="list-style-type: none"> <li>□ DES, AES, AES</li> </ul> </li> <li>■ Encrypting larger messages</li> <li>■ 'Provably secure' crypto</li> </ul>	<p><b>Cryptology:</b></p>  <p><b>Cryptography</b> The art of making</p>  <p><b>Cryptanalysis</b> The art of breaking</p>	<p><b>Security Goals and Cryptography</b></p> <ul style="list-style-type: none"> <li>■ Confidentiality</li> <li>■ Authenticity</li> <li>■ Data integrity</li> <li>■ Non-repudiation</li> <li>■ <i>Privacy</i></li> <li>■ <i>Availability</i></li> </ul>
---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Encrypting data aims to protect its confidentiality. There are also cryptographic techniques such as digital signatures which allow checking integrity. Authenticity and providing non-repudiation can also be achieved using encryption algorithms. Several privacy enhancing techniques, e.g. Direct Anonymous Attestation, use techniques similar to asymmetric cryptography, and indirectly, through confidentiality, cryptography can contribute to privacy. However, privacy is typically not a direct goal of cryptography; your data is under control of other entities which have control over the data. Privacy requirements restrict what they may do with the data, not the ability to get to the data. See also the lecture on Privacy and Anonymity (Chapter ??).

Clearly encryption has a negative impact on availability. Decryption aims to maintain availability in the presence of encryption.

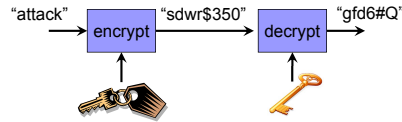
<p><b>Example: What's the message</b></p> <p>Greetings to all at Oxford. Many thanks for your letter and for the summer examination package all Entry forms and Fess Forms should be ready for final dispatch to the Syndicate by Friday 20<sup>th</sup> or at the very latest, I'm told, by the 21<sup>st</sup>. Admin has improved here, though there's room for improvement still; just give us all two or three more years and we'll really show you! Please don't let these wretched 16+ proposals destroy your basic A and O pattern. Certainly this sort of change, if implemented immediately would bring chaos.</p>	<p><b>Another Example: What's the message</b></p> <p>Welcome back to Oxford. Thanks again, this letter explains the winter examination method and its related forms. Early submission does guarantee full and early feedback but does not influence the grading of the quality of the work done. A full grade report will be available once the deadline for submissions has passed. In it the evaluation is explained. The evaluation is final as the criteria for the work are now known.</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

The text above contains a hidden message - try to discover it. It may seem to be well hidden; at least it is not directly obvious. However, once the method is known it will no longer work (it relies on obscurity of the encryption method); once you know how to find the message in the first example the second will be trivial. The final example below will likely also be quite easy to break.

## Final Example: What's the message

A final greeting to our Oxford graduates. Though with a slight delay, we hope this letter finds you well. The new variation in the forms attached shows how our alumni will continue to play a key role in our school and will not be forgotten. Instead we hope that you continue to work with us, and any contribution that you can bring, either directly or indirectly, will be appreciated.

## Algorithms + keys



Cipher (aka cryptosystem)

"Public" algorithm + Secret keys (Kerckhoffs' principle)

6

7

## 2.1 Symmetric Cryptography

Kerckhoff's principle tells us that the security of an encryption technique should not rely on secrecy of the algorithm used. Instead, keeping the key used in the encryption secret should be sufficient.

In the basic setup of a symmetric key cipher system there is an *encryption algorithm* and a *secret key* that is used by both. If the same key is used in the encryption and decryption then the original *plain text* should be recovered. Both the *cipher text*, i.e. the result of encrypting the plain text and decryption of the cipher text with an incorrect key should make no sense to an attacker.

The only difference between the legitimate user and the attacker is that the legitimate user knows the key. To ensure that the attacker cannot simply guess the key should be chosen randomly. In many symmetric key algorithms the key can be any bitstring of a given length and selecting a random string of this length is sufficient to generate a good key. However, for some ciphers, especially asymmetric ones, the keys have some structure and generating a good random key is more complex; a cipher should then also come with a probabilistic *key generation algorithm*.

We can now formalize "makes no sense" as having no correlation to the plain text at all. In this case, the security is *unconditional*; no matter the computational resources of an attacker, the attacker cannot learn anything new about the plain text from the cipher text (or in other words - our attacker model assumes no limits on the computational resources the attacker has available.)

**Definition 1 (Unconditionally Secure)** We say a (symmetric) cipher with encryption *Enc* is *unconditionally or information theoretically secure* if encryption with a random key gives a cipher text that is not correlated to the plain text.

Formally, consider random variable  $\mathbf{k}$  (the randomly selected key) we require that for any cipher text  $c$  and potential plain texts  $p_1, p_2$ :

$$\mathbf{P}(c = Enc_{\mathbf{k}}(p_1)) = \mathbf{P}(c = Enc_{\mathbf{k}}(p_2))$$

i.e. if one does not know the key each plain text is equally likely to have been the one that was encrypted.

An alternative, equivalent, definition is that for any distribution over plain texts (the attacker's a-priori estimate of how likely a given plain text is) the attacker learns nothing new by obtaining the cipher text; the conditional probability of plain texts given an encryption with a random key is the same as in the original distribution over plain text.

To summarize; the term 'unconditionally secure' cipher refers to the security goal 'confidentiality of the plain text' with security policy 'those who do not know the randomly generated key may learn anything about the plain text' and the attacker is one with unlimited computational resources, knows (Kerckhoff's principle) the cipher including the probability distribution of the

keys (i.e. knows the key generation algorithm) but does not know the actual key (outcomes of the random choices in the key generation algorithm).

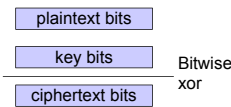
### When is message 'safe'?

- Suggestion 1: 'cannot know the message'.
  - Kill the king with a @#\$%~!.
- Suggestion 2: 'cannot know even a single bit'.
  - 99% chance "Kill the king", 1% "Drink coffee"...
- ... lets find a definition...
- For ciphertext each plaintext equally likely
  - Can this be done?

8

### Yes(\*)!: One time pad

- Vernam's one time pad is **information theoretically secure**



Note: random key equally long as message

9

Unconditional security imposes a very strong requirement on the cipher. Is it actually possible to create a cipher which achieves this? Assuming the number of possible plain texts is finite (see also Exercises) this is indeed possible and achieved by a surprisingly simple system; the one-time pad.

Though simple and easy to understand this system is rather impractical in use. It uses a bitwise exclusive or (XOR,  $\oplus$ ) with a key which is as long as the plain text and which can only be used for a single plain text (see item (c) of Exercise 6). (Decryption is done by repeating this same operation.) For communication this means that we first need to safely share a key which is equally long as the message that we want to send. But if we have a secure way of sending a key then why not use this for the message directly? Thus the use is limited to cases we can share a key in a way we cannot share the message itself - e.g. meet to share a key in advance, and later use this to enable communication over an insecure channel or share keys over a channel on which eavesdropping can be detected (as in quantum key distribution - see the course 'Physical aspects of digital security' (2IC35) for more on security meets quantum theory).

Similarly if we use the one time pad for securely storing (rather than sending) some data we would need to securely store a key of equal length. Here the only remaining benefit is that an attacker would have to obtain both the key and encrypted data, which we could try to make difficult e.g. by storing them in different places. (This is a form of secret sharing, see Section ?? for more on this.)

So the one time pad is secure<sup>1</sup> but needs impractically long keys. Could we not think of a more efficient system which achieves the same level of security? Unfortunately such a system cannot exist (see Exercises). Thus people have been trying to create practical encryption systems which are 'good enough'.

**Substitution Ciphers** The Caesar cipher, which as its name suggests was already used in Roman times, is a simple substitution cipher. Given a plain text we simply shift each letter by a fixed number of places (e.g. *A* becomes *D*, *B* becomes *E* etc.) to obtain the cipher text. To decrypt we shift back by the same number of places.

plain text letters	H	E	L	L	O	...
key letter (repeats)	C	C	C	C	C	...
cipher text letters	K	H	O	O	R	...

