Serangan Aljabar pada Algoritme S-IDEA dan Miniatur S-IDEA

Nadya Marta Matanggwan[1)], Sa'aadah Sajjana Carita[2)]
(1) *Badan Siber dan Sandi Negara, nadya.marta@bssn.go.id*
(2) *Politeknik Siber dan Sandi Negara, sajjana.carita@bssn.go.id*

Abstrak

Serangan aljabar dapat dilakukan dalam dua tahapan yaitu mendapatkan sistem persamaan polinomial dan menentukan solusi dari sistem persamaan polinomial tersebut. Pada penelitian ini dilakukan serangan aljabar pada S-IDEA. Proses enkripsi satu round S-IDEA terdiri dari 14 langkah sementara sampai dengan Langkah ke-7 sudah diperoleh persamaan polinomial yang besar yaitu terdiri dari 4.721 monomial. Oleh karena keterbatasan sumber daya, dibuat miniatur S-IDEA agar serangan aljabar dapat dilakukan pada setiap langkah secara utuh. Algoritme *miniatur S-IDEA terdiri dari 2,5 round yang setiap round-nya terdiri dari 14 langkah seperti halnya S-IDEA. Proses serangan aljabar pada miniatur S-IDEA menghasilkan 8 persamaan polinomial dengan monomial maksimal yang diperoleh yaitu sebanyak 1.109 monomial. Penentuan solusi dari persamaan polinomial yang diperoleh dilakukan dengan metode XL algorithm dan basis Gröbner. Metode XL algorithm dilakukan sebanyak 4 dari 5 tahap, yaitu tahap linierisasi. Tahap linierisasi menghasilkan 136 persamaan yang didalamnya terdapat 1512 monomial. Konstanta dari persamaan linier tersebut dapat direpresentasikan ke dalam bentuk matriks berukuran 1512×136. Besarnya sistem persamaan hasil linierisasi yang diperoleh menyebabkan nilai kunci belum bisa didapatkan secara langsung melainkan harus dilakukan analisis lebih lanjut mengenai persamaan mana saja yang perlu digunakan untuk tahap selanjutnya pada XL algorithm. Sementara itu penentuan solusi dengan basis Gröbner menghasilkan 34 persamaan baru yang cukup panjang, sehingga nilai kunci belum dapat diperoleh secara langsung. Dengan disimulasikannya pencarian solusi sistem persamaan polinomial dengan basis Gröbner dapat diketahui beberapa hal, salah satunya yaitu walaupun sudah ditemukan sistem persamaan polinomial yang merepresentasikan bit teks sandi ternyata tetap diperlukan komputasi yang cukup besar untuk menentukan solusinya.*

Kata kunci: Serangan aljabar, Miniatur S-IDEA, S-IDEA, XL *algorithm.*

1. PENDAHULUAN

Serangan aljabar adalah teknik kriptanalisis yang menyederhanakan kriptanalisis terhadap sistem kriptografi yang diserang menjadi permasalahan memperoleh dan menyelesaikan sistem persamaan polinomial [1]. Serangan ini juga merupakan serangan yang berpotensi kuat pada *block cipher*. Serangan aljabar dimulai dengan membangun sistem persamaan polinomial dari bit-bit kunci, bit-bit teks terang, dan bit-bit teks sandi dan mencoba untuk menemukan solusi dari sistem persamaan polinomial tersebut [2].

Secara umum, terdapat dua tahap dalam melakukan serangan aljabar, yaitu memperoleh sistem persamaan polinomial dan menentukan solusi dari sistem persamaan polinomial yang telah didapatkan tersebut [1]. Salah satu syarat pada serangan aljabar yaitu diperolehnya persamaan polinomial dari S-Box atau operasi non-linier lainnya [3]. Serangan ini dinilai cukup menarik karena hanya memerlukan satu *known-plaintext* untuk melakukan pemulihan kunci [4]. Serangan ini sudah pernah diterapkan pada beberapa algoritme *block cipher* seperti misalnya pada DES [4], pada S-DES [5], dan pada Mini-AES [6].

Serangan aljabar ini juga dapat diterapkan pada algoritme *block cipher* lainnya seperti *International Data Encryption Algorithm* (IDEA) dengan syarat yaitu diperolehnya persamaan polinomial dari operasi non-linier.

IDEA merupakan *block cipher* yang mengenkripsi 64-bit blok teks terang menjadi 64-bit blok teks sandi menggunakan 128-bit kunci. Algoritme ini terdiri dari 8 *round* identik dan setengah *round* sebagai transformasi akhir [7]. Sampai tahun 2015, tidak ada kelemahan linier atau aljabar yang berhasil ditemukan [8].

Pada tahun 2007 Nick Hoffman mengembangkan bentuk sederhana dari IDEA yang bernama *Simplified* IDEA (S-IDEA). S-IDEA mengenkripsi 16-bit teks terang menjadi 16-bit teks sandi menggunakan 32-bit kunci. Algoritme ini terdiri dari 4 *round* identik dan setengah *round* sebagai transformasi akhir. Algoritme S-IDEA memiliki struktur dan sifat yang sama dengan algoritme IDEA namun dengan ukuran yang lebih kecil.

Pada penelitian ini dilakukan serangan aljabar pada algoritme S-IDEA dan diperdalam dengan serangan aljabar pada miniatur S-IDEA yang bertujuan untuk lebih memahami metode serangan aljabar pada algoritme S-IDEA. Miniatur S-IDEA merupakan penyederhanaan dari algoritme S-IDEA yang setiap *round*-nya juga terdiri dari 14 langkah. Penjelasan lebih lengkap mengenai miniatur S-IDEA dapat dilihat pada penjelasan selanjutnya. Proses enkripsi pada miniatur S-IDEA menghasilkan 8 persamaan polinomial yang merepresentasikan relasi antara bit teks terang, kunci, dan teks sandi. Penentuan solusi dari sistem persamaan polinomial pada penelitian ini menggunakan XL *algorithm* dan basis Gröbner.

2. LANDASAN TEORI

2.1 Serangan Aljabar

Serangan aljabar pertama kali diperkenalkan oleh Kipnis dan Shamir [1]. Serangan aljabar ini dikembangkan dari ide awal berupa menemukan dan menyelesaikan sistem persamaan polinomial multivariat pada lapangan hingga. Serangan aljabar terdiri dari 2 langkah utama yaitu:

a. Menemukan sistem persamaan polinomial, dan
b. Menyelesaikan sistem persamaan polinomial.

