ENCRYPTING A PLAINTEXT MESSAGE WITH AUTHENTICATION

Cross Reference to Related Applications

[0001] The present invention claims priority from U.S. provisional patent application number 60/595,720, filed on 08/01/2005, the entire contents of which are incorporated by reference herein.

Field of the Invention

[0002] The present invention relates generally to the technical field of data communication and storage. Specifically, the invention relates to cryptographic methods and systems that allow for both the encryption and authentication of information through the use of a generic pseudorandom permutation.

Background of the Invention

[0003] Data encryption methods provide privacy of the encrypted information over insecure communication channels. Encryption methods alone, however, lack manipulation detection abilities to ensure the integrity or authenticity of the information. Data authentication methods are required to detect when the received message was altered by an adversary during communication.

[0004] Many known algorithms provide authentication separate from privacy. One of the most well known methods for providing data authentication generates an authentication tag or Message Authentication Code (MAC) through the use of a key-dependent one-way hash function. A one-way hash function is designed such that it is comparatively easy to compute but almost impossible to reverse. Because the length of the authentication tag is usually fixed and shorter than the length of the initial message, the authentication tag cannot ensure a one-to-one mapping of messages to authentication tags. The length of the authentication tag, however, is designed to be long enough to thwart brute force attacks.

[0005] In the method for exchanging a message with an authentication tag, the sender initiates the exchange by generating an authentication tag from the authentic

message using a shared key. The sender then transfers the message and authentication tag to the receiver. At the receiving end, the receiver must generate an authentication tag from the received message using the shared key as well. The receiver then compares his or her generated authentication tag with the received authentication tag. If the two tags match, then the receiver can be assured that the message has not been modified during transmission and that it was sent by someone who knows the secret key.

[0006] The use of an authentication tag or MAC consumes time on the receiving end, because it requires the receiver to generate a MAC for comparison. When combined with a data encryption method, the receiver must decrypt the message and generate an authentication tag before the received information can be used. This conventional approach requires two passes over the same message on both the sending and receiving end often with the same basic algorithm. Furthermore, this conventional approach often requires the use of separate keys for each function. The use of two separate functions uses excessive processing power, memory, and time.

[0007] In applications focused on minimizing latency such as Supervisory Control and Data Acquisition (SCADA) networks, Remote Frequency Identification (RFID), and other real-time data exchange systems, received information must be used immediately making it impossible to use a separate MAC for data authentication. The devices used in such applications present further restrictions on processing power, code space, and memory. These applications highlight the need for methods that provide message integrity integrated with strong cryptographic privacy to minimize the latency and overhead imposed by separate conventional methods.

[0008] In response to the disadvantages of the conventional approaches, various methods have been suggested. Based on a new proof in error detection, the SCADA community including the AGA12 committee suggested assured error-propagation as a means for providing integrity without the need for a traditional MAC. Various methods exist that include error-propagation to provide some level of integrity. Depending on the level of error-propagation, a one-bit modification to the transmitted ciphertext results in some amount of randomization of subsequent bits in order to provide enhanced manipulation detection. One such method, Propagating Cipher Block Chaining (PCBC) was designed to fully propagate a one-bit modification to all subsequent bits. Since its

design, however, PCBC mode has been found to be vulnerable to some straight-forward attacks. For example, switching two ciphertext blocks leaves the rest of the message unchanged.

[0009] Andrew Wright et al. recently proposed another solution, AES PE-mode for use in SCADA networks that was designed based on the error detection proof to assure at least six bits of randomization following a one-bit manipulation (viz., A. K. Wright, J. A. Kinast, and J. McCarty. Low-Latency Cryptographic Protection for SCADA Communications. In: Proc. 2nd Int. Conf. on Applied Cryptography and Network Security, ACNS 2004). While PE-mode lacks the straight-forward vulnerabilities of PCBC, PE-mode imposes a great deal of latency and overhead, because it is essentially a cascade cipher of two AES encryption modes. In addition to encrypting the message twice, PE-mode is designed to be used with a separate message authentication algorithm such as a CBC-MAC. The drawbacks of PCBC and PE-mode illuminate the need for an error-propagating encryption algorithm that is both fast and small and does not require further steps to achieve integrity.

Summary

[0010] The defined invention provides methods and systems for efficiently integrating integrity and strong encryption through assured error-propagation and an automatically generated authentication tag. The present invention is designed to work with considerably low levels of needed code space, processing resources, memory, and latency requirements. Briefly, the present invention consists of a multi-stage encryption system, wherein a plaintext chunk is passed through a sequence of pseudorandom permutations. The system generates intermediate data-dependent cryptographic variables at each stage, which are systematically combined into feedback loops to produce assured error-propagation. At the conclusion of encryption, the invention generates a cryptographic hash using the final data-dependent cryptographic variables.

[0011] The invention in question can be implemented in numerous ways including as a method, a system, a process, a device, a stand-alone cryptographic algorithm, or a mode of an existing cryptographic algorithm. Several inventive embodiments of the present invention are described below.

[0012] In one embodiment of the present invention, a method for multi-stage data encryption and authentication is defined wherein each stage is a pseudorandom permutation. The method comprises the steps of: receiving plaintext data, partitioning the plaintext into equal size plaintext blocks, passing the each plaintext block through a sequence of pseudorandom permutations, modifying the states of the pseudorandom permutations for the next block based on each current block's intermediate stage cryptographic variables each of which is the output of one stage in the sequence, and generating ciphertext from the output of each plaintext block's final stage. It should be noted that pseudorandom permutations are usually considered stateless. The present invention creates what will be referred to as states by storing variables that are used to modify the input to each pseudorandom permutation. Since the state of these variables directly affects the output of each pseudorandom permutation, the permutations can be seen as having states.