Figure 2.1: Caesar cipher with key = C (+3)

It is clear that this cipher does not offer much security; there are only 26 possible key values (of which one is not really useful; consider which and why) so we can only hope for  $\log_2 26$  (i.e. less

<sup>1</sup>What did that mean again in this setting ...

than 5) bits of security at best. However, this cipher can be broken even more easily. The patterns in the text remain unaltered (e.g. in figure 2.1 the double L becomes double O) allowing knowledge of text patterns to be used to easily find the key. E.g. by looking at the frequency of the letters one can conclude which letter is most likely encryption of the letter E which also directly gives the key used.

The Caesar cipher uses a very linear operation to encode letters; a simple rotation in which each letter moves by the same amount. If instead as substitution we would use an arbitrary permutation of the letters, where e.g. *A* could be encoded as *E* but *B* as *Z* etc., the number of possible keys grows to  $26!$  which corresponds to 88 bits. Yet breaking such a cipher can still be done, even by hand, by looking at patterns in the text rather than trying all possible keys (there is actually a type of puzzle based on this idea; in a filled in crossword each letter is replaced by a number and the goal is to find the corresponding mapping between numbers and letters.)

### Some History: the Caesar cipher

- Monoalphabetic substitution
- Replace letter by letter 3 places further

Plaintext	A	B	C	D	E	F	G	H	...
Ciphertext	D	E	F	G	H	I	J	K	...

- Example:  
"attackatdawn" → "dwwdfndwgdzq"

- Letter frequency undisturbed
- Nr of keys: 26 (25)

A=1, B=2, C=3, ...  
 Encrypt: C = P+3  
 Decrypt: P = C-3

### Vigenere cipher

- Polyalphabetic substitution
- Key is keyword
- Encrypt: Add keyword (letter by letter)  
□ Modulo 26 with A=0, B=1, etc.
- Decrypt: Subtract keyword
- Example

wearediscoveredsaveyourself  
 +  
 deceptive  
 ZICVTWQNGRZGVTWAVZHCQYGLMGJ

Thus substitution of a single letter at a time is insufficient. What if we increase the block size, i.e. always encrypt a number of letters at the same time. If we apply this principle to the Caesar cipher and thus use a word instead of a letter as the key we obtain the Vigenere cipher. (The Caesar cipher is a Vigenere cipher with a blocksize 1).

plain text letters	H	E	L	L	O	...
key word (repeats)	B	Y	E	B	Y	...
cipher text letters	J	D	Q	N	N	...

Figure 2.2: Vigenere cipher with key=BYE

With a block size of  $n$  letters the size of the key space becomes  $26^n$ . However, again an attacker can mount a more effective attack than just trying all possible keys; first the attacker guesses the length of the key  $n$ . This guess can be verified e.g. by checking that the frequencies of each  $n$ -th letter show the same pattern as the frequencies normally occurring in a text or simply by the fact that the next step succeeds. Finding the key then an easy exercise of attacking a Caesar cipher  $n$  times.

So simply increasing the block size does not by itself solve the problem. The high amount of linearity in the encoding allows recognition of patterns. As with single letters we could try using substitutions that are arbitrary permutation of words of length  $n$ , yielding  $26^n!$  possible keys (which for  $n = 2$  already corresponds to more than 5,000 bits). But how would one store such a key? Typically a substitution would be stored as a table which, for a single letter would already be 26 by 26 in size, and for words of size  $n$  would give an  $26^n$  by  $26^n$  table, which quickly becomes unrealistic. A second problem that remains is patterns which are bigger than the block size (e.g. repeating words) may still allow analysis of the text.

### Cryptanalysis – plaintext structure

- (English) Text
  - Distribution of characters known
  - Distribution of bi-graphs also known:
- Data
  - Format known

E: 12%  
 T: 9%  
 A,I,N,O,R: 8%

TH: 3.2%  
 HE: 3.1%  
 ER: 2.1%

<account>87539</account>  
 <amount>1234</amount>

15

### Transposition cipher

- Change order of letters in the message

"meet me after the toga party"

M e m a t r h t g p r y  
e t e f e t e o a a t

"mematrhgtpryeteefeteoaat"

16

**Transposition Ciphers** Instead of changing the letters in a text we could also try to hide the text meaning by changing the order of the letters, i.e. transposing the text; applying a permutation to the text. This in itself, however, is easy to detect: the frequency of letters does not change so it may be easy to recognize. Also, if a fixed permutation is used there is no key so once the method is known the security is lost and if a random permutation is used we have the problem of how to effectively store what permutation we are using.

Still by mixing substitutions (confusion) with permutations (diffusion) one can try to simulate an encryption which is a large random permutation of all plain text. Confusion is done by mixing with a key and non-linear s-boxes which are tables describing substitutions. The confusion is done on relatively small sub-blocks so the s-boxes remain sufficiently small. The diffusion is then used to mix and combine the different sub-blocks. This process is then repeated several times to ensure the end result is as close as possible to completely random (or at least: each output bit depends on each input bit and in a non-linear manner). This is the basic idea of modern substitution-permutation ciphers. Below we will see two examples of this in some more detail; the Data Encryption Standard and its successor, the Advanced Encryption Standard. First we will address the basics of asymmetric or public key cryptography.

### Modern Block Cipher

- Principle: Combine
  - Confusion (substitution)
  - Diffusion (transposition)
- Design: Iterate a round function
- Two common types:
  - Feistel network (e.g. DES)
  - Substitution-permutation network (e.g. AES)

n bit plaintext block → encrypt → n bit ciphertext block → decrypt → n bit plaintext block

More on this next week – Now first: asymmetric (public key) cryptography

17

### Many symmetric keys needed

Bob, Carol, ..., Zeke

Alice

To send to Alice, everyone needs a **different key**

To receive, Alice needs all these keys

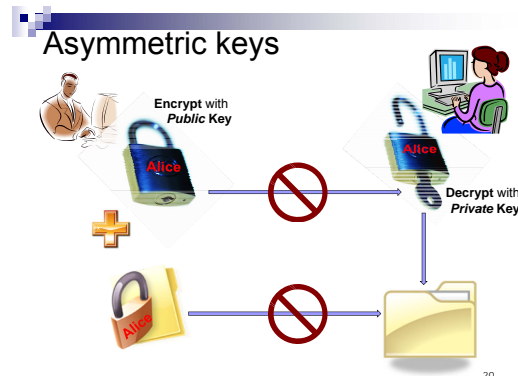
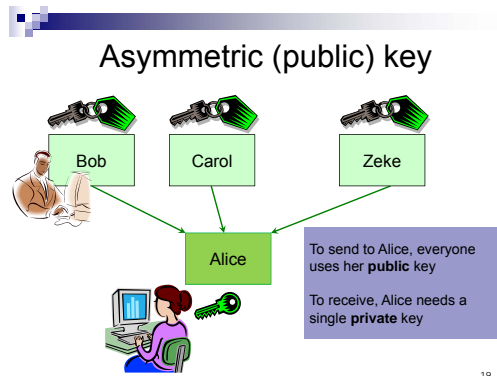
18

## 2.2 Public Key Cryptography

Suppose that Alice wants to be able to securely communicate with a lot of different people without any of them being able to eavesdrop on communications with the others. She could use the symmetric cryptography techniques we discussed above to achieve this. However, Alice will need to share a *different* key with each of these people and will have to store all of these keys securely.<sup>2</sup>

<sup>2</sup>You are of course thinking: what do we mean by this here?

A solution to this problem is the use of asymmetric cryptography. An asymmetric system uses different keys to encrypt and decrypt data. Using such a system Alice could have a *public key* that she could share with everyone and that other can use to encrypt messages meant for Alice so only she can decrypt it with her *private key*. (Hence asymmetric cryptography is also referred to as public key cryptography.) If Alice wants to send a message herself, e.g. to Bob, she uses his public key to encrypt it.