Untuk menyelesaikan sistem persamaan yang sudah ditemukan dapat digunakan beberapa cara seperti *Extended Linierization Algorithm* atau XL *Algorithm*, Gröbner *bases*, *Resultant-based matrices*, SAT *Solvers* dan Buchbergers *Algorithm*.

2.2 *Simplified International Data Encryption Algorithm* (S-IDEA)

Algoritme S-IDEA pertama kali diperkenalkan oleh Nick Hoffman [7]. Algoritme ini merupakan algoritme kunci simetris *block cipher* yang merupakan penyederhanaan dari algoritme IDEA. Algoritme IDEA sendiri diperkenalkan pertama kali pada tahun 1991 oleh Lai, Massey dan Murphy. Pada awalnya algoritme ini bernama *Improved Proposed Encryption Standard* (IPES) yang merupakan perbaikan dari algoritme *Proposed Encryption Standard* (PES). Kemudian pada tahun 1992 namanya diubah menjadi IDEA. Algoritme ini merupakan salah satu Algoritme kandidat NESSIE *Project* dan saat ini tergabung di dalam *Pretty Good Privacy* (PGP).

Algoritme IDEA mengenkripsi 64-bit teks terang menjadi 64-bit teks sandi menggunakan kunci sepanjang 128-bit. Algoritme ini terdiri dari delapan *round* identik dan setengah *round* tambahan sebagai transformasi akhir. Terdapat tiga operasi yang digunakan pada algoritme IDEA yaitu *Exclusive* OR per bit, penjumlahan tertutup di dalam modulus 2^{16} dan perkalian tertutup di dalam modulus $2^{16}+1$.

Pada algoritme sederhananya yaitu S-IDEA, teks terang yang dienkripsi yaitu 16 bit menghasilkan 16 bit teks sandi menggunakan kunci sepanjang 32 bit. Algoritme ini terdiri dari empat *round* identik dan setengah *round* tambahan sebagai transformasi akhir. Kunci yang digunakan pada masing-masing *round* diatur dalam *key scheduling*. *Key scheduling* pada algoritme S-IDEA menggunakan operasi *rotate shift* atau rotasi geser dalam prosesnya untuk menghasilkan kunci. Input kunci pada algoritme S-IDEA berjumlah 32-bit yang dalam penelitian ini dinotasikan sebagai $k1,k2,k3,\ldots,k32$ dan dari bit-bit tersebut diambil 24-bit untuk *round* ke-1 lalu dilakukan *rotate shift* ke kiri sebanyak 6-bit. Setelah dilakukan *rotate shift* lalu subkunci untuk *round* selanjutnya ditentukan. Subkunci yang dihasilkan selanjutnya digunakan untuk empat setengah *round* dengan total seluruhnya 112-bit dari kunci input sepanjang 32-bit.

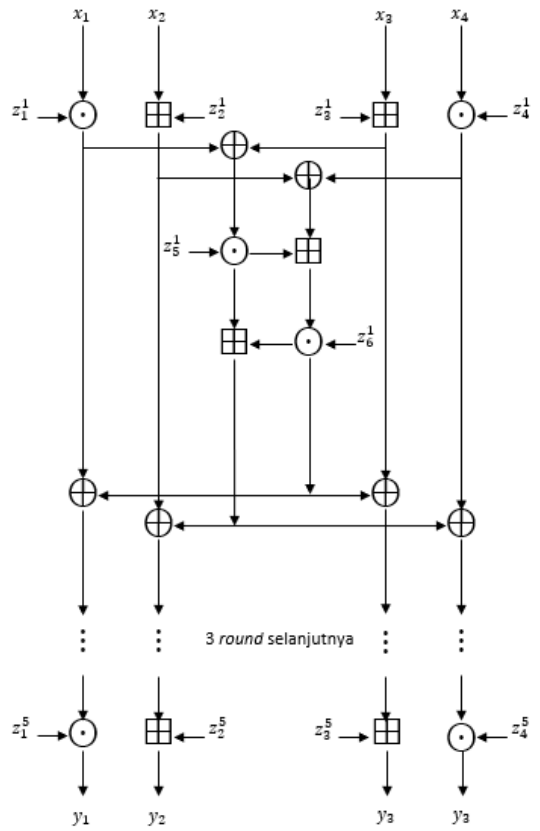
Total panjang subkunci pada setiap *round*-nya yaitu 24-bit kecuali pada setengah *round* terakhir yaitu enam belas bit. Masing-masing subkunci yang dihasilkan oleh *key scheduling* dinotasikan dengan Z_{ji} yang merupakan subkunci enkripsi ke-j pada *round* ke-i. Sebagai contoh Z_{21} merupakan subkunci enkripsi ke-2 pada *round* ke-1.

Proses enkripsi algoritme S-IDEA diawali dengan membagi teks terang menjadi 4 blok yang dinotasikan dengan x_i dengan i merupakan nomor blok. Pada [7] dijelaskan bahwa pada satu *round* proses enkripsi Algoritme S-IDEA terdiri dari 14 langkah yang dijabarkan pada Tabel 1.

Tabel 1. Proses Enkripsi pada 1 *Round* Algoritme S-IDEA

Langkah	Proses
1	Mengalikan x_1 dengan subkunci pertama Z_1.
2	Menambahkan x_2 dengan subkunci kedua Z_2.
3	Menambahkan x_3 dengan subkunci kedua Z_3.
4	Mengalikan x_4 dengan subkunci pertama Z_4.
5	*Bitwise* XOR hasil dari Langkah ke-1 dengan 3.
6	*Bitwise* XOR hasil dari Langkah ke-2 dengan 4.
7	Mengalikan hasil dari Langkah ke-5 dengan subkunci kelima Z_5.
8	Menambahkan hasil dari Langkah ke-6 dengan 7.
9	Mengalikan hasil dari Langkah ke-8 dengan subkunci keenam Z_6.
10	Menambahkan hasil dari Langkah ke-7 dengan 9.
11	*Bitwise* XOR hasil dari Langkah ke-1 dengan 9.
12	*Bitwise* XOR hasil dari Langkah ke-3 dengan 9.
13	*Bitwise* XOR hasil dari Langkah ke-2 dengan 10.
14	*Bitwise* XOR hasil dari Langkah ke-4 dengan 10.

