Eliminating variables in Boolean equation systems

Bjørn Møller Greve[1,2] Håvard Raddum[2] Gunnar Fløystad[3] Øyvind Ytrehus[2]

[1]Norwegian Defence Research Establishment

[2]Simula@UiB

[3]Dept. of Mathematics, UiB

July 5, 2017

simula@uib

FFI Forsvarets forskningsinstitutt
Norwegian Defence Research Establishment

Boolean functions

- $B[1,n] = \mathbb{F}_2[x_1,\ldots,x_n]/(x_i^2 + x_i | i = 1,\ldots,n)$
- Set of Boolean equations $F = \{f_1,\ldots,f_s\}$ in $B[1,n] \leftrightarrow F$ generate an ideal $I(F) = (f_1,\ldots,f_s)$, with zero set $Z(I(F)) = \{\mathbf{a} \in \mathbb{F}_2^n | f(\mathbf{a}) = 0 \text{ for every } f \in I(F)\}$.

Elimination of variables from Boolean functions

- Objective: Given $I(F) \subset B[1,n]$ we want to find $I'(F) \subset B[2,n]$ s.th $Z(I'(F)) = \pi_1(Z(I(F))) \leftrightarrow$ Compute $J \subset I'(F)$ as large as possible given computational restrictions.
- In general: We can eliminate more variables in the same fashion $\rightarrow k$'th elimination ideal $I(F) \cap B[k+1,n]$.
- Without loss of generality we eliminate variables in the order $x_1, x_2, \ldots, x_n$.

Boolean functions

- $B[1,n] = \mathbb{F}_2[x_1,\dots,x_n]/(x_i^2 + x_i | i = 1,\dots,n)$
- Set of Boolean equations $F = \{f_1,\dots,f_s\}$ in $B[1,n] \leftrightarrow F$ generate an ideal $I(F) = (f_1,\dots,f_s)$, with zero set $Z(I(F)) = \{\mathbf{a} \in \mathbb{F}_2^n | f(\mathbf{a}) = 0 \text{ for every } f \in I(F)\}$.

Elimination of variables from Boolean functions

- Objective: Given $I(F) \subset B[1,n]$ we want to find $I'(F) \subset B[2,n]$ s.th $Z(I'(F)) = \pi_1(Z(I(F))) \leftrightarrow$ Compute $J \subset I'(F)$ as large as possible given computational restrictions.
- In general: We can eliminate more variables in the same fashion $\rightarrow k$'th elimination ideal $I(F) \cap B[k+1,n]$.
- Without loss of generality we eliminate variables in the order $x_1, x_2, \dots, x_n$.

Boolean functions

- $B[1,n] = \mathbb{F}_2[x_1,\dots,x_n]/(x_i^2 + x_i | i = 1,\dots,n)$
- Set of Boolean equations $F = \{f_1,\dots,f_s\}$ in $B[1,n] \leftrightarrow F$ generate an ideal $I(F) = (f_1,\dots,f_s)$, with zero set $Z(I(F)) = \{\mathbf{a} \in \mathbb{F}_2^n | f(\mathbf{a}) = 0$ for every $f \in I(F)\}$.

Elimination of variables from Boolean functions

- Objective: Given $I(F) \subset B[1,n]$ we want to find $I'(F) \subset B[2,n]$ s.th $Z(I'(F)) = \pi_1(Z(I(F))) \leftrightarrow$ Compute $J \subset I'(F)$ as large as possible given computational restrictions.
- In general: We can eliminate more variables in the same fashion $\rightarrow k$'th elimination ideal $I(F) \cap B[k+1,n]$.
- Without loss of generality we eliminate variables in the order $x_1, x_2, \dots, x_n$.

Boolean functions

- $B[1,n] = \mathbb{F}_2[x_1,\dots,x_n]/(x_i^2 + x_i | i = 1,\dots,n)$
- Set of Boolean equations $F = \{f_1,\dots,f_s\}$ in $B[1,n] \leftrightarrow F$ generate an ideal $I(F) = (f_1,\dots,f_s)$, with zero set $Z(I(F)) = \{\mathbf{a} \in \mathbb{F}_2^n | f(\mathbf{a}) = 0$ for every $f \in I(F)\}$.

