Wired Equivalent Privacy dan Metode Kriptanalisisnya

Arie Minandar Anggiat – NIM : 13503074

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if13074@students.if.itb.ac.id

Abstraksi

Wired Equivalent Privacy (WEP) merupakan protokol keamanan untuk komunikasi dalam *Wireless Local Area Network* (*Wireless* LAN). WEP dispesifikasi dalam standar IEEE 802.11 (Wi-fi). WEP dirancang untuk memberikan keamanan yang setara dengan *wired LAN*. Pada *wired LAN*, jaringan diproteksi secara fisik, misalnya dengan pembatasan akses gedung. Sedangkan pada *Wireless* LAN, komunikasi dilakukan dengan menggunakan gelombang radio sehingga data yang ditansmisikan dapat ditangkap oleh siapapun yang berada dalam jangkauan transmisi.

Pengamanan yang dilakukan oleh WEP adalah dengan menggunakan enkripsi data yang akan ditransmisikan. Dengan enkripsi, walaupun data yang ditransmisikan dapat ditangkap oleh orang yang tidak berhak, namun data tersebut tidak dapat dimengerti oleh orang lain selain pengirim dan penerima, dalam hal ini *client* dan *Access Point* (AP).

Walaupun demikian, pengaman yang diakukan pada WEP memiliki beberapa celah. Celah-celah ini dapat digunakan untuk memperoleh kunci ataupun melakukan serangan lainnya tanpa perlu terlebih dahulu mengetahui kunci.

***Kata kunci**: Wired Equivalent Privacy, WEP, encryption, decryption, enkripsi, dekripsi, RC4, wireless security, IEEE 802.11*

1. Pendahuluan

Teknologi jaringan komputer berkembang dengan pesat seiring dengan meningkatnya kebutuhan penggunaan akan teknologi ini. Jaringan komputer dapat dibedakan berdasarkan cakupannya. Jaringan komputer yang mencakup area lokal dengan jarak fisik antar *node* yang relatif dekat, seperti misalnya di rumah, di kantor, ataupun di universitas, dikenal dengan nama *Local Area Network* (LAN).

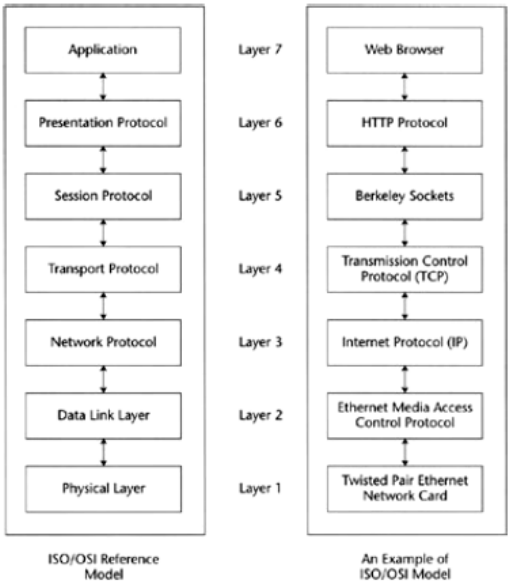
Wireless LAN merupakan LAN yang menggunakan gelombang radio frekuensi tinggi sebagai pengganti kabel untuk komunikasi antar *node*-nya. *Wireless LAN* dispesifikasikan dalam kelompok standar IEEE 802.11 atau yang lebih dikenal dengan Wi-fi. Karena tidak menggunakan kabel, maka *wireless LAN* memungkinkan mobilitas yang tinggi bagi para penggunanya. Selain itu, pembangunan infrastruktur *wireless* LAN juga menjadi lebih mudah dan tidak memerlukan biaya besar.

Namun di sisi lain, karena transmisi data dilakukan dengan menggunakan gelombang radio, maka masalah keamanan pada komunikasi *wireless* tidak lagi sesederhana bila dibandingkan dengan jaringan kabel. Apabila pada jaringan kabel, pengamanan fisik saja sudah dirasakan cukup (seperti misalnya dengan pembatasan akses gedung), maka pada jaringan *wireless* pembatasan fisik merupakan hal yang mustahil untuk dilakukan. Gelombang radio yang digunakan pada jaringan *wireless* dapat menembus tembok-tembok bangunan sehingga segala informasi yang dipertukaran melalui gelombang radio ini dapat diakses oleh siapapun, bahkan oleh orang yang tidak berhak.

Oleh karena itu, suatu mekanisme pengamanan lain perlu dilakukan. Mekanisme pengamanan yang dirasakan paling tepat dan memungkinkan adalah enkripsi. Dengan enkripsi, meskipun transmisi mengalami kebocoran dan diakses oleh orang yang tidak berhak, informasi yang terkandung di dalamnya tidak dapat diketahui tanpa kunci yang tepat. Salah satu protokol enkripsi yang digunakan dan juga merupakan protokol yang pertama pada jaringan *wireless* adalah *Wired Equivalent Privacy* (WEP). Sesuai dengan namanya, WEP diharapkan memberikan keamanan yang setara dengan jaringan kabel

atau *wired LAN*. Protokol ini sudah dispesifikasikan dalam standar IEEE 802.11 (Wi-fi), walaupun pada kenyataannya, pemanfaatan WEP masih bersifat optional.

2. Wired Equivalent Privacy (WEP)



Gambar 1 Model OSI [KHA03]

Protokol WEP diimplementasikan pada level *data link* (Gambar 1) oleh semua perangkat yang disertifikasi oleh *Wireless Ethernet Compatibility Alliance* (WECA). Protokol ini menyediakan proses otentikasi dan enkripsi untuk perangkat *wireless*.

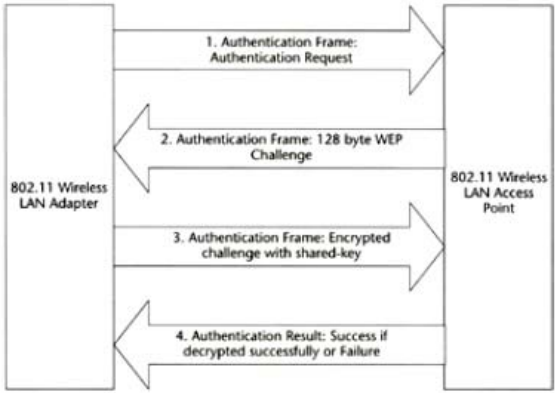
Otentikasi

Standar IEEE 802.11 menspesifikasikan dua mode proses otentikasi, yaitu *open system authentication* dan *shared key authentication*. Pada mode *open system authentication* atau yang dikenal juga dengan *null authentication*, tidak ada proses otentikasi yang dilakukan. Perangkat *wireless* apapun dapat melakukan asosiasi dengan *access point* dan membaca *traffic* jaringan dalam *plaintext*. Pada mode ini, aspek privasi dan keamanan tidak diperhatikan.

Sedangkan pada mode *shared key authentication*, proses otentikasi dengan menggunakan *shared key* dilakukan. Implementasi mode ini dilakukan dengan menggunakan WEP. Dengan menggunakan WEP, hanya client yang memiliki *shared key* yang sesuai saja yang dapat berkomunikasi dalam jaringan.

Proses otentikasi pada *shared key authentication* adalah sebagai berikut (Gambar 2):

1. Perangkat *wireless* mengirimkan *request* untuk melakukan otentikasi kepada *access point*.
2. Ketika *access point* menerima *request*, *access point* tersebut mengirimkan *challenge* dengan ukuran 128 *byte* kepada perangkat yang ingin melakukan otentikasi.
3. Perangkat *wireless* akan mengenkripsi *challenge* tersebut dengan menggunakan *shared key* yang dimilikinya dan mengirimkannya pada *access point*. Kemudian, *access point* melakukan dekripsi terhadap *challenge* terenkripsi yang dikirimkan oleh perangkat *wireless* tersebut dan membandingkannya dengan nilai *challenge* yang dikirimkan sebelumnya.
4. *Access point* akan mengirimkan hasil proses otentikasi pada perangkat *wireless*.