Masing-masing *round* diakhiri dengan *swap* antara hasil dari Langkah ke-12 dengan Langkah ke-13 kecuali pada *final transformation*. *Source code* enkripsi S-IDEA empat setengah *round* dilakukan dengan menggunakan bahasa pemrograman C++. Skema enkripsi algoritme S-IDEA dapat dilihat pada Gambar 1.



Gambar 1. Skema Enkripsi Algoritme S-IDEA

2.3 Miniatur S-IDEA

Miniatur S-IDEA pada penelitian ini didefinisikan sebagai penyederhanaan dari algoritme S-IDEA. Miniatur S-IDEA mengenkripsi 8-bit teks terang menjadi 8-bit teks sandi menggunakan kunci sepanjang 16-bit. Algoritme ini terdiri dari dua *round* identik dan setengah *round* tambahan sebagai transformasi akhir. Terdapat tiga operasi yang digunakan pada algoritme IDEA yaitu *Exclusive* OR per bit, penjumlahan tertutup di dalam modulus 2^2 dan perkalian tertutup di dalam modulus $2^2 + 1$.

Kunci yang digunakan pada masing-masing *round* diatur dalam *key scheduling*. *Key scheduling* pada miniatur S-IDEA menggunakan operasi *rotate shift* atau rotasi geser dalam prosesnya untuk menghasilkan kunci. Input kunci pada algoritme S-IDEA berjumlah 16-bit. Setelah itu, bit-bit tersebut diambil 12-bit untuk *round* ke-1 lalu dilakukan *rotate shift* ke kiri sebanyak 3-bit. Setelah dilakukan *rotate shift* lalu subkunci untuk *round* selanjutnya ditentukan. Proses enkripsi dan dekripsi telah dibuktikan dapat dilakukan dengan bantuan *source code* Bahasa pemrograman C++.

2.4 XL *Algorithm*

Seperti namanya yaitu *Extended Linierization*, maka secara garis besar algoritme ini akan menambahkan persamaan yang diperlukan untuk menemukan solusi dari sistem persamaan polionimial. Berikut merupakan tahapan-tahapan pada XL *Algorithm* [11]:

a. Memilih $D > d$ (biasanya $D = d + 1$). D adalah derajat dari sistem persamaan polinomial yang akan dibentuk, dan d adalah derajat dari sistem persamaan polinomial yang diberikan.
b. Membuat daftar seluruh monomial yang berderajat $\leq D - d$, termasuk monomial "1" yang berderajat 0.
c. Mengalikan seluruh persamaan polinomial yang diberikan dengan seluruh monomial yang telah didaftarkan.
d. Mengubah sistem persamaan polinomial yang terbentuk menjadi sistem persamaan linier (melakukan linierisasi).
e. Menentukan solusi dari sistem persamaan linier menggunakan aljabar linier.

2.5 Basis Gröbner

Basis Gröbner pertama kali diperkenalkan oleh Bruno Bucherger pada tesisnya [12] pada tahun 1965. Tesis tersebut kemudian diterjemahkan ke dalam Bahasa Inggris pada tahun 2006. Basis Gröbner merupakan suatu teknik yang menyediakan solusi algoritmik untuk berbagai permasalahan aljabar komutatif dan geometri. Teknik Basis Gröbner telah diaplikasikan untuk menyelesaikan persamaan polinomial yang terdiri dari beberapa variabel. Teori dasar dari basis Gröbner merupakan konsep dari suatu ideal yang dibangkitkan oleh himpunan hingga dari polinomial multivariat [13].

3. METODE PENELITIAN

Penelitian ini dilakukan dengan dua metode penelitian. Metode penelitian yang digunakan yaitu metode telaah kepustakaan dan metode eksperimen. Metode telaah kepustakaan dilakukan dengan cara studi literatur sumber-sumber yang mendukung dalam penelitian ini seperti buku, paper, tesis, tugas akhir, dan sumber-sumber lainnya. Studi literatur ini bertujuan untuk memahami konsep dan teori terkait serangan aljabar, algoritme S-IDEA, dan XL *algorithm*.

Metode eksperimen dilakukan dengan cara menerapkan serangan aljabar pada algoritme S-IDEA dan dilakukan dengan bantuan perangkat lunak Microsoft Excel, Matlab R2013a, bahasa pemrograman C++, dan SageMath 9.1. Perangkat lunak tersebut dijalankan pada laptop dengan processor Intel®Core™i3-6006U CPU @ 2.00GHz dan RAM 4GB.

Metode eksperimen juga dilakukan pada penerapan serangan aljabar pada miniatur S-IDEA. Pembuatan miniatur S-IDEA ini dilakukan untuk mengetahui proses pencarian sistem persamaan polinomial S-IDEA secara lengkap dalam satu *round* dan untuk mengetahui proses penyelesaian sistem persamaannya.

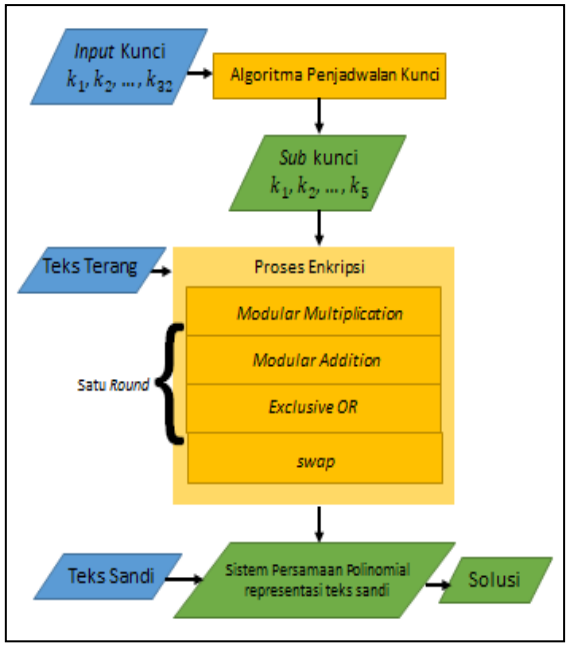
3.1 Pembatasan Masalah

Pembatasan masalah yang digunakan pada penelitian ini:

a. Serangan aljabar pada algoritme S-IDEA menggunakan sepasang teks terang dan teks sandi yang diperoleh dari pemilihan sembarang.
b. Serangan aljabar pada algoritme S-IDEA dilakukan sampai dengan pencarian polinomial Langkah ke-7.
c. Serangan aljabar pada miniatur S-IDEA menggunakan sepasang teks terang dan teks sandi yang diperoleh dari pemilihan sembarang.
d. Pencarian solusi sistem persamaan polinomial dilakukan pada satu setengah *round* miniatur S-IDEA.
e. Basis Gröbner tidak termasuk pada pembahasan.
f. Serangan dilakukan berdasarkan kekuatan sumber daya yang digunakan