[0013] In one further aspect of the present invention, the implementation of each pseudorandom permutation can be achieved by one or multiple of the following: generating shifted permutation tables also known as S-Boxes, using existing cryptographic algorithms, including but not limited to AES, RC5, TEA, IDEA, TWOFISH, or electronically implementing classical rotors.

[0014] In one further aspect of the present invention, the modification of the states of each pseudorandom permutation is accomplished by following the pattern which will be referred to as the "312 rule". The pattern outlines the following steps:

a. The state of the first pseudorandom permutation is modified by the output of the next to the last pseudorandom permutation.
b. The states of the middle pseudorandom permutations are modified by the output of previous pseudorandom permutation.
c. The state of the last pseudorandom permutation is modified by both the state and output of the first pseudorandom permutation.

[0015] In one further aspect of the present invention, the method generates an authentication tag using the final plaintext block's state variables. The generation of the authentication tag is accomplished by either concatenating the final state variables or

masking the final state variables by combining them with the initial permutation states before concatenation.

[0016] In one further aspect of the present invention, the method includes the step of initializing the beginning state variables. The initialization process is conducted by encrypting an nonce using a non-initialized version of the defined method and using the generated ciphertext and authentication tag as the beginning variables.

[0017] In one further aspect of the present invention, an internal counter is used to further modify the states of the pseudorandom permutations. The addition of a counter, designed to eliminate short cycles, is performed by storing a N counter variables where N is the number of pseudorandom permutation stages, incrementing the counter variables in an odometric fashion, and modifying each pseudorandom permutation state by the associated counter variable.

[0018] In one further aspect of the present invention, the number of needed pseudorandom permutations is reduced by substituting the inverse of pseudorandom permutations in use for some number of other pseudorandom permutations. For example, in a 4-stage method with pseudorandom permutations ABCD, the number of pseudorandom permutations required could be reduced by using the permutations $ABA^{-1}B^{-1}$ where A^{-1} and B^{-1} are the inverses of A and B respectively.

[0019] In one further aspect of the present invention, the number of needed pseudorandom permutations can be reduced on one side of the communications channel by supplying only pseudorandom permutations on one side while the other side has both pseudorandom and inverse pseudorandom permutations. For example the server side can have both ABCD and $A^{-1}B^{-1}C^{-1}D^{-1}$ permutations while the client side can have only ABCD permutations. Communication from the server to the client is accomplished by first decrypting the plaintext message (has the same effect as encrypting). The client can then recover the message by encrypting the ciphertext (has the same effect as decrypting). Communication from the client to the server is done in the normal fashion i.e. client encrypts message, server decrypts message.

[0020] In one further embodiment of the present invention, a data decryption method that is the inverse of the multi-stage data encryption and authentication method is defined. The method comprises the steps of: receiving ciphertext data, partitioning the

ciphertext into equal size ciphertext blocks, passing each block through a sequence of pseudorandom permutations where each permutation is an inverse of the permutations used in the encryption method, modifying the states of the pseudorandom permutations for the next block based on each current block's intermediate stage cryptographic variables, and generating plaintext from the output of each ciphertext block's final stage. The decryption method passes each ciphertext block backwards through the sequence in the encryption method. It should be noted that the chosen embodiment of the decryption method should match those in the chosen embodiment of the encryption method. For example, if the chosen embodiment of the encryption method uses RC5 as the pseudorandom permutation, generates an authentication tag, and utilizes counters, the chosen embodiment of the decryption method should also use RC5, generate an authentication tag, and use counters.

[0021] In one further embodiment of the present invention, a method for performing an integrity check is defined. The method consists of the steps of: performing the encryption method defined in order to generate ciphertext and an authentication tag, performing the decryption method defined on said ciphertext in order to generate plaintext and a second authentication tag, and comparing the two authentication tags for equality. It can be assured with high probability that the ciphertext was not modified after encryption if the two authentication tags are equal.

Brief Description of the Drawings

[0022] For a more complete understanding of the present invention and for further features and advantages, reference is now made to the following description, taken in conjunction with the accompanying drawings, in which:

[0023] FIG. 1 is a first flowchart in accordance with a preferred embodiment of the present invention;

[0024] FIG. 2 is a second flowchart in accordance with a preferred embodiment of the present invention;

[0025] FIG. 3 illustrates an encryption system in accordance with a preferred embodiment of the present invention;

[0026] FIG. 4 illustrates a decryption system in accordance with a preferred embodiment of the present invention;

[0027] FIG. 5 illustrates initializing variables using a nonce in accordance with a preferred embodiment of the present invention;

[0028] FIG. 6 illustrates generating an authentication tag from final state variables in accordance with a preferred embodiment of the present invention;

[0029] FIG. 7 illustrates generating a masked authentication tag from a combination of the initial and final state variables in accordance with a preferred embodiment of the present invention;

[0030] FIG. 8 illustrates decrypting and verifying the integrity of the message using a received authentication tag in accordance with a preferred embodiment of the present invention;

[0031] FIG. 9 illustrates the encryption method from FIG. 3 with the addition of counters, in accordance with a preferred embodiment of the present invention;

[0032]FIG. 10 illustrates the decryption method from FIG. 4 with the addition of counters, in accordance with a preferred embodiment of the present invention; and

[0033] FIG. 11 illustrates incrementing counters in accordance with a preferred embodiment of the present invention.