Gambar 2 Shared Key Authentication [KHA03]

Enkripsi

Untuk enkripsi, WEP menggunakan *stream cipher* RC4 yang diciptakan oleh Ron Rivest dari RSA Security, Inc. Algoritma enkripsi RC4 merupakan algoritma enkripsi kunci simetri, yang menggunakan kunci enkripsi dan kunci dekripsi yang sama. Penggunaan algoritma enkripsi RC4 dalam aplikasi sendiri sangat banyak. Selain pada WEP, RC4 juga digunakan pada *Secure Sockets Layer* (SSL) dan *Wi-fi Protect Access* (WPA).

RC4 membangkitkan aliran bit *pseudorandom*, yang lebih dikenal dengan istilah *keystream*, berdasarkan pada kunci yang diterimanya (*shared key*). *Keystream* yang dibangkitkan ini kemudian digunakan untuk mengenkripsi

plaintext melalui operasi XOR. Proses dekripsi dilakukan dengan cara yang serupa. *Keystream* dibangkitkan dan di-XOR-kan dengan *ciphertext*.

Standar IEEE 802.11 menyediakan dua buah skema pendefinisian kunci WEP yang digunakan pada *wireless LAN*. Pada skema yang pertama, *shared key* digunakan bersama-sama oleh seluruh perangkat pada suatu sistem *wireless LAN*. Ketika *client* mendapatkan kunci yang tepat, maka *client* tersebut dapat turut berkomunikasi dalam sistem tersebut. Kekurangan pada skema ini terletak pada penggunaan kunci yang sama yang mempermudah dilakukannya kompromi terhadap sistem. Pada skema yang kedua, untuk setiap komunikasi yang dibangun antara dua buah perangkat, digunakan *shared key* yang berbeda. Skema ini jauh lebih aman bila dibandingkan dengan skema pertama. Namun seiring dengan meningkatnya jumlah perangkat yang digunakan, pendistribusian penggunaan kunci menjadi lebih rumit.

WEP menggunakan *Initialization Vector* (IV). IV merupakan rangkaian karakter yang dibangkitkan secara random dan digunakan bersama-sama dengan *shared key* dalam melakukan enkripsi. Tidak seperti *shared key*, yang nilainya senantiasa tetap, nilai IV berganti-ganti secara periodik. Penggabungan *shared key* dengan nilai IV yang baru akan menghasilkan kunci yang berbeda dengan kunci sebelumnya. Penggunaan IV mengakibatkan *lifetime* dari *shared key* menjadi lebih panjang. Umumnya, nilai IV yang digunakan berbeda untuk setiap pesan yang dikirimkan, dan karena IV juga ikut dikirimkan bersama dengan pesan, maka proses dekripsi dapat dilakukan oleh penerima pesan yang memiliki *shared key*.

WEP standar (64-bit) menggunakan *shared key* sepanjang 40-bit dan IV dengan panjang 24-bit. Pada perkembangannya, panjang *shared key* yang digunakan bervariasi, antara lain WEP 128-bit yang menggunakan *shared key* sepanjang 104-bit dan WEP 256-bit dengan panjang *shared key* 232-bit. Sedangkan panjang IV yang digunakan pada WEP 128-bit dan WEP 256-bit tidak mengalami perubahan, yaitu sepanjang 24-bit.

Untuk lebih jelasnya, prosedur enkripsi pada WEP terlihat sebagai berikut:

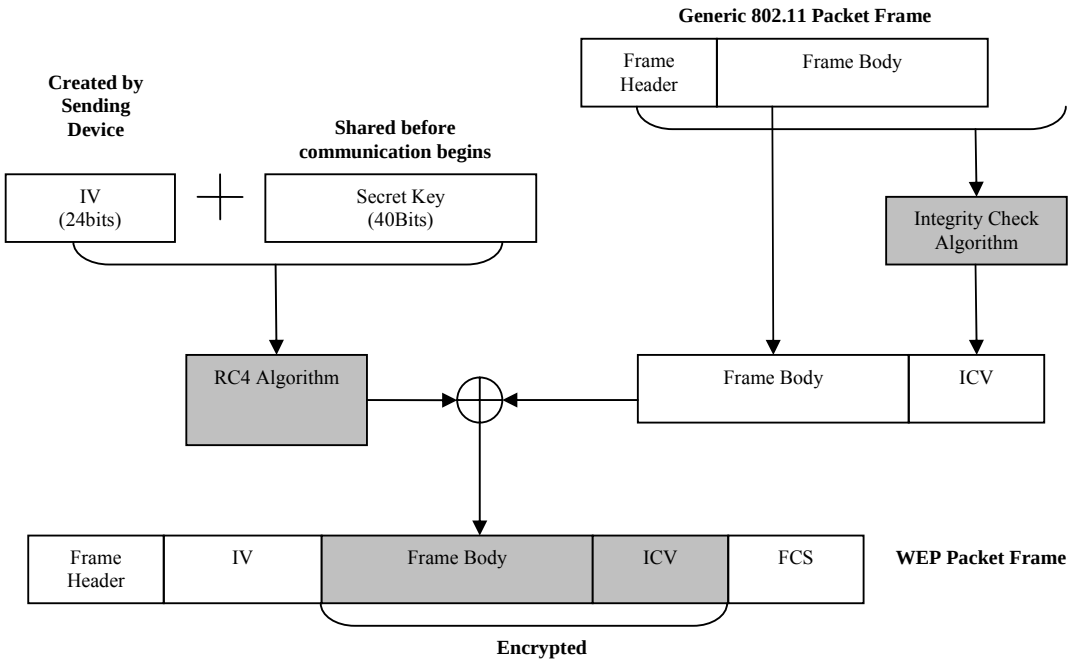
1. Pembangkitan kunci enkripsi dari *shared key*.
   Shared key sepanjang 40-bit digabungkan dengan 24-bit *Initialization Vector (IV)* yang dibangkitkan secara random. Hasilnya berupa kunci sepanjang 64-bit. Kunci ini kemudian digunakan sebagai masukan bagi algoritma RC4 yang menghasilkan rangkaian kunci enkripsi.
2. Enkripsi data dengan menggunakan kunci enkripsi.
   Cyclic Redundancy Check (CRC) merupakan algoritma yang digunakan untuk menjaga integritas data. Algoritma ini memiliki fungsi untuk menjaga data dari modifikasi ketika data ditransmisikan. CRC dihasilkan dari proses komputasi terhadap bit-bit pada pesan yang akan dikirimkan. Hasil komputasi ini kemudian ditambahkan pada bagian terakhir pesan sebelum ditransmisikan. Penerima dapat mendeteksi adanya kesalahan pada pesan yang diterima dengan melakukan kalkulasi untuk menghasilkan CRC yang baru dan membandingkan nilainya dengan CRC pada pesan.

   Operasi CRC 32-bit dilakukan terhadap pesan yang akan dikirimkan. Hasil operasi ini akan menghasilkan 4 *byte checksum*. *Checksum* ini kemudian ditambahkan pada akhir pesan. Selanjutnya, pesan baru ini di-XOR-kan dengan menggunakan rangkaian kunci yang dibangkitkan oleh algoritma RC4.

   Pesan terenkripsi ini kemudian dikirimkan bersama dengan IV yang ditambahkan di awal pesan (Gambar 3).

Proses dekripsi dilakukan sebagai berikut:

1. Dekripsi data
   24-bit pertama pesan yang merupakan IV (yang digunakan pada proses enkripsi) digunakan bersama dengan *shared key* untuk membangkitan rangkaian kunci dekripsi. Operasi XOR antara kunci yang dibangkitkan ini dengan *ciphertext* menghasilkan *plaintext*.
2. Otentikasi pesan yang diterima
   Kalkulasi CRC atau *Integrity Check Value* (ICV) dilakukan terhadap bit-bit pesan. Hasilnya kemudian dibandingkan dengan CRC pada bagian akhir pesan. Apabila hasil perhitungan CRC tidak sesuai, maka pesan yang diterima tidak konsisten.