3.2 Pencarian Sistem Persamaan

Pada penelitian ini langkah untuk menemukan sistem persamaan dapat dilihat pada Gambar 2. Kotak berwarna kuning menunjukkan proses, kotak berwarna biru menunjukkan input dan kotak berwarna hijau menunjukkan *output*. Seperti yang terlihat dalam Gambar 2., diperlukan tiga input yaitu kunci, teks sandi dan teks terang yang melibatkan dua proses yaitu algoritme penjadwalan kunci dan enkripsi. Algoritme penjadwalan kunci menghasilkan subkunci. Selanjutnya subkunci ini digunakan sebagai salah satu input pada proses enkripsi. Proses enkripsi menghasilkan sistem persamaan polinomial yang merepresentasikan relasi antara bit teks terang, kunci, dan teks sandi. Sistem persamaan polinomial inilah yang dicari solusinya.



Gambar 2. Proses Penentuan Sistem Persamaan Polinomial

Modular addition dan *modular multiplication* merupakan fungsi non-linier di dalam algoritme S-IDEA. Fungsi non-linier tersebut diubah ke dalam bentuk persamaan polinomial. Hal ini bertujuan untuk mendapatkan persamaan polinomial yang merepresentasikan relasi antara bit teks terang, kunci, dan teks sandi. Sehingga didapatkan 4 persamaan polinomial yang merepresentasikan fungsi non-linier *modular addition* dan 4 persamaan polinomial yang merepresentasikan fungsi non-linier *modular multiplication*.

Pencarian persamaan polinomial dari dua fungsi non-linier yaitu *modular addition* dan *modular multiplication* dilakukan dengan metode yang sama seperti pengubahan S-Box menjadi persamaan polinomial pada [2], dengan cara:

a. Mendaftar semua kemungkinan nilai *output* dari fungsi non-linier
b. Mendaftar semua kemungkinan nilai *output* dari fungsi non-linier
c. Mencari kombinasi yang sesuai dengan bit *output*
d. Mencari persamaan polinomial yang merepresentasikan relasi antara bit teks terang, kunci, dan teks sandi.
e. Penyelesaian sistem persamaan polinomial

4. HASIL DAN PEMBAHASAN

Pencarian polinomial pada proses enkripsi algoritme S-IDEA dari Langkah ke-1 sampai dengan Langkah ke-7, menggunakan teks terang sepanjang 16 bit yaitu 1001010001110010. Langkah ke-1 dan seterusnya dilakukan setelah teks terang dibagi menjadi 4 blok yang masing-masing bloknya berukuran 4 bit.

Input kunci pada algoritme S-IDEA berjumlah 32-bit sehingga pada penelitian ini dinotasikan sebagai $k_1, k_2, k_3, \ldots, k_{32}$. Langkah ke-1 sampai dengan Langkah ke-7 pada S-IDEA memerlukan 5 subkunci yang totalnya yaitu 20 bit. Lima subkunci ini digunakan pada Langkah ke-1, Langkah ke-2, Langkah ke-3, Langkah ke-4, dan Langkah ke-7 secara berturut-turut. Pada 32-bit kunci S-IDEA, diambil dua puluh bit pertama untuk subkunci dari Langkah ke-1 sampai Langkah ke-7. Selanjutnya dua puluh bit kunci tersebut dibagi menjadi lima bagian seperti pada Tabel 2.

Tabel 2. Subkunci untuk S-IDEA dari Langkah ke-1 sampai Langkah ke-7

Subkunci	Bit subkunci
Z_1	$k_1k_2k_3k_4$
Z_2	$k_5k_6k_7k_8$
Z_3	$k_9k_{10}k_{11}k_{12}$
Z_4	$k_{13}k_{14}k_{15}k_{16}$
Z_5	$k_{17}k_{18}k_{19}k_{20}$

Subkunci tersebut digunakan di dalam proses enkripsi algoritme S-IDEA. Sehingga hasil dari persamaan polinomial pada Langkah ke 7 yaitu:

$$k_1 + k_2 + k_3 + k_9 + k_{10} + k_{18} + k_{19} + k_{20} + k_1k_3 + k_1k_4 + k_2k_3 + k_2k_4 + k_3k_4 + k_1k_{11} + k_2k_{10} + k_1k_{12} + k_2k_{11} + k_2k_{12} + k_1k_{17} + k_1k_{18} + k_2k_{17} + k_2k_{19} + k_4k_{17} + \ldots + k_2k_3k_4k_{10}k_{11}k_{12}k_{18}k_{19}k_{20} \quad (1)$$

$$k_1 + k_2 + k_{10} + k_{19} + k_{20} + k_3k_4 + k_1k_{19} + k_2k_{18} + k_3k_{17} + k_3k_{18} + k_2k_{20} + k_4k_{18} + k_4k_{19} + \ldots + k_2k_3k_4k_{10}k_{11}k_{12}k_{18}k_{19}k_{20} \quad (2)$$

$$k_2 + k_{11} + k_{12} + k_{17} + k_{18} + k_{19} + k_3k_4 + k_1k_{17} + k_1k_{19} + k_3k_{17} + k_3k_{18} + k_4k_{17} + k_4k_{19} + k_9k_{18} + k_{10}k_{17} + k_9k_{19} + k_{11}k_{17} + k_{10}k_{19} + k_{11}k_{18} + k_{12}k_{17} + k_{12}k_{18} + \ldots + k_2k_3k_4k_{10}k_{11}k_{12}k_{19}k_{20} + 1 \quad (3)$$

$$k_3 + k_4 + k_{12} + k_{17} + k_{18} + k_{19} + k_{20} + k_1k_{18} + k_2k_{17} + k_1k_{19} + k_2k_{19} + k_3k_{18} + k_4k_{18} + k_4k_{19} + k_9k_{19} + k_{10}k_{18} + k_{11}k_{17} + \ldots + k_2k_3k_4k_{10}k_{11}k_{12}k_{18}k_{19}k_{20} \quad (4)$$

Persamaan (1) sampai Persamaan (4) merupakan persamaan polinomial yang didapatkan dari Langkah ke-7. Proses pencarian polinomial ini dari Langkah ke-1 sampai Langkah ke-7 dilakukan dengan bantuan *source code* Matlab. Persamaan polinomial yang dihasilkan pada Langkah ke-7 paling panjang terdiri dari 4721 monomial. Untuk mengetahui penerapan aljabar secara lengkap pada setiap proses enkripsi S-IDEA dilakukan penerapan serangan aljabar pada Miniatur S-IDEA.

