Application of Synchronous Dynamic Encryption System in Mobile Wireless Domains

Hamdy S. Soliman and Mohammed Omari
Computer Science Department
New Mexico Institute of Mining and Technology
801 Leroy Place, Socorro, NM 87801
1(505)835-5170
{hss, omari}@nmt.edu

ABSTRACT

Motivated by the tradeoff between security and efficiency performance parameters that has been imposed on all modern wireless security protocols, we designed a novel security system that gained in both parameters. Our system is based on stream ciphers for their speed, but maintaining a much more solid and proven security. Such security strength stems from the novel deployment of permutation vectors and the data records in the regeneration of the secret key. Moreover, the involvement of the former results in an adaptive and efficient data integrity mechanism that relies on error propagations in the data stream. Simulation results show that our security protocol is much faster than peer mechanisms such as WEP and CCMP. Hence, we anticipate a great opportunity to deploy our system in environments with scarce bandwidth, which are the most vulnerable; specifically the wireless domain.

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Applications (SMTP, FTP, etc.)

General Terms

Performance, Design, Experimentation, Security, Human Factors, Standardization.

Keywords

Mobile network security, dynamic encryption, permutation vectors, flexible integrity, integrity violations, seamless handover.

1. INTRODUCTION

The main advantage of wireless communications is seen in ubiquitous and location-independent computing in restricted spatial domains such as offices, factories, enterprise facilities, hospitals, and campuses [2]. In such environments, wireless networks complement and expand the coverage areas of existing wired networks. Wireless networks are attractive because of their cost-effectiveness in terms of installation and communication. One particular advantage of wireless networks is the fact that they can be quickly installed in an ad hoc configuration, without preplanning and without a supporting backbone network. Even though, many efficient wireless protocols have been developed, however, we are still faced with the great challenge of securing wireless communications efficiently. The rapid progress in wireless communication prompts organizations to develop new security mechanisms that are equivalent to the existing wired-based security protocols. These wired-based protocols presume that the existing wired network jacks are located inside buildings that are already secured from unauthorized access through the use of keys, badge access, etc [5]. In contrast, a wireless access point (AP) may be accessed from off the premises if the signal is detectable.

The 802.11 standard [3] for wireless LAN communications introduced the Wired Equivalent Privacy (WEP) protocol in an attempt to bring the security level of wireless systems closer to that of wired ones. The primary goal of WEP is to protect the confidentiality of user data from eavesdropping. The integration of WEP in wireless network cards helped in its widespread use. Unfortunately, WEP failed to accomplish its security goals, despite deploying the believed- to-be-secure RC4 stream cipher [3][14]. Hence, a temporal key integrity protocol (TKIP) was introduced to strengthen the WEP, albeit with the cost of slowing down the protocol. Finally, the proliferation of the stream cipher approach led to the introduction if a new, faster and more secure block cipher protocol. The Counter-Mode-CBC-MAC-Protocol (CCMP), with the Advanced Encryption System (AES) as its encryption engine and Cipher Block Chaining Message Authentication Code (CBC-MAC) for integrity, definitively replaces the TKIP. CCMP is a data-confidentiality protocol that handles both packet authentication and encryption. The extra time and packet size overhead required by the CCMP motivated us to design a new protocol that returns to the stream cipher approach, yet manifests all of the security strength of CCMP (if not more), with much faster speed.

Motivated by the aforementioned problems in the wireless security techniques, we designed a novel security mechanism that alleviates the existing security and efficiency problems in the wireless domain. Our mechanism is proof that the philosophy of stream ciphers is still sound, if a system is implemented properly. Moreover, gains of security and efficiency are not really out of reach.

The rest of this paper is organized as follows. Section 2 is devoted to WEP protocol, enhancements and deficiencies. In Section 3, we will describe briefly the new 802.11 proposed standard (Counter Mode CBC-MAC Protocol). Our proposed security protocol is illustrated in detail in Section 4, including confidentiality, authentication, and integrity services. Section 5 presents wireless simulation results corresponding to the deployment of the three aforementioned security mechanisms. Section 6 is the conclusion.

2. WIRED EQUIVALENCY PROTOCOL

2.1 WEP Security

The 802.11 standard describes the protocol of wireless local area networks (WLANs). The Wired Equivalent Privacy (WEP) algorithm [4] is used to protect wireless communication from eavesdropping. WEP relies on a secret key that is shared between a supplicant mobile station (e.g., a laptop with a wireless card) and an access point (AP) base station. The secret key is used to encrypt packets before they are transmitted, and an integrity check is used to ensure that packets are not modified in transit by an intruder. The standard does not discuss how the shared key is established. In practice, most installations use a single key that is shared among all mobile stations and access points.