Gambar 3 Enkripsi pada WEP

3. Kelemahan pada WEP

Kelemahan utama WEP terletak pada proses pembangkitan *keystream* RC4. Pembangkitkan *keystream* dilakukan dengan menggunakan *shared key* sepanjang 40-bit yang digabungkan IV sepanjang 24-bit. Panjang *shared key* yang hanya 40-bit ini sangat beresiko terhadap serangan *brute force*. Selain itu, ukuran IV juga dirasakan terlalu pendek. IV 24-bit hanya memungkinkan dibangkitkannya 16.777.216 kunci enkripsi untuk suatu *shared key*. Dengan kunci sejumlah itu, pada jaringan umumnya, perulangan kunci akan terjadi setelah 5-8 jam. Sedangkan pada jaringan yang sibuk, perulangan dapat terjadi dalam hanya beberapa jam saja. Perulangan IV yang digunakan, mengakibatkan penentuan korelasi antara *plaintext* dan *ciphertext* menjadi semakin mudah.

Bagaimana cara IV dibangkitkan bervariasi pada setiap implementasi. Beberapa perangkat menggunakan *counter* sebagai nilai IV. Sedangkan beberapa perangkat lainnya membangkitkan nilai IV secara random. Nilai IV yang dibangkitkan secara random umumnya mengalami perulangan setelah 5000 kali pembangkitan. Eksploitasi terhadap kelemahan ini dipublikasikan oleh Scott Fluhrer, Itsik Mantin and Adi Shamir pada tahun 2001.

Selain itu, implementasi RC4 pada WEP diketahui memiliki kunci yang lemah. Dengan kunci yang lemah, korelasi antara kunci dengan cipher menjadi semakin besar. Apabila suatu paket dienkripsi dengan menggunakan kunci yang lemah, 3 *byte* pertama dari kunci diperoleh dari IV yang ditransmisikan sebagai *plaintext* bersama dengan paket. Dari sekitar 16 juta nilai IV yang dibangkitkan, terdapat sekitar 9 ribu nilai IV yang lemah. Kelemahan ini dapat dieksploitasi dengan menggunakan serangan secara pasif. Dalam serangan ini, *attacker* harus menangkap paket-paket yang ditransmisikan sebanyak mungkin. Setelah itu, paket-paket dengan nilai-nilai IV yang lemah, yang akan berkorespondensi dengan kunci yang lemah, dapat diperoleh. Kemudian, *attacker* perlu mencoba sejumlah kecil kemungkinan-kemungkinan kunci dan paket untuk menentukan kunci yang digunakan. Terdapat suatu eksperimen yang membuktikan bahwa untuk menentukan kunci WEP sepanjang 104-bit hanya diperlukan 2 ribu hingga 4 ribu paket. Dalam jaringan enterprise, jutaan paket dipertukarkan dalam jaringan setiap harinya. Karena itu, waktu yang dibutuhkan untuk memecahkan kunci WEP 128-bit tereduksi menjadi beberapa hari saja.

Kedua kelemahan WEP seperti yang dijelaskan di atas dapat dieksploitasi dengan serangan pasif. Maksudnya, interaksi langsung antara *attacker* dengan *client* maupun *access point* tidak diperlukan. Selanjutnya, pada bahasan berikut ini dijelaskan akan serangan-serangan aktif yang dapat dilakukan pada WEP:

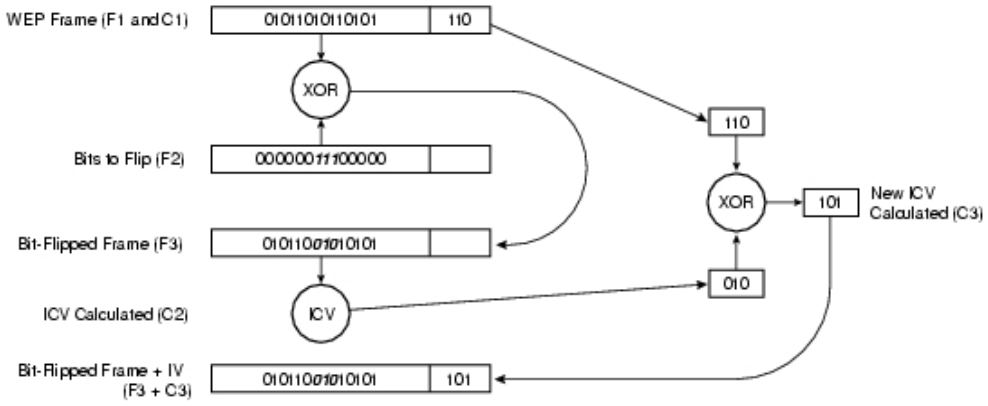
1. Serangan *bit-flipping*:

Algoritma *integrity check* yang digunakan untuk membangkitkan *checksum* atau ICV (*Integrity Check Value*), yaitu algoritma CRC32, tidaklah aman bila ditinjau dari sudut pandang kriptografi. CRC umumnya digunakan untuk mendeteksi terjadinya *bit-error* pada saat transmisi dengan menggunakan komputasi sederhana. Namun, CRC tidak dapat diandalkan untuk benar-benar melakukan otentikasi terhadap integritas pesan. Berikut ini adalah skenario serangan terhadap ICV (Gambar 4):

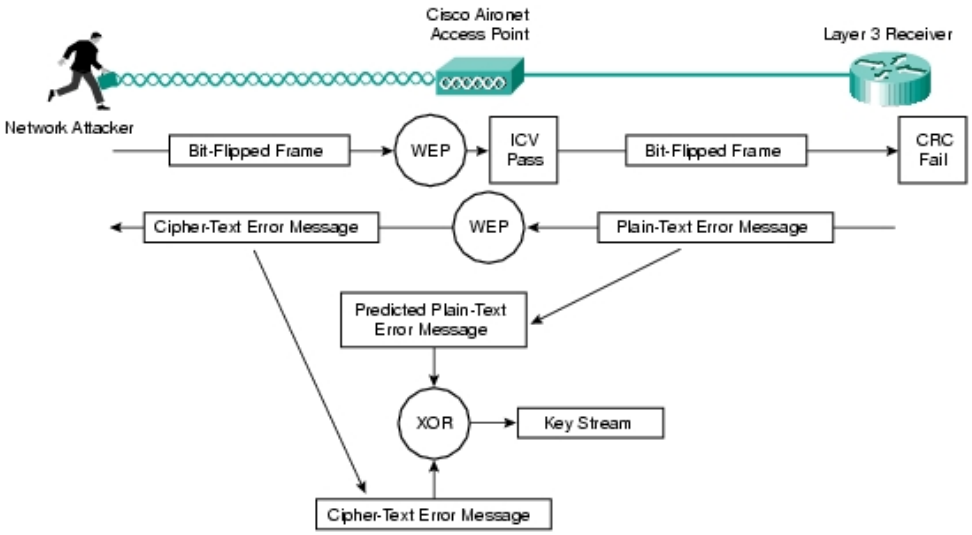
a. Suatu frame (F1) mempunyai ICV (C1).
b. Sebuah frame baru dibangkitkan, yaitu F2, dengan panjang sama dengan F1 dan nilai bit tertentu.
c. Frame F3 dapat diperoleh dengan melakukan proses XOR antara F1 dan F2.
d. ICV untuk F3 (C2) dikalkulasi.
e. ICV C3 dibangkitkan dengan melakukan XOR terhadap C1 dan C2.

Kelemahan ini dapat dimanfaatkan pada serangan bit-flipping, yang dijelaskan sebagai berikut (Gambar 5):

a. *Attacker* melakukan sniffing terhadap jaringan.
b. *Attacker* melakukan intersepsi terhadap paket yang dienkripsi dengan menggunakan WEP
c. *Attacker* melakukan serangan bit-flipping terhadap paket tersebut dan melakukan melakukan modifikasi *checksum* (ICV), seperti yang telah dijelaskan di atas.
d. *Attacker* mengirimkan frame yang telah dimodifikasi kepada *receiver*.
e. *Receiver* (*client* ataupun *access point*) menerima *frame* dan melakukan perhitungan ulang *ICV* berdasarkan pada konten frame.
f. *Receiver* membandingkan ICV yang dihitung dengan ICV pada frame. Karena nilai ICV sesuai, maka *receiver* menerima *frame*.
g. *Receiver* melakukan proses dekapsulasi *frame* dan memproses paket *layer* 3.
h. Karena bit pada paket layer 3 ter-*flip*, proses *checksum* layer 3 gagal.
i. *Stack* IP pada *receiver* membangkitkan pesan *error* yang mudah diprediksi.
j. Pesan *error* dienkripsi dan dikirimkan oleh *receiver*.
k. *Attacker* melakukan *sniffing* untuk menangkap pesan *error* yang terenkripsi.
l. Berdasarkan pada pesan *error* yang diketahui dan pesan *error* yang terenkripsi, *attacker* dapat menurunkan *keystream*.