Selanjutnya dilakukan penerapan serangan aljabar pada algoritme miniatur S-IDEA. Sama halnya dengan algoritme S-IDEA, pada miniatur S-IDEA juga terdapat fungsi non-linier *modular addition* dan *modular multiplication*. Kedua fungsi non-linier ini juga diubah ke dalam bentuk persamaan polinomial untuk dapat menghasilkan persamaan polinomial yang menghasilkan bit-bit teks sandi. Jika digambarkan ke dalam bentuk tabel maka operasi *modular addition* (2^2) dan *modular multiplication* ($2^2 + 1$) dalam bentuk desimal pada miniatur S-IDEA terlihat seperti pada Tabel 3 dan 4 secara berturut-turut.

Sehingga didapatkan 2 persamaan polinomial yang merepresentasikan fungsi non-linier *modular addition* dan 2 persamaan polinomial yang merepresentasikan fungsi non-linier *modular multiplication.*

Tabel 3. Hasil *Modular addition* dengan Input 2 Bit

	0	1	2	3
0	0	1	3	3
1	1	2	3	0
2	2	3	0	2
3	3	0	1	2

Tabel 4. Hasil *Modular Multiplication* dengan Input 2 Bit

	4	1	2	3
4	1	0	3	2
1	0	1	3	3
2	3	3	0	1
3	2	3	1	0

$$x_1 + x_3 + x_2x_4 \quad (5)$$

$$x_2 + x_4 \quad (6)$$

$$x_1 + x_3 \quad (7)$$

$$x_2 + x_4 + x_1x_3 + 1 \quad (8)$$

Selain itu diperlukan juga penentuan bit subkunci pada proses enkripsi miniatur S-IDEA. Input kunci pada miniatur S-IDEA berjumlah 16-bit sehingga pada penelitian ini dinotasikan sebagai $k_1, k_2, k_3, \ldots, k_{16}$. Pada *round* ke-1 miniatur S-IDEA memerlukan 6 subkunci yang totalnya yaitu 12 bit. Enam subkunci ini digunakan pada Langkah ke-1, Langkah ke-2, langkah3, Langkah ke-4, Langkah ke-7 dan Langkah ke-9. Pada 16-bit subkunci miniatur S-IDEA diambil 12 bit pertama sebagai subkunci *round* ke-1. Selanjutnya 12-bit kunci tersebut dibagi menjadi enam bagian seperti pada Tabel 5.

Tabel 5. Subkunci untuk Miniatur S-IDEA *Round* 1

Subkunci	Bit subkunci
Z_1^1	k_1k_2
Z_2^1	k_3k_4
Z_3^1	k_5k_6
Z_4^1	k_7k_8
Z_5^1	k_9k_{10}
Z_6^1	$k_{11}k_{12}$

Selanjutnya sisa 4-bit kunci digunakan sebagai subkunci untuk *round* berikutnya yaitu *round* 2. Untuk menambahkan sisa subkunci yang diperlukan pada *round* 2, kunci di-*rotate shift* ke kiri sebanyak 3 bit. Setelah itu sisa subkunci untuk *round* 2 diambil. Subkunci untuk *round* ke-2 ditunjukkan pada Tabel 6.

Tabel 6. Subkunci untuk Miniatur S-IDEA *Round* 2

Subkunci	Bit subkunci
Z_1^2	$k_{13}k_{14}$
Z_2^2	$k_{15}k_{16}$
Z_3^2	k_4k_5
Z_4^2	k_6k_7
Z_5^2	k_8k_8
Z_6^2	$k_{10}k_{11}$

Setelah didapatkan subkunci untuk *round* ke-2, maka sisa bit kunci yang telah di-*rotate shift* sebelumnya dimasukkan menjadi subkunci untuk *round* selanjutnya yaitu *round* terakhir. Subkunci untuk *round* terakhir ditunjukkan pada Tabel 7.

Tabel 7. Subkunci untuk Miniatur S-IDEA *Round* 3

Subkunci	Bit subkunci
Z_1^3	$k_{12}k_{13}$
Z_2^3	$k_{14}k_{15}$
Z_3^3	$k_{16}k_1$
Z_4^3	k_2k_3

Selanjutnya dilakukan pencarian polinomial pada proses enkripsi miniatur S-IDEA. Pada percobaan ini dinggunakan teks terang sepanjang 8 bit yaitu 01010010. Proses enkripsi dilakukan setelah teks terang dibagi menjadi 4 blok yang masing masing bloknya berukuran 2 bit. Proses pada dua setengah *round* miniature S-IDEA menghasilkan 8 persamaan polinomial yang merepresentasikan relasi antara bit teks terang, kunci, dan teks sandi:

$$c_1 = k_3 + k_5 + k_8 + k_9 + k_{11} + k_{13} + k_2k_4 + k_2k_7 + k_2k_8 + k_4k_6 + k_6k_7 + k_4k_{10} + k_6k_8 + k_7k_{10} + k_8k_{10} + k_1k_4k_9 + k_1k_7k_9 + k_1k_8k_9 + k_4k_5k_9 + k_5k_7k_9 + k_5k_8k_9 + 1$$