WEP uses the RC4 stream cipher encryption algorithm [4] [9]. A stream cipher operates by expanding a short key into an infinite pseudo-random key stream. The sender applies XOR to the key stream with the plaintext to produce ciphertext. The receiver holds a copy of the same key, and uses it to generate an identical key stream. XORing the key stream with the ciphertext yields the original plaintext. To ensure packet integrity, WEP uses an Integrity Check (IC) field in the packet. To avoid encrypting two plaintexts with the same key stream, an Initialization Vector (IV) is used to augment the shared secret key and produce a different RC4 key for each packet. The IV is also included in the transmitted packet. However, both of these measures are implemented inaccurately, resulting in poor security.

2.2 WEP Vulnerabilities

The first attack on the WEP was due to the limited space of the IV permutations in the RC4 (only 224). In a heavily trafficked network, a passive eavesdropper can intercept all wireless packets with repeated IVs. By XORing two packets that use the same IV, the attacker obtains the XOR of the two plaintext messages [4][15][16]. The resulting XOR can be used to infer the contents of the two messages, given that the IP traffic is often very predictable and includes much redundancy. Such redundancy can be used to eliminate many possibilities for the contents of messages. Further educated guesses about the contents of one or both of the messages can be used to statistically reduce the search space of possible messages, and in some cases it is possible to determine the exact contents, hence cracking the WEP key.

The second attack is also a direct consequence of the problems described previously. If an attacker knows the exact plaintext for one encrypted message, such knowledge can be used to forge valid encrypted packets [4]. The procedure involves constructing a new message, calculating the CRC-32, and performing bit flips on the original encrypted message to change the plaintext to the new message. The basic property is that RC4(X) xor X xor RC4(Y) =Y. This fake packet can be sent to the access point or mobile station, and it will be accepted as a valid packet.

2.3 WEP Enhancements: Temporal Key Integrity Protocol

In order to alleviate the above attacks, the TGi company responded to the need to improve the security of the already-deployed 802.11 equipment [9]. TGi (IEEE 802.11i) has proposed a temporal key integrity protocol (TKIP) as a mandatory security enhancement for 802.11, and patches implementing it will likely be available for most equipment. TKIP is a suite of algorithms wrapping WEP that is used to achieve the best security that can be obtained, given the problem design constraints. The TKIP algorithms are designed explicitly for implementation on legacy hardware, hopefully without unduly disrupting performance. TKIP adds four new algorithms to the WEP: 1) A cryptographic message integrity code, or MIC, called Michael, defeating any forgery attempts; 2) A new IV sequencing discipline, eliminating any replay attacks from the attacker’s arsenal; 3) A per-packet key mixing function, de-correlating the public IVs from weak keys; and 4) A rekeying mechanism, providing fresh encryption and integrity keys, undoing the threat of attacks stemming from key reuse. The TKIP key mixing function can construct at most 216 IVs, which implies that TKIP requires a key-update mechanism operating at least every 216 packets. The instantiation of new temporal keys requires careful coordination, using special rekey messages. The rekey message distributes keying material from which both the station and AP derive the next set of temporal keys. This exchange must itself be secure, or an attacker can compromise the temporal keys by compromising the rekey message.

This general outline of rekeying is necessary but not sufficient to provide TKIP security [9][17]. Other problems must be resolved. One major issue is how to support roaming by stations from one access point to another in large infrastructure deployments. A full reauthentication by the 802.1X authentication server can be too slow to support real-time applications like voiceover IP traffic, so a faster mechanism may be required. This suggests some sort of key passing architecture. Key passing is especially prone to attack [18][19], so design of this type of architecture is quite delicate, an issue that is being currently investigated.

3. THE CCMP PROTOCOL

The Counter-Mode-CBC-MAC-Protocol, like TKIP, addresses all known WEP deficiencies, but without the shackles of already-deployed hardware [13]. The Advanced Encryption System (AES) was selected as its core encryption algorithm.

The use of single-key (block cipher) is mainly to provide confidentiality and integrity, reducing key management overhead and minimizing the time spent computing AES key schedules. Another feature provided by CCMP is maintenance of integrity protection for the plaintext packet header, as well as integrity and confidentiality of the packet payload [17].