Gambar 4 Kelemahan ICV

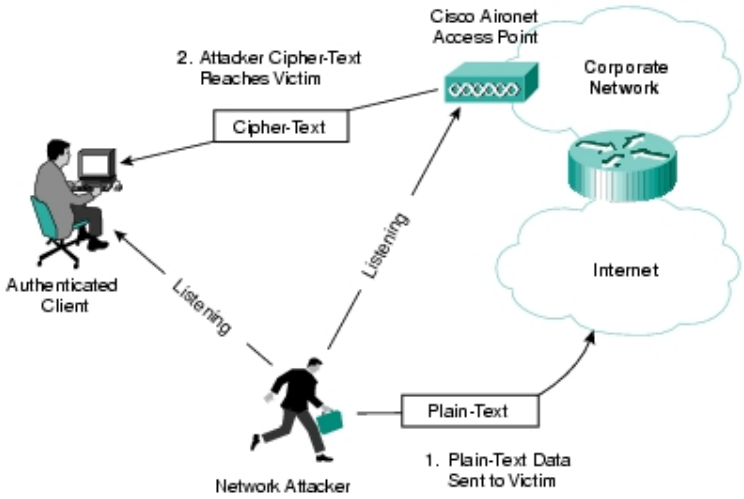


Gambar 5 Serangan Bit-flipping

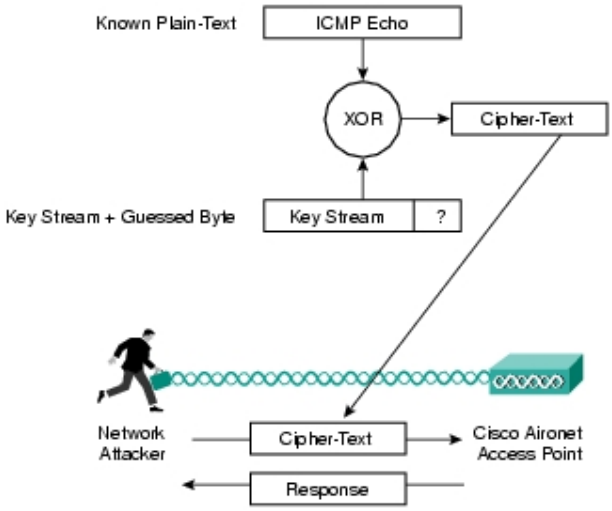
2. Serangan *Initialization Vector* (IV) *replay*

Serangan ini dilakukan pada fakta bahwa IV dan kunci WEP dapat digunakan ulang secara terus menerus untuk membangkitkan keystream yang cukup besar. Langkah-langkah yang dilakukan adalah sebagai berikut (Gambar 6):

a. Pesan dengan *known plaintext* dikirimkan kepada *client wireless* LAN. Pesan ini dapat diperoleh dari serangan *bit-flipping* yang telah dijelaskan sebelumnya ataupun pesan lain yang memiliki pola tertentu, misalnya *email*.
b. *Attacker* melakukan *sniffing* untuk mendapatkan *ciphertext* yang memiliki *known plaintext*.
c. *Attacker* menemukan frame yang dimaksud dan menurunkan *keystream*.
d. *Attacker* dapat melakukan ekspansi *keystream* dengan menggunakan IV (atau pasangan kunci WEP) tersebut.



Gambar 6 Serangan *Initialization Vector* (IV) Replay



Gambar 7 Ekspansi *Keystream*

Setelah *keystream* berhasil diturunkan untuk ukuran *frame* tertentu, *keystream* tersebut dapat diekspansi ke dalam ukuran apapun. Proses tersebut dijelaskan sebagai berikut (Gambar 7):

a. *Attacker* dapat membangun *frame* yang satu *byte* lebih besar dari *keystream* yang diketahui. Pada prakteknya, frame yang ideal digunakan adalah *Internet Control Message Protocol* (ICMP) *echo frame* karena *access point* akan mengembalikan respon terhadap frame ini.
b. *Attacker* melakukan ekspansi *keystream* sebesar satu *byte*.
c. *Byte* tambahan ini dapat ditebak karena kemungkinan nilai yang mungkin hanya 256 (1 *byte* = 8 bit, 2^8=256)
d. Ketika nilai yang ditebak *attacker* benar, respon akan diterima. Apabila frame yang dikirimkan adalah ICMP *echo frame*, maka pesan yang diterima adalah *ICMP echo replay*.

Proses ini dilakukan berulang-ulang hingga panjang *keystream* yang diinginkan tercapai.

4. Fluhrer, Mantin, and Shamir *attack*

4.1. Algoritma RC4

Algoritma *stream cipher* RC4 terbagi menjadi dua bagian, yaitu algoritma *key scheduling* KSA yang mengubah kunci random (biasanya antara 40-256 bit) menjadi *initial permutation* S {0, ..., N-1} dan *output generator* PGRA yang menggunakan permutasi S untuk menghasilkan rangkaian output yang *pseudo-random*.

```
KSA(K)
Initialization:
    For i = 0 ... N - 1
        S[i] = i
    j = 0
Scrambling:
    For i = 0 ... N - 1
        j = j + S[i] + K[i mod ℓ]
    Swap(S[i], S[j])
```

Gambar 8 Algoritma *Key Scheduling* [FLU01]

```
PRGA(K)
Initialization:
    i = 0
    j = 0
Generation loop:
    i = i + 1
    j = j + S[i]
    Swap(S[i], S[j])
    Output z = S[S[i] + S[j]]
```

Gambar 9 Algortima *Pseudo-Random Generator* [FLU01]

PRGA melakukan inisialisasi i dan j dengan nilai 0 dan melakukan proses loop untuk empat operasi sederhana, yaitu melakukan penambahan nilai *counter* i, melakukan penambahan nilai j secara *pseudo random*, mempertukarkan nilai S[i] dan S[j], dan kemudian mengeluarkan nilai dari S[S[i] +

S[j]]. Sebagai catatan, setiap nilai dari S dipertukarkan paling tidak sekali (dapat juga dengan dirinya sendiri) dalam N putaran, sehingga permutation S mengalami perubahan seiring dengan proses pembangkitan output.

Algoritma KSA terdiri dari N buah loop yang sama dengan jumlah loop pada PRGA. KSA melakukan inisialisasi nilai S sedemikan hingga nilai S[i] sama dengan nilai i (atau dapat disebut sebagai *identity permutation*) dan nilai j menjadi 0. Kemudian KSA melakukan proses loop sebanyak N putaran dengan i berperan sebagai *counter*. Untuk setiap putaran, nilai j di-*update* dengan menambahkan nilai j sebelumnya dengan S[i] dan nilai karakter dari kunci, S[i mod ℓ], dengan ℓ adalah panjang kunci.

Mengingat panjang kunci efektif dari RC4 sangat besar, maka penyerangan terhadap algoritma PRGA tampak tidak mungkin. Bahkan, serangan pada algoritma ini diketahui memerlukan waktu lebih dari 2^{700}. Satu-satunya hasil praktis terkait dengan PRGA adalah pembangunan *distinguisher*. Fluhrer and McGrew dalam paper mereka yang berjudul "*Statistical Analysis of the Alleged RC4 Keystream Generator*" **(**[FLU00]) menjelaskan mengenai bagaimana membedakan (*distinguish*) output-output RC4 dari string random menggunakan 230 data. *Distinguisher* yang lebih baik, yang hanya membutuhkan 28 data, dijelaskan oleh Mantin dan Shamir dalam "*A Practical Attack on Broadcast RC4*" ([MAN01]). Bagaimanapun juga, *distinguisher* ini hanya dapat digunakan untuk melakukan serangan partial pada RC4 dalam aplikasi *broadcast*.