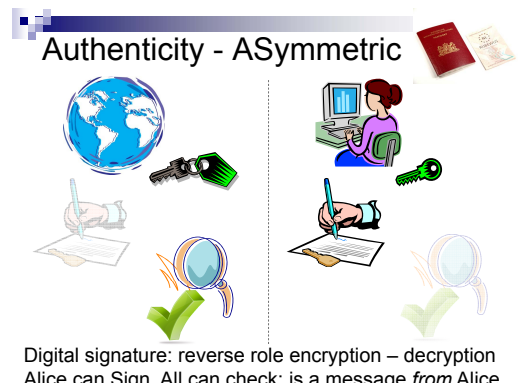
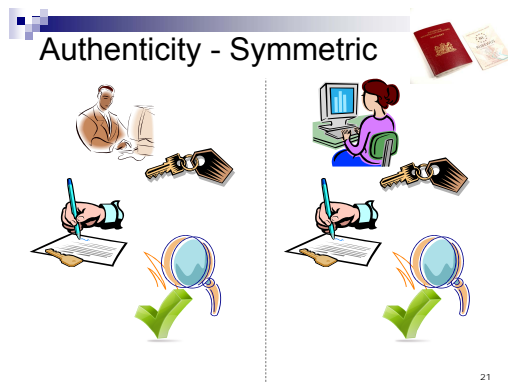
One can compare a public key system to a padlock; anyone who has the padlock (public key) can lock it (encrypt a message) but only the one with the (private) key can unlock (decrypt) it. A symmetric system on the other hand is like a lockbox where you need the (same) key both to put things in and take things out.

So how does this help; Alice still needs a different (public) key of everyone she wants to send messages to in addition to her private key? The advantage is that while in a symmetric system she has to keep all keys confidential with the public key system this is only needed for her own private key. (In both systems we will need to protect the integrity of all keys.)

For the public key approach above to work we need to satisfy the following:

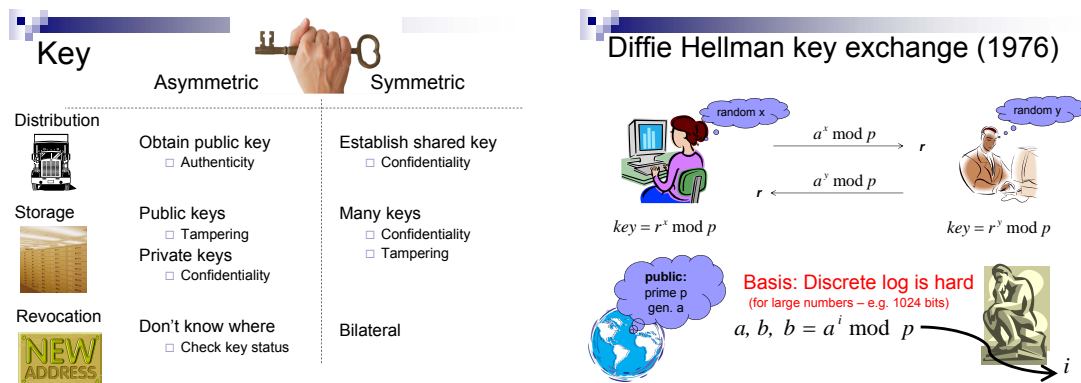
- It should not be possible to derive information about encrypted messages or the secret key from the public key.
- Alice needs to be sure the 'public key of Bob' really belongs to Bob.

The need for the first property is quite obvious but less obvious is how we can achieve this; not only do we have to keep information secret but we have to keep it secret from somebody who has 'half of the answer' (the public key). The second requirement is related to key management; though Alice will not need to keep the public key of Bob secret, she does need to make sure it is correct and not the key of say Mallory.



The discussion so far focusses on achieving confidentiality of messages. Lets, for a moment, consider the other security attributes that we may want to achieve. Suppose we wish to achieve authenticity (and with it integrity) rather than confidentiality. With a symmetric key system Alice knows that if she decrypts a message and it makes sense, then it must have been created by Bob (or by herself); no one else has the key so no one can make cipher texts that decrypt to meaningful plain texts. With public key cryptography we can also achieve authenticity but we have to 'turn' the public key cryptography schema around; Alice *signs* a message with her private key and everybody is able to check this *digital signature* using her public key. Often this 'turning' can be achieved by simply using decryption for signing and encryption for checking a signature.

Note that we could also use digital signatures to protect the integrity of public keys. For example, if Alice signs 'Bobs key is 2ef8d83441e...' and stores this she can then tell if someone has tampered with the stored keys as the signature will no longer be correct. Changing both the stored key and the signature is not an option for an attacker as only Alice can make valid signatures. This is an example of a *certificate* which we will treat in more detail in a later lecture (see Chapter ??). Here we first give some idea of the mathematics that make public key schemas possible by looking at the Diffie Hellman key exchange protocol.



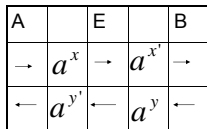
**Diffie-Hellman Key Exchange** Consider the following setup: Alice and Bob want to create a shared key, e.g. to be able to use a symmetric cipher, but are worried that someone may overhear their communication. Diffie-Hellman Key Exchange could be used to solve this. In this system two public parameters are set: a large prime number  $p$  and a *generator*  $a < p$ . (For the multiplicative group modulo  $p$ . Recall:  $a$  is a generator means that any element of the group  $(1, \dots, p - 1)$  can be expressed as a power of  $a$ ) Alice and Bob both generate a random number,  $x$  and  $y$  respectively, raise  $a$  to the power of this random number and send this to the other. The number  $r$  they receive they again raise to the power of their random number and the result is the shared secret key. Alice and Bob share the same key as Alice uses  $r_b^x \bmod p = (a^y \bmod p)^x \bmod p = a^{yx} \bmod p$  and Bob uses  $r_a^y \bmod p = (a^x \bmod p)^y \bmod p = a^{xy} \bmod p$ .

An eavesdropper could obtain  $r_a$  and  $r_b$ , however to make the key e.g. from  $r_a$  one would need  $y$ . The attacker could try to obtain  $y$  from  $r_b$  because  $r_b = a^y \bmod p$  in which only  $y$  is unknown. However, this is an instance of solving a *discrete logarithm* problem in a finite group for which no efficient algorithms are known. Thus if we use a large prime number  $p$  an eavesdropper is highly unlikely to obtain the shared secret key.

Note that the attacker model is very important for the security of this scheme. For example, the scheme is not secure against a *man in the middle* attack. In the man in the middle attacker model the attacker is able to not only listen to messages but actually intercept and change them. If Eve can alter messages sent by Alice and Bob she could replace  $a^x$  and  $a^y$  by  $a^{x'}$  and  $a^{y'}$  after which both Alice and Bob share keys with Eve while thinking they share a key with each other.

## DH - Soundness and Security

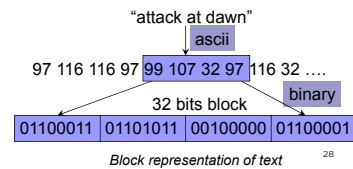
- Alice key equals  
 $r^x \text{ mod } p = (a^x \text{ mod } p)^r \text{ mod } p = a^{xr} \text{ mod } p$
- Bobs key equals  
 $r^y \text{ mod } p = (a^y \text{ mod } p)^r \text{ mod } p = a^{yr} \text{ mod } p$
- Eavesdropper sees  
 $a^x \text{ mod } p$   
 $a^y \text{ mod } p$
- Vulnerable to man-in-the-middle attack



26

## Encrypting Larger messages

- Seen methods to encrypt block
- Split into blocks (padding to fill last block)
- Treat blocks separately?



28

## 2.3 Block modes: Encrypting more than one block

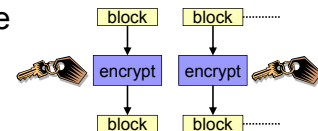
Block ciphers take a fixed size (64 bits, 128 and 256 bits are common sizes) block to encrypt. Asymmetric cryptography typically takes an element of a group of size  $n$ , thus encoding at most  $\log_2(n)$  bits (e.g. 256, 512, 1.024 and 2.048 are common sizes for  $n$  in bits). Thus though different algorithms and settings for these algorithms lead to different size blocks that we can encode, to encode large messages, we will need to split the message into multiple blocks.

### Encrypting larger messages

- Operation modes
  - Electronic codebook (ECB)
  - Cipher Block Chaining (CBC)
  - Cipher Feedback (CFB)
  - Output Feedback (OFB)

32

### ECB mode

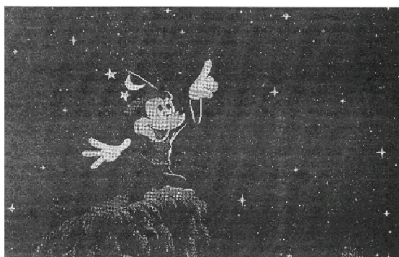


- Same plaintext block maps to same ciphertext block
  - Reordering, replacing possible
- No error propagation
  - Bit changes only
  - Bit deletions/omissions are a problem

29

We can divide our message up into blocks of the correct size and simply treat each block separately. This approach, called Electronic Code Book (ECB) mode, has several drawbacks. For example; the same block will encrypt to the same cipher text every time, thus patterns in the original text will still be visible in the encrypted version of the text. (This is the same problem as for the Caesar cipher but now with respect to whole blocks instead of letters).