$$c_2 = k_4 + k_6 + k_7 + k_8 + k_{10} + k_{11} + k_{12} + k_{13} + k_{14} + k_1k_9 + k_1k_{11} + k_3k_{11} + k_5k_9 + k_3k_{13} + k_5k_{11} + k_5k_{13} + k_8k_{11} + k_9k_{11} + k_8k_{13} + k_9k_{13} + k_{11}k_{13} + k_2k_4k_{11} + k_2k_4k_{13} + k_2k_7k_{11} + k_2k_8k_{11} + k_4k_6k_{11} + k_2k_7k_{13} + k_2k_8k_{13} + k_4k_6k_{13} + k_6k_7k_{11} + k_4k_{10}k_{11} + k_6k_8k_{11} + k_6k_7k_{13} + k_4k_{10}k_{13} + k_6k_8k_{13} + k_7k_{10}k_{11} + k_8k_{10}k_{11} + k_7k_{10}k_{13} + k_8k_{10}k_{13} + k_1k_4k_9k_{11} + k_1k_4k_9k_{13} + k_1k_7k_9k_{11} + k_1k_8k_9k_{11} + k_4k_5k_9k_{11} + k_1k_7k_9k_{13} + k_1k_8k_9k_{13} + k_4k_5k_9k_{13} + k_5k_7k_9k_{11} + k_5k_8k_9k_{11} + k_5k_7k_9k_{13} + k_5k_8k_9k_{13} + 1$$

$$c_3 = k_7 + k_{12} + k_{15} + k_1k_{11} + k_2k_{11} + k_2k_{12} + k_3k_{11} + k_5k_{11} + k_6k_{11} + k_6k_{12} + k_8k_{11} + k_9k_{11} + k_{10}k_{11} + k_{10}k_{12} + k_7k_{16} + k_8k_{16} + k_{11}k_{16} + k_{12}k_{16} + k_1k_2k_{11} + k_2k_3k_{11} + k_1k_6k_{11} + k_2k_5k_{11} + k_3k_6k_{11} + k_1k_9k_{11} + k_2k_8k_{11} + k_1k_9k_{12} + k_1k_{10}k_{11} + k_2k_9k_{11} + k_5k_6k_{11} + k_3k_{10}k_{11} + k_5k_9k_{11} + k_6k_8k_{11} + k_5k_9k_{12} + k_5k_{10}k_{11} + k_6k_9k_{11} + k_1k_{11}k_{16} + k_8k_{10}k_{11} + k_3k_{11}k_{16} + k_9k_{10}k_{11} + k_5k_{11}k_{16} + k_8k_{11}k_{16} + k_9k_{11}k_{16} + k_1k_3k_9k_{11} + k_3k_5k_9k_{11} + k_1k_8k_9k_{11} + k_2k_4k_{11}k_{16} + k_5k_8k_9k_{11} + k_2k_7k_{11}k_{16} + k_2k_8k_{11}k_{16} + k_4k_6k_{11}k_{16} + k_6k_7k_{11}k_{16} + k_4k_{10}k_{11}k_{16} + k_6k_8k_{11}k_{16} + k_7k_{10}k_{11}k_{16} + k_8k_{10}k_{11}k_{16} + k_1k_4k_9k_{11}k_{16} + k_1k_7k_9k_{11}k_{16} + k_1k_8k_9k_{11}k_{16} + k_4k_5k_9k_{11}k_{16} + k_5k_7k_9k_{11}k_{16} + k_5k_8k_9k_{11}k_{16} + 1$$

$$c_4 = k_7 + k_8 + k_{11} + k_{12} + k_{16} + k_1k_{11} + k_3k_{11} + k_5k_{11} + k_8k_{11} + k_9k_{11} + k_2k_4k_{11} + k_2k_7k_{11} + k_2k_8k_{11} + k_4k_6k_{11} + k_6k_7k_{11} + k_4k_{10}k_{11} + k_6k_8k_{11} + k_7k_{10}k_{11} + k_8k_{10}k_{11} + k_1k_4k_9k_{11} + k_1k_7k_9k_{11} + k_1k_8k_9k_{11} + k_4k_5k_9k_{11} + k_5k_7k_9k_{11} + k_5k_8k_9k_{11}$$

$$c_5 = k_1 + k_3 + k_4 + k_8 + k_9 + k_{11} + k_2k_4 + k_2k_5 + k_2k_7 + k_4k_5 + k_2k_8 + k_4k_6 + k_5k_7 + k_5k_8 + k_6k_7 + k_4k_{10} + k_5k_9 + k_6k_8 + k_5k_{10} + k_5k_{12} + k_7k_{10} + k_8k_{10} + k_1k_4k_9 + k_1k_5k_9 + k_1k_5k_{11} + k_1k_7k_9 + k_1k_8k_9 + k_4k_5k_9 + k_3k_5k_{11} + k_5k_7k_9 + k_5k_8k_9 + k_5k_8k_{11} + k_5k_9k_{11} + k_2k_4k_5k_{11} + k_2k_5k_7k_{11} + k_2k_5k_8k_{11} + k_4k_5k_6k_{11} + k_4k_5k_9k_{11} + k_5k_6k_7k_{11} + k_4k_5k_{10}k_{11} + k_5k_6k_8k_{11} + k_5k_7k_9k_{11} + k_5k_7k_{10}k_{11} + k_5k_8k_9k_{11} + k_5k_8k_{10}k_{11} + k_1k_4k_5k_9k_{11} + k_1k_5k_7k_9k_{11} + k_1k_5k_8k_9k_{11} + 1$$

$$c_6 = k_2 + k_4 + k_5 + k_7 + k_8 + k_{10} + k_{11} + k_{12} + k_1k_9 + k_1k_{11} + k_3k_{11} + k_5k_9 + k_5k_{11} + k_8k_{11} + k_9k_{11} + k_2k_4k_{11} + k_2k_7k_{11} + k_2k_8k_{11} + k_4k_6k_{11} + k_6k_7k_{11} + k_4k_{10}k_{11} + k_6k_8k_{11} + k_7k_{10}k_{11} + k_8k_{10}k_{11} + k_1k_4k_9k_{11} + k_1k_7k_9k_{11} + k_1k_8k_9k_{11} + k_4k_5k_9k_{11} + k_5k_7k_9k_{11} + k_5k_8k_9k_{11}$$

$$c_7 = k_3 + k_4 + k_6 + k_{12} + k_1k_{11} + k_2k_{11} + k_2k_{12} + k_3k_{11} + k_5k_{11} + k_6k_{11} + k_6k_{12} + k_8k_{11} + k_9k_{11} + k_{10}k_{11} + k_{10}k_{12} + k_1k_2k_{11} + k_2k_3k_{11} + k_1k_6k_{11} + k_2k_5k_{11} + k_3k_6k_{11} + k_1k_9k_{11} + k_2k_8k_{11} + k_1k_9k_{12} +$$