3.1 Counter Mode (CTR)

In the Counter Mode (CTR), a counter is encrypted first using a block cipher algorithm, yielding an encryption key K-CTR. Then, the MPDU plaintext (Message Protocol Data Unit) is simply XORed with K-CTR to produce an MPDU cipher. Subsequently, the counter is incremented in order to encrypt the next plaintext.

3.2 CCM mode

A new mode called CCM was designed to meet the expected security criteria. CCM merges two well-known and widely deployed techniques. CCM uses counter mode (CTR) for encryption and the Cipher Block Chaining Message Authentication Code (CBC-MAC) for integrity protection [13][20]. Both algorithms employ only the encryption primitive at both the sender and the receiver.

CCM uses the same key for both confidentiality and integrity. This is normally a dangerous practice, but CCM avoids the pitfalls of this usage by guaranteeing that the space for counter mode never overlaps with that used by the CBC-MAC initialization vector.

3.3 CCMP Protocol

The protocol using CCM has many properties in common with TKIP. Freedom from constraints associated with current hardware leads to a more elegant solution. As with TKIP, CCMP employs a 48-bit IV, ensuring that the lifetime of the AES key is longer than any possible association. In this way, key management can be confined to the beginning of an association and ignored for its lifetime. CCMP uses a 48-bit IV as a sequence number to provide replay detection, just like TKIP.

4. OUR SECURITY MECHANISM: SYNCHRONOUS DYNAMIC ENCRYPTION SYSTEM (SDES)

4.1 Security Design Guidelines

4.1.1 Security Using Stream Ciphers

Stream ciphers are meant to enhance the speed of encryption while maintaining a high level of security, compared to other block cipher techniques [1]. Moreover, they use different keys to protect against cryptanalysis and replay attacks. Hence, we based our security system on the stream ciphering mechanism. Specifically, we involve, among other parameters, the previous key to generate the next key in the key stream, avoiding key duplication.

4.1.2 Protection Against Key Compromise

In an effort to further strengthen our security model, we aim to minimize the effect of any security violation. For example, the compromise of one key results in breaking only one data record. In order to achieve such goal, the key management function involves a random data record (previously sent/received) as another parameter. Thus, obtaining one key would not be enough to generate any previous/next key in the keystream, since the intruder also needs all previously communicated data. The idea of involving the data in key management also protects against attacks that involve detecting key-stream patterns. As in RC4 cryptosystem [6][7],the key stream is periodic; hence obtaining one key instance from a cycle leads to the total compromise of such cycle. The involvement of the data (with its randomness) in our system results in acyclic key stream generation process.

4.1.3 Protection Against "Biased Bytes" Analysis

In order to break the cryptosystem, a cryptanalyst always seeks any flaw in the encryption mechanism (e.g., repeated ciphers, weak encryption keys, low entropy, etc), avoiding brute-force attacks. Since the raw data is involved in the key management, the key's bytes values could inherit the bias from the data itself. Consequently, if the intruder has some knowledge about the nature of the transmitted data (text, satellite images, landscape backgrounds, etc), a possible analysis of particular bytes at certain positions might reveal a key of the keystream. Therefore, it is also desirable to maintain an equidistribution of the key values by involving another parameter in the key generation process.

4.1.4 Protection Against Integrity Violation

In traditional block and stream ciphering, maintaining cipher integrity might involve rigid communication and computation overhead per packet. The Cipher Block Chaining Message Authentication Code (CBC-MAC) and the Cyclic Redundancy Check (CRC) are used in block and stream cipher techniques, respectively [1]. On the other hand, the cipher integrity of our approach is inherent, since data are involved in the synchronized key generation. Hence, any cipher alteration will automatically result in key missynchronization that will propagate to the consecutive keys, based on the aforementioned key generation mechanism. Such a property will allow us the flexibility to check cipher integrity on demand, instead of per packet, alleviating the overhead encountered by other peer mechanisms.

4.2 Theory

One of the most important aspects of stream ciphers is the manufacturing of encryption keys. Modern stream ciphers utilize permutation techniques in the manufacturing of their encryption keys, for better diversion. Our technique still follows the same philosophy with some subtle differences in order to achieve much more solid security that is amenable to mathematical proof.