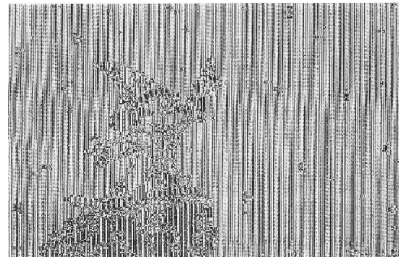
### Example: Mickey Mouse



■ Original picture

30

### Example: Mickey Mouse

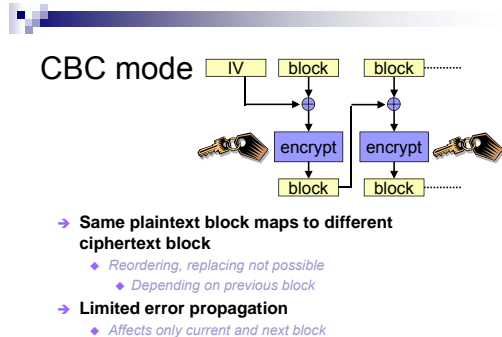


■ Encrypted in ECB mode

31

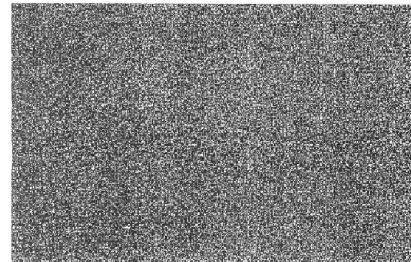


The effect is evident when encrypting an image with a symmetric cipher in ECB mode. If we simply treat the encrypted data as an image it is garbled but the original image can still be recognized. A clear case of 'Micky Mouse encryption' one could say.



33

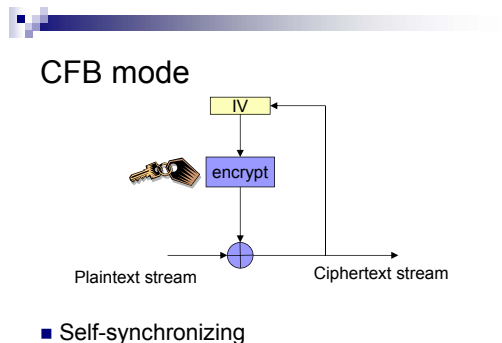
**Example: Mickey Mouse**



■ Encrypted in CBC mode

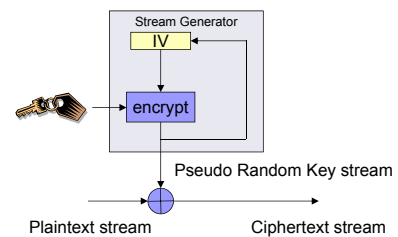
35

Instead of treating blocks separately, one could combine a block to be encrypted with the previous blocks (and/or their encryption) in some way. Then the encryption of the block becomes dependent on the previous block(s) and any structure in the plain text is hidden. One way to do this is by using Cipher Block Chaining (CBC) mode. In this mode the first XORs the previous encrypted block with the plain text before encrypting it. For the first block, where no previous encrypted block is available, an 'initialization vector' is used. With CBC the data that is encrypted is basically random, ensuring no structure remains that could be recognized in the cipher text blocks. If we encrypt the same picture as before with CBC mode no discernable pattern remains.



36

**Stream Ciphers and OFB mode**



37

Two other modes are Cipher Feedback (CFB) and Output Feedback (OFB) mode. Output feedback mode turns a block cipher into a *stream cipher*. In a stream cipher a pseudo random stream of key bits is generated from the shared symmetric key and this key stream is XORed with the plain text to obtain the cipher text. Note the similarity with the one-time pad; if the key stream is 'sufficiently random' then so is the cipher text. Also, like with the one-time pad, the same key(stream) should not be reused. With OFB mode, an initialization vector is encrypted to give the first block of key bits which then also replaces the initialization vector and is re-encrypted to form the next block of key bits, etc. (Using a different initialization vector enables one to generate multiple streams from the same key in OFB mode.) CFB mode is a slight variation of CBC which uses the structure resembling a stream cipher. In CBC an initialization vector is encrypted to get the first block of random bits and the resulting cipher text is then re-encrypted, thus re-seeding the pseudo random number stream, but here with data which also depends on the plain text.

## 2.4 Example Algorithms

To get a better idea of the inner workings of algorithms we now look at some common algorithms in somewhat more detail.

### 2.4.1 Data Encryption Standard (DES)

#### DES

- Data Encryption Standard
  - published by NIST as FIPS PUB 46 in 1977
- Based on Lucifer by IBM
- NSA changed the design
  - Fear of weaknesses
- Used extensively by banks
  - E.g. ATM
- With whitening in Win2K encrypted FS
- Becoming less common (move towards AES)

#### DES properties

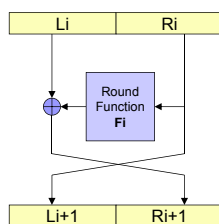
- Block size 64 bit
- Key size 64 bit
  - 56 bit real key data
  - Remaining 8 bits are parity bits
- 16 rounds Feistel network
- Complement property:
  - $E(k, x^c) = E(kc, x)^c$

43

44

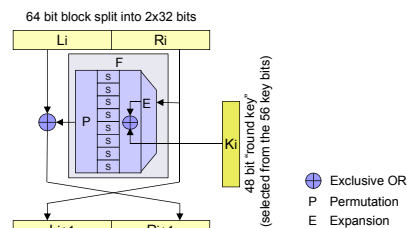
The Data Encryption Standard (DES) was adopted by the US National Bureau of Standards (NBS), now named National Institute of Standards and Technology (NIST), in 1976. The key used by DES is effectively 56 bits long. (It specifies 64 bit keys but 8 of these are parity bits.) DES uses a 16 rounds *Feistel network* structure. In this structure a round consists of splitting input into two parts, applying a function  $F$  to the right hand side and XOR-ing the result with the left hand side. This gives the new left hand side. The new right hand side is the old left hand side. A simple key schedule determines which of the 56 bits of the key form the 48 bits of the round key for each round. The input ( $R_i$ ) is extended (by repeating some bits) to 48 bits and the result XOR-ed with the round key. The eight different DES S-boxes, each taking 6-bits of input and producing 4 bits of output are then applied yielding 32 bits of output. The final step in computing  $F$  is permuting these 32 bits.

#### One Feistel round



42

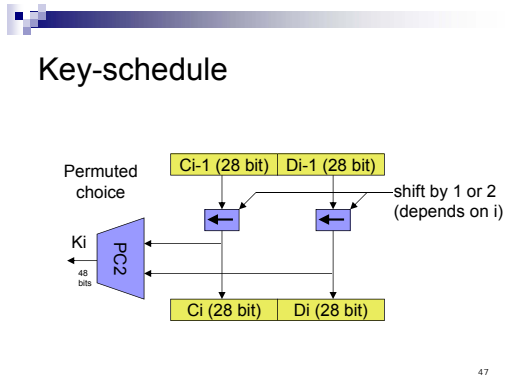
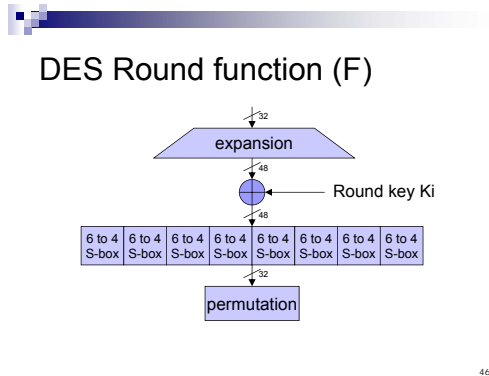
#### One Feistel round



45

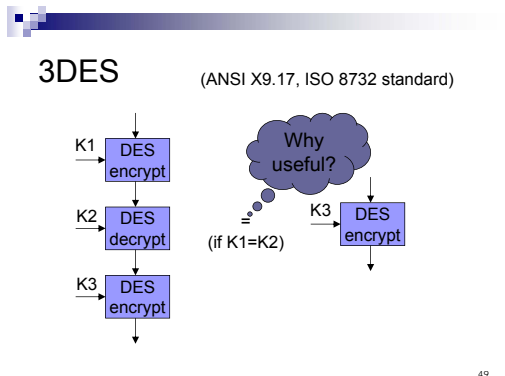
One of the advantages of the Feistel network structure is that the Decryption is very similar to the Encryption: undoing a single round is exactly the same just with left and right parts swapped. (Check this for yourself; apply a round with key  $K_i$  and input  $R_{i+1}, L_{i+1}$  to see it gives as output  $R_i, L_i$ .) Note that, in particular, the function  $F$  does not have to be invertible. The DES S-boxes for one are clearly not invertible as they have less bits of output than input.

