Key Length

- The security of a symmetric cryptosystem is a function of:
    - strength of the algorithm
    - length of the key
- If we assume that the strength of the algorithm is such that it is technically unbreakable, then the best way to break the cipher is to try every possible key using a brute-force attack.
- The key length used for a cryptographic protocol determines the highest security it can offer. Thus, key lengths must be chosen in accordance with the desired security.
- Selecting appropriate key lengths requires a good understanding of the relation between security and key lengths, and a good insight into the design of the cryptographic methods and the mathematics involved in the attempts at breaking them.
- Key lengths indicate the number of bits contained in a certain cryptographic key or related arithmetic structure. They are a measure of the level of security that may be achieved.
- However, the relationship between key lengths and security is not that straightforward. For example, key lengths 80, 160, and 1024, may imply the same level of security when 80 bits is the key length for a symmetric encryption method, 160 bits for a hash length, and 1024 bits as the length of an RSA modulus.
- In other words the cryptographic algorithm plays a large part in the relationship between key lengths and the level of security achieved.
- As another example, consider the difference of key lengths between DES and Triple-DES. DES has a key length of 56 bits.
- Triple-DES applies DES three times implying a key size of 168 bits. However it uses three keys k1, k2, and k3, but k1 and k2 are the same which means it is actually 112 bits. That is what was meant with the statement that the relationship between key lengths and security is not that straightforward.

Time and Estimates for Brute-force Attacks

- Brute-force attack involves systematically checking all possible key combinations until the correct key is found and is one way to attack when it is not possible to take advantage of other weaknesses in an encryption system.

- Two parameters determine the speed of a brute-force attack the number of keys to be tested and the speed of each test.

- The key length used in the encryption determines the practical feasibility of performing a brute-force attack, with longer keys exponentially more difficult to crack than shorter ones.

- Consider the following table (Table 7.1) from your textbook that provides time and cost estimates for 1995 computer architectures:

Cost ($)	Key Length (Bits)					
	40	**56**	**64**	**80**	**112**	**128**
$100K	2 sec	35 hours	1 year	70,000 years	10^{14} years	10^{19} years
$1 M	0.2 sec	3.5 hours	37 days	7000 years	10^{13} years	10^{18} years
$10 M	0.02 sec	21 min	4 days	700 years	10^{12} years	10^{17} years
$100 M	2 msec	2 min	9 hours	70 years	10^{11} years	10^{16} years
$1 B	0.2 msec	1.3 sec	1 hr	7 years	10^{10} years	10^{15} years
$10 B	0.02 msec	1 sec	5.4 min	245 days	10^{9} years	10^{14} years
$100 B	2 µsec	0.1 sec	32 sec	24 days	10^{8} years	10^{13} years
$1 T	0.2 µsec	0.01 sec	3 sec	2.4 days	10^{7} years	10^{12} years
$10 T	0.02 µsec	1 msec	0.3 sec	6 hrs	10^{6} years	10^{11} years

- Now, according to Moore's Law: Computing power doubles approximately every 18 months. This means costs go down a factor of 10 every five years; what cost $1 million to build in 1995 will cost a mere $100,000 in the year 2000 and $10,000 in 2015.

- This does not take into consideration highly-pipelined and multi-core architectures, as well as computing clusters. All this would decrease computing times as well as decreased costs.

- A state-of-the-art modern supercomputer can go through 38,360,000,000,000,000 keys per second as an estimate.

- Check out the following link for fun: http://calc.opensecurityresearch.com/

- The main consideration to try to estimate the minimum "value" of a key: How much value can be trusted to a single key before it makes economic sense to try to break?

- To give an extreme example, if an encrypted message is worth $1.39, then it wouldn't make much financial sense to set a $10-million cracker to the task of recovering the key.

- On the other hand a $1.39 message which may result in saving lives and preventing severe damage to a country may very be worth the cost. It is also important to note that in this example the value of the message is very short-lived.

- On the other hand, if the plaintext message is worth $100 million, then decrypting that single message would justify the cost of building the cracker.