Detailed Description of the Preferred Embodiment

[0034] FIGS. 1 and 2 represent two versions of the flow chart explaining the steps of encryption for the present invention. FIG. 1 was the original diagram as can be found in the provisional patent of the present invention. While maintaining the essential core of the invention, FIG. 2 is the revised encryption diagram with a more clear representation and the elimination of unneeded steps. Before explaining the details of the revised diagram, it is good to note the difference between the two diagrams.

[0035] The original diagram uses the term Variable Exchange Table (VET) which is now referred to as the more generally used and understood term, pseudorandom permutation. Furthermore, what was originally denoted as a VET Setting (VS) is now referred to as a state variable (SV), and the Output of a VET is now referred to as an

intermediate cryptographic variable (CV). The terms have been modified for ease of understanding.

[0036] FIG. 1 contains all of the same steps as the revised diagram except the generation of the authentication tag **270** and the initialization of the state variables **500**. At the time of submission of the provisional patent, the cryptographic strength of the present invention was still undetermined. In order to compensate for the uncertainty, additional steps **115**, **120,** and **150** were added to the encryption method to facilitate the combination of the output of the final pseudorandom permutation with an AES keystream through an exclusive or (XOR) function to produce ciphertext. Said additional steps were thought to further protect the ciphertext from attacks. Further consideration and evaluation have eliminated the need for said additional steps, and therefore they have been removed from the revised diagram. Note that corresponding steps in the two diagrams have been numbered the same (ex **125** corresponds to **225**).

[0037] FIG. 2 illustrates the steps involved in an encryption embodiment of the present invention. From the start, a key and counter are loaded **200** in order to initialize the pseudorandom permutations if necessary **205** and **210**. The next step initializes the state variables and counters with an nonce **500** which is described in further detail in FIG. 5. Once the plaintext is acquired **225**, the first plaintext block is combined with the initialized state variables and stepped through a series of four pseudorandom permutations **230-245** resulting in the first ciphertext block **255**. Before the next plaintext block can be encrypted, the state variables are updated using the intermediate cryptographic variables **260**. This cycle continues **265** and **225** for all plaintext blocks. Optionally, the final state variables can be combined to form an authentication tag **270**. The details of the embodied encryption method are described to a greater extent in the next diagram.

[0038] FIG. 3 represents an encryption embodiment of the present invention wherein m plaintext blocks P_i **301** are each passed through a sequence of four pseudorandom permutations **303** resulting in m ciphertext blocks **304**. In this embodiment each of the four permutations **303** are keyed with different keys k1, k2, k3, and k4.The embodied method includes the step of initializing the state variables **302** by passing an nonce **310** through a randomization function **500** that is discussed in detail

below. Once the state variables are initialized, the first plaintext block P_1 **301a** is combined with the initial state variable $SV1_0$ **302a** through modular 2^n addition where n is the size of a plaintext block. The result of said combination is passed into the first pseudorandom permutation F_{k1} **303a** producing an intermediate cryptographic variable $CV12_1$ (the cryptographic variable between the first pseudorandom permutation Fk_1 **303a** and the second Fk_2 **303b**) which will be fed forward to encrypt the next plaintext block P_2 **301b**. Continuing with the encryption of P_1 **301a**, $CV12_1$ is combined with the second initialized state variable $SV2_0$ **302b** through modular 2^n addition and passed into the second pseudorandom permutation F_{k2} **303b** resulting in $CV23_1$. The encryption continues to follow the same pattern for the two remaining pseudorandom permutations F_{k3} **303c** and F_{k4} **303d** where the result of F_{k4} **303d** is the first ciphertext block C_1 **304a**.

[0039] For the encryption of the next plaintext block P_2 **301b**, the state variables **305** must be updated using a feedback mechanism as will be described. The first state variable $SV1_1$ **305a** produced following the encryption of the first plaintext block P_1 **301a** is generated by combining the previous state variable $SV1_0$ **302a** with the output from the previous block's third permutation $CV34_1$ through modular 2^n addition where n is the size of a plaintext block. The second state variable $SV2_1$ **305b** is generated by combining the previous state variable $SV2_0$ **302b** with the output from the previous block's first permutation $CV12_1$ through modular 2^n addition. Similarly, the third state variable $SV3_1$ **305c** is generated by combining the previous state variable $SV3_0$ **302c** with the output from the previous block's second permutation $CV23_1$ through modular 2^n addition. The fourth state variable $SV4_1$ **305d** is generated by combining the previous state variable $SV4_0$ **302d** with the output from the previous block's first permutation $CV12_1$ and the current block's first state variable $SV1_1$ **305a**, through modular 2^n addition. It should be noted that the calculation of $SV1_1$ **305a** should occur before the calculation of $SV4_1$ **305d**. Furthermore, while the described embodiment of the present invention stores the state variables SV1, SV2, SV3, and SV4, derived embodiments could entail the same spirit of the present embodiments without actually storing the state variables. The step of storing state variables is disclosed in the present invention for ease of understanding.

[0040] The encryption of all further plaintext blocks P_2 **301b** through P_m **301c** are conducted in the same manner as the encryption of P_1 **301a**. For example, the second

plaintext block P_2 **301b** is conducted in the same manner as the encryption of the first plaintext block P_1 **301a** substituting the updated state variables **305** for the previous state variables **302**.