Elimination of variables from Boolean functions

- Objective: Given $I(F) \subset B[1,n]$ we want to find $I'(F) \subset B[2,n]$ s.th $Z(I'(F)) = \pi_1(Z(I(F))) \leftrightarrow$ Compute $J \subset I'(F)$ as large as possible given computational restrictions.
- In general: We can eliminate more variables in the same fashion $\rightarrow k$'th elimination ideal $I(F) \cap B[k+1,n]$.
- Without loss of generality we eliminate variables in the order $x_1, x_2, \dots, x_n$.

Boolean functions

- $B[1,n] = \mathbb{F}_2[x_1,\ldots,x_n]/(x_i^2 + x_i | i = 1,\ldots,n)$
- Set of Boolean equations $F = \{f_1,\ldots,f_s\}$ in $B[1,n] \leftrightarrow F$ generate an ideal $I(F) = (f_1,\ldots,f_s)$, with zero set $Z(I(F)) = \{\mathbf{a} \in \mathbb{F}_2^n | f(\mathbf{a}) = 0$ for every $f \in I(F)\}$.

Elimination of variables from Boolean functions

- Objective: Given $I(F) \subset B[1,n]$ we want to find $I'(F) \subset B[2,n]$ s.th $Z(I'(F)) = \pi_1(Z(I(F))) \leftrightarrow$ Compute $J \subset I'(F)$ as large as possible given computational restrictions.
- In general: We can eliminate more variables in the same fashion $\rightarrow k$'th elimination ideal $I(F) \cap B[k+1,n]$.
- Without loss of generality we eliminate variables in the order $x_1, x_2, \ldots, x_n$.

Boolean functions

- $B[1,n] = \mathbb{F}_2[x_1,\ldots,x_n]/(x_i^2 + x_i | i = 1,\ldots,n)$
- Set of Boolean equations $F = \{f_1,\ldots,f_s\}$ in $B[1,n] \leftrightarrow F$ generate an ideal $I(F) = (f_1,\ldots,f_s)$, with zero set
  $Z(I(F)) = \{\mathbf{a} \in \mathbb{F}_2^n | f(\mathbf{a}) = 0$ for every $f \in I(F)\}$.

Elimination of variables from Boolean functions

- Objective: Given $I(F) \subset B[1,n]$ we want to find $I'(F) \subset B[2,n]$ s.th $Z(I'(F)) = \pi_1(Z(I(F))) \leftrightarrow$ Compute $J \subset I'(F)$ as large as possible given computational restrictions.
- In general: We can eliminate more variables in the same fashion $\rightarrow k$'th elimination ideal $I(F) \cap B[k+1,n]$.
- Without loss of generality we eliminate variables in the order $x_1, x_2, \ldots, x_n$.

The Elimination Theorem

Theorem

If $G(F)$ is a Gröbner basis for the ideal $I(F)$ with respect to the (lex) order $x_1 > x_2 > \cdots > x_n$, then

$$G_k(F) = G(F) \cap B[k+1, n]$$

is a Gröbner basis of the k'th elimination ideal $I^k(F)$.

\+ Computes the full elimination ideal

\+ Preserves all "exact" solutions of the original system

1. — We have to compute the *full* Gröbner basis *before* elimination.
2. — Eliminates one monomial at the time.
3. — Gröbner bases are hard to compute → high complexity (All possible degrees)

The Elimination Theorem

Theorem

If $G(F)$ is a Gröbner basis for the ideal $I(F)$ with respect to the (lex) order $x_1 > x_2 > \cdots > x_n$, then

$$G_k(F) = G(F) \cap B[k+1, n]$$

is a Gröbner basis of the k'th elimination ideal $I^k(F)$.

- + Computes the full elimination ideal
- + Preserves all "exact" solutions of the original system

1. − We have to compute the *full* Gröbner basis *before* elimination.
2. − Eliminates one monomial at the time.
3. − Gröbner bases are hard to compute → high complexity (All possible degrees)

The Elimination Theorem

Theorem

If $G(F)$ is a Gröbner basis for the ideal $I(F)$ with respect to the (lex) order $x_1 > x_2 > \cdots > x_n$, then

$$G_k(F) = G(F) \cap B[k+1, n]$$

is a Gröbner basis of the k'th elimination ideal $I^k(F)$.