Fakta bahwa proses inisialisasi RC4 sangat sederhana menimbulkan pertimbangan untuk melakukan riset pada mekanisme RC4 ini. Roos dalam papernya yang berjudul "*A Class of Weak Keys in the RC4 Stream Cipher*" ([ROO95]) menemukan bahwa sebuah kumpulan kunci lemah mengurangi keefektifan mereka sebesar 5 bit. Kemudian Grosul dan Wallach dalam "*A Related-key Cryptanalysis of RC4*" ([GRO00]) menunjukan bahwa untuk kunci-kunci besar yang ukurannya mendekati N huruf, RC4 lemah terhadap serangan *related key*.

4.2. Kelemahan *Invariance*

```
KSA(K)
    For i = 0 ... N - 1
        S[i] = i
    i = 0
    j = 0
    Repeat N times
        j = j + S[i] + K[i mod ℓ]
        Swap(S[i], S[j])
        i = i +1
```

Gambar 10 KSA [FLU01]

```
KSA*(K)
    For i = 0 ... N - 1
        S[i] = i
    i = 0
    j = 0
    Repeat N times
        i = i +1
        j = j + S[i] + K[i mod ℓ]
        Swap(S[i], S[j])
```

Gambar 11 KSA* [FLU01]

Perbedaan antara KSA pada gambar 10 dengan KSA* pada gambar 11 yaitu pada KSA*, proses *update* nilai i terjadi di awal loop, sedangkan pada KSA proses *update* nilai i terjadi di akhir loop.

Definisi

Definisi 1 Asumsikan S permutasi dari {0, …, N-1}, t indeks dalam S, dan b nilai integer. Jika $S[t] \overset{\text{mod } b}{\equiv} t$, maka permutasi S dikatakan *b-conserve* indeks t. Sebaliknya, permutasi S dikatakan *b-unconserve* indeks t.

Permutasi S dan indeks i dan j setelah putaran ke-t dari KSA* kita sebut sebagai S_t, i_t, and j_t. Jumlah indeks pada permutasi *b-conserve* kita sebut sebagai $I_b(S)$. Namun, untuk kemudahan, $I_b(S_t)$ seringkali kita tuliskan sebagai I_t

Definisi 2 Suatu permutasi S {0, ..., N-1} adalah *b-conversing* jika $I_b(S) = N$, dan hampir *b-conserving* jika $I_b(S) \geq N - 2$.

Definisi 3 Asumsikan b, n adalah integer, dan asumsikan K adalah kunci sepanjang ℓ. K dikatakan kunci yang *b-exact* jika untuk suatu index t, K[t mod ℓ] ≡ (1 - t) (mod b). Pada kasus K[0] = 1 and msb(K[1]) = 1, K disebut kunci yang *special b-exact*.

Perhatikan bahwa untuk kondisi ini dipertahankan, diperlukan (tetapi tidak cukup) untuk b | ℓ.

Kelemahan

Teorema 1 Asumsikan q ≤ n dan ℓ adalah integer, dan $b \overset{def}{=} 2^q$. Jika that b | ℓ dan K adalah kunci yang *b-exact* sepanjang ℓ, maka permutasi S = KSA*(K) adalah *b-conserving*.

Berikut ini adalah pembuktian *auxiliary lemma*.

Lemma 1 If $i_{t+1} \equiv j_{t+1} \pmod b$, then $I_{t+1} = I_t$.

Proof: Satu-satunya operasi yang mungkin mempengaruhi S (dan mungkin juga I) adalah operasi pertukaran. Bagaimanapun juga, ketika i_{t+1} dan j_{t+1} *equivalent* (mod b), S_{t+1} *b-conserve* i_{t+1} (j_{t+1}) jika dan hanya jika S_t *b-conserve* j_t (i_t). Jadi, jumlah index S yang *b-conserve* tetap sama.

Proof: (dari Teorema 1) Kita akan membuktikan dengan induksi t bahwa untuk 1 ≤ t ≤ N apapun, akan menghasilkan $I_b(S_t) = N$ dan $i_t \equiv j_t \pmod b$. Hal ini mengimplikasikan $I_N = N$, yang mengakibatkan permutasi output *b-conserving*.

Untuk t=0 (sebelum putaran pertama), klaim bersifat trivial karena $i_0 = j_0 = 0$ dan S_0 merupakan *identity permutation* yang *b-conserving* untuk semua b. Jika $j_t \equiv i_t$ dan S_t *b-conserving*, maka $i_{t+1} = i_t + 1$ dan $j_{t+1} = j_t + S_t[i_{t+1}] + K[i_{t+1} \bmod \ell] \overset{\bmod b}{\equiv} i_t + i_{t+1} + (1 - i_{t+1}) = i_t + 1 = i_{t+1}$

Jadi, $i_{t+1} \equiv j_{t+1} \pmod b$ dan dengan mengaplikasikan Lemma 1 kita mendapatkan $I_{t+1} = I_t = N$ dank arena itu S_{t+1} *b-conserving*.

Jadi, KSA* mentransformasikan pola-pola khusus dalam kunci ke dalam pola-pola yang berkoresponden pada permutasi awal. Fraksi dari bit-bit *determined permutation* proporsional terhadap fraksi dari bit-bit *fixed key*. Sebagai contoh, aplikasikan hasil ini ke dalam $RC4_{n=8,\ell=6}$ and q = 1, 6 dari 48 bit-bit kunci sepenuhnya menentukan 252 dari 1684 bit-bit permutasi.

Perbedaan kecil antara KSA* dan KSA penting dalam hal KSA, diaplikasikan dalam kunci yang *b-exact*, tidak mempertahankan equivalensi (mod b) dari i dan j bahkan sesudah putaran pertama. Analisa eksekusi pada kunci yang *b-exact* menghasilkan

$j_1 = j_0 + S_0[i_1] + K[i_1] = 0 + S_0[0] + K[0] = K[0] \overset{\bmod b}{\equiv} 1 \overset{\bmod b}{\neq} 0 = i_1$

Jadi, struktur yang dideskripsikan sebelumnya tidak dapat dijaga dengan penggunaan K yang siklik. Bagaimanapun juga, kelemahan *invariance* dapat disesuaikan terhadap KSA yang sebenarnya dan perubahan yang tepat diformulasikan pada teorema berikut.

Teorema 2 Asumsikan q ≤ n and ℓ adalah integer dan $b \overset{def}{=} 2^q$. Jika b | ℓ dan asumsikan K merupakan kunci yang *special b-exact* sepanjang ℓ, maka

Pr[KSA(K) is almost b-conserving] ≥ 2/5

ketika probabilitasnya melebihi sisa dari bit-bit kunci.

Pada putaran pertama dua deviasi dari eksekusi KSA* terjadi. Deviasi yang pertama, i dan j yang *non-equivalence*, yang diharapkan mengakibatkan entri-entri yang *non-equivalent* ditukar pada putaran selanjutnya, sehingga merusak stuktur yang dipertahankan selama eksekusi KSA*. Sedangkan deviasi yang kedua, S menyebabkan *b-unconserve* dua buah indeks, yaitu i_1=0 dan j_1=K[0]. Bagaimanapun juga, kita dapat membatalkan *discrepancy* ij dengan memaksa K[0] (dan j_1) menjadi 1. Pada kasus ini, *discrepancy* dalam $S[j_1]$ (K[1]) menyebabkan nilai yang ridak tepat ditambahkan pada j, jadi memperbaiki *non-equivalence* pada i selama putaran kedua. Pada saat ini, masih terdapat dua buah index yang *unconserve dan* aberasi ini terbawa pada keseluruhan proses eksekusi dan berakibat pada hasil permutasi. Walaupun entri-entri yang rusak ini mungkin berpengaruh pada proses *update* nilai j, nilai j pseudo-random dapat mencapai mereka sebelum mereka digunakan untuk melakukan *update* nilai j dan mengirimkan mereka ke dalam suatu area dalam S di mana mereka tidak dapat mempengaruhi nilai j. Kemungkinan terjadinya event ini diperkuat oleh fakta bahwa entri yang rusak adalah i_1=0, yang tidak tersentuh sampai pada terminasi KSA terkait dengan jaraknya dari lokasi i saat ini, dan $j_2 = 1 + K[1] > N/2$ (ingat juga bahwa msb K[1]=1) jauh dari $i_1 = 2$, yang memberikan peluang untuk j mencapainya sebelum i.

