Breaking Cryptography

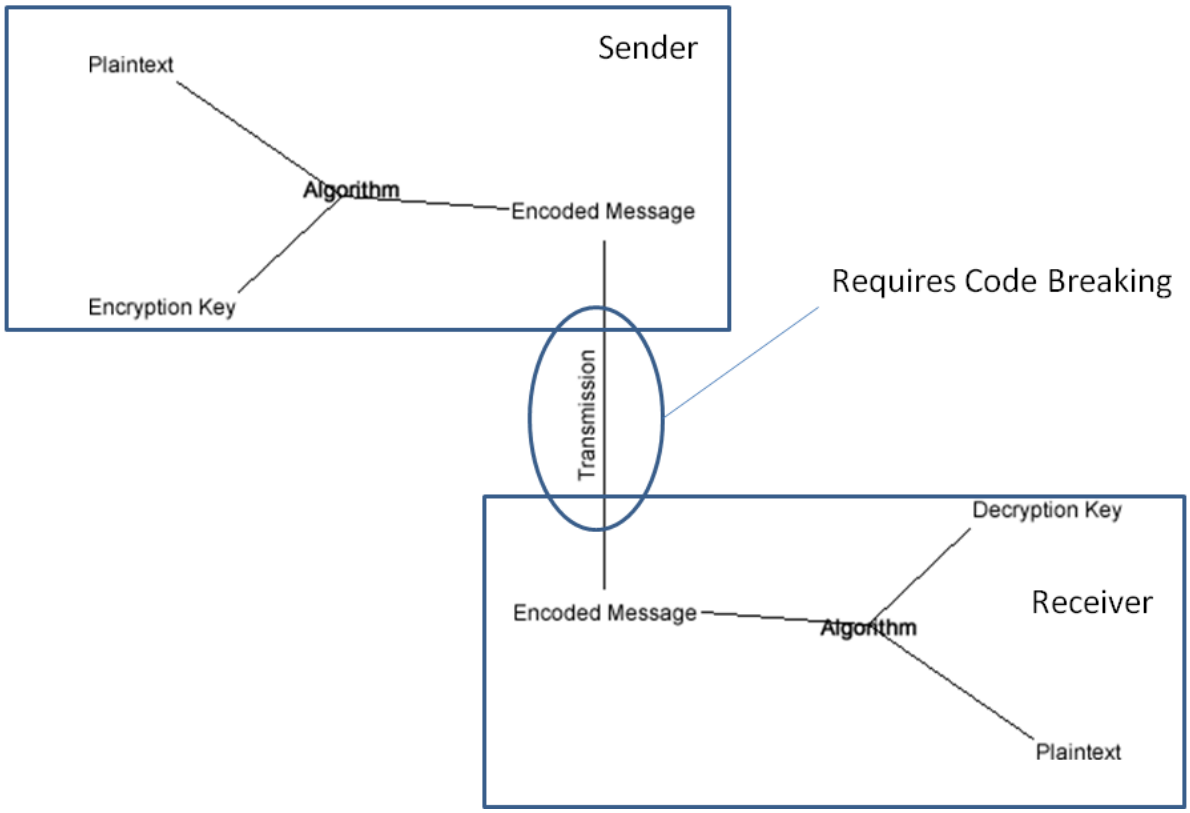
Parallel Decryption

Brad Heap

May 2009

Encryption 101

Encryption is the process by which a piece of information is transformed into a state that cannot be easily understood without a reverse transformation being applied to it to recover the original data. Encryption is used by both Governments and private companies around the world to ensure that sensitive data cannot be read by unauthorised parties. Transmission of data through encryption usually takes the form shown below:



At the sender's end plain data (plaintext) is combined with a key and through the use of an algorithm an encrypted form of the data (ciphertext) is produced. This is then transmitted to another party via some medium (which may be public). At the receiver's end they apply a reverse algorithm on the received data, which combines their unlock key to receive the normal data.

Types of Algorithms

There are three basic algorithm forms that are used in encryption of data.

Symmetric Key

Symmetric Key algorithms require that both sender and receiver use the same key to encrypt and decrypt a message. This usually simplifies the encryption process which provides a large time speed up when there is a lot of data that is being encrypted. To ensure that data is strongly encrypted some symmetric key algorithms allow for multiple encryption passes over the data.