Fear that the influence of NSA on the design introduced specific weaknesses that they could exploit was later countered by the relatively good resistance of the cipher to 'differential cryptanalysis', a techniques published in the late 1980s but apparently already known to the NSA and the IBM designers at the time DES was made.



A main issue with DES is its limited key size. While, when initially published it was considered that a machine that would break the cipher in a reasonable amount of time would be unrealistically expensive, already in 1977 a theoretical 20M dollar machine which could break keys in a day was described [?]. In 1998 the Electronic Frontier Foundation (EFF) actually built Deep Crack for less than \$250,000 which can crack a key in a few days [?]. Currently 'brute-force' techniques for cryptanalysis not only benefit from individually faster machines but also from massive parallelism such as distributed computing and the use of graphic cards.

- ### DES: discussion
- Extensively studied
    - No severe weaknesses found
  - However, 56 bit key too short
    - 3DES
    - AES as new standard



In short DES cannot be considered secure<sup>3</sup> anymore. As a solution to the short key triple DES was introduced. As the name suggests triple DES uses three DES operations with a set of three 56 bit keys; two DES encryptions with a DES decryption in between. By choosing the first two keys equal the decryption cancels out the first encryption and the end result is the same as doing a single DES encryption with the third key. In this way we get backward compatibility with single DES; triple DES encryption hardware can also be used to do DES encryptions.

By choosing all three keys different we get an encryption scheme with a 168 bit key. It is also possible to take the first key and second key different but the third key the same as the first key. This gives an encryption scheme with a 112 bit key. Of course the key length along does not tell use how much security a cipher really offers. Estimates of security offered against the best known attacks are 112 bits for the first and 80 bits for the second scheme. (The detailed cryptanalysis of this method are beyond the scope of this course.)

<sup>3</sup>(remember to consider "what does secure mean in this context...")

## 2.4.2 Advanced Encryption Standard (AES)

A new standard was introduced in 2001: the Advanced Encryption Standard (AES) based on the Rijndael cipher. This cipher was chosen from amongst several competing encryption schemes (the 'AES candidates'). AES uses a 128, 192 or 256 bit key to encrypt 128bit blocks. Here we will focus on the 128 bits version of the algorithm.

The input to AES is ordered into a 4 x 4 matrix of bytes and bytes are seen as elements of finite field  $GF(2^8)$ . The following four basic operations are used in AES:

**AddRoundKey** which XORs the state with the round key. The round key is also represented by a 4x4 byte matrix and XOR-ed with the state. The round key is derived from the main key according to a key schedule that we will not treat here.

**SubBytes** which applies an S-box substitution on each byte of the state. The S-box that is used is given below (in hexadecimal format) Note that unlike the DES S-box the AES S-box is invertible. Also, the AES S-box can be expressed as combination of several functions, allowing it to be computed rather than stored.

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

**ShiftRows** which rotates each row a number of steps. Row  $n$  is rotated  $n - 1$  places to the left, i.e. the first row is not changed, the second row is rotated 1 place to the left etc. As an effect, any column after the shift depends on every column before the shift.

**MixColumn** which combines the four bytes in each column to a new column by multiplying with the following matrix:

$$\begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix}$$

Within  $GF(2^8)$  addition and multiplication work as follows: Addition is XOR. Multiplication by 2 is a shift 1-bit to the left. If the most significant bit is 1 this results in a result larger than 8 bits so the modulus operation in  $GF(2^8)$  is applied which corresponds to doing an XOR with 1B (hexadecimal). (Note that the result fits in 8 bits again.) Mixcolumns can be inverted by multiplying with a different matrix.

The basic function are combined into ten rounds as follows: In the first round a single AddRound-Key is performed. Rounds two to nine perform the sequence SubBytes, ShiftRows, MixColumns, AddRoundKey and the final round leaves out MixColumns, thus performing SubBytes, ShiftRows, AddRoundKey.

Each AES operation is invertable. Decryption simply inverts each encryption step in turn.

## 2.4.3 RSA

### RSA

- By Rivest, Shamir and Adleman in 1978
- First “public” public key system
- Most popular
- Patent expired September 2000
- Large keys (1024 bits or more)

50

### RSA preliminaries

- Euler Totient Function  $\phi$
- $\phi(n) = \# \{ i \mid i < n, i \text{ relatively prime with } n \}$
- $\phi(p * q) = (p - 1)(q - 1)$  for  $p, q$  prime
- $a^{\phi(n)} \bmod n = 1$ 
  - If  $a, n$  relatively prime
  - For  $n=p*q$  also without  $a, n$  relatively prime
- Inverse modulo  $n$  ‘easy’ to find.

51

In 1978 Rivest, Shamir and Adleman were the first to published a public key system. Their RSA system works as follows:

### RSA Key generation

- Pick two large primes  $p, q$  and set  $n = p * q$ 
  - $p \neq q$
- Pick  $e, d$  such that
  - $ed = 1 \bmod \phi(n)$
  - i.e.  $ed = 1 \bmod (p-1)(q-1)$
- Destroy  $p, q$
- Public key:  $(e, n)$
- Private key:  $(d, n)$

52

### RSA Key generation Example

- Choose  $p, q$ :  $p=7$  and  $q=17$
- Gives  $n=119$  and  $\phi(n) = 6 * 16 = 96$
- Pick  $e$  relatively prime with 96, e.g.  $e=5$
- Compute  $d$  with  $ed = 1 \bmod 96$ .
  - Result:  $d=77$
  - Verify:  $77 * 5 = 385 = 4 * 96 + 1$
- Public key:  $(5, 96)$  Private key  $(77, 96)$

56

*Setup and key generation:* Randomly choose two large primes  $p, q$  and compute modulus  $n = p * q$ . Pick an  $e$  and  $d$  such that  $ed = 1 \bmod \phi(n)$ . (knowing  $p$  and  $q$  one can compute  $d$  for a given  $e$  e.g. by using the extended Euclidean algorithm). The public key is  $(e, n)$  and the private key  $(d, n)$ . I.e.  $e$  and modulus  $n$  are made public and keep  $d$  is kept secret.

*Encryption* A plain text can be any number less than  $n$ . To encrypt a plain text  $P$  raise it to the power  $e$  (modulo  $n$ ),  $c = P^e \bmod n$ .

*Decryption* To decrypt a cipher text  $C$  raise it to the power  $d$  (modulo  $n$ ),  $P = C^d \bmod n$ .

### RSA Encryption, Decryption

- Encrypt  $P$ :  $C = P^e \bmod n$
  - Decrypt  $C$ :  $P = C^d \bmod n$
- Public key:  $(e, n)$   
 Private key:  $(d, n)$
- Why it works:
- $$C^d \bmod n = (P^e \bmod n)^d \bmod n = P^{ed} \bmod n$$
- $$[ed = 1 \bmod \phi(n)] = P \times P^{\phi(n)k} \bmod n$$
- $$[P^{\phi(n)} \bmod n = 1] = P$$

53

### RSA Encrypt/Decrypt Example

- Public key:  $(5, 96)$
- Encrypting  $P=19$ :
  - $19^5 = 2476099 = 20807 * 119 + 66$
  - Ciphertext is 66
- Private key  $(77, 96)$
- Decrypting 66
  - $66^{77} = 19 \bmod 96$

57

The way of choosing  $e$  and  $d$  guarantees that  $a^{ed} = a \bmod n$  for any  $a$  (see Exercise ??). Thus

decryption works because  $C^d = P^{ed} \bmod n = P \bmod n$ . The public information ( $e$  and  $n$ ) is not sufficient to find  $d$ ,  $p$ ,  $q$  or  $\phi(n)$  and this is essential as knowing any of these would allow (finding  $d$  and) decrypting messages.