+ Computes the full elimination ideal
+ Preserves all "exact" solutions of the original system

1. — We have to compute the *full* Gröbner basis *before* elimination.
2. — Eliminates one monomial at the time.
3. — Gröbner bases are hard to compute → high complexity (All possible degrees)

The Elimination Theorem

Theorem

If $G(F)$ is a Gröbner basis for the ideal $I(F)$ with respect to the (lex) order $x_1 > x_2 > \cdots > x_n$, then

$$G_k(F) = G(F) \cap B[k+1, n]$$

is a Gröbner basis of the k'th elimination ideal $I^k(F)$.

+ Computes the full elimination ideal
+ Preserves all "exact" solutions of the original system

1. − We have to compute the *full* Gröbner basis *before* elimination.
2. − Eliminates one monomial at the time.
3. − Gröbner bases are hard to compute → high complexity (All possible degrees)

The Elimination Theorem

Theorem

If $G(F)$ is a Gröbner basis for the ideal $I(F)$ with respect to the (lex) order $x_1 > x_2 > \cdots > x_n$, then

$$G_k(F) = G(F) \cap B[k+1, n]$$

is a Gröbner basis of the k'th elimination ideal $I^k(F)$.

+ Computes the full elimination ideal
+ Preserves all "exact" solutions of the original system

1. − We have to compute the *full* Gröbner basis *before* elimination.
2. − Eliminates one monomial at the time.
3. − Gröbner bases are hard to compute → high complexity (All possible degrees)

Symmetric cryptography

- Defined over the binary field $GF(2)$ $\rightarrow$ block encryption algorithms $E_K(P) = C$ takes a fixed length plaintext P and a secret key K as inputs, and produces a ciphertext C.
- Divides the data into blocks of fixed size, and then encrypting each block separately. The encryption usually consists of iterating a *round function*, consisting of suitable linear and nonlinear transformations
- A known plaintext attack: Assume both P and C are known. Objective: Extract the secret key K.

Boolean functions in cryptography

Ciphers defined over $GF(2)$ can always be described as a system of Boolean equations of degree 2 $\rightarrow$ introduce enough auxiliary variables $\rightarrow$ Solving this system of equations w.r.t K: Algebraic cryptanalysis.

- The bits of the cipher states during encryption can always be described as polynomials in the user-selected key!
- Over multiple rounds in a block cipher algorithm, the degree of the polynomials in only user-selected key bits grow fast, making the equations hard to solve.

Symmetric cryptography

- Defined over the binary field $GF(2) \rightarrow$ block encryption algorithms $E_K(P) = C$ takes a fixed length plaintext P and a secret key K as inputs, and produces a ciphertext C.
- Divides the data into blocks of fixed size, and then encrypting each block separately. The encryption usually consists of iterating a *round function*, consisting of suitable linear and nonlinear transformations
- A known plaintext attack: Assume both P and C are known. Objective: Extract the secret key K.

Boolean functions in cryptography

Ciphers defined over $GF(2)$ can always be described as a system of Boolean equations of degree $2 \rightarrow$ introduce enough auxiliary variables $\rightarrow$ Solving this system of equations w.r.t K: Algebraic cryptanalysis.

- The bits of the cipher states during encryption can always be described as polynomials in the user-selected key!
- Over multiple rounds in a block cipher algorithm, the degree of the polynomials in only user-selected key bits grow fast, making the equations hard to solve.

Symmetric cryptography

- Defined over the binary field $GF(2) \rightarrow$ block encryption algorithms $E_K(P) = C$ takes a fixed length plaintext P and a secret key K as inputs, and produces a ciphertext C.
- Divides the data into blocks of fixed size, and then encrypting each block separately. The encryption usually consists of iterating a *round function*, consisting of suitable linear and nonlinear transformations
- A known plaintext attack: Assume both P and C are known. Objective: Extract the secret key K.

Boolean functions in cryptography

Ciphers defined over $GF(2)$ can always be described as a system of Boolean equations of degree $2 \rightarrow$ introduce enough auxiliary variables $\rightarrow$ Solving this system of equations w.r.t K: Algebraic cryptanalysis.

- The bits of the cipher states during encryption can always be described as polynomials in the user-selected key!
- Over multiple rounds in a block cipher algorithm, the degree of the polynomials in only user-selected key bits grow fast, making the equations hard to solve.