A permutation describes an arrangement, or ordering, of numbers [8]. Many algorithmic problems seek the best way to order a set of objects, including traveling salesman (the least-cost order to visit n cities), bandwidth (order the vertices of a graph on a line so as to minimize the length of the longest edge), and graph isomorphism (order the vertices of one graph so that it is identical to another). Any algorithm for solving such problems exactly must construct a series of permutations along the way.

There are $n!$ permutations of n items, which grow exponentially to generate all permutations. Numbers like these should calm the urge of anyone interested in exhaustive search and help explain the importance of generating random permutations.

Fundamental to any permutation-generation algorithm is a notion of order, the sequence in which the permutations are constructed, from first to last. The most natural generation order is "lexicographic;" the order in which they would appear if they were sorted numerically. Lexicographic order for $n = 3$ is {1, 2, 3}, {1, 3, 2}, {2, 1, 3}, {2, 3, 1}, {3, 1, 2}, and finally {3, 2, 1}. Although lexicographic order is aesthetically pleasing, there is often no particular reason to use it. Indeed, non-lexicographic orders lead to faster and simpler permutation generation algorithms [8] [11].

The generation of random permutations is an important problem to solve, in order to simplify security algorithms. One way [12] is the following two-line, linear-time algorithm. We assume that *Random(i,n)* generates a random integer between i and n.

```
for i = 1 to n do a[i] = i;            // a =[1, 2, ..., n]
    for i = 1 to n do swap[ a[i], a[ Random(i, n) ];
```

It is not obvious that this algorithm generates all permutations uniformly. However, a security algorithm that is based on such types of generating permutations might be as secure as harder peer algorithms, yet the generation process is still linear.

The generation of permutation vectors can be performed recursively. Given a permutation vector PV, the generation of the next permutation vector PV` is based on PV and some other parameter that provides the randomness. In this section, we will focus on the additional parameter selection, and we will show that the best-case brute force of such a parameter requires exponential computation complexity.

Our goal is to generate a large sequence of permutations (or permutation vectors) such that it is hard to guess their order. In fact, both communication parties can utilize the shared secret key (SK) as a "seed" of the random generation process that can be used in the abovementioned algorithm. Next is our proposed linear algorithm to generate permutation vectors:

```
for i = 1 to n do PV[i] = i;
for i = 1 to n do swap[ PV[i], PV[ SK[i] ];  // 1 ≤SK[i] ≤n
```

In order to prove that breaking the permutation vectors' generation is hard, we need to show that given any input parameters (especially PV and PV`), a cryptanalyst is restricted to the brute-force option only in order to break the random permutation order.

TABLE 1: All possible key combinations, for key size *n* =3

PV	PV`	All Possible values for SK that permutes PV to PV`
[1, 2, 3]	[1, 2, 3]	[3, 2, 1], [1, 3, 2], [2, 1, 3], [1, 2, 3]
[1, 2, 3]	[1, 3, 2]	[3, 1, 1], [2, 3, 1], [2, 1, 2], [1, 2, 2], [1, 3, 3]
[1, 2, 3]	[2, 1, 3]	[3, 3, 1], [3, 1, 2], [2, 3, 2], [1, 1, 3], [2, 2, 3]
[1, 2, 3]	[2, 3, 1]	[1, 3, 1], [1, 1, 2], [2, 2, 2], [3, 1, 3], [2, 3, 3]
[1, 2, 3]	[3, 1, 2]	[1, 1, 1], [2, 2, 1], [3, 2, 2], [3, 3, 3]
[1, 2, 3]	[3, 2, 1]	[2, 1, 1], [1, 2, 1], [3, 3, 2], [3, 2, 3]

Table 1 shows that SK (of size *n*=3) always has possible combinations greater than or equal to 2^{n-1}. Unfortunately, the time complexity to verify this result for higher key size values is exponential with respect to *n*. Yet, it can be proven mathematically, by induction (out of the scope of this paper), that for any permutation vector PV of size *n*+1 and one of its possible permutations PV`, there are at least $2^n$ different keys SK that permutes PV to PV`, given that for any permutation vector *pv* of size *n* and one of its possible permutations pv`, there are at least $2^{n-1}$ different secrete keys sk, of size n, that permutes *pv* to *pv*`.

4.3 SDES Implementation

Based on the above security design guidelines where we explored the main advantages of our system over existing peers, we can confidently state that our system is the most amenable for deployment in the wireless domain. Our system is fast since it is based on non-IV stream ciphers, yet it maintains a very solid security, as shown above. Such a property lends itself easily to the wireless domain where security is much needed (broadcast system), yet resources are more limited than in the wired domain.