Thermodynamic Limitations

- Longer key lengths are better, but only up to a point. AES will have 128-bit, 192-bit, and 256-bit and even longer key lengths. This is far longer than needed for the foreseeable future. In fact, we cannot even imagine a world where 256-bit brute force searches are possible.

- It requires some fundamental breakthroughs in physics and our understanding of the universe. One of the consequences of the second law of thermodynamics is that a certain amount of energy is necessary to represent information.

- To record a single bit by changing the state of a system requires an amount of energy no less than kT, where T is the absolute temperature of the system and k is the Boltzman constant.

- Given that $k = 1.38 \times 10^{-16}$ erg/K, and that the ambient temperature of the universe is 3.2 Kelvin, an ideal computer running at 3.2 K would consume 4.4×10^{-16} ergs every time it set or cleared a bit. To run a computer any colder than the cosmic background radiation would require extra energy to run a heat pump.

- Now, the annual energy output of our sun is about 1.21×10^{41} ergs. This is enough to power about 2.7×10^{56} single bit changes on our ideal computer; enough state changes to put a 187-bit counter through all its values.

- Say we built a Dyson sphere around the sun and captured ***all*** its energy for 32 years, without ***any*** loss, we could power a computer to count up to 2^{192}. Of course, it wouldn't have the energy left over to perform any useful calculations with this counter.

- But that's just one star, and not a very substantial one in the grand scheme of the universe. A typical supernova releases something like 10^{51} ergs. (About a hundred times as much energy would be released in the form of neutrinos).

- If all of this energy could be channeled into a single orgy of computation, a 219-bit counter could be cycled through all of its states.
- These numbers have nothing to do with the technology of the devices; they are the maximums that thermodynamics will allow. And they strongly imply that **brute-force attacks against 256-bit keys will be infeasible until computers are built from something other than matter and occupy something other than space.**

Public-Key Key Length

- Modern dominant public-key encryption algorithms are based on the difficulty of factoring large numbers that are the product of two large primes.

- Multiplying two large primes is essentially a one-way function (discussed earlier); it's easy to multiply the numbers to get a product but hard to factor the product and recover the two large primes. Public-key cryptography uses this idea to make a trap-door one-way function.

- The traditional factoring algorithm is known as the quadratic sieve. The modern algorithm is the general number field sieve. These algorithms are continuously evolving and making the factoring process (and hence breaking keys) faster and faster.

- In 1989 mathematicians suggested that the general number field sieve would never be practical. In 1992 they suggested that it was practical, but only faster than the quadratic sieve for numbers greater than 130 to 150 digits or so.

- The general number field sieve can factor a 512-bit number over 10 times faster than the quadratic sieve.

- The following tables show the number of MIPS-years required to factor numbers of different sizes for both algorithms:

Factoring Using the General Number Field Sieve	
# of bits	**MIPS-years required to factor**
512	30,000
768	$2x10^{8}$
1024	$3x10^{11}$
1280	$1x10^{14}$
1536	$3x10^{16}$
2048	$3x10^{20}$

Factoring Using the General Number Field Sieve	
# of bits	**MIPS-years required to factor**
512	< 200
768	100,000
1024	$3x10^{7}$
1280	$3x10^{9}$
1536	$2x10^{11}$
2048	$4x10^{14}$

- To determine how long a key will meet one's security requirements requires an analysis of both the intended security and lifetime of the key, and the current state-of-the-art of factoring.

- Currently a 1024-bit number provides the same level of security as a 512-bit number in the early 1980s. Thus, if we want our keys to remain secure for 20 years, 1024 bits is likely too short.

- Assume that dedicated private cryptanalysts can acquire 10,000 MIPS-years, a large corporation can get 10^7 MIPS-years, and that a large government can get 109 MIPS-years.

- Also assume that computing power will increase by a factor of 10 every five years, and that advances in factoring mathematics allow us to factor general numbers at the speeds of the special number field sieve. (currently not possible, but a breakthrough could occur at any time.)