### RSA: choices & requirements

- $e = 3$ ,  $e = 7$  or  $e = 65537 (= 2^{16} + 1)$ 
  - Salt append random bits (e.g. 64) to plaintext
  - Otherwise attacks exist to find private key and
  - encryption small  $m$  less than  $n$ ; easily recovered
- All users must pick distinct modulus  $n$ 
  - Any  $e, d$  with  $ed = 1 \bmod \phi(n)$  allows factoring  $n$
  - Easy to compute any  $d'$  from  $e'$

60

### RSA: choices & requirements (2)

- $d$  roughly the same size as  $n$ 
  - Otherwise it can be found efficiently from  $e$  and  $n$
- factoring  $n$  must be hard
  - $p, q$  sufficiently big
  - $p, q$  roughly the same size
  - still  $p - q$  sufficiently large

61

The large prime numbers  $p$  and  $q$  are randomly chosen; an attacker should not be able to guess them. They should be sufficiently large as should their difference ( $|p - q|$ ) to ensure factoring is  $n$  is actually difficult. All users should have their own, distinct modulus  $n$ . The public key  $e$  is typically chosen to be convenient value such as 3, 7 or  $2^{16} + 1$  as this allows for efficient encryption (and as we will give away this value anyway there is no need for it to be random). The privacy key  $d$  on the other hand should be big (close is size to  $n$ ) to prevent it being guessed or derived efficiently from  $e$  and  $n$ .

### RSA: Setup and Security

- Given  $p, q$ , it is easy to find  $e, d$  such that
 
$$ed = 1 \bmod \phi(n) = 1 \bmod (p-1)(q-1)$$
- Without  $p, q$ 
  - computing  $\phi(n)$  is hard
  - finding  $d$  given  $e$  as hard as finding  $p, q$
  - finding private key as hard as factoring

58

### RSA Special properties

- $E(m * m') = E(m) * E(m') \bmod n$ 
  - Add redundancy to sign messages
- Blinding with a random  $r$ 

$$E(mr^e) = (mr^e)^d \bmod n = m^d r \bmod n$$
  - Hide message from signer
    - Application: Anonymous money
- RSA can be used to sign or encrypt
  - signing = decrypting
  - use separate key pairs

59

As we are working modulo  $n$  there are up to  $n$  possible values. Thus if we work in block as with symmetric ciphers, the maximum block size with RSA is  $\log_2(n)$ . There are several ways to represent blocks as numbers in  $0, \dots, n$ . Typically some form of padding is used, for example by forming the most significant bits from random *salt* creating a randomized encryption. The padding ensures that the numbers are not too small which would give several problems: if plain text  $P$  is a small number then  $P^e$  may be smaller than  $n$  (recall that  $e$  may be a small) and thus we are doing normal integer arithmetic where taking an  $e$ th-root is easy, instead of modular calculations where such an operation is hard.

Note that for RSA encrypting and decrypting are the same except for the use of a different key and, as  $de = ed = 1 \bmod \phi(n)$  we can reverse the role of the encryption and decryption key; i.e. thus first decrypting a message and then encrypting it also results in the same message. With this we can also use RSA to perform signing; by decrypting the message we want to sign we generate a signature than anyone can check with the public key and that only the holder of the secret key can create. (We typically sign hashes of message rather than messages themselves, see Chapter ??.)

The structure of RSA encryption and decryption also gives several other properties, for example

that the encryption of the product of two messages is the product of the encryptions (modulo  $n$ ),  $E(m * m', (e, n)) = E(m, (e, n)) * E(m', (e, n)) \bmod n$ . Combined with the observation above this allows one to create *blind signatures*, i.e. to have Alice sign Bob's message without Alice learning the message: Bob generates a random mask  $r$ , encrypt this mask with Alice's public key  $(e, n)$  and multiplies the message (modulo  $n$ ) with the encrypted mask. The result,  $m * r^e$ , is given to Alice to sign (decrypt). This text does not reveal anything about  $m$  to Alice and by signing it she creates the signature of  $m$  masked (i.e. multiplied) with  $r$ .

$$D(m * r^e, (d, n)) = (m * r^e)^d \bmod n = m^d * r \bmod n$$

Bob can now remove the mask by dividing by  $r$ .

## 2.5 Computational Security

RSA requires a significant amount of computation to encrypt and especially decrypt. Also, the mathematical structure used for RSA implies that keys have to be big to prevent attackers from e.g. factoring the modulus  $n$ . So how large a key is needed and how does the performance compare to e.g. a block cipher?

### RSA vs DES performance

- RSA ~ 1000 slower in hardware
- RSA ~ 100 time slower in software
- Gets worse with longer keys
- How long a key is needed?
  - Estimate effort needed by attacker

62

### Hypotheses

- 56 bit DES key was strong enough in 1982
  - Breaking it requires 500,000 Mips Years
    - 1 Mips Year = 20 hours on 450Mhz Pentium II
- Computing per \$ doubles every 18 months
  - Variant of Moore's law
  - Every 10 years, 100x computing power per \$
- Budget of organisations doubles every 10 years
- Algorithmic improvement
  - Computation required halves every 18 months

63

Public key systems are typically significantly slower than asymmetric systems and often require larger keys for the same level of security. When considering how large a key should be one should not only consider the capabilities of an attacker now. If the encrypted data is to stay secret for three decades then we need to plan ahead and consider the computational power of an attacker may have in 30 years.

### Overview

Year	DES	RSA	DSA	EC	Mips years
1982	56	417	102	...	$5 \times 10^5$
2002	72	1028	127	139	$2 \times 10^{10}$
2012	80	1464	141	165	$4 \times 10^{12}$
2022	87	1995	154	193	$8 \times 10^{14}$

64

### Definitions of security



- Information theoretical (aka unconditional)
  - Possible in public key setting ?
  - Public key known: Anyone can encrypt.
  - Try all possible private keys
    - (Recall why is this not possible for one time pad...)
- Computational
  - Breaking cipher is mathematically hard problem

So if we use a large key is an algorithm such as RSA secure? What does this exactly mean; when is RSA broken. In the symmetric setting we can achieve 'unconditional security' (or: information theoretically security) with the one-time pad. Even an attacker with unlimited computational power could not break this cipher because there was no way of validating whether a

guessed/found key is correct. For RSA unconditional security is clearly not achievable; an attacker with unlimited computational power could theoretically factor modulus  $n$  for any size key which is enough to compute the private key from the public key. In the public key setting in general the attacker can always encrypt its own messages and thus check whether these decrypt correctly, allowing elimination of incorrect keys. Thus we cannot expect any unconditionally secure cipher in this setting.

Luckily no attacker has unlimited resources. In the argumentation above we argued that breaking the system would require solving a problem that would be too hard. Thus we aim for *computational security*: breaking the cipher should not be computationally feasible for the attacker. But what exactly is hard and how can we be sure that the attacker cannot think up a way to break the system without solving this hard problem? To answer the first part of this question we recall some basic complexity theory.

### What is a hard problem (1)



- Algorithm for short or long instances
  - Running time depends on length of instance
  - E.g.: Sorting 10 numbers takes less time than sorting 10.000 numbers
- For some problems minimum number of steps for **any algorithm** known
  - Sorting  $n$  numbers takes at least  $n \log n$  steps
  - Very hard to prove



### What is a hard problem (2)

- 'Hard' problem: requires at least an **exponential** number of steps to solve
  - I.e. nr of steps more than any polynomial.
  - in size of problem (= *security parameter*)
- No hard problems in NP known
- **Known** solutions take exponential time:
  - Factoring a product of two primes
  - Computing the discrete logarithm

68

**Definition 2** Consider a problem which can have instances of different sizes. We say such a problem is in complexity class  $\mathcal{P}$  (polynomial time) if there exists a (deterministic) algorithm which solves the problem and the time it takes grows at most polynomial in the size of the problem.