Our Synchronous Dynamic Encryption System (SDES) is a stream cipher crypto system based on permutation vector generation. In order to maintain the highest level of security and avoid as much vulnerability as possible, our mechanism minimizes the key exchange process between supplicants (SUP) and access points (AP), as well as the authentication server (AS), in the wireless domain. The main idea is to keep mobile users and the AS/AP in synch at all times, with respect to the secret and encryption keys. Unless a node (SUP/AP) is pre-registered with the AS, it is nearly impossible for a non-member node (e.g., intruder) to have the same corresponding synchronized dynamic secret key that is maintained at the AS. The only explicit node identity verification is carried out once, at the time of the node's first registration with the AS. Any subsequent authentication is implicitly processed, without the overhead of securely exchanging plain keys.

In our system, there are two types of dynamic keys: secret authentication keys (SAK) and secret session keys (SSK). The AS is responsible for generating an initial SAK for every registered station (SUP/AP). Upon securely receiving the initial SAK from AS, the station and the AS use the shared SAK for any subsequent mutual authentication. In case of SUP-AS authentication, once the SUP is initially authenticated by the AS, the AS forwards the SUP's SAK to its associated AP.

On the other hand, the SSK is generated per any communication session between APs, and SUPs as well, for the duration of the session only. In case of an AP-to-AP session, the generation and delivery of the initial SSK is securely generated and carried out by the AS. However, when two supplicants request a secure communication, the AP associated with the source SUP is responsible for generating a SSK and sends it securely to both supplicants. Recall that both SAK and SSK are shared keys; they are involved in the process of shuffling permutation vectors that are used for encryption.

4.4 Encryption/Decryption

The encryption function is simplified in order to minimize the overhead cost at the authentication stage. Thus, a simple XOR is performed between the data record d^t and the corresponding generated permutation vectors PV^t, resulting in a cipher c^t to be transmitted. The decryption function is performed in the same manner of the encryption function. The cipher record c^t is XORed with the same corresponding permutation vector PV^t (generated at the recipient side) producing the original data record d^t. Both communication parties generate a new permutation vector (PV^{t+1}), based on SAK/SSK, to be used in the next encryption/decryption operations, synchronously.

4.5 Key Management

The SAK and SSK each serve as a seed of permutation to generate a stream of encryption permutation vectors. There are three modes to regenerate these keys that tune between different levels of security and corresponding efficiency.

4.5.1 Static shared keys

This option provides a low security profile. The secret key is a "sitting duck" at both communication parties, which makes it vulnerable to key compromising attack. Furthermore, the permutation vectors generation might lead to a constant stream of keys (*PV* and *PV`* are identical), which results in breaking our system using the "known plaintext-ciphertext" attack. However, prolonged experimentation failed to realize the above case (running an experiment for one month, the obtained *PV`*[*i*] was always different than *PV*[*i*], $1\leq i \leq n$). The clear advantage of such a mode is the elimination of the secret key management.

4.5.2 Stream of shared keys

In order to alleviate the static security problem, a second option is to modify the shared key after each data record encryption, for a dynamic key generation. Hence, the implementation of the shared key management process is as follows.

for i = 1 to n do SK[i] = (SK[i] + PV[i]) modulo n

Then, the shared key is not as easy a target for cryptanalysts, as in the above mode. Moreover, experimentally, the shared key generation is not vulnerable to "biased byte" analysis since the involved permutation vector is a good source of byte diversity. Hence, the increase of security costs only one addition operation. Moreover, it is possible to generate a stream of keys in advance (offline), speeding up the process of secure communication. However, in case of opening more than one session between the same two authenticated clients, all sessions generate the same stream of shared keys, lacking security independence: breaking one session breaks all.

4.5.3 Dynamic stream of shared keys

For ultimate security, the communicated data is involved in the shared key generation, as a third mode. The new shared-key-management process is as follows:

for i = 1 to n do SK[i] = (SK[i] + PV[i] + d[i]) % n;

The idea behind involving the data is to provide a different set of generated shared keys for different sessions (assuming that they are of different data to communicate), eliminating the second option security violation above. In addition to the permutation vector, the incorporation of the data in the key management process adds another dimension of diversity, increasing secret key entropy. Another big advantage is that the data integrity is equivalent to the key integrity; i.e., if the communicated cipher is altered, the generated shared keys at both sender and recipient sites will missynchronize. The price of the additional security and advantageous data integrity is the infeasibility of the efficient offline generation of keys, and two extra addition operations.