Symmetric cryptography

- Defined over the binary field $GF(2) \rightarrow$ block encryption algorithms $E_K(P) = C$ takes a fixed length plaintext P and a secret key K as inputs, and produces a ciphertext C.
- Divides the data into blocks of fixed size, and then encrypting each block separately. The encryption usually consists of iterating a *round function*, consisting of suitable linear and nonlinear transformations
- A known plaintext attack: Assume both P and C are known. Objective: Extract the secret key K.

Boolean functions in cryptography

Ciphers defined over $GF(2)$ can always be described as a system of Boolean equations of degree $2 \rightarrow$ introduce enough auxiliary variables $\rightarrow$ Solving this system of equations w.r.t K: Algebraic cryptanalysis.

- The bits of the cipher states during encryption can always be described as polynomials in the user-selected key!
- Over multiple rounds in a block cipher algorithm, the degree of the polynomials in only user-selected key bits grow fast, making the equations hard to solve.

Symmetric cryptography

- Defined over the binary field $GF(2) \rightarrow$ block encryption algorithms $E_K(P) = C$ takes a fixed length plaintext P and a secret key K as inputs, and produces a ciphertext C.
- Divides the data into blocks of fixed size, and then encrypting each block separately. The encryption usually consists of iterating a *round function*, consisting of suitable linear and nonlinear transformations
- A known plaintext attack: Assume both P and C are known. Objective: Extract the secret key K.

Boolean functions in cryptography

Ciphers defined over $GF(2)$ can always be described as a system of Boolean equations of degree $2 \rightarrow$ introduce enough auxiliary variables $\rightarrow$ Solving this system of equations w.r.t K: Algebraic cryptanalysis.

- The bits of the cipher states during encryption can always be described as polynomials in the user-selected key!
- Over multiple rounds in a block cipher algorithm, the degree of the polynomials in only user-selected key bits grow fast, making the equations hard to solve.

Symmetric cryptography

- Defined over the binary field $GF(2) \rightarrow$ block encryption algorithms $E_K(P) = C$ takes a fixed length plaintext P and a secret key K as inputs, and produces a ciphertext C.
- Divides the data into blocks of fixed size, and then encrypting each block separately. The encryption usually consists of iterating a *round function*, consisting of suitable linear and nonlinear transformations
- A known plaintext attack: Assume both P and C are known. Objective: Extract the secret key K.

Boolean functions in cryptography

Ciphers defined over $GF(2)$ can always be described as a system of Boolean equations of degree $2 \rightarrow$ introduce enough auxiliary variables $\rightarrow$ Solving this system of equations w.r.t K: Algebraic cryptanalysis.

- The bits of the cipher states during encryption can always be described as polynomials in the user-selected key!

Symmetric cryptography

- Defined over the binary field $GF(2)$ $\rightarrow$ block encryption algorithms $E_K(P) = C$ takes a fixed length plaintext P and a secret key K as inputs, and produces a ciphertext C.
- Divides the data into blocks of fixed size, and then encrypting each block separately. The encryption usually consists of iterating a *round function*, consisting of suitable linear and nonlinear transformations
- A known plaintext attack: Assume both P and C are known. Objective: Extract the secret key K.

Boolean functions in cryptography

Ciphers defined over $GF(2)$ can always be described as a system of Boolean equations of degree 2 $\rightarrow$ introduce enough auxiliary variables $\rightarrow$ Solving this system of equations w.r.t K: Algebraic cryptanalysis.

- The bits of the cipher states during encryption can always be described as polynomials in the user-selected key!
- Over multiple rounds in a block cipher algorithm, the degree of the polynomials in only user-selected key bits grow fast, making the equations hard to solve.

The block cipher problem

If we start with a description of a block cipher as a system of equations of degree 2 using "many" variables, is it possible to efficiently eliminate all the auxiliary variables, such that we end up with *some* low-degree equations in which the only variables are the bits of K?

NB!

We are guaranteed that the correct key K is one solution to this system, but restricting the degree means that we get many false keys as well.

How to solve equations after elimination

1. The general method: Enumerating the possible solutions to the final system and "lifting" these through the intermediate systems to filter out false solutions.
2. The block cipher method: Repeating the process of variable elimination using other known plaintext/ciphertext pairs and build up a low-degree system of equations in only user-selected key variables that has K as a unique solution.
3. Low degree system $\leftrightarrow$ solve by re-linearization if we have enough polynomials $\leftrightarrow$ repeat elimination until by brute force is possible.