Asymmetric Key (Public/Private Key)

Asymmetric Key algorithms do not require the sender and receiver to have the same key. However, in most cases the two keys are mathematically linked. Prime numbers are most commonly used in the generation of Asymmetric keys because the factorisation of semi-prime numbers (the product of two prime numbers) is a very long process which can make data encrypted in this form very hard to break. Asymmetric encryption is very slow in comparison to symmetric key encryption. In practice it is common for data to be encoded with a symmetric algorithm but for the symmetric key to be transmitted to the receiver through an encrypted asymmetric method.

One Way Messages

A third form of data encoding is the use of one way encryption. This is used where the content of the plaintext does not matter but the result does, for instance in the use of storing pin numbers. Just storing the raw number on a card or in a computer is dangerous because someone could compromise security and find out the pin numbers to cards. However, one way functions allow us to calculate a hash or checksum of the pin number and store that instead. The checksum cannot be reversed back to a raw pin number, and to check if a pin number is correct someone merely has to calculate the checksum of the given pin.

However, a major vulnerability in one way messages is the possibility of a data collision where two different numbers hash to the same value. For instance if the checksum function was to sum all four digits in a pin together e.g. Pin: 1234, Checksum: 1 + 2 + 3 + 4 = 9 and another pin was 2341, Checksum 2 + 3 + 4 +1 = 9. In this example a wrong pin number could still give out cash.

Breaking the Key

If you know the algorithm used to encrypt a set of data then the easiest way to decrypt it is to break the key. There are a number of methods that can be used to achieve this.

Brute Force

A brute force attack involves trying every possible key combination. This method is very time consuming and although theoretically it will always result in the correct key it is often impractical to perform on a real world problem.

Example One: 16 Digit Key

- 10 Possible Numbers Per digit (0-9)
- Permutations: 9,999,999,999,999,999 values to check.
- Checking Process is naturally parallel.
- Assume 256 node cluster computer at 100 checks per second.
- Maximum Time: 390,624,999,999 seconds (12,735 years)
- In this example it would take on average (half time) over 6,000+ years to break a single code.

Example Two: 8 Alphanumeric Character String

- 90 Possible Characters Per character.
- Permutations: 4,304,672,100,000,000 to check.
- Checking Process is naturally parallel.
- Assume 256 node cluster computer at 100 checks per second.
- Maximum Time: 168,151,253,906 seconds (131,573 years)
- In this example it would take on average (half time) over 60,000+ years to break a single code.

These two examples have shown how slow a brute force attack is on breaking a key. However, there are ways to limit the number of keys that we need to check and therefore speed up the process. This is achieved through learning about the key and algorithm that is used in the encryption of the data. Typically the more that we know about the design of an encryption system the fewer the number of keys that we need to check; therefore the faster that we can break the key.

Dictionary Attack

A dictionary attack is very similar to a brute force attack where every single possible key is tried. However, a dictionary attack only attempts every possible key within a set of given keys or an algorithm to generate possible keys. The dictionary is generated based off information that you can gather about both the key and the algorithm in use.

Example One

Extending Example One from above, say we have learnt that the key is a prime number. There are 29,844,570,422,669 16 digit primes. Assuming the same computing power as before it would now only take a maximum of 1,165,803,532 seconds (912 years) to break.

Example Two

Extending Example Two from above, say we have learnt that the 8 characters spell out an English word. There are around 100,000 words in the English language. We are still unsure if it is spelt in capital, lowercase or a combination of the two cases. Taking this into account we are limited to 2,965,420,000 possible orders. Assuming same computing power as before it would now only take a maximum of 115,836 seconds (32 hours) to break.

In the above two examples the number of keys to test was still huge, however in comparison to a brute force attack the speed up was many thousands of times faster, and for the second example brought the ability to crack the key within easy reach of someone with a powerful system available to them.

To improve a dictionary even further, when you know that a key has been generated by either a human or a non-random system, it is possible to study these systems and order your dictionary in such a way that the keys that are expected to occur more frequently to be checked first.

Random Numbers

To ensure complete security many algorithms rely on keys that are completely random and cannot be guessed or predicted. Unfortunately getting truly random numbers is a very difficult task. Humans cannot be relied upon to generate true random numbers. Studies of human behaviour have shown that we are very prone to avoiding duplication of numbers such as 111 and overly alternate between high and low digits in random numbers. Computers also fail the randomness test as well because they rely on algorithms to generate random numbers. One of the few truly random sequences of random numbers is the value of PI.

Because of the predictability of “random” numbers generated by both humans and computers an attempt to break a key that uses random numbers can often be achieved through the study of the algorithm and or the processes used in the implementation of it. If information can be understood about the source of the random numbers then the code breaker can implement a system to predict the random numbers.