[0041] FIG. 4 represents a decryption embodiment of the present invention wherein m ciphertext blocks C_i **404** are each passed through a sequence of four inverse pseudorandom permutations F_K^{-1} **403** resulting in m plaintext blocks P_i **401**. In this embodiment each of the four inverse permutations F_K^{-1} **403** are keyed with the same keys used in the encryption in Fig. 3. The embodied method includes the step of initializing the state variables **402** by passing an nonce **410** through a randomization function **500** that is discussed in detail below. Once the state variables **402** are initialized, the first ciphertext block C_1 **404a** is passed into the first inverse pseudorandom permutation F_{k4}^{-1} **403d.** The result of said inverse pseudorandom permutation F_{k4}^{-1} **403d** is combined with the initial state variable $SV4_0$ **402d** through modular 2^n subtraction where n is the size of a ciphertext block producing an intermediate cryptographic variable $CV34_1$ (the cryptographic variable between F_{k3}^{-1} **403c** and F_{k4}^{-1} **403d**) which will be fed forward to decrypt the next ciphertext block C_2 **404b**. Continuing with the decryption of C_1 **404a**, $CV34_1$ is passed into the second inverse psuedorandorandom permutation F_{k3}^{-1} **403c.** The result of said inverse permutation F_{k3}^{-1} **403c** is combined with $SV3_0$ using modular 2^n subtraction producing $CV23_1$. The decryption continues to follow the same pattern for the two remaining inverse pseudorandom permutations F_{k2}^{-1} **403b** and F_{k1}^{-1} **403a** where the result of F_{k1}^{-1} **403a** is combined with $SV1_0$ **402a** using modular 2^n subtraction to produce the first plaintext block P_1 **401a**.

[0042] For the decryption of the next ciphertext block C_2 **404b**, the state variables **405** must be updated using a feedback mechanism as will be described. The state variable $SV1_1$ **405a,** produced following the decryption of the first ciphertext block C_1 **404a,** is generated by combining the previous state variable $SV1_0$ **402a** with the input from the previous block's second inverse permutation $CV34_1$ through modular 2^n addition where n is the size of a ciphertext block. The second state variable $SV2_1$ **405b** is the output of the previous block's third inverse permutation F_{k2}^{-1} **403b**. Similarly, the state variable $SV3_1$ **405c** is the output of the previous block's second inverse permutation F_{k3}^{-1} **403c**. The state variable $SV4_1$ **405d** is generated by combining the previous state

variable $SV4_0$ **402d** with the input from the previous block's fourth inverse permutation $CV12_1$ and the current block's state variable $SV1_1$ **405a**, through modular 2^n addition. It should be noted that the calculation of $SV1_1$ **405a** should occur before the calculation of $SV4_1$ **405d**. Furthermore, while the described embodiment of the present invention stores the state variables SV1, SV2, SV3, and SV4, derived embodiments could entail the same spirit of the present embodiments without actually storing the state variables. The step of storing state variables is disclosed in the present invention for ease of understanding.

[0043] The decryption of all further ciphertext blocks C_2 **404b** through C_m **404c** are conducted in the same manner as the decryption of C_1 **404a**. For example, the second ciphertext block C_2 **404b** is conducted in the same manner as the decryption of the first ciphertext block C_1 **404a** substituting the updated state variables **405** for the previous state variables **402**.

[0044] FIG. 5 illustrates the function of generating initial values by randomizing a nonce as used in FIGS. 3, 4, 9, and 10. The purpose said function is to initialize the state variables and counters to unique and unpredictable values. The nonce or input to the function may be a random number, an incrementing counter, or any value as long as it has not been used before in the context of a given key(s). It should be noted that the nonce need not be secret. The initialization function parses a unique value into m blocks N_i **501** and passes each block through a sequence of m pseudorandom permutations **503** resulting in values that are used in the initial setup of both the encryption and decryption methods. Padding may be necessary in order to facilitate equal sized blocks. The number of blocks m and the number of pseudorandom permutations m must always be the same. In the present embodiment of the initialization function, m is equal to 4. The randomization function keys each of the four permutations F_K **503** with different keys k1, k2, k3, and k4. The embodied method includes the step of initializing the state variables **502** to a constant such as zero. Once the state variables **502** are initialized, the first block N_1 **501a** is combined with the initial state variable $SV1_{n1}$ **502a** through modular 2^n addition where n is the size of a block. The result of said combination is passed into the first pseudorandom permutation F_{k1} **503a** producing an intermediate cryptographic variable $CV12_1$ (the cryptographic variable between the first pseudorandom permutation F_{k1} **503a** and the second F_{k2} **503b**) which will be fed forward to encrypt the next block N_2

501b. Continuing with the randomization function of N_1 **501a**, $CV12_1$ is combined with the second initialized state variable $SV2_{n1}$ **502b** through modular 2^n addition and passed into the second pseudorandom permutation F_{k2} **503b** resulting in $CV23_1$. The randomization continues to follow the same pattern for the two remaining pseudorandom permutations F_{k3} **503c** and F_{k4} **503d** where the result of F_{k4} **503d** is the first CTR value $CTR1_0$ **504a**. It should be noted that some embodiments may not use the generated CTR **504** values.

