Michael Guidry
March 11, 2017

Rethinking cryptographic implementations to create dynamic algorithms

Encryption algorithms are used to turn plaintext information into unreadable data. The purpose is to keep information secret from eavesdroppers. It is how credit cards, and other information gets communicated securely over the internet. Encryption algorithms can be used to encrypt entire hard drives, USB drives, and other forms of data containments. It is also used by criminals to protect evidence from being used in court cases. Governments use it to securely transmit important information such as cables between embassies similar to the leaks exposed by Bradley Manning.

Many different types of encryption algorithms exists which have various features, and methods for their particular features. The cryptographic keys needs to be secure for the communications to be unavailable to a third party. Encryption algorithms are used in practically every application requiring even the slightest security today. Encryption works by using a secret key, and then modifying data using this key. The other party which is attempting to communicate securely needs the same key to be able to recover the original data. It is important that the key stay secure otherwise the whole purpose of the encryption algorithms becomes pointless.

Keys are generated using random number generators. The amount of security a key has directly relates to the strength of the random number generator being less predictable. Random number generators will take various information sources such as process identifier numbers, data file sizes, username information, and other supposed random information. It is possible to brute force some of these variables which would decrease the security of the key. This implementation doesn’t directly relate to the possible mechanisms for key recovery of other cryptographic solutions.

Enigma is considered to be the turning point of World War 2. Allen Turing created a machine which was able to calculate, and perform different tasks which now became a major factor in our world full of electronics. If cryptography is implemented in this way it will require such a solution which would be generally considered the next step towards more complex machines. I personally believe it will require artificial intelligence, or a more advanced machine learning which isn’t being discussed publicly at this time.

Random number generators are historically known to have faults, or backdoors which could decrease the amount of time to determine the actual key used to encrypt the information. RSA had a backdoor in one of its random number generators, and supposedly the NSA had paid $10 million for them to set it as a default for cryptographic operations. It has been a severe topic because it is very difficult to obtain ‘truly’ random numbers on a mass scale which can be used to generate encryption keys for sessions, or starting places to find prime numbers.

Cracking cryptographic algorithms is possible because the algorithms are static, and it can get distributed among variable amounts of processors, and allows a calculable amount of time to solve. The fact that quantum computers are no longer a thing of the future is of great concern while security continues to become more relied on for communications, or data transfer.

Quantum computers are only solutions for a subset of problems which relate to mathematical equations that cannot easily be solved without processing all possible paths. Quantum computers perform tasks such as finding prime numbers faster by orders of magnitudes quicker than current processors such as Internet CPUs.

If you were to create an infinite amount of paths which were devised by the data itself while also modifying the algorithms during the process it would exponentially increase the effort required to create systems that may crack the cryptographic keys. Rethinking implementations is required to create this scenario which will ensure truly secure communications for the foreseeable future.

Cryptographic communications have historically used a single algorithm, and a key. The amount of bits within a key usually relates to peoples perception of the strength of the secured transmission. It is noticeable when generating keys that the higher bitrate requires more time to process. It is directly proportional to the time it takes to attempt to brute force, or use other techniques to break that same key in an adversaries control.

The most insecure encryption algorithm with a layer of ‘dynamic cryptography’ may become a small percent of the most secure possible communications. Dynamic cryptography means the actual algorithms being used, and their parameters becomes another layer of cryptography on top of the algorithms themselves. It will require not only mathematical solutions but reverse engineering, emulation, and possibly manual analysis of the changes with the numerous states. This process will be useless to assist future attempts to break different communications using a system which implements in this manner.

Infinite path creation is possible by putting all encryption algorithms into a structure containing a link to their functions, key schedulers, and internal states of their initialization vectors. The structure is used to instruct the overall implementation to use, swap, and manipulate the encryption algorithm, cryptographic key, and internal states of each algorithm. The implementation can generate these instructions ahead of time, or dynamically using the data being encrypted, or decrypted. It would ensure that infinite paths exist because the data itself becomes a part of the cryptographic “algorithm.” It is this idea which allowed me to coin the term “dynamic cryptography.”

The dynamic encryption algorithm should function similarly to processors. It may allow modifying states, or the instruction information itself for use further down the cryptographic stream. If these cryptographic opcodes were generated dynamically using the data itself which is being encrypted then it would further increase the exponential processing power required to break the cryptographic key, and recover the plaintext data. The opcodes could only affect things further down the cryptographic stream since it wouldn’t be able to be used with data which is before the opcode was used. Cryptographic opcodes would allow an entirely new effective type of solutions for secure communications.

Dynamic encryption opcodes should allow a push, pop, and execute from stack instructions. Pushing to specific positions in the stack would increase complexities. If these features are implemented properly then it would dramatically increase the amount of time for solvers such as SAT to exponentially increase. Communication channels could have layers on top which allow other connections such as paired, prior or global, or random to affect the particular sessions master key.

The most important part of implementing this to allow truly unbreakable cryptography, and infinite paths is to ensure the instruction set swaps algorithms every so many bytes of data being encrypted. The amount of bytes before it swaps to another algorithm can be generated in the same way as stated before such as from an encryption key, or the actual data itself. It may randomly choose for example: AES-256-CBC for 5 bytes, AES-128-ECB for 1000 bytes, Blowfish-CBC for 100 bytes, etc. The individual keys for each algorithm used for these algorithms would be modified as well using those instructions which were generated from the 'master key.' The individual keys could be reset, or have their state kept the same from the most recent time the individual algorithm was used.

Encrypting insignificant data during the process using either instructions, stack, or opcodes would further increase difficulty of solving the plaintext without the initial master key. The insignificant data could be a blank buffer, or random data in a buffer. This buffer wouldn't be used in the plaintext, or the secure encrypted end result. It would allow the implementation to advance the key scheduler of individual algorithms. An adversary without the information of how many bytes were skipped, or the key scheduler was advanced forward wouldn't be able to use traditional cryptanalysis methods without heavy modifications. Modifications of these sorts also defeats random number generator backdoors.