Probabilitas dari N/2 pseudo random j untuk mencapai nilai tertentu berada di bawah 2/5, dan eksperimen lebih lanjut mengindikasikan bahwa probabilitas ini mendekati setengah.

4.3. Korelasi Key-Output

Saat ini, kita akan menganalisis propagasi dari pola-pola kunci yang lemah ke dalam output yang dibangkitkan. Pertama-tama, kita membuktikan Claim 1 yang terkait dengan perilaku yang bias dari varian PRGA yang dilemahkan, diaplikasikan pada permutasi yang b-conserving. Selanjutnya kita, akan mendiskusikan bahwa prefiks dari output PRGA yang original sangat berkorelasi dengan prefiks dari varian yang *swapless* (dalam *initial permutation* yang sama), yang mengimplikasikan adanya kebiasan dalam distribusi PRGA untuk kunci-kunci yang lemah ini.

Claim 1 Asumsikan RC4* merupakan variant RC4 yang dilemahkan dengan menghilangkan operasi *swap*. Asumsikan $q \leq n$, $b \stackrel{def}{=} 2^q$ dan S_0 permutasi yang b-conserving. Asumsikan $\{X_t\}_{t=1}^{\infty}$ rangkaian output yang dibangkitkan dengan mengaplikasikan RC4* pada S_0, dan $x_t \stackrel{def}{=} X_t \bmod b$. Maka, rangkaian $\{x_t\}_{t=1}^{\infty}$ bersifat konstan.

Karena operasi pertukaran (*swap*) tidak dilakukan, permutasi tidak berubah dan tetap *b-conserving* selama proses pembangkitan. Perhatikan bahwa semua dari S diketahui (mod b), begitu juga halnya dengan indeks inisial i = j = 0 ≡ 0 (mod b), sehingga operasi putaran (dan nilai-nilai ouputnya) dapat disimulasikan (mod b) independen terhadap S. Akibatnya, rangkaian output (mod b) bersifat konstan dan analisis yang lebih dalam menunjukkan bahwa rangkaian output bersifat periodik dengan periode 2b, seperti tampak pada Gambar 12 untuk q =1

i	j	$S[i]$	$S[j]$	$S[i]+S[j]$	Out
0	0	0	0	0	/
1	1	1	1	0	0
0	1	0	1	1	1
1	0	1	0	1	1
0	0	0	0	0	0
1	1	1	1	0	0
⋮	⋮	⋮	⋮	⋮	⋮

Gambar 12 Putaran pada RC4*, diaplikasikan pada 2-Conserving Permutation [FLU01]

Pada setiap langkah dalam PRGA, S berubah paling banyak di dua lokasi, dan kita dapat mengharapkan prefiks dari pembangkitan stream output oleh RC4 dari permutasi S_0 memiliki korelasi yang tinggi dengan stream yang dibangkitkan dengan S_0 yang sama (atau sedikit berbeda) dibangkitkan oleh RC4*. Lebih jelasnya, stream yang dibangkitkan oleh RC4 dari permutasu yang *almost b-conserving* diharapkan memiliki korelasi yang tinggi dengan substream konstan pada Claim 1.

Dengan demikian, kita telah membuktikan korelasi dengan probabilitas yang besar antara beberapa bit pada kunci rahasia dan bit-bit pada stream output untuk kumpulan besar kunci yang lemah.

4.4. Aplikasi Kripanalitik terhadap Kelemahan *Invariance*

Dalam [MAN01], Mantin dan Shamir mendeskripsikan suatu kebiasan statistik yang signifikan dalam output *word* kedua pada RC4. Dengan menggunakan kebiasan ini, mereka membangun suatu algoritma mampu membedakan antara output-output RC4 dan rangkaian yang benar-benar random hanya dengan melakukan analisis terhadap satu *word* dari O(N) stream-stream output yang berbeda. Algoritma ini merupakan algoritma *distinguiher* yang sangat efisien. Namun, hal ini dapat dihindari dengan mudah dengan cara mengabaikan dua *word* pertama dari setiap *stream* output. Apabila dua *word* ini diabaikan, *distinguisher* yang terbaik sekalipun membutuhkan 2^{30} stream output.

Suatu obeservasi menghasilkan *distinguisher* baru berdasarkan pada fakta bahwa pada bagian-bagian kunci yang signifikan, terdapat sejumlah *word* output inisial yang mengandung pola yang mudah dikenali.

Kebiasan ini gagal ketika kunci dipilih dari distribusi yang *uniform*. Namun, kebiasan ini tidak sepenuhnya hilang dan dapat digunakan untuk membangun sebuah *distinguisher* yang efisien, bahkan ketika dua *word* dari rangkaian output dihilangkan.

Perhatikan bahwa probabilitas dari *special* 2^q - *exact key* yang ditransformasikan menjadi 2^q – *conserving permutation* tidak tergantung pada panjang kunci ℓ. Bagaimanapun juga, jumlah bit yang telah ditentukan sebelumnya linier dengan ℓ, karena ity, ukutan dari kebiasan ini juga tergantung pada ℓ. Hasilnya, untuk kunci 64 bit *distinguisher* ini hanya membutuhkan

221 data, lebih baik daripada sebelumnya yang membutuhkan 2^{30} *word* output.

Perlu diperhatikan bahwa, kualitas dari *distinguisher* hampir tidak terpengaruh oleh proses pengabaian beberapa *word* pertama. Sebagai contoh, mengabaikan 2 *word* pertama mengakibatkan data yang dibutuhkan oleh *distinguisher* bertambah dengan perbandingan $2^{0.5}$ hingga 2^2 tergantung pada panjang ℓ.

4.5. Known IV Attack

Seperti yang telah dijelaskan sebelumnya, RC4 terdiri dari bagian, yaitu algoritma *key scheduling* dan *output generator*. Pada WEP, algoritma *key scheduling* dapat menggunakan 64-bit kunci (40-bit *secret key* ditambah dengan 24-bit IV) atau 128-bit kunci (104-bit *secret key* ditambah dengan 24-bit IV) untuk membangun RC4 tabel permutasi S {0, ..., 255}. Output generator menggunakan S untuk membangkitkan rangkaian *pseudo-random*.

Serangan ini memanfaatkan *byte* pertama output dari rangkaian pseudo-random. Persamaan dari *byte* pertama adalah S[S[1]+S[S[1]]]. Jadi, setelah proses inisialisasi S, *byte* pertama ini tergantung pada tiga nilai S, yaitu (S[1], S[S[1]], S[S[1]+S[S[1]]]). Serangan ini bergantung pada kemampuan untuk memperoleh informasi kunci dengan mengamati nilai-nilai ini.

Untuk melakukan serangan, kita perlu mencari IV yang menempatkan algoritma *key scheduling* pada kondisi memberikan informasi mengenai kunci (Kondisi pemberian informasi kunci ini kita sebut sebagai *resolved*). Merupakan hal yang mudah untuk melakukan pemeriksaan apakah suatu paket menyediakan IV dan *byte* output yang mengakibatkan kondisi *resolved*. Setiap paket yang *resolved* memberikan informasi sekitar satu *byte* kunci dan kita harus menebak setiap *byte* kunci dengan tepat sebelum suatu paket memberikan informasi mengenai *byte* kunci selanjutnya. Dengan kata lain, kita harus menebak setiap *byte* kunci. Setiap paket yang *resolved* memberikan peluang 5% untuk menebak *byte* kunci yang tepat dan memberikan peluang 95% untuk tebakan yang salah. Bagaimanapun juga, dengan mengamati banyak kasus yang *resolved*, kita dapat berharap menemukan *byte* kunci yang tepat.

5. Implementasi dari Fluhrer, Mantin, and Shamir *attack*

Airsnort adalah *software open source* yang merupakan implementasi pertama dari Fluhrer, Mantin, and Shamir *attack* yang terbuka untuk publik. Airsnort, yang dikembangkan Blake Hegerle dan Jeremy Bruestle, dirilis pada Agustus 2001. Airsnort membutuhkan rata-rata lima sampai dengan sepuluh juta paket terenkripsi untuk memecahkan kunci WEP.