[0045] For the next block N_2 **501b**, the state variables **505** must be updated using a feedback mechanism as will be described. The first state variable $SV1_{n2}$ **505a** produced following the randomization of the first block N_1 **501a** is generated by combining the previous state variable $SV1_{n1}$ **502a** with the output from the previous block's third permutation $CV34_1$ through modular 2^n addition where n is the size of a block. The second state variable $SV2_{n2}$ **505b** is generated by combining the previous state variable $SV2_{n1}$ **502b** with the output from the previous block's first permutation $CV12_1$ through modular 2^n addition. Similarly, the third state variable $SV3_{n2}$ **505c** is generated by combining the previous state variable $SV3_{n1}$ **502c** with the output from the previous block's second permutation $CV23_1$ through modular 2^n addition. The fourth state variable $SV4_{n2}$ **505d** is generated by combining the previous state variable $SV4_{n1}$ **502d** with the output from the previous block's first permutation $CV12_1$ and the current block's first state variable $SV1_{n2}$ **505a**, through modular 2^n addition. It should be noted that the calculation of $SV1_{n2}$ **505a** should occur before the calculation of $SV4_{n2}$ **505d**. Furthermore, while the described embodiment of the present invention stores the state variables SV1, SV2, SV3, and SV4, derived embodiments could entail the same spirit of the present embodiments without actually storing the state variables. The step of storing state variables is disclosed in the present invention for ease of understanding.

[0046] The randomization of all further plaintext blocks N_2 **501b** through N_4 **501d** are conducted in the same manner as the randomization of N_1 **501a**. For example, the second plaintext block N_2 **501b** is conducted in the same manner as the randomization of the first plaintext block N_1 **501a** substituting the updated state variables **505** for the previous state variables **502**. After the four blocks **501** are each randomized, the resulting state variables $SV1_0$, $SV2_0$, $SV3_0$, and $SV4_0$ **508** can be used as initial state variables for

FIG 3, 4, 9,10. Similarly, the resulting randomized values, $CTR1_0$, $CTR2_0$, $CTR3_0$, and $CTR4_0$ **504** can be used as initial counters for FIG 9,10.

[0047] FIG. 6 presents an elevated look at the method for generating an authentication tag from the results of the previously described encryption embodiment. The diagram includes an abbreviated version of the encryption method **300** in which each sequence of pseudorandom permutations is depicted in a single encryption function E_i **601**. The final encryption function E_m **601c** produces four final state variables **602** which are concatenated to form an authentication tag **603**. As explained previously, an authentication tag is used to provide an integrity check on encrypted data.

[0048] FIG. 7 represents an alternative embodiment of the method for generating an authentication tag from the results of the encryption embodiment. As in FIG. 6, the diagram includes an abbreviated version of the encryption method **300**. In this alternative embodiment, each final state variable **702** is combined with its corresponding initial state variable **701** through an XOR function **703** before being concatenated to form the authentication tag **704**. This alternative embodiment masks the final state variables from being openly accessible to an attacker and may serve to increase the cryptographic strength of the present invention.

[0049] FIG.8 represents an embodied method for performing an integrity check of a message after decryption. The diagram includes an abbreviated version of the decryption method **400** in which each sequence of inverse pseudorandom permutations is depicted in a single decryption function D_i **802**. The received message includes a previously generated authentication tag AT **805** in addition to the ciphertext **801**. Said authentication tag was previously generated during encryption as is depicted in FIG. 6. The final decryption function D_m **802c** produces four final state variables **803** which are concatenated to form an authentication tag AT' **804**. The received authentication tag AT **805** identifies the original message that was encrypted, while the newly generated authentication tag AT' **804** identifies the received message. With the two authentication tags, an integrity check **806** is performed as follows. If the two authentication tags are not equal, the message was modified between its encryption and decryption and should be rejected. Conversely, if the authentication tags are equal, it can be assured with high probability that the message has not been tampered with and can be accepted. It should

be noted that an integrity check could also be performed using a previously generated authentication tag as in FIG. 7. The method for generating an authentication tag during decryption would match the encryption method in FIG. 7 followed by an integrity check as in the present figure.

[0050] FIG. 9 represents a further aspect the present invention wherein counters are added. In the same manner as the embodiment in FIG. 3, m plaintext blocks P_i **901** are each passed through a sequence of four pseudorandom permutations F_k **903** resulting in m ciphertext blocks C_i **904**. Each of the four permutations F_k **903** are keyed with different keys k1, k2, k3, and k4. The embodied method includes the step of initializing the state variables **902** and counters **906** by passing a nonce **900** through a randomization function **500** that has been previously defined. Once the state variables and counters are initialized, the first plaintext block P_1 **301a** is combined with the initial state variable $SV1_0$ **902a** through modular 2^n addition where n is the size of a plaintext block. The result of said combination is passed into the first pseudorandom permutation F_{k1} **903a** producing an intermediate cryptographic variable $CV12_1$ (the cryptographic variable between the first pseudorandom permutation F_{k1} **903a** and the second F_{k2} **903b**) which will be fed forward to encrypt the next plaintext block P_2 **901b**. Continuing with the encryption of P_1 **901a**, $CV12_1$ is combined with the second initialized state variable $SV2_0$ **902b** through modular 2^n addition and passed into the second pseudorandom permutation F_{k2} **903b** resulting in $CV23_1$. The encryption continues to follow the same pattern for the two remaining pseudorandom permutations F_{k3} **903c** and F_{k4} **903d** where the result of F_{k4} **903d** is the first ciphertext block C_1 **904a**.