4.6 Authentication

At the network initialization stage, all APs go through a registration process authenticating themselves with the AS (once in their life cycle). Then, every AP is authenticated with its neighboring APs via the AS that generates and transmit a private SSK to each pair of authenticated APs. When a supplicant joins the network, with the pair (MAC[1], SAK) installed in its wireless card, it sends a first-authentication request to its local AP. The AP forwards the SUP's request to the AS in order to authenticate the supplicant, and transfer its newly generated SAK back to the AP. Figure 1 explains in detail the protocol sequence of the supplicant initial authentication.

1. SUP→AP : SUP requests a secure connection by generating a nonce N and sending (MAC, E(N|N)) to the AP.
2. AP→AS : AP forwards (MAC, E(N|N)) to the AS.
3. At the AS : Search for the local SAK corresponding to the received MAC and decrypt E(N|N). If the decrypted two nonces are different, abort.
4. AS→AP : AP sends the supplicant's SAK, encrypted with AP's SAK.
5. AP→SUP :AP encrypts N using the received supplicant's DAK, and send E(N) to the supplicant.
6. At the SUP : SUP decrypts E(N) using the local SAK. If $N \neq D(E(N))$, abort. Otherwise, the supplicant is mutually authenticated with AP.
E(N|M) represents the encryption of the concatenation of N and M

Figure 1: Initial authentication between the supplicant and the access point.

Notice that the SUP authentication with the AS is done only once; subsequent authentications are performed directly with the associated AP. Only in case a supplicant remains out of range with its AP for long time, would it need to reauthenticate with the AS again.

4.7 Handover

When the communication signal between the supplicant and its currently associated AP (say AP_1) gets weak, the supplicant roams for another AP (say AP_2) of stronger signal. Then, the supplicant sends a handover request to AP_1 including AP_2's info. Usually, AP_1 and AP_2 are adjacent and wired; therefore, they are already preauthenticated to each other via the AS. Following the rule of "a trusted by a trusted is trusted," AP_1 sends a secure handover request to AP_2 including the supplicant's authentication information. This AP-AP communication is secured via their private shared SSK. Then, AP_1 sends a secure "integrity check" message to the supplicant in order to check the previously received data integrity. Figure 2 explains in detail the protocol sequence of the handover process.

1. Initialization: Triggered by a weak signal from the current AP (AP_1), the SUP starts roaming for another AP with a stronger communication signal (say, AP_2).
2. SUP→AP_1 : SUP securely sends a handover request including AP_2's info.
3. AP_1→SUP : AP_1 securely sends an "integrity check" message, and verifies the integrity of the previously sent data.
4. AP_1→AP_2 : AP_1 securely sends the SUP's SAK (integrity verified one) to AP_2.
5. AP_2→SUP : AP_2 initiates the authentication procedure (no need to communicate with the AS).

Figure 2: The seamless secure handover of the supplicant.

[1] Mac address

4.8 Integrity

In SDES, altering communicated data records would propagate to the remaining decrypted data, interfering with the key-key synchronization, since the key regeneration is a function of the altered data. In a highly vulnerable environment, error detection and/or correction could be accomplished using checksum, incurring extra bytes (bandwidth) of permanent overhead. The SDES introduces a much more flexible and powerful mechanism to replace the checksum-like mechanisms where errors are detected via key missynchronization.

Our system divides each session into sections of size *R* records, with *R*-1 data records and a duplicate of the last record. The receiver validates the integrity of the entire section by simply verifying the equality of the last two received data records, ignoring the duplicate record. If an integrity violation is detected, the sender needs to encrypt and retransmit the previous *R* data records, which may degrade the performance of our mechanism in hostile environments. Still, an advantage of our mechanism is the flexibility of adjusting R based on the environment hostility.

Hence, the communication throughput is (*R*-1)/*R*, approaching 99% for large *R*, in a peaceful environment, whereas the efficiency of other mechanisms that utilize CRC is about 80% (e.g., in 128-block ciphers, using 32-bit CRC field). Moreover, our mechanism has the huge advantage of detecting many more violations, such as cipher shuffling, injection, and deletion and session hijacking. Other mechanisms will incur a huge overhead (e.g., adding CBC mode) in order to protect against the same set of cipher attacks.