The block cipher problem

If we start with a description of a block cipher as a system of equations of degree 2 using "many" variables, is it possible to efficiently eliminate all the auxiliary variables, such that we end up with *some* low-degree equations in which the only variables are the bits of K?

NB!

We are guaranteed that the correct key K is one solution to this system, but restricting the degree means that we get many false keys as well.

How to solve equations after elimination

1. The general method: Enumerating the possible solutions to the final system and "lifting" these through the intermediate systems to filter out false solutions.
2. The block cipher method: Repeating the process of variable elimination using other known plaintext/ciphertext pairs and build up a low-degree system of equations in only user-selected key variables that has K as a unique solution.
3. Low degree system $\leftrightarrow$ solve by re-linearization if we have enough polynomials $\leftrightarrow$ repeat elimination until by brute force is possible.

The block cipher problem

If we start with a description of a block cipher as a system of equations of degree 2 using "many" variables, is it possible to efficiently eliminate all the auxiliary variables, such that we end up with *some* low-degree equations in which the only variables are the bits of K?

NB!

We are guaranteed that the correct key K is one solution to this system, but restricting the degree means that we get many false keys as well.

How to solve equations after elimination

1. The general method: Enumerating the possible solutions to the final system and "lifting" these through the intermediate systems to filter out false solutions.
2. The block cipher method: Repeating the process of variable elimination using other known plaintext/ciphertext pairs and build up a low-degree system of equations in only user-selected key variables that has K as a unique solution.
3. Low degree system $\leftrightarrow$ solve by re-linearization if we have enough polynomials $\leftrightarrow$ repeat elimination until by brute force is possible.

The block cipher problem

If we start with a description of a block cipher as a system of equations of degree 2 using "many" variables, is it possible to efficiently eliminate all the auxiliary variables, such that we end up with *some* low-degree equations in which the only variables are the bits of K?

NB!

We are guaranteed that the correct key K is one solution to this system, but restricting the degree means that we get many false keys as well.

How to solve equations after elimination

1. The general method: Enumerating the possible solutions to the final system and "lifting" these through the intermediate systems to filter out false solutions.
2. The block cipher method: Repeating the process of variable elimination using other known plaintext/ciphertext pairs and build up a low-degree system of equations in only user-selected key variables that has K as a unique solution.
3. Low degree system $\leftrightarrow$ solve by re-linearization if we have enough polynomials $\leftrightarrow$ repeat elimination until by brute force is possible.

The block cipher problem

If we start with a description of a block cipher as a system of equations of degree 2 using "many" variables, is it possible to efficiently eliminate all the auxiliary variables, such that we end up with *some* low-degree equations in which the only variables are the bits of K?

NB!

We are guaranteed that the correct key K is one solution to this system, but restricting the degree means that we get many false keys as well.

How to solve equations after elimination

1. The general method: Enumerating the possible solutions to the final system and "lifting" these through the intermediate systems to filter out false solutions.
2. The block cipher method: Repeating the process of variable elimination using other known plaintext/ciphertext pairs and build up a low-degree system of equations in only user-selected key variables that has K as a unique solution.
3. Low degree system $\leftrightarrow$ solve by re-linearization if we have enough polynomials $\leftrightarrow$ repeat elimination until by brute force is possible.

The block cipher problem

If we start with a description of a block cipher as a system of equations of degree 2 using "many" variables, is it possible to efficiently eliminate all the auxiliary variables, such that we end up with *some* low-degree equations in which the only variables are the bits of K?

NB!

We are guaranteed that the correct key K is one solution to this system, but restricting the degree means that we get many false keys as well.

How to solve equations after elimination

1. The general method: Enumerating the possible solutions to the final system and "lifting" these through the intermediate systems to filter out false solutions.
2. The block cipher method: Repeating the process of variable elimination using other known plaintext/ciphertext pairs and build up a low-degree system of equations in only user-selected key variables that has K as a unique solution.
3. Low degree system $\leftrightarrow$ solve by re-linearization if we have enough polynomials $\leftrightarrow$ repeat elimination until by brute force is possible.