Algoritma Airsnort:

1. Menangkap semua frame dan mencari frame yang memiliki *Initialization Vector* (IV)
2. Mencocokan IV dengan *byte* kunci yang familiar. Kemudian mengkategorikan sampel berdasarkan pada *byte* kunci tersebut dan menggunakan sampel untuk memecahkan kunci. Untuk memperoleh kunci, sampel yang dibutuhkan lebih kecil dari 100.

Sebelumnya, Adam Stubblefield, mahasiswa *Rice University* membuat aplikasi serupa ketika sedang magang di AT&T Labs. Aplikasi yang dibuatnya mampu memecahkan kunci pada jaringan *wireless* AT&T (dengan izin dari administrator) dalam waktu di bawah dua jam. Riset ini dipublikasikan di Internet, sedangkan aplikasi yang dibuatnya tidak dirilis untuk publik.

Di samping kedua aplikasi tersebut, terdapat WEPCrack yang juga mengimplementasikan Fluhrer, Mantin, and Shamir *attack*. WEPCrack dirilis tidak lama setelah Airsnort. Aplikasi lainnya, antara lain Aircrack dan WepLab.

Berikut ini akan dijelaskan tahapan-tahapan yang dilakukan untuk memecahkan kunci WEP yang dilakukan oleh Adam Stubblefield:

1. Mengumpulkan paket data
   Proses ini merupakan proses yang paling banyak menghabiskan waktu dari keseluruhan proses serangan. Untuk melakukan hal ini dapat digunakan aplikasi *packet capturer*, seperti misalnya *Wildpacket's AiroPeek*.
2. Serangan dasar
   Berikut ini merupakan algoritma dasar (Gambar 13) untuk melakukan serangan terhadap WEP. Walaupun Fluhrer, Mantin, dan Shamir mempostulatkan bahwa empat juta paket data cukup untuk melakukan serangan terhadap WEP, namun dengan menggunakan algoritma ini dibutuhkan lima hingga enam juta paket data.

```
RecoverWEPKey()
        Key[0 ... KeySize] = 0
        for KeyByte = 0 ... KeySize
                Counts[0 ... 255] = 0
                foreach packet → P
                        if P.IV ∈ {(KeyByte+3,0xFF,N) | N ∈ 0x00 ... 0xFF}
                                Counts[SimulateResolved(P,Key)]+ = 1
                Key[KeyByte] = IndexOfMaximumElement(Counts)
        return Key
```

Gambar 13 Algoritma Serangan Dasar

Fungsi SimulateResolved melakukan perhitungan seperti yang dideskripsikan dalam bab 7.1 paper yang ditulis oleh Fluhrer, Mantin, dan Shamir [FLU01]

3. Perbaikan terhadap serangan
   Modifikasi dilakukan untuk meningkatkan performasi serangan, baik dari segi waktu maupun dari segi penggunaan memori.
   a. Memilih *Initialization Vector* (IV)
      Pada serangan dasar, hanya IV dalam bentuk tertentu yang digunakan, yaitu yang berkoresponden dengan KeyByte+3,0xFF,N, dengan KeyByte adalah KeyByte yang sedang kita cari dan N tidak terbatas. Namun, ternyata terdapat IV lain yang dapat menhasilkan *resolved state*. Selain itu, proses pemeriksaan semua IV (berlainan dengan yang disarankan oleh Fluhrer, Mantin, Shamir yang menyarankan untuk mengambil subset IV) dapat dilakukan secara paralel dengan proses menerima paket.
   b. Menebak *key bytes* awal
      Mengingat Fluhrer, Mantin, Shamir *attack* bekerja dengan mengkonstruksi key bytes yang ditemukan sebelumnya, menemukan key bytes awal merupakan proses yang kritikal. Terdapat dua pendekatan untuk melakukan hal ini. Pendekatan pertama memanfaatkan bagaimana IV dibangkitkan, lebih jelasnya, kita menerima paket-paket yang telah *resolved* untuk berbagai macam *key bytes* yang berbeda sebelum menerima cukup paket yang *resolving* untuk memprediksi *key bytes* awal. Untuk itu, kasus *resolving* yang telah diterima akan digunakan untuk memperkecil kemungkinan *key bytes* awal. Selanjutnya, kandidat kunci dapat dites dengan menentukan apakah checksum WEP pada paket terdekripsi benar. Pendekatan kedua dilakukan dengan mengeksploitasi manajemen kunci yang buruk pada implementasi WEP. Karena kunci WEP harus dimasukkan secara manual, dapat diasumsikan bahwa kunci yang digunakan bukan string panjang yang digit-digitnya berupa heksadesimal, melainkan *passphrase* yang mudah diingat. Pemeriksaan *passphrase* ini mudah dilakukan, cukup dengan pemeriksaan apakah nilai byte dari *passphrase* sama dengan huruf, angka, atau simbol-simbol bahasa. Kedua optimisasi ini dengan sangat menakjubkan berhasil menurunkan jumlah paket yang dibutuhkan. Secara paralel dengan menerima paket, proses menebak kunci dilakukan dengan memilih kandidat yang paling dekat berdasarkan pada *resolved* case yang telah dikumpulkan sebelumnya. Prioritas *byte* yang diberlakukan adalah huruf kecil, huruf besar, angka, simbol, dan *byte* lainnya
   c. Kasus *resolved* spesial
      Seperti yang telas Shamir kemukakan, terdapat beberapa kasus di mana kasus *resolved* dapat menyediakan indikasi yang lebih baik kepada *key byte* tertentu. Jika terdapat duplikasi antara tiga nilai pada posisi S[1], S[S[1]], dan S[S[1]+S[S[1]]] (sebagai contoh, hanya terdapat dua nilai berbeda), maka kemungkinan posisi ini dalam permutasi S tidak berubah adalah antara $e^{-3} \approx 5\%$ dan $e^{-2} \approx 13\%$. Karena itu, kasus ini dapat diberlakukan sama dengan kasus *resolved* standar.

```
RecoverWEPKeyImproved(CurrentKeyGuess, KeyByte)
        Counts[0 ... 255] = 0
        foreach packet → P
                if Resolved?(P.IV )
                        Counts[SimulateResolved(P,CurrentKeyGuess)]+=
                                Weight(P,CurrentKeyGuess)
        foreach SelectMaximalIndexesWithBias(Counts)→ ByteGuess
                CurrentKeyGuess[KeyByte] = ByteGuess
                if Equal?(KeyByte,KeyLength)
                        if CheckChecksums(CurrentKeyGuess)
                                return CurrentKeyGuess
                else
                        Key = RecoverWEPKeyImproved(CurrentKeyGuess,KeyByte+1)
                        if notEqual?(Key,Failure)
                                return Key
                return Failure
```

Gambar 14 Algoritma Serangan yang Telah Diperbaiki

d. Penggabungan optimisasi
Dengan menggabungkan proses optimisasi yang dijelaskan di atas, diperoleh penurunan jumlah paket yang dibutuhkan dari sekitar lima juta paket menjadi satu juta paket (Gambar 14). Fungsi CheckChecksums memeriksa apakah kunci menyebabkan *checksum* pada paket WEP benar. Fungsi SelectMaximalIndexesWithBias terkait dengan fungsi optimasi pada bahasan 3b. Fungsi Weight mengembalikan nilai 3 apabila kasus *resolved* merupakan kasus *resolved* spesial. Predikat Resolved? Memeriksa apakah paket yang diberikan sebagai parameter menghasilkan kondisi *resolved*.