Trapdoor Function

Some encryption algorithms allow for a special set of functions to be operated on the encrypted data to reveal the plain data without the use of a key. These functions are deliberately designed into the algorithms used in encryption and allow governments or other authorised parties access to encrypted data. It is very hard without knowledge of the trapdoor function to be able to stumble upon it because rather than just testing keys with a known algorithm you must design a whole algorithm that does not rely on a key.

Breaking the Text

As shown breaking the key used in the encryption of an encoded message can be nearly impossible. Another approach is to try and break the encrypted text or part of it. Breaking part of the text can help with the development of a key to unlock the rest, or be used to make assumptions about possibilities in the encryption used in the rest of the text and break it without the use of a key.

Pattern Matching

Finding patterns in the text is an effective way to break an encrypted text. Say for instance part of the encrypted text contained the sequence: “+83(88”. If you assume that the ‘8’ is really an ‘e’ you get the text “+e3(ee”. From here you can work on the rest of the text to find other patterns and unlock other characters and then reveal the word “degree”.

Of course not all codes keep each letter the same in the encryption that is what makes codes like the enigma so hard to break. However, if a pattern can be matched early on in the code, the method used to determine the next character can also be determined and this can also help unlock the complete message.

Parallel computers are a vital aide in pattern matching as they can be used to quickly check many different methods for finding patterns in codes, and once a partial pattern is matched the combined use of a number of cores can unlock the rest of the message in a much shorter time frame than a single computer.

Permutations

One of the fundamental issues with the brute force method of key cracking is the number of permutations that must be checked. In many codes there are a huge number of permutations that would never be used in a real code because they cause the encrypted message to transpose too closely to the plaintext.

For instance an alphabet of ABCD there is 24 permutations of this alphabet; however, 21 of these permutations either transpose or reverse onto themselves, leaving only three possible secure permutations (This isn't a very secure alphabet!). For instance the permutation DACB cannot be used because the C will always transpose onto the C and will never become encrypted.

BADC can also not be used because although a first pass through will encrypt the message, a second pass will reverse the code back to its original form. Only DCBA, CDAB, BADC are permutations that do not transpose onto themselves. If a code is limited by the number of permutations that are secure it will greatly reduce the number of keys that need to be checked and/or leave obvious unencrypted data in the code that can easily be pattern matched.

Summary

Breaking encryption is a naturally parallel problem. There are many different forms of encryption and in turn many different methods for attempting to break them. The rise of faster machines has seen many unbreakable codes broken, but has also seen many more ways to encrypt data that is currently considered unbreakable. However, there are still many methods of encryption that are considered unbreakable; a short discussion of these is presented below.

Secure Encryption

The Question of Trust

Receiving an encrypted message through either legitimate or illegitimate means reveals nothing about the identity of the sender and if the message is authentic. In fact it is possible for someone to break an asymmetric code and send a message perpetrating to be for a legitimate source to extract secrets. To prevent this there must be a way of verifying the identity to the sender and receiver (to ensure the message also gets to the right place). The only effective method of doing this is to send the message through a trusted median. In real world this is a very difficult task and a number of companies (such as VeriSign) have been set up to ensure that messages that are sent and received are authentic.

One Time Pad

A one-time pad is a pad where each page contains a series of truly random numbers that form a key for use in encryption. It is an asymmetric algorithm and requires that both sender and receiver have the same pad and page in use to unlock the encoded message. The strength of the one-time pad is that each key is only used once. In reality one-time pad codes have been cracked but only as a result of repeatedly using the same page on a pad, a pad falling into the wrong hands, or through brute force. A brute force attack on a one-time pad will only reveal one message if the algorithm has been used correctly which given the amount of time a brute force attack takes adds to the beauty of the method.