Our contribution

- Trade-off: The ability to control the degree vs the ability to stay close to the elimination ideal $I \cap B[k+1, n]$.
- Minimize complexity $\leftrightarrow$ Only consider polynomials of degree ≤ 3 $\leftrightarrow$ $F = \{f_1, \ldots, f_c\}$, $G = \{g_1, \ldots, g_q\}$, f_i's have degree 3 and the g_i's degrees 2.
- Objective: Find as many polynomials in the ideal $I(F, G)$ of degree ≤ 3 as we can $\leftrightarrow$ Try to produce degree 3 or less in only *key variables* when applied to block ciphers.

— Eliminating variables while keeping degree ≤ 3 $\rightarrow$ introduce false solutions.

- $L = \{1, x_1, \ldots, x_n\} \rightarrow \langle L \rangle \rightarrow$ vector space spanned by the Boolean polynomials.
- Eliminate variables from the vector space $\langle F \cup LG \rangle$ $\leftrightarrow$ $LG = \{lg$ where $l \in L$ and $g \in G\}$.

Our contribution

- Trade-off: The ability to control the degree vs the ability to stay close to the elimination ideal $I \cap B[k+1, n]$.
- Minimize complexity $\leftrightarrow$ Only consider polynomials of degree $\leq 3 \leftrightarrow$ $F = \{f_1, \ldots, f_c\}$, $G = \{g_1, \ldots, g_q\}$, f_i's have degree 3 and the g_i's degrees 2.
- Objective: Find as many polynomials in the ideal $I(F, G)$ of degree ≤ 3 as we can $\leftrightarrow$ Try to produce degree 3 or less in only *key variables* when applied to block ciphers.

— Eliminating variables while keeping degree $\leq 3 \rightarrow$ introduce false solutions.

- $L = \{1, x_1, \ldots, x_n\} \rightarrow \langle L \rangle \rightarrow$ vector space spanned by the Boolean polynomials.
- Eliminate variables from the vector space $\langle F \cup LG \rangle \leftrightarrow$ $LG = \{lg \text{ where } l \in L \text{ and } g \in G\}$.

Our contribution

- Trade-off: The ability to control the degree vs the ability to stay close to the elimination ideal $I \cap B[k+1, n]$.
- Minimize complexity $\leftrightarrow$ Only consider polynomials of degree $\leq 3 \leftrightarrow$ $F = \{f_1, \ldots, f_c\}$, $G = \{g_1, \ldots, g_q\}$, f_i's have degree 3 and the g_i's degrees 2.
- Objective: Find as many polynomials in the ideal $I(F, G)$ of degree ≤ 3 as we can $\leftrightarrow$ Try to produce degree 3 or less in only *key variables* when applied to block ciphers.

– Eliminating variables while keeping degree $\leq 3 \rightarrow$ introduce false solutions.

- $L = \{1, x_1, \ldots, x_n\} \rightarrow \langle L \rangle \rightarrow$ vector space spanned by the Boolean polynomials.
- Eliminate variables from the vector space $\langle F \cup LG \rangle \leftrightarrow$ $LG = \{lg \text{ where } l \in L \text{ and } g \in G\}$.

Our contribution

- Trade-off: The ability to control the degree vs the ability to stay close to the elimination ideal $I \cap B[k+1, n]$.
- Minimize complexity $\leftrightarrow$ Only consider polynomials of degree $\leq 3 \leftrightarrow$ $F = \{f_1, \ldots, f_c\}$, $G = \{g_1, \ldots, g_q\}$, f_i's have degree 3 and the g_i's degrees 2.
- Objective: Find as many polynomials in the ideal $I(F, G)$ of degree ≤ 3 as we can $\leftrightarrow$ Try to produce degree 3 or less in only *key variables* when applied to block ciphers.

– Eliminating variables while keeping degree $\leq 3 \rightarrow$ introduce false solutions.

- $L = \{1, x_1, \ldots, x_n\} \rightarrow \langle L \rangle \rightarrow$ vector space spanned by the Boolean polynomials.
- Eliminate variables from the vector space $\langle F \cup LG \rangle \leftrightarrow$ $LG = \{lg \text{ where } l \in L \text{ and } g \in G\}$.