6. Solusi

Berikut ini adalah beberapa solusi yang dapat dilakukan untuk mencegah terjadinya serangan terhadap WEP, baik secara aktif maupun secara pasif:

1. Serangan pada *Integrity Check Value* (ICV) dapat dicegah dengan menggunakan *Message Identity Check* (MIC). Pengirim paket menambahkan beberapa *byte* (MIC) ke dalam paket sebelum melakukan enkripsi dan pengiriman paket. Ketika *receiver* menerima paket tersebut, *receiver* akan mendekripsi paket dan melakukan pemeriksaan terhadap MIC. Apabila MIC pada *frame* sama dengan MIC yang dikalkulasi, maka paket diterima, sebaliknya paket ditolak. Proses ini sama dengan proses pada ICV, hanya saja MIC dispesifikasikan untuk menjaga integritas data. Dengan MIC ini, dimungkinkan paket yang telah diubah di dalam perjalanan di-*drop* sehingga serangan yang memanfaatkan kelemahan ini dpat dicegah.
2. *Wi-fi Protected Access* (WPA)
Wi-fi Protected Access atau WPA merupakan algoritma enkripsi pengganti WEP. Tujuan dari dibuatnya WPA adalah untuk menangani kelemahan-kelemahan yang terdapat pada WEP. Untuk mencapai hal tersebut, maka dilakukan dua pengembangan, yaitu:
   a. memperbaiki enkripsi data yang sudah terbukti lemah pada WEP.
   b. otentikasi *user*, yang hampir tidak ada pada WEP.

7. Kesimpulan

Terdapat beberapa variabel yang dapat mempengaruhi performasi serangan terhadap kunci WEP, yaitu pemilihan IV, pemilihan kunci, dan RC4 .

1. Pemilihan IV
Karena standar WEP tidak menspesifikasikan bagaimana IV dipilih, terdapat variasi dalam pembangkitan nilai IV dalam beberapa perangkat. Kebanyakan perangkat tampaknya menggunakan salah satu dari tiga metode berikut, yaitu *counter*, pemilihan random, atau *value flipping* (sebagai contoh pertukaran antara dua buah nilai IV). Serangan ini mungkin dilakukan terhadap metode pembangkitan IV yang pertama dan kedua. *Value flipping* dapat mencegah terjadinya serangan ini. Namun sebagai

gantinya, metode ini menggunakan kembali stream *pseudo-random* untuk semua paket yang lain. *Trade off* ini tampak tidak beralasan. Metode pembangkitan IV dengan *counter* merupakan metode yang paling mungkin diserang dengan serangan ini. Pada metode ini, nilai IV ditambah untuk setiap paket yang dikirimkan (nilai IV awal dapat dimulai dari 0 ataupun dibangkitkan secara random). Perangkat dengan pembangkitan IV ini memberikan distribusi yang baik dalam proses memperoleh *key byte* dari paket. Pembangkitan nilai IV secara random juga tidak lebih baik, mengingat terdapat kasus-kasus *resolved* yang terjadi walaupun distribusinya tidak sebaik pada metode *counter*. Singkat kata, apapun metode pembangkitan IV yang digunakan, efek serangan ini tidak dapat dihilangkan tanpa secara eksplisit tanpa melakukan testing terhadap setiap pasangan IV dan kunci untuk memeriksa apakah pasangan kunci ini menghasilkan kondisi *resolved* sebelum dikirimkan.

2. Pemilihan kunci
   Lemahnya proses manajemen kunci pada WEP secara langsung berkontribusi pada kemudahan dalam proses memecahkan kunci WEP. Pada jaringan *wireless* umumnya, *shared key* tunggal biasanya digunakan untuk komunikasi antara *base station* dan *node-node mobile*. Selain serangan dapat saja dilakukan oleh eks-pegawai yang mengetahui kunci, terdapat juga permasalahan dalam pendistribusian kunci. Banyak site yang menggunakan *password* yang mudah diingat untuk mempermudah pendistribusian kunci. Namun, sebenarnya tidak ada standar pemetaan dari password ini menjadi kunci WEP. Solusi paling umum adalah dengan memetakan nilai ASCII secara langsung menjadi nilai *key byte*. Sangat disarankan untuk menggunakan kunci WEP yang tidak dapat diingat atau menggunakan aplikasi pembangkit kunci yang menggunakan fungsi hash kriptografi untuk memetakan *password*. Perlu diingat bahwa solusi ini tidak mencegah terjadinya serangan, melainkan hanya membuat serangan menjadi lebih sulit. Solusi terbaik adalah dengan mengganti kunci setiap *user* setelah pengiriman 10 ribu paket.
3. RC4
   RC4 merupakan stream cipher yang efisien yang dapat digunakan secara aman. Implementasi RC4 pada SSL tidak terpengaruh oleh Fluhrer, Mantin, dan Shamir *attack*. Alasannya, SSL melakukan pra-proses terhadap kunci enkripsi dan IV dengan melakukan *hashing* kedua kunci tersebut dengan MD5 dan SHA-1. Akibatnya, sesi yang berbeda menggunakan kunci yang tidak berhubungan. Sebagai tambahan, dalam SSL, *state* RC4 pada paket sebelumnya digunakan untuk paket selanjutnya sehingga kunci algoritma tidak berubah setelah setiap paket dikirimkan. Lebih jauh, dapat digunakan aplikasi untuk membuang 256 byte pertama output dari RC4. Hal ini mungkin sedikit lebih mahal untuk paket yang kecil, namun apabila *state* dari sesi dipertahankan, maka *cost* dapat terbayarkan.

   Secara keseluruhan, RC4 dapat digunakan sebagai bagian dari solusi keamanan. Namun, implementasinya harus dilakukan dengan cermat sehingga material kunci tidak bocor. Salah penyebab terdapatnya celah dari suatu algoritma kriptografi adalah desainer protokol yang tidak memahami dengan dalam kriptografi dan ketidakperdulian terhadap implementasi keamanan dengan baik. Kedua hal inilah yang terjadi pada WEP.

Jaringan *wireless* tidak akan pernah aman. Oleh karena itu, disarankan bagi setiap orang yang menggunakan jaringan *wireless* untuk:

1. mengasumsikan *link layer* tidak menyediakan keamanan.
2. menggunakan mekanisme keamanan pada level yang lebih tinggi, seperti IPSec dan SSH, daripada mengandalkan WEP.
3. memperlakukan seluruh sistem yang terhubung melalui jaringan *wireless* sebagai jaringan eksternal. Posisikan semua *access point* di luar *firewall* yang melindungi jaringan utama.
4. Mengasumsikan bahwa semua orang yang berada dalam jangkauan fisik jaringan *wireless* dapat melakukan komunikasi pada jaringan sebagai pengguna yang valid. Perlu juga diperhatikan bahwa, pihak tertentu dapat saja menggunakan perangkat antena yang canggih dengan jangkauan yang lebih jauh dari yang di-*support* oleh perangkat-perangkat *wireless* pada PC.

Pengalaman dengan WEP menunjukkan bahwa keamanan merupakan hal yang sulit untuk diperoleh dan dijaga. Celah-celah keamanan dapat terjadi pada level apapun, termasuk

protokol, desain, implementasi, dan bahkan *deployment*. Celah-celah tersebut dapat mengakibatkan resiko yang fatal bagi sistem.

DAFTAR PUSTAKA

[FLU00] Fluhrer, Scott, McGrew. (2000). Statistical Analysis of the Alleged RC4 Keystream Generator. Fast Software Encryption.

[FLU01] Fluhrer, Scott, Itsik Mantin, Adi Shamir. (2001). Weaknesses in the Key Scheduling Algorithm of RC4. In Eighth Annual Workshop on Selected Areas in Cryptography, Toronto, Canada.

[GRO00] Grosul, A. L., D. S. Wallach. (2000). A Related-key Cryptanalysis of RC4.

[HON03] Hong, Jumnit, Riad Lemhachheche. (2003). WEP Protocol Weaknesses and Vulnerabilities. Oregon State University.

[KHA03] Khan, Jahanzeb, Anis Khawaja. (2003) Building Secure Wireless Networks with 802.11. Wiley Publishing, Inc.

[MAN01] Mantin, Itsik, Adi Shamir. (2001). A Practical Attack on Broadcast RC4. Fast Software Encryption.

[ROO95] Roos, A. (1995) A Class of Weak Keys in the RC4 Stream Cipher.

[STU01] Stubblefield, Adam, John Ioannidis, Aviel D. Rubin. (2001). Using the Fluhrer, Mantin, and Shamir Attack to Break WEP.

[TAN03] Tanenbaum, Andrew. (2003). Computer Networks, Fourth Edition. Prentice Hall.