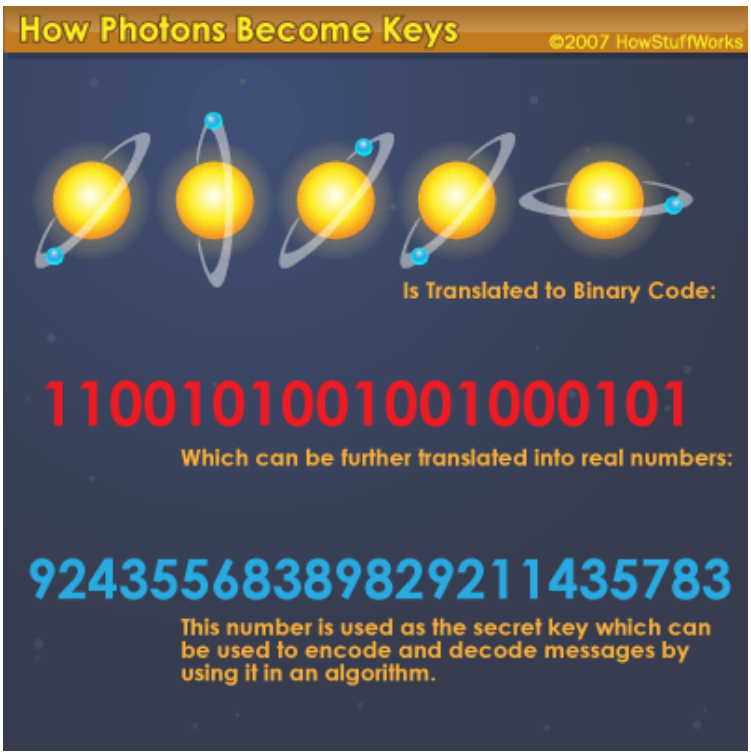
Salting

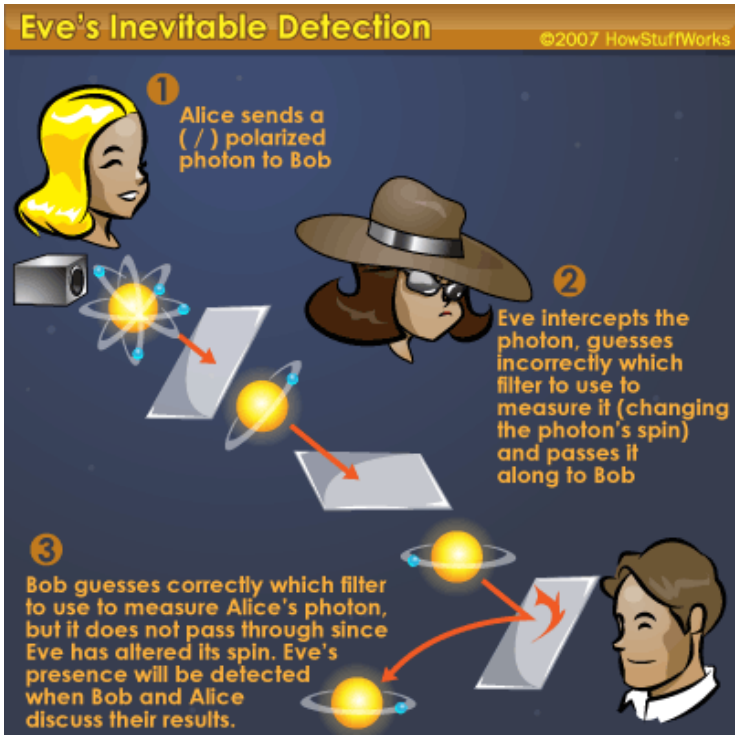
Salting is a method through which additional redundant data is added to plaintext to confuse anyone who may crack the encryption. Salting can be used in two ways. The first is to add additional data to the front and back of a plaintext message. This is often used in the encryption of passwords. Say for instance my password was auyewj, if I added jkh to the front and hjk to the rear the stored password becomes jkhquyewjhik.

Provided that I never reveal the algorithm I used to produce the salting it would be very hard to determine the actual password. The other method is to encode a message in amongst other plaintext. Take the following statement for instance: Holden engines live longer occasionally. If I take the first letters of each word I spell out Hello, revealing a message hidden in plain sight.

Quantum Encryption

Quantum Encryption is most likely the form that standard encryption will take in the future. It relies on the polarisation of light photons. It is far too complex to explain here, but because each time the code is encrypted or decrypted the polarisation of the photons changes and any attempt to view the data is revealed through this. The following diagrams provide some basic understanding:





Further Reading

- Code Breaking. Rudolf Kippenhahn. 1999.
- The Code Book. Simon Singh. 2000.
- A good maths text-book. Particularly something on discrete mathematics.
- How Encryption Works. Jeff Tyson. http://computer.howstuffworks.com/encryption.htm
- How Quantum Cryptology Works. Josh Clark. http://science.howstuffworks.com/quantum-cryptology.htm
- Cryptanalysis. Wikipedia. http://en.wikipedia.org/wiki/Cryptanalysis