- Given the above assumptions, the following table shows recommendations for different key lengths for security during different years:

Year	Individual	Corporation	Government
1995	768	1280	1536
2000	1024	1280	1536
2005	1280	1536	2048
2010	1280	1536	2048
2015	1536	2048	2048

- If we consider the fact that in every decade we can factor numbers twice as long as in the previous decade, the following table shows in interesting curve:

Long-range Factoring Predictions	
Year	**Key Length (in bits)**
1995	1024
2005	2048
2015	4096
2025	8192
2035	16,384
2045	32,768

Comparing Symmetric and Public-Key Key Length

- A maxim of network security in general is that a system is only as strong as its weakest link. In a system that uses both symmetric and public-key cryptography, the key lengths for each type of cryptography should be chosen so that it is equally difficult to attack the system via each mechanism.

- It makes no sense to use a symmetric algorithm with a 128-bit key together with a public-key algorithm with a 386-bit key, just as it makes no sense to use a symmetric algorithm with a 56-bit key together with a public-key algorithm with a 1024-bit key.

- The following table lists public-key modulus lengths whose factoring difficulty roughly equals the difficulty of a brute-force attack for popular symmetric key lengths.

Symmetric and Public-key Key Lengths with Similar Resistances to Brute-Force Attacks	
Symmetric Key Length	**Public-key Key Length**
56 bits	384 bits
64 bits	512 bits
80 bits	768 bits
112 bits	1792 bits
128 bits	2304 bits

- The data in the table suggests that for a symmetric algorithm with a 112-bit key, a modulus length for the public-key algorithm of about 1792 bits is adequate enough to address the security requirements.

- In general, a public-key length that is more secure than your symmetric-key length is a requirement. Public keys tend to be used over a longer term, and are used to protect much more critical information.

Birthday Attacks against One-Way Hash Functions

- A birthday attack is a type of cryptographic attack that exploits the mathematics behind the birthday problem in probability theory.

- This attack can be used to subvert communication between two or more parties. The attack depends on the higher likelihood of collisions found between random attack attempts and a fixed degree of permutations.

- As an example, consider the scenario in which a teacher with a class of 30 students asks for everybody's birthday, to determine whether any two students have the same birthday (corresponding to a hash collision as described earlier).

- Intuitively, this chance may seem small. In the first attack, if the teacher picked a specific day (say October 1), then the chance that at least one student was born on that specific day is about 7.9%, using: $(1 - \left(\frac{364}{365}\right)^{30}$

- For the second attack however, the probability that at least one student has the same birthday as any other student is around 70% (using the following formula for n = 30): $\frac{1-365!}{((365-n)! \times 365^n)}$

- Given a function ***f***, the goal of the attack is to find two different inputs x_1 and x_2 such that $f(x_1) = f(x_2)$.

- Such a pair is called a **collision**. The method used to find a collision is simply to evaluate the function for different input values that may be chosen randomly or pseudorandomly until the same result is found more than once.

- Analogous to the above discussion, there are two brute-force attacks against a one-way hash function:

    - The first is: Given the hash of message, ***H*(*M*)**, an adversary would like to be able to create another document, ***M´,*** such that ***H*(*M*) = *H*(*M´*)**.

    - The second attack is more subtle: An adversary would like to find two random messages, ***M,*** and ***M´,*** such that ***H*(*M*)** = ***H*(*M´*)**. This is called a **collision,** and it is a far easier attack than the first one.

- Finding someone with a specific birthday is analogous to the first attack; finding two people with the same random birthday is analogous to the second attack. The second attack is commonly known as a **birthday attack**.

- Assume that a one-way hash function is secure and the best way to attack it is by using brute force. It produces an ***m***-bit output.

- Finding a message that hashes to a given hash value would require hashing 2^m random messages. Finding two messages that hash to the same value would only require hashing $2^{m/2}$ random messages.

- A machine that hashes a million messages per second would take 600,000 years to find a second message that matched a given 64-bit hash. The same machine could find a pair of messages that hashed to the same value in about an hour.