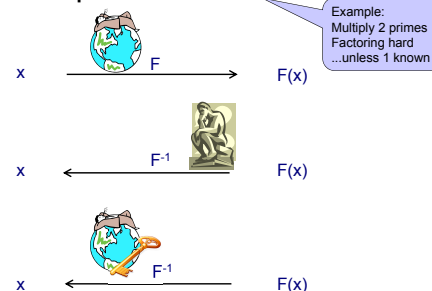
We say a problem is in complexity class  $\mathcal{NP}$  if there exists a non-deterministic algorithm which solves the problem in polynomial time. (Equivalently we could say that there exists an algorithm that can check a given solution to the problem in at most polynomial time.)

### P vs NP

- P
  - Solving takes polynomial time
- NP
  - Solution can be **checked** in polynomial time
    - But finding solution **may** take exponential time
- NP contains P
- It is **unknown** whether  $P = NP$

69

### Trapdoor function: F



70

We say that a problem is 'hard' when it is not in  $\mathcal{P}$ . Having a hard problem, however, is not enough to construct a public key cryptography system; we need a problem that is hard for an attacker but easy for the holder of the private key. We need a *trapdoor* function, i.e. a function that is:

- easy to compute in one direction (to be able to encrypt)
- hard to compute the inverse without the secret (an attacker should not be able to decrypt)
- easy to compute the inverse with the secret (the secret key holder should be able to decrypt)



A first natural question to ask is: do such functions actually exist? The perhaps surprising answer is that this is not known. A fundamental unanswered question of complexity theory is whether  $\mathcal{P}$  is equal to  $\mathcal{NP}$ . As computing the inverse of a trapdoor function must be in  $\mathcal{NP}$ ; a non-deterministic algorithm could first 'guess' the secret and then perform the same computations as the key holder. Yet it cannot be in  $\mathcal{P}$  as it must be a hard problem. Thus if one finds such a trapdoor function one has also shown that  $\mathcal{P} \neq \mathcal{NP}$ .

Still, though no-one has proven the existence of trapdoor functions, many candidates exist which have been studied extensively and for which no efficient algorithm to compute the inverse are known. One example is the factoring the product of two primes; creating the product from the primes is easy while factoring a (large) number is difficult. However, if you know one of the two primes finding the other is easy again.

**Showing Computational security?** We've just seen that there are no provably hard trapdoor functions. As such it would be impossible to prove that any asymmetric crypto system is actually computationally secure. For example, the security of RSA depends on the fact that factoring of two primes is hard. Thus its security inherently relies on an unproven fact. However, this problem has been studied so extensively that we can be reasonably certain it is at least hard enough that an attacker will not manage to solve it in general (if they did they could win a Nobel prize instead of decrypting our data...) So perhaps we can safely *assume* that this problem is hard. But even then is that enough to make our crypto system secure? Clearly we have to make the right choices (large keys, etc.) but we also need to know there is no way around our system without breaking the 'hard' problem. This is where provable security comes in: we prove that breaking the cipher implies solving some assumedly hard problem.

First we need to be more precise on what 'breaking' the cipher means; what information does the attacker get and what does she need to learn to consider the cipher broken? To answer these question we need to specify the exact setup, knowledge of the attacker and their goal. To do this in an easy to understand way we express security properties in the form of a game between us as keyholder (also called challenger) and the attacker (opponent). In the game for property IND-CPA we give the opponent the encryption of one out of two texts and the opponent has to guess which one it is.

### Security Game 1 (IND-CPA)

*Indistinguishable under chosen-plaintext attack*

1. **Opponent** picks two plain texts
2. **We** randomly pick one, encrypt it & give cipher text to opponent
3. **Opponent** guesses which text was encrypted

Opponent advantage:  $|P(\text{correct guess}) - 1/2|$

Good cipher: opponent advantage small

72

### Security Game 2 (IND-CCA2)

- **Opponent** has **Enc, Dec** oracle
- **Opponent** picks two plain texts
  - Can use **Enc/Dec** as wanted before choosing
- **We** randomly pick one and encrypt it
- **Opponent** gets cipher text (**challenge**)
- **Decryption** oracle not for **challenge**
- **Opponent** guesses which text encrypted

79

Of course the opponent could simply randomly guess, giving a probability of 1/2 of being correct. But perhaps the attacker can do better by examining the cipher text we give in the second step. How much better the attacker can do we call the *attacker advantage*:  $|P(\text{correct guess}) - 1/2|$ . (Note that being able to guess wrong most of the time also implies it is also possible to guess correctly most of the time, hence the use of the absolute value here.)

We say the attacker 'wins the game' if there is a distinct attacker advantage. The attacker advantage will depend on the *security parameter* which captures the 'problem size', for example the length of the key used. As the problem grows it is no longer realistic for an attacker to e.g. try decrypting with all possible keys. Formally we can thus put that a cipher is *IND-CPA secure* if

the attacker advantage is negligibly small (goes to zero faster than  $1/p$  for any polynomial  $p$ ) for any polynomial-time attacker. Paraphrasing; the advantage of attacker with a realistic amount of resources will go down exponentially as the key size grows.

The setup of the IND-CPA game matches a situation where the attacker has a single cipher text to examine and captures the worst case scenario for this case. If the attacker can tell that any plain text is more likely than any other for this cipher text, this game will give her an advantage and thus allow her to win the game. Thus IND-CPA security gives us that the attacker will not be able to learn anything from a single cipher text. But what if the attacker has even more information, for example several cipher texts or even several plain-cipher text pairs? In this case she might be able to break the cipher using this extra information even when she cannot win the IND-CPA security game. To state that a cipher is suitable for such situations we need a stronger notion of security expressed by an easier to win (by the opponent/attacker) security game. There are several different security games to describe security of ciphers in different usage scenarios. The indistinguishability under adaptive chosen-ciphertext attack (IND-CCA2) security game is one such game where the attacker has access to an encryption and decryption oracle, i.e. can encrypt any plain text and decrypt all cipher texts except, of course, the challenge itself. From the games it is clear that IND-CCA2 security implies IND-CPA security. (Any winning strategy for the IND-CPA game can also be used in the IND-CCA2 setting.)

**Security Proofs** As mentioned above, being able to factor large numbers implies the ability to break RSA. But is the reverse also true; is breaking RSA (i.e. winning the security game) really as hard as factoring large numbers (or another difficult problem<sup>4</sup>)?

A security proof of an asymmetric cryptography system shows that breaking the system (i.e. winning the security game) implies breaking some problem that is assumed to be hard. A cipher for which such a proof exists is then referred to as being 'provably secure'. Although such a proof is a very useful validation of the encryption one has to be careful in the interpretation of the term 'provably secure': it shows that breaking the cipher in a specific setting is at least as hard as some problem which is *assumed* to be hard in *general*. This by itself does not guarantee that our specific instance of the problem is not breakable (perhaps the problem was not really hard or we chose our parameters poorly (e.g. keys too small) leading to a specific instance of the problem that is easy to solve).

### Example: ElGamal

- Multiplicative Group  $Z_q = \{1 \dots q-1\}$ 
  - Multiply, divide, exponentiation easy, log hard
- Key creation: sample  $x$  from  $Z_q$ ,
  - $x$  is the private key,  $g^x$  is the public key
- Encryption: sample  $y$  from  $Z_q$  (salt)
  - $enc(m, g^x) = (c, k) = (m * g^{xy}, g^y)$
- Decryption:
  - $dec((c, k), x) = c / k^x$

$$\begin{aligned}
 dec(enc(m, g^x), x) &= dec((m * g^{xy}, g^y), x) \\
 &= m * g^{xy} / g^{yx} \\
 &= m
 \end{aligned}$$

73

### Security Assumption (the 'hard' problem)

- Decisional Diffie Hellman (DDH):

"no effective attacker can distinguish between  $(g^x, g^y, g^z)$  and  $(g^x, g^y, g^{xy})$ "

- Exists  $\epsilon$  such that, for any attacker, any  $q$ :
  - Random  $x, y, z$  in  $Z_q$ ;
  - $guess1 = Attacker(g^x, g^y, g^z)$ ;
  - $guess2 = Attacker(g^x, g^y, g^{xy})$ ;
  - $|P(guess1) - P(guess2)| < \epsilon(q)$

76

As an example of a security proof we show that the algorithm ElGamal satisfies security property IND-CPA, assuming that the Decisional Diffie Hellman (DDH) assumption holds. Note that if we can take a discrete log then DDH does not hold; DDH is a stronger assumption than the assumption that discrete log is hard.