Our contribution

- Trade-off: The ability to control the degree vs the ability to stay close to the elimination ideal $I \cap B[k+1, n]$.
- Minimize complexity $\leftrightarrow$ Only consider polynomials of degree $\leq 3 \leftrightarrow$ $F = \{f_1, \dots, f_c\}$, $G = \{g_1, \dots, g_q\}$, f_i's have degree 3 and the g_i's degrees 2.
- Objective: Find as many polynomials in the ideal $I(F, G)$ of degree ≤ 3 as we can $\leftrightarrow$ Try to produce degree 3 or less in only *key variables* when applied to block ciphers.

– Eliminating variables while keeping degree $\leq 3 \rightarrow$ introduce false solutions.

- $L = \{1, x_1, \dots, x_n\} \rightarrow \langle L \rangle \rightarrow$ vector space spanned by the Boolean polynomials.
- Eliminate variables from the vector space $\langle F \cup LG \rangle \leftrightarrow$ $LG = \{lg \text{ where } l \in L \text{ and } g \in G\}$.

Our contribution

- Trade-off: The ability to control the degree vs the ability to stay close to the elimination ideal $I \cap B[k+1, n]$.
- Minimize complexity $\leftrightarrow$ Only consider polynomials of degree $\leq 3 \leftrightarrow$ $F = \{f_1, \ldots, f_c\}$, $G = \{g_1, \ldots, g_q\}$, f_i's have degree 3 and the g_i's degrees 2.
- Objective: Find as many polynomials in the ideal $I(F, G)$ of degree ≤ 3 as we can $\leftrightarrow$ Try to produce degree 3 or less in only *key variables* when applied to block ciphers.

– Eliminating variables while keeping degree $\leq 3 \rightarrow$ introduce false solutions.

- $L = \{1, x_1, \ldots, x_n\} \rightarrow \langle L \rangle \rightarrow$ vector space spanned by the Boolean polynomials.
- Eliminate variables from the vector space $\langle F \cup LG \rangle \leftrightarrow$ $LG = \{lg \text{ where } l \in L \text{ and } g \in G\}$.

Our contribution

- Trade-off: The ability to control the degree vs the ability to stay close to the elimination ideal $I \cap B[k+1, n]$.
- Minimize complexity $\leftrightarrow$ Only consider polynomials of degree $\leq 3 \leftrightarrow$ $F = \{f_1, \dots, f_c\}$, $G = \{g_1, \dots, g_q\}$, f_i's have degree 3 and the g_i's degrees 2.
- Objective: Find as many polynomials in the ideal $I(F, G)$ of degree ≤ 3 as we can $\leftrightarrow$ Try to produce degree 3 or less in only *key variables* when applied to block ciphers.

– Eliminating variables while keeping degree $\leq 3 \rightarrow$ introduce false solutions.

- $L = \{1, x_1, \dots, x_n\} \rightarrow \langle L \rangle \rightarrow$ vector space spanned by the Boolean polynomials.
- Eliminate variables from the vector space $\langle F \cup LG \rangle \leftrightarrow$ $LG = \{lg$ where $l \in L$ and $g \in G\}$.

The monomial orders

A. Monomials containing x_1 are largest: Split variable

Gauss eliminate monomials containing x_1 from the sets F and G producing $\langle F_{x_1}, G_{x_1} \rangle$ and $\langle F_{\overline{x_1}}, G_{\overline{x_1}} \rangle = \langle F, G \rangle \cap B[2, n]$.

B. Monomials of degree 3 are largest: Split deg 2/3

- $\langle F \cup LG \rangle$ may contain more quadratic polynomials than just G.
- Produce a larger set of quadratic polynomials $G^{(2)}$ by Gaussian elimination on degree 3 monomials in order to try to produce some polynomials of degree 2.

3-*normal forms*: Normalizing cubics with respect to quadratics

- Eliminate particular monomials containing x_1 from F using G as basis.
- A polynomial $f \in B$ is said to be in *normal form* f^{Norm} with respect to G, if no monomial in f is divisible by the leading term of any polynomial in $G \rightarrow$ Achieve f^{Norm} by successively subtracting multiples of the polynomials in G.
- The effect of this procedure is that there is a rather large set of monomials containing x_1 that can not appear in the cubic polynomials output at the end.

The monomial orders

A. Monomials containing x_1 are largest: Split variable

Gauss eliminate monomials containing x_1 from the sets F and G producing $\langle F_{x_1}, G_{x_1}\rangle$ and $\langle F_{\overline{x_1}}, G_{\overline{x_1}}\rangle = \langle F, G\rangle \cap B[2, n]$.