$k_1k_{10}k_{11} + k_2k_9k_{11} + k_5k_6k_{11} + k_3k_{10}k_{11} + k_5k_9k_{11} + k_6k_8k_{11} + k_5k_9k_{12} + k_5k_{10}k_{11} + k_6k_9k_{11} + k_8k_{10}k_{11} + k_9k_{10}k_{11} + k_1k_3k_9k_{11} + k_3k_5k_9k_{11} + k_1k_8k_9k_{11} + k_5k_8k_9k_{11}$

$c_8 = k_4 + k_7 + k_{11} + k_{12} + k_3k_6 + k_4k_6 + k_1k_{11} + k_3k_{11} + k_5k_{11} + k_6k_{11} + k_8k_{11} + k_9k_{11} + k_2k_4k_{11} + k_2k_6k_{11} + k_2k_6k_{12} + k_2k_7k_{11} + k_2k_8k_{11} + k_4k_6k_{11} + k_6k_7k_{11} + k_4k_{10}k_{11} + k_6k_8k_{11} + k_6k_{10}k_{11} + k_6k_{10}k_{12} + k_7k_{10}k_{11} + k_8k_{10}k_{11} + k_1k_2k_6k_{11} + k_2k_3k_6k_{11} + k_2k_5k_6k_{11} + k_1k_4k_9k_{11} + k_1k_6k_9k_{11} + k_2k_6k_8k_{11} + k_1k_6k_9k_{12} + k_1k_6k_{10}k_{11} + k_1k_7k_9k_{11} + k_2k_6k_9k_{11} + k_1k_8k_9k_{11} + k_4k_5k_9k_{11} + k_3k_6k_{10}k_{11} + k_5k_6k_9k_{11} + k_5k_6k_9k_{12} + k_5k_6k_{10}k_{11} + k_5k_7k_9k_{11} + k_5k_8k_9k_{11} + k_6k_8k_{10}k_{11} + k_6k_9k_{10}k_{11} + k_1k_3k_6k_9k_{11} + k_3k_5k_6k_9k_{11} + k_1k_6k_8k_9k_{11} + k_5k_6k_8k_9k_{11} + 1$

Penyelesaian sistem persamaan polinomial yang merepresentasikan relasi antara bit teks terang, kunci, dan teks sandi satu setengah *round* miniatur S-IDEA dilakukan dengan XL *algorithm* dan basis Gröbner. Penggunaan XL *algorithm* pada penilitian ini didasarkan pada kesederhanaan tahapan dalam XL *algorithm*. Sistem persamaan polinomial yang diambil merupakan persamaan-persamaan polinomial yang merepresentasikan relasi antara bit teks terang, kunci, dan teks sandi pada 1,5 *round* miniatur S-IDEA.

Tahapan yang dilakukan dengan XL *algorithm* yaitu:

a. Memilih $D > d$

Nilai d pada tahap ini berarti derajat dari persamaan polinomial pada satu setengah *round* miniatur S-IDEA yaitu polinomial yang merepresentasikan C_1 sampai C_8, yaitu $d = 5$. Nilai D yang digunakan biasanya adalah $D = d + 1$, oleh karena itu dipilihlah $D = 6$.

b. Mendaftar seluruh monomial berderajat $\leq D - d$ termasuk monomial "1" yang berderajat 0.

Dari persamaan polinomial C_1 sampai C_8 dilakukan pendaftaran seluruh monomial. Hasil pendaftaran monomial yaitu termasuk monomial berderjat $\leq D - d$.

c. Mengalikan C_1 sampai C_8 dengan monomial berderajat $\leq D - d$

Pada tahap ini dilakukan perkalian persamaan polinomial C_1 sampai C_8 yang berjumlah delapan persamaan dengan monomial berderajat $\leq D - d$. Hal ini bertujuan untuk memudahkan proses analisis sehingga pada persamaan hasil perkalian memiliki derajat yang tidak terlalu tinggi. Perkalian dilakukan dengan *source code* pada program Matlab. Perkalian ini menghasilkan 136 persamaan polinomial baru. Jumlah ini didapatkan dari 8 persamaan C_1 sampai C_8 dikalikan dengan 17 monomial berderajat $\leq D - d$.

d. Mengubah sistem persamaan polinomial menjadi sistem persamaan linier.

Pada tahap ini dilakukan linierisasi terhadap hasil perkalian. Hal ini dikarenakan jumlah monomial yang ada pada 136 persamaan polinomial yang baru lebih banyak dari pada jumlah persamaan polinomialnya. Monomial yang terdaftar pada sistem persamaan linier disubstitusi sedemikian sehingga persamaan polinomial menjadi persamaan linier. Subtitusi yang dilakukan berjumlah $C_1^{16} + C_2^{16} + \cdots + C_{16}^{16} = 65535$, hal ini dikarenakan panjang kunci pada miniatur S-IDEA adalah 16 bit. Selanjutnya, linierisasi dilakukan dengan bantuan *source code* program Matlab. Persamaan hasil linierisasi mengandung 1512 suku. Jika persamaan ini direpresentasikan ke dalam bentuk matriks, maka akan didapatkan matriks berukuran 1512x136. Validasi dari persamaan hasil linierisasi dapat dilakukan dengan melakukan substitusi kembali dari variabel linier menjadi monomial yang bersesuaian. Setelah itu persamaan polinomial dapat disubstitusi dengan informasi yang sudah dimiliki sebelumnya.

e. Mencari solusi dari sistem persamaan linier menggunakan aljabar linier

Pencarian solusi dari sistem persamaan linier dapat dilakukan menggunakan beberapa metode seperti substitusi dan eliminasi, metode Gauss Jordan, *divide and conquer*, ataupun gabungan beberapa metode. Penerapan metode untuk pencarian solusi dari sistem persamaan linier dapat memanfaatkan data sebelumnya ataupun dilakukan ekspansi lebih lanjut. Penerapan metode Gauss Jordan pada penelitian ini hanya dilakukan sampai tahap linierisasi dikarenakan terbatasnya komputasi.