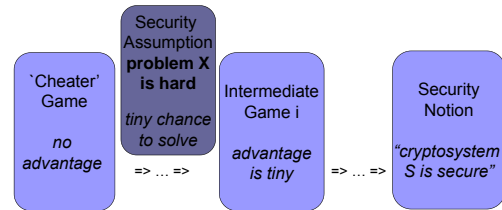
<sup>4</sup>If one is able to take the  $e$ -th root then one can brake RSA. It is not known whether this is equivalent with factoring  $n$ .

### 'Cheater' Game for ElGamal

- Opponent picks two plain texts
- We randomly pick one and encrypt it.
- Opponent gets  $g^z$  for random  $z$
- Opponent guesses which text encrypted.
  
- Opponent advantage:  $|P(\text{correct guess}) - 1/2|$
- Information opponent independent of choice
  - no opponent advantage possible

75

### Game stepping (reduction)



- Tasks:
1. Show security of basic game
  2. Show correctness of implications (games steppings)

74

To show security of ElGamal we show that there is no way to tell the difference between the IND-CPA game and a game in which we cheat so the attacker can never win; telling the difference between the two games is the same as solving the DDH problem. As such any attacker that can win the security game breaks DDH. In the game where we cheat we basically give back a random value ( $g^z$ ) instead of an encrypted message - as such no information about which message is encrypted is available at all and the attacker can impossibly win. With a small transformation we can replace the random  $g^z$  by (message times random  $m * g^z$ ) which is just as random as  $g^z$  itself (similar to a one-time pad). The only remaining difference with the security game is whether we use  $g^z$  or  $g^{xy}$ . The DDH assumption exactly states we cannot tell those two situations apart. Thus whether we cheat or not has no significant influence on the attackers advantage; i.e. the attacker has no significant advantage.

### Transformation

- Use property of  $*$  to conclude
  - Random  $x, y, z$  in  $Z_q$
  - guess-game1 = Attacker( $g^x, g^y, m * g^z$ );
  - guess-game2 = Attacker( $g^x, g^y, m * g^{xy}$ )
  - $|P(\text{guess-game1}) - P(\text{guess-game2})| < \epsilon(q)$
- If can tell difference  $m * a$  and  $m * b$  then can tell difference  $a, b$

77

### Difference Cheat - Security Game

- In real game attacker gets
  - Public key:  $g^x$
  - From the cipher text:  $g^y$  and  $m * g^{xy}$
- In the basic game the opponent
  - Public key:  $g^x$
  - From the cipher text:  $g^y$  and  $m * g^z$
- If the attacker can distinguish then also between  $(g^x, g^y, g^z)$  and  $(g^x, g^y, g^{xy})$ 
  - Play security game with this input.
  - For first will be basic game, for second security game

78

So we have proven that ElGamal satisfied is 'secure' (or more precise: satisfies IND-CPA). Again a word of caution though. Not all 'assumedly hard' problems are created equal; for example DDH is (assumed to be) a stronger assumption (so gives a weaker security property) than the discrete log assumption. Yet other problems may have been less extensively studied and the fact that no efficient way to solve the problem have been found is not a sufficiently strong indication that they are actually hard. Even if the problem is theoretically hard you have to choose an instance that is also hard in practice; see e.g. the discussion on key choices for RSA above as an example factoring can become easy if you do not choose your primes carefully. Finally security proofs are difficult to make, requiring subtle probabilistic reasoning, and many mistakes have been discovered in published proofs.

## 2.6 Conclusions and where to go from here

The goal after this lecture/chapter is that you will have a good idea of the basics of cryptography; know how symmetric and public key ciphers can be used and are able to give examples of these ciphers. It should also be clear what 'cipher X is secure' means in any given context and you should be able to check whether this is the case for a given setting and scheme.

This chapter provides an introduction to cryptography. We will look at digital signatures in more details yet in the lecture on hashing and certificates. The handbook of applied cryptography [5] (see e.g. Chapters 7 and 8) gives more technical and detailed descriptions of the algorithms, security notions and types of attacks. If you wish to learn more about cryptography there are also several Master courses on this subject.

### 2.6.1 Literature

Suggested reading (check the course page [1] for the most up to date list of suggested reading materials):

- Security Engineering Introduction [2, Ch 5].
- Handbook of applied cryptography [5, Ch 7,8]

## 2.7 Exercises

The exercises marked with (\*) indicate questions that are more challenging and/or technically involved than what you can expect on the exam. The other questions include old exam questions as well as some practise questions.

1. For the online music store scenario in Exercise 2 of the previous chapter. consider potential weaknesses and attacks that could be addressed. Indicate how you would apply the counter-measures and what trade-offs this imposes and what effect do they have on both the attacks as well as on the goals of the legitimate actors in the system, and what new goals/actors do you need to introduce.
2. Decipher the following (English) text:
 

Ftq Husqzqdq oubtqd ue zaf hqdk eqogdq.
3. Suppose that amongst six people each pair wants to be able communicate securely without the others being able to eavesdrop.
  - (a) How many different keys will they need in total if they use a symmetric encryption algorithm and how many if they use an asymmetric encryption algorithm?
  - (b) How many keys does a person have to store secretly in both cases?
4. Four methods to encrypt multi-block messages are 'Electronic Code Book', 'Cipher Block Chaining', 'Output Feedback', 'en 'Cipher Feedback'.
 

[ (See Figures Above) ]

  - (a) Give the corresponding decryption schemas.
  - (b) CBC has limited error propagation. Explain why this is so and give the error propagation for the other schemas.
  - (c) Compare the methods on the aspects secrecy, integrity and performance.

- (d) Two users notice that under the same conditions the same message always has the same encryption and decide to randomize the encryption by prefixing message with a randomly chosen block, i.e. by encrypting  $R, B_1, B_2, \dots, B_n$  rather than  $B_1, \dots, B_n$ .
- Why would you want to randomize the encryption?
  - For which of the methods above would this work?
  - Could you use the IV factor (for the methods that use one) instead? (Explain)
5. Entropy describes the amount of randomness in an unknown event/piece of data; a higher entropy indicates that more information is needed to be able to predict the outcome.
- What has a higher entropy?
    - The roll of a fair dice or an unfair dice.
    - A coin toss or a roll of a dice.
    - The state of the union address of 2008 or a pincode
  - The (Shannon) entropy (in bits) can be calculated with  $-\sum p(x) \cdot \log_2 p(x)$  where  $x$  ranges over all possible outcomes and  $p(x)$  is the probability of outcome  $x$ . Check your intuition in the previous part by calculating the entropies. (Choosing any appropriate probability distribution for the 'unfair dice'.)
  - (\*) Show that a function cannot increase entropy ( $Ent(f(X)) \leq Ent(X)$ ).
  - (\*) Show that for a domain of size  $2^n$  the maximum entropy for any random variable on that domain is  $n$  (which is achieved only by taking the uniform distribution).
6. The one-time pad makes a message undistinguishable from messages of equal length. With letters (instead of bits) you can apply the one time pad by adding the key to the message modulo 26.
- This looks a lot like the Vigenere cipher. Why does the analysis method to attack that cipher not work here?  
The encryption is secure no matter the amount of computational power the attacker has available; even trying all possible keys will not help the attacker.
  - Find two keys such that ciphertext 'AFIGHT' decodes to 'YESYES' and 'NONONO'.  
The one-time pad is not very convenient in use; the key is as long as the message and can only be used once.
  - What happens if the key of a one-time pad is reused?
  - Would it be possible to make an unconditionally secure cipher with a shorter or reusable key? (Remember question 3)?
7. The El-Gamal crypto algorithm is a variant of the Diffie-Hellman cryptosystem. Given a large random prime number  $p$  and a generator  $g$ , Alice chooses her private sleutel  $x$  randomly such that  $1 \leq x \leq p - 2$ . Alice's public key is  $(p, g, g^x)$ .
- To encode a message  $m$  for Alice, Bob selects a random  $r$  and sends  $(g^r, mh^r)$  to Alice, where  $h = g^x$  is obtained from the public key of Alice.
- How can Alice decode message (c,d) ?
  - Why can only Alice decode this message?
  - Why does Bob need to create and use a random  $r$  ?
  - Does Alice have to know that Bob's number  $r$  is really random?
  - How many bits of information can Bobs message contain.
  - What can Bob do if he wants to send a larger message?