B. Monomials of degree 3 are largest: Split deg 2/3

- $\langle F \cup LG\rangle$ may contain more quadratic polynomials than just G.
- Produce a larger set of quadratic polynomials $G^{(2)}$ by Gaussian elimination on degree 3 monomials in order to try to produce some polynomials of degree 2.

3-*normal forms*: Normalizing cubics with respect to quadratics

- Eliminate particular monomials containing x_1 from F using G as basis.
- A polynomial $f \in B$ is said to be in *normal form* f^{Norm} with respect to G, if no monomial in f is divisible by the leading term of any polynomial in G → Achieve f^{Norm} by successively subtracting multiples of the polynomials in G.
- The effect of this procedure is that there is a rather large set of monomials containing x_1 that can not appear in the cubic polynomials output at the end.

The monomial orders

A. Monomials containing x_1 are largest: Split variable

Gauss eliminate monomials containing x_1 from the sets F and G producing $\langle F_{x_1}, G_{x_1}\rangle$ and $\langle F_{\overline{x_1}}, G_{\overline{x_1}}\rangle = \langle F, G\rangle \cap B[2, n]$.

B. Monomials of degree 3 are largest: Split deg $2/3$

- $\langle F \cup LG\rangle$ may contain more quadratic polynomials than just G.
- Produce a larger set of quadratic polynomials $G^{(2)}$ by Gaussian elimination on degree 3 monomials in order to try to produce some polynomials of degree 2.

3-*normal forms*: Normalizing cubics with respect to quadratics

- Eliminate particular monomials containing x_1 from F using G as basis.
- A polynomial $f \in B$ is said to be in *normal form* f^{Norm} with respect to G, if no monomial in f is divisible by the leading term of any polynomial in G → Achieve f^{Norm} by successively subtracting multiples of the polynomials in G.
- The effect of this procedure is that there is a rather large set of monomials containing x_1 that can not appear in the cubic polynomials output at the end.

The monomial orders

A. Monomials containing x_1 are largest: Split variable

Gauss eliminate monomials containing x_1 from the sets F and G producing $\langle F_{x_1}, G_{x_1} \rangle$ and $\langle F_{\overline{x_1}}, G_{\overline{x_1}} \rangle = \langle F, G \rangle \cap B[2, n]$.

B. Monomials of degree 3 are largest: Split deg $2/3$

- $\langle F \cup LG \rangle$ may contain more quadratic polynomials than just G.
- Produce a larger set of quadratic polynomials $G^{(2)}$ by Gaussian elimination on degree 3 monomials in order to try to produce some polynomials of degree 2.

3-*normal forms*: Normalizing cubics with respect to quadratics

- Eliminate particular monomials containing x_1 from F using G as basis.
- A polynomial $f \in B$ is said to be in *normal form* f^{Norm} with respect to G, if no monomial in f is divisible by the leading term of any polynomial in G $\rightarrow$ Achieve f^{Norm} by successively subtracting multiples of the polynomials in G.
- The effect of this procedure is that there is a rather large set of monomials containing x_1 that can not appear in the cubic polynomials output at the end.

The monomial orders

A. Monomials containing x_1 are largest: Split variable

Gauss eliminate monomials containing x_1 from the sets F and G producing $\langle F_{x_1}, G_{x_1}\rangle$ and $\langle F_{\overline{x_1}}, G_{\overline{x_1}}\rangle = \langle F, G\rangle \cap B[2, n]$.

B. Monomials of degree 3 are largest: Split deg $2/3$

- $\langle F \cup LG\rangle$ may contain more quadratic polynomials than just G.
- Produce a larger set of quadratic polynomials $G^{(2)}$ by Gaussian elimination on degree 3 monomials in order to try to produce some polynomials of degree 2.

3-*normal forms*: Normalizing cubics with respect to quadratics

- Eliminate particular monomials containing x_1 from F using G as basis.
- A polynomial $f \in B$ is said to be in *normal form* f^{Norm} with respect to G, if no monomial in f is divisible by the leading term of any polynomial in G $\rightarrow$ Achieve f^{Norm} by successively subtracting multiples of the polynomials in G.
- The effect of this procedure is that there is a rather large set of monomials containing x_1 that can not appear in the cubic polynomials output at the end.

The monomial orders

A. Monomials containing x_1 are largest