Disamping hal tersebut, dilakukan juga pencarian solusi dengan mengunakan basis Gröbner. Pencarian solusi dengan basis Gröbner didasarkan pada persamaan linier hasil linierisasi XL *algorithm* yang masih cukup panjang. Basis Gröbner dilakukan terhadap 8 persamaan polinomial yang merepresentasikan relasi antara bit teks terang, kunci, dan teks sandi 1,5 *round* miniatur S-IDEA dengan bantuan perangkat lunak SageMath 9.1. Basis Gröbner menghasilkan 34 polinomial baru yang mayoritas panjangnya hampir sama dengan 8 persamaan polinomial awal. Sehingga masih belum bisa diperoleh nilai kunci secara langsung.

4.1 Pembahasan

Berdasarkan penelitian yang telah dilakukan dan hasil yang diperoleh yaitu persamaan polinomial pada Langkah ke-7 algoritme S-IDEA, persamaan polinomial yang merepresentasikan relasi antara bit teks terang, kunci, dan teks sandi dari miniatur S-IDEA, dan 136 persamaan hasil linierisasi maka diketahui perbandingan jumlah monomial di dalam persamaan polinomial Langkah ke-1 sampai Langkah ke-7 algoritme S-IDEA dan miniatur S-IDEA ditunjukkan pada Tabel 8.

Tabel 8. Perbandingan Jumlah Monomial S-IDEA dan Miniatur S-IDEA Langkah ke-1 sampai Langkah ke-7

Langkah	Jumlah maksimal monomial pada S-IDEA	Jumlah maksimal monomial pada miniatur S-IDEA
1	14	1
2	2	2
3	8	1
4	12	3
5	22	2
6	13	4
7	4721	6

Tabel 8 menunjukkan perbandingan jumlah monomial yang cukup besar. Hal tersebut disebabkan adanya beberapa perbedaan pada algoritme yang diserang. Pada algoritme S-IDEA digunakan teks terang dan teks sandi sepanjang 16 bit, sementara pada miniatur S-IDEA menggunakan teks terang dan teks sandi sepanjang 8 bit. Selain itu, panjang kunci yang digunakan pun berbeda. Pada S-IDEA panjang kunci yang digunakan yaitu 32 bit, sementara pada miniatur S-IDEA adalah 16 bit. Ukuran blok pun memiliki perbedaan, pada S-IDEA ukuran blok yaitu 4 bit, sementara pada miniatur S-IDEA ukuran bloknya adalah 2 bit.

5. KESIMPULAN

Kesimpulan dari penelitian yang telah dilakukan adalah sebagai berikut:

a. Telah dilakukan pencarian dan didapatkan persamaan polinomial sampai dengan Langkah ke-7 algoritme S-IDEA dan diperoleh 4 persamaan polinomial dengan panjang maksimal terdiri dari 4271 monomial.
b. Telah dilakukan pencarian dan didapatkan persamaan polinomial 2,5 *round* miniatur S-IDEA dan diperoleh 8 persamaan polinomial dengan panjang maksimal terdiri dari 1109 monomial.
c. Besar operasi per blok, jumlah bit kunci, dan jumlah fungsi non-linier yang terlibat mempengaruhi persamaan polinomial yang merepresentasikan relasi antara bit teks terang, kunci, dan teks sandi.
d. Telah dilakukan penerapan XL-*Algorithm* dari Langkah ke-1 sampai Langkah ke-4 dengan nilai D=6 dan diperoleh 136 persamaan linier hasil expansi dan linierisasi.
e. Telah dilakukan penerapan basis Gröbner terhadap 8 persamaan polinomial yang merepresentasikan relasi antara bit teks terang, kunci, dan teks sandi dan dihasilkan 34 persamaan polinomial baru.
f. Meskipun sudah didapatkan sistem persamaan polinomial yang merepresentasikan relasi antara bit teks terang, kunci, dan teks sandi, tetap diperlukan komputasi yang cukup besar ataupun analisis lebih lanjut terhadap sistem persamaan polinomial untuk memperoleh bit kunci yang digunakan.

REFERENSI

[1] M. Voros, *Algebraic Attack on Stream Ciphers*, Comenius University, 2007.

[2] S. Simmons, *Algebraic Cryptanalysis of Simplified AES*, Cryptologia, pp. 305-314, 2009.

[3] B. Preneel, J. Vandewalle dan J. Nakahara, *Cryptanalysis and Design of Block cipher,* Leuven: Katholieke Universiteit Leuven, 2003.

[4] G. V. Bard dan N. T. Courtois, *Algebraic Cryptanalysis of the Data Encryption Algorithm*, IMA International Conference on Cryptography and Coding, pp. 152-169, 2007.

[5] F. Paradise, Serangan Aljabar pada *Simplified Data Encryption Standar* (S-DES) dengan metode *XL algorithm*, Bogor: Sekolah Tinggi Sandi Negara, 2018.

[6] T. Sundari, Serangan Aljabar pada Algoritme Mini-AES, Sekolah Tinggi Sandi Negara, 2015.

[7] N. Hoffman, *A Simplified IDEA Algorithm*, Crytplogia, pp. 143-151, 2007.

[8] H. K. Sahu, V. Jadhav, S. Sonavane dan R. K. Sharma, Cryptanalytic Attacks on *International Data Encryption Algorithm Block cipher*, Defence Science Journal, vol. 66, pp. 582-589, 2016.

[9] A. Kipnis dan A. Shamir, Cryptanalysis of the HFE Public *Key* Cryptosystem by Relinearization, Annual International Cryptology Conference, pp. 19-30, 1999.

[10] N. Courtois, A. Klimov, J. Patarin dan A. Shamir, *Efficient Algorithm for Solving Overdefined Systems of Multivariate Polynomial Equations*, EUROCRYPT 2000, pp. 392-407, 2000.

[11] G. V. Bard, *Algebraic Cryptanalysis*, Springer Science & Bussiness Media, 2009.

[12] B. Buchberger, Bruno Buchberger's PhD thesis 1965: *An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal*, Symbolic Computation, vol. 41, pp. 475 - 551, 2006.

[13] I. Ajwa, Z. Liu dan P. Wang, *Gröbner Bases Algorithm*, Ohio: Kent State University, 2003.