[0051] For the encryption of the next plaintext block P_2 **901b**, the state variables **905** must be updated using counters and a feedback mechanism as will be described. The first state variable $SV1_1$ **905a** produced following the encryption of the first plaintext block P_1 **901a** is generated by combining the previous state variable $SV1_0$ **902a** with the output from the previous block's third permutation $CV34_1$ and a counter $CTR1_0$ **906a** through modular 2^n addition where n is the size of a plaintext block. The second state variable $SV2_1$ **905b** is generated by combining the previous state variable $SV2_0$ **902b** with the output from the previous block's first permutation $CV12_1$ and a counter $CTR2_0$ **906b** through modular 2^n addition. Similarly, the third state variable $SV3_1$ **905c** is

generated by combining the previous state variable $SV3_0$ **902c** with the output from the previous block's second permutation $CV23_1$ and a counter $CTR3_0$ **906c** through modular 2^n addition. The fourth state variable $SV4_1$ **905d** is generated by combining the previous state variable $SV4_0$ **902d** with the output from the previous block's first permutation $CV12_1$ and the current block's first state variable $SV1_1$ **905a** and a counter $CTR4_0$ **906d** through modular 2^n addition. The counters **906** are then incremented using function **1100**. It should be noted that the calculation of $SV1_1$ **905a** should occur before the calculation of $SV4_1$ **905d**. Furthermore, while the described embodiment of the present invention stores the state variables SV1, SV2, SV3, and SV4, derived embodiments could entail the same spirit of the present embodiments without actually storing the state variables. The step of storing state variables is disclosed in the present invention for ease of understanding.

[0052] The encryption of all further plaintext blocks P_2 **901b** through P_m **901c** are conducted in the same manner as the encryption of P_1 **901a**. For example, the second plaintext block P_2 **901b** is conducted in the same manner as the encryption of the first plaintext block P_1 **901a** substituting the updated state variables **905** for the previous state variables **902**.

[0053] FIG. 10 represents a decryption embodiment of the present invention wherein m ciphertext blocks C_i **1004** are each passed through a sequence of four inverse pseudorandom permutations **1003** resulting in m plaintext blocks P_i **1001**. In this embodiment each of the four inverse permutations **1003** are keyed with the same keys used in the encryption in Fig. 9. The embodied method includes the step of initializing the state variables **1002** and initial counters **1006** by passing a nonce **1000** through a randomization function **500** that has been previously defined. Once the state variables and counters are initialized, the first ciphertext block C_1 **1004a** is passed into the first inverse pseudorandom permutation F_{k4}^{-1} **1003d.** The result of said inverse pseudorandom permutation F_{k4}^{-1} **1003d** is combined with the initial state variable $SV4_0$ **1002d** through modular 2^n subtraction where n is the size of a ciphertext block producing an intermediate cryptographic variable $CV34_1$ (the cryptographic variable between F_{k3}^{-1} **1003c** and F_{k4}^{-1} **1003d**) which will be fed forward to decrypt the next ciphertext block C_2 **1004b**. Continuing with the decryption of C_1 **1004a**, $CV34_1$ is passed into the second inverse

psuedorandorandom permutation F_{k3}^{-1} **1003c.** The result of said inverse permutation F_{k3}^{-1} **1003c** is combined with $SV3_0$ using modular 2^n subtraction producing $CV23_1$. The decryption continues to follow the same pattern for the two remaining inverse pseudorandom permutations F_{k2}^{-1} **1003b** and F_{k1}^{-1} **1003a** where the result of F_{k1}^{-1} **1003a** is combined with $SV1_0$ **1002a** using modular 2^n subtraction to produce the first plaintext block P_1 **1001a**.

[0054] For the decryption of the next ciphertext block C_2 **1004b**, the state variables **1005** must be updated using a feedback mechanism as will be described. The state variable $SV1_1$ **1005a** produced following the decryption of the first ciphertext block C_1 **1004a** is generated by combining the previous state variable $SV1_0$ **1002a** with the input from the previous block's second inverse permutation $CV34_1$ and a counter $CTR1_0$ **1006a** through modular 2^n addition where n is the size of a ciphertext block. The second state variable $SV2_1$ **1005b** is the output from the previous block's third inverse permutation F_{k2}^{-1} **1003b** and a counter $CTR2_0$ **1006b** through modular 2^n addition. Similarly, the state variable $SV3_1$ **1005c** is the output from the previous block's second pseudorandom permutation F_{k3}^{-1} **1003c** and a counter $CTR3_0$ **1006c** through modular 2^n addition. The state variable $SV4_1$ **1005d** is generated by combining the previous state variable $SV4_0$ **1002d** with the input from the previous blocks fourth inverse permutation $CV12_1$ and the current block's state variable $SV1_1$ **1005a** and a counter $CTR4_0$ **1006a** through modular 2^n addition. The counters **1006** are then incremented using function **1100**. It should be noted that the calculation of $SV1_1$ **1005a** should occur before the calculation of $SV4_1$ **1005d**. Furthermore, while the described embodiment of the present invention stores the state variables SV1, SV2, SV3, and SV4, derived embodiments could entail the same spirit of the present embodiments without actually storing the state variables. The step of storing state variables is disclosed in the present invention for ease of understanding.

[0055] The decryption of all further ciphertext blocks C_2 **1004b** through C_m **1004c** are conducted in the same manner as the decryption of C_1 **1004a**. For example, the second ciphertext block C_2 **1004b** is conducted in the same manner as the decryption of the first ciphertext block C_1 **1004a** substituting the updated state variables **1005** for the previous state variables **1002**.

[0056] FIG. 11 represents an embodied method for modifying the counters from one block encipherment to the next. The method takes as input four counters $CTR1_i$ through $CRT4_i$ and produces four counters $CTR1_{i+1}$ through $CRT4_{i+1}$. The steps taken in the embodied method model a typical mileage odometer from an automobile where CTR1 is the lowest order of magnitude and CTR4 is the highest order of magnitude. The embodied method always begins by incrementing the lowest order counter CTR1 **1105** through modular 2^n addition where n is the size of the counter in bits. If CTR1 has reset itself and is equal to zero **1110**, the embodied method continues to increment CTR2 **1115** in the same manner as CTR1. If CTR1 is not zero **1110**, the method exits **1140a** and the resulting counters are stored for use in encrypting or decrypting the next block. Each subsequent counter is incremented in the same manner as long as all lower order counters are equal to zero.

[0057] In one embodiment of the present invention, a method for encrypting a plaintext message comprises receiving at least one plaintext message, wherein the plaintext message forms at least one plaintext block, encrypting said plaintext block by applying 2 or more pseudorandom permutations to each block, and modifying an input to each said pseudorandom permutation by at least one state variable which is modified for each plaintext block by at least one of previously generated permutation outputs, previously generated permutation inputs, ciphertext, and plaintext. The method comprises generating at least one ciphertext block from the output of each plaintext block's final pseudorandom permutation, partitioning the plaintext message into a plurality of equal size plaintext blocks, padding the plaintext message to facilitate the equal sized plaintext blocks, wherein the modification of the state variables comprises at least one of: modifying the state variable for a first pseudorandom permutation by an output of a next to the last pseudorandom permutation from the previous block, modifying the state variable for a final permutation by an output of the first pseudorandom permutation from the previous block and the state variable for the first pseudorandom permutation from the current block, and modifying the state variables for all other pseudorandom permutations by an output of the preceding pseudorandom permutation from the previous block, wherein the state variables are modified using at least one of modular 2^n addition and

modular 2^n subtraction wherein n represents the size of a block, and wherein the state variables are modified using a bitwise exclusive or (XOR).

[0058] The method comprises initializing the state variables before encrypting the first plaintext block by randomizing a nonce and padding the nonce in order to facilitate the initialization of the state variables, wherein the initialized state variables are unique from other initialized state variables in a context of a session key, wherein the number of pseudorandom permutations determines the number of state variables, wherein the pseudorandom permutations are at least one of: block ciphers, keyed substitution tables, S-Boxes, and rotors, wherein each pseudorandom permutation is keyed by at least one different key, wherein each pseudorandom permutation is keyed by a same key, wherein a portion of the pseudorandom permutations may be substituted for the inverses of a remaining portion of the pseudorandom permutations, and wherein the pseudorandom permutations and inverse pseudorandom permutations may be arranged in any order.

[0059] The method comprises generating an authentication tag from a combination of the state variables, wherein the generation consists of concatenating the resulting state variables after the encryption of the final plaintext block, wherein the generation consists of concatenating the resulting state variables after the encryption of a chosen plaintext block, wherein the generation consists of concatenating the resulting state variables after the encryption of the final plaintext block, concatenating the initial state variables, and combining the two sets of concatenated variables through an exclusive or (XOR), comprises attaching the authentication tag to a ciphertext message, wherein the number of state variables determines the size of the authentication tag, and comprises modifying the input to a pseudorandom permutation by at least one counter, and initializing the counters before encrypting the first plaintext block by randomizing a nonce.

[0060] In another embodiment of the present invention, an apparatus for encrypting a plaintext message comprises logic to form at least one nonce block from at least one nonce, memory to store at least one state variable, an initializer to set the at least one state variable to at least one initial value, wherein the logic is coupled to the memory and to the initializer, wherein the logic includes at least two pseudorandom permutations to sequentially randomize each nonce block, wherein the logic combines the at least one

state variable with inputs to the pseudorandom permutations, and wherein the logic generates the at least one state variable of a current nonce block from at least one of: state variables of a previous nonce block, outputs from the previous nonce block's pseudorandom permutations, and inputs to the previous nonce block's pseudorandom permutations, wherein the memory stores outputs of final pseudorandom permutations as initial values to use in an encryption or decryption, wherein the memory stores final state variables as initial values for use in an encryption or decryption, wherein the logic adds at least one bit of padding to the nonce to generate equal sized nonce blocks, wherein the number of pseudorandom permutations is equal to the number of nonce blocks and the number of state variables, wherein the pseudorandom permutations are at least one of: block ciphers, keyed substitution tables, S-Boxes, and rotors, wherein a portion of the pseudorandom permutations may be substituted for inverses of a remaining portion of the pseudorandom permutations.

[0061] In a further embodiment of the present invention, a computer readable medium comprising instructions for: receiving at least one plaintext message, wherein the plaintext message forms at least one plaintext block, encrypting said plaintext block by applying 2 or more pseudorandom permutations to each block, modifying an input to the pseudorandom permutations by at least one state variable, modifying the at least one state variable after each plaintext block is encrypted for use in encrypting a next plaintext block, modifying the at least one state variable for a first pseudorandom permutation by an output of a next to last pseudorandom permutation from a previous block, modifying the at least one state variable for a final permutation by an output of the first pseudorandom permutation from the previous block and the at least one state variable for the first pseudorandom permutation from the current block, and modifying the at least one state variable for all other pseudorandom permutations by an output of a preceding pseudorandom permutation from the previous block.

[0062] The computer readable medium comprises instructions for initializing the at least one state variable before encrypting a first plaintext block by randomizing a nonce, modifying the input to a pseudorandom permutation by an internal counter, generating an authentication tag from a combination of the state variables, generating at least one ciphertext block from an output of each plaintext block's final pseudorandom

permutation, wherein the pseudorandom permutations are at least one of: block ciphers, keyed substitution tables, S-Boxes, and rotors.

[0063] Although an exemplary embodiment of the system of the present invention has been illustrated in the accompanied drawings and described in the foregoing detailed description, it will be understood that the invention is not limited to the embodiments disclosed, but is capable of numerous rearrangements, modifications, and substitutions without departing from the spirit of the invention as set forth and defined by the following claims. For example, the capabilities of the invention can be performed fully and/or partially by one or more of the elements. Also, these capabilities may be performed in the current manner or in a distributed manner and on, or via, any device able to provide and/or receive information. Further, although depicted in a particular manner, various modules or blocks may be repositioned without departing from the scope of the current invention. Still further, although depicted in a particular manner, a greater or lesser number of modules and connections can be utilized with the present invention in order to accomplish the present invention, to provide additional known features to the present invention, and/or to make the present invention more efficient. Also, the information sent between various modules can be sent between the modules via at least one of a wireless source, and a wired source and via plurality of protocols.

Claims

What Is Claimed Is:

1. A method for encrypting a plaintext message, comprising:

receiving at least one plaintext message, wherein the plaintext message forms at least one plaintext block;

encrypting said plaintext block by applying 2 or more pseudorandom permutations to each block; and

modifying an input to each said pseudorandom permutation by at least one state variable which is modified for each plaintext block by at least one of previously generated permutation outputs, previously generated permutation inputs, ciphertext, and plaintext.

2. The method of claim 1, comprising generating at least one ciphertext block from the output of each plaintext block's final pseudorandom permutation.

3. The method of claim 1, comprising partitioning the plaintext message into a plurality of equal size plaintext blocks.

4. The method of claim 3, comprising padding the plaintext message to facilitate the equal sized plaintext blocks.

5. The method of claim 1, wherein the modification of the state variables comprises at least one of:

modifying the state variable for a first pseudorandom permutation by an output of a next to the last pseudorandom permutation from the previous block;

modifying the state variable for a final permutation by an output of the first pseudorandom permutation from the previous block and the state variable for the first pseudorandom permutation from the current block; and

modifying the state variables for all other pseudorandom permutations by an output of the preceding pseudorandom permutation from the previous block.

6. The method of claim 5, wherein the state variables are modified using at least one of modular 2^n addition and modular 2^n subtraction wherein n represents the size of a block.

7. The method of claim 5, wherein the state variables are modified using a bitwise exclusive or (XOR).

8. The method of claim 1, comprising initializing the state variables before encrypting the first plaintext block by randomizing a nonce.

9. The method of claim 8, comprising padding the nonce in order to facilitate the initialization of the state variables.

10. The method of claim 8, wherein the initialized state variables are unique from other initialized state variables in a context of a session key.

11. The method of claim 1, wherein the number of pseudorandom permutations determines the number of state variables.

12. The method of claim 1, wherein the pseudorandom permutations are at least one of: block ciphers, keyed substitution tables, S-Boxes, and rotors.

13. The method of claim 1, wherein each pseudorandom permutation is keyed by at least one different key.

14. The method of claim 1, wherein each pseudorandom permutation is keyed by a same key.

15. The method of claim 1, wherein a portion of the pseudorandom permutations may be substituted for the inverses of a remaining portion of the pseudorandom permutations.

16. The method of claim 15, wherein the pseudorandom permutations and inverse pseudorandom permutations may be arranged in any order.

17. The method of claim 1, comprising generating an authentication tag from a combination of the state variables.

18. The method of claim 17, wherein the generation consists of concatenating the resulting state variables after the encryption of the final plaintext block.

19. The method as defined in claim 17, wherein the generation consists of concatenating the resulting state variables after the encryption of a chosen plaintext block.

20. The method of claim 17, wherein the generation consists of concatenating the resulting state variables after the encryption of the final plaintext block, concatenating the initial state variables, and combining the two sets of concatenated variables through an exclusive or (XOR).

21. The method of claim 17, comprising attaching the authentication tag to a ciphertext message.

22. The method of claim 17, wherein the number of state variables determines the size of the authentication tag.

23. The method of claim 1, comprising modifying the input to a pseudorandom permutation by at least one counter.

24. The method of claim 23, comprising initializing the counters before encrypting the first plaintext block by randomizing a nonce.

25. An apparatus for encrypting a plaintext message, comprising:

logic to form at least one nonce block from at least one nonce;

memory to store at least one state variable;

an initializer to set the at least one state variable to at least one initial value;

wherein the logic is coupled to the memory and to the initializer;

wherein the logic includes at least two pseudorandom permutations to sequentially randomize each nonce block;

wherein the logic combines the at least one state variable with inputs to the pseudorandom permutations; and

wherein the logic generates the at least one state variable of a current nonce block from at least one of: state variables of a previous nonce block, outputs from the previous nonce block's pseudorandom permutations, and inputs to the previous nonce block's pseudorandom permutations.

26. The apparatus of claim 25, wherein the memory stores outputs of final pseudorandom permutations as initial values to use in an encryption or decryption.

27. The apparatus of claim 25, wherein the memory stores final state variables as initial values for use in an encryption or decryption.

28. The apparatus of claim 25, wherein the logic adds at least one bit of padding to the nonce to generate equal sized nonce blocks.

29. The apparatus of claim 25, wherein the number of pseudorandom permutations is equal to the number of nonce blocks and the number of state variables.

30. The apparatus of claim 25, wherein the pseudorandom permutations are at least one of: block ciphers, keyed substitution tables, S-Boxes, and rotors.

31. The apparatus of claim 25, wherein a portion of the pseudorandom permutations may be substituted for inverses of a remaining portion of the pseudorandom permutations.