In hostile environments, our mechanism tunes *R* based on the violation probability. Let *p* be the probability of violating the integrity of one transmitted cipher. In order to analyze the successful transmission of *R* records (*R*-1 regular data records, and one record for integrity checking process), we need to calculate the expected value of the total transmitted and retransmitted records in function of *p* and *R*.

5. SIMULATIONS

We designed a wireless network simulator that implements several security models in a wireless infrastructured environment. The simulator was developed using Java Development Kit JSDK 1.5.0. In addition, a very useful library of generating random variable of different distributions was downloaded and added to the kit (refer to [10] for more detail of the Renesys Raceway educational version).

In order to truly demonstrate the performance of every technique, we preferred to limit the number of access points, network bottle-neck, to two. Also, the wireless area is limited to a dimension of 100×100 (unit^2), with 256 wireless nodes that are confined to move in that area, each with a maximum velocity of 5 (units/sec).

For more accurate results, we chose 12 different probability distribution scenarios for packet generation rates, with a mean $m = 50$ msec, to ensure reasonable utilization of resources. These distributions include:

1. The fixed rate m.
2. The Uniform distribution, with packet rates ranging between 0 and 2m.
3. The Exponential distribution, with $\lambda = 1/m$.
4. The Beta distribution, with two scenarios: ($\alpha = 0.5$, $\beta = 0.7$) and ($\alpha = 0.7$, $\beta = 0.5$).
5. The Gamma distribution, with two scenarios: ($\alpha = 0.5$, $\lambda = 1/m$) and ($\alpha = 2$, $\lambda = 1/m$).
6. The Normal distribution, with the mean m.
7. The Pareto distribution, with K = 1/m, and $\alpha = 1.0$.
8. The Pareto2 distribution, with K =1.0, and $\alpha = 1/m$, and $\mu = 1.0$.
9. The CHI-Square distribution, with freedom = m.
10. The Logarithmic distribution, with probability = 0.85.

In addition, the simulator challenges the security technique in question with different integrity violation rates. In fact, the simulator alters the outgoing packet randomly, following a uniform distribution.

Our simulator implements the three security models WEP, CCMP, and SDES including their confidentiality, authentication and integrity checking mechanisms. Each experiment is run 20 times, with a simulation confidence of 96.37% ± 1.78%.

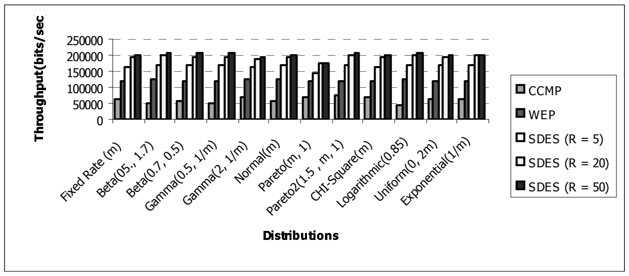

Figure 3: Different security mechanisms' throughputs with no integrity violations.

The first set of experiments targets the efficiency of every security model in a network that is considered to be very immune to integrity violations. The experiment results (Figure 3) shows that SDES outperformed both the fast WEP and the secure CCMP in terms of throughput. Moreover, when a higher round size of integrity checking process (*R*) is selected, a better performance was achieved, as we have expected in the above section analysis.

In the second set of experiments, we challenged the simulated systems with an integrity violation rate of 0.001%. The experimental results (Figure 4) show that SDES still outperformed both WEP and CCMP in terms of useful throughput. Also, when a higher round size of integrity checking process was selected, a better performance was achieved.

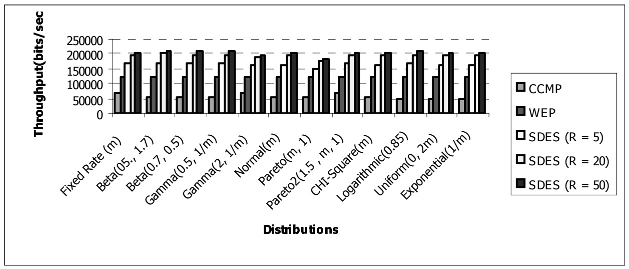

Figure 4: Different security mechanisms' throughputs in a hostile environment (violation probability = 0.001%).