- Therefore, to protect against a birthday attack, select a hash-value twice as long as you otherwise might think you need. For example, if you want to drop the odds of someone breaking your system to less than 1 in 2^{80}, use a 160-bit one-way hash function.

- The following table shows number of hashes ***n(p)*** needed to achieve the given probability of success, assuming all hashes are equally likely.

Bits	**Possible outputs (H)**	**Desired probability of random collision (p)**									
		10^{-18}	**10^{-15}**	**10^{-12}**	**10^{-9}**	**10^{-6}**	**10^{-3}**	**10^{-2}**	**25%**	**50%**	**75%**
16	**66000**	< 2	< 2	< 2	< 2	< 2	11	36	190	300	430
32	**4.3×10^{9}**	< 2	< 2	< 2	3	93	2900	9300	50000	77000	110000
64	**1.8×10^{19}**	6	190	6100	190000	$6.1X10^{6}$	$1.9X10^{8}$	$6.1X10^{6}$	$3.3X10^{9}$	$5.1X10^{9}$	$7.2X10^{9}$
128	**3.4×10^{38}**	$2.6X10^{10}$	$8.2X10^{11}$	$2.6X10^{13}$	$8.2X10^{14}$	$2.6X10^{16}$	$8.3X10^{17}$	$2.6X10^{18}$	$1.4X10^{19}$	$2.2X10^{19}$	$3.1X10^{19}$
256	**1.2×10^{77}**	$4.8X10^{29}$	$1.5X10^{31}$	$4.8X10^{32}$	$1.5X10^{34}$	$4.8X10^{35}$	$1.5X10^{37}$	$4.8X10^{37}$	$2.6X10^{38}$	$4.0X10^{38}$	$5.7X10^{38}$
384	**3.9×10^{115}**	$8.9X10^{48}$	$2.8X10^{50}$	$8.9X10^{51}$	$2.8X10^{53}$	$8.9X10^{54}$	$2.8X10^{56}$	$8.9X10^{56}$	$4.8X10^{57}$	$7.4X10^{57}$	$1.0X10^{58}$
512	**1.3×10^{154}**	$1.6X10^{68}$	$5.2X10^{69}$	$1.6X10^{71}$	$5.2X10^{72}$	$1.6X10^{74}$	$5.2X10^{75}$	$1.6X10^{76}$	$8.8X10^{76}$	$1.4X10^{77}$	$1.9X10^{77}$

Digital signature susceptibility

- Digital signatures can be susceptible to a birthday attack. A message ***m*** is typically signed by first computing ***f(m)***, where ***f*** is a cryptographic hash function, and then using some secret key to sign ***f(m)***.

- Suppose Mallory wants to trick Bob into signing a fraudulent contract. Mallory prepares a fair contract ***m*** and a fraudulent one ***m'***.

- She then finds a number of positions where ***m*** can be changed without changing the meaning, such as inserting commas, empty lines, one versus two spaces after a sentence, replacing synonyms, etc.

- By combining these changes, she can create a huge number of variations on ***m*** which are all fair contracts.

- In a similar manner, Mallory also creates a huge number of variations on the fraudulent contract ***m'.***

- She then applies the hash function to all these variations until she finds a version of the fair contract and a version of the fraudulent contract which have the same hash value, ***f(m)*** = ***f(m')***. She presents the fair version to Bob for signing.

- After Bob has signed, Mallory takes the signature and attaches it to the fraudulent contract. This signature then "proves" that Bob signed the fraudulent contract.

- The probabilities differ slightly from the original birthday problem, as Mallory gains nothing by finding two fair or two fraudulent contracts with the same hash.

- Mallory's strategy is to generate pairs of one fair and one fraudulent contract. The birthday problem equations apply where ***n*** is the number of pairs. The number of hashes Mallory actually generates is ***2n***.

- To avoid this attack, the output length of the hash function used for a signature scheme can be chosen large enough so that the birthday attack becomes computationally infeasible, i.e. about twice as many bits as are needed to prevent an ordinary brute-force attack.