6. CONCLUSION

A novel approach of stream cipher key management is introduced in this paper. It possesses the high speed of stream ciphers, while maintaining a high level of security that compels the intruder to brute-force a large key space with a time complexity of $\Omega(2^n)$. Unique to our approach is its recursive generation of a secret key per each record (different from the encryption key), involving the data and the encryption key; this led to a flexible and much more efficient data integrity mechanism compared to other, peer techniques. Moreover, the encryption key is recursively generated via a simple permutation based on the secret key, which led to the above exponential complexity. Hence, our security system revives the misimplementation of the stream ciphering concept, from the insecure WEP to the inefficient TKIP.

Simulation results showed that our mechanism has dramatically enhanced network throughput; up to fourfold compared to peer wireless security mechanisms. In addition, the dynamics of our data integrity checking process showed much higher protocol efficiency even with a considerable degree of environment hostility. Based on the obtained results, our protocol is the most amenable to be deployed in the wireless domain, which aids in providing quality of service in a secure network environment.

7. REFERENCES

[1]A. Menezes, P. Van Oorschot and S. Vanstone, Handbook of Applied Cryptography. CRC Press, Inc. 1997.

[2] Se Hyun Park, Aura Ganz, Zvi Ganz. "Secure Protocol for 802.11 Wireless Local Area Network". Mobile Networks and Applications, September 1998, Volume 3 Issue 3

[3] L. M. S. C. of the IEEE Computer Society. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE Standard 802.11, 1999 Edition, 1999.

[4]http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html (Insecurity of the WEP algorithm).

[5] Dell Inc. "Wireless Security in 802.11 (WI-FI) Networks". Dell White Papers, January 2003.

[6] Rick Wash, Lecture Note on Stream Ciphers and RC4. http://www.crimelabs.net/docs/stream.pdf , unpublished.

[7] Knudsen, L., Meier, W., Preneel, B., Rijmen, V., Verdoolaege, S. "Analysis method for (alleged) RC4". In Ohta, K., Pei, D., eds.: Advances in Cryptology, Proc Asiacrypt '98. Volume 1514 of LNCS., SpringerVerlag (1998)

[8] A. Nijenhuis and H. Wilf. "Combinatorial Algorithms for Computers and Calculators." Academic Press, Orlando FL, second edition, 1978.

[9] Jesse Walker, "802.11 Security Series, Part II: The Temporal Key Integrity Protocol (TKIP)," Network Security Architect, Platform Networking Group Intel Corporation. (refer http://cedar.intel.com/media/pdf/security/80211_part2.pdf)

[10] Renesys Raceway. "Java Package for Random Variable Generations: Educational Version". https://gradus.renesys.com/exe/Raceway

[11] S. Skiena. "Implementing Discrete Mathematics." Addison-Wesley, Redwood City, CA, 1990.

[12] R. Sedgewick. "Permutation Generation Methods." Computing Surveys, 9:137-164, 1977.

[13] Communication of the ACM: Wireless Network Security. Vol. 46, No. 5, May 2003.

[14] Adam Stubblefield, John Ioannidis, Aviel D. Rubin. "A key recovery attack on the 802.11b wired equivalent privacy protocol (WEP)." ACM Transactions on Information and System Security (TISSEC), May 2004.

[15] Nikita Borisov, Ian Goldberg, David Wagner. "Intercepting mobile communications: the insecurity of 802.11". Proceedings of the 7th annual international conference on Mobile computing and networking. July 2004.

[16] Yasir Zahur, T. Andrew Yang. "Wireless LAN security and laboratory designs." Journal of Computing Sciences in Colleges, January 2004.

[17] Nancy Cam-Winget, Russ Housley, David Wagner, Jesse Walker. "Wireless networking security: Security flaws in 802.11 data link protocols." Communications of the ACM, May 2003.

[18] Bob Briscoe, Ian Fairman. "Nark: receiver-based multicast non-repudiation and key management". Proceedings of the 1st ACM conference on Electronic commerce. November 1999.

[19] Eli Biham, Michel Boyer, P. Oscar Boykin, Tal Mor, Vwani Roychowdhury. "A proof of the security of quantum key distribution (extended abstract)." Proceedings of the thirty-second annual ACM symposium on Theory of computing, May 2000.

[20] Phillip Rogaway, Mihir Bellare, John Black, Ted Krovetz. "Cryptosystems: OCB: a block-cipher mode of operation for efficient authenticated encryption." Proceedings of the 8th ACM conference on Computer and Communications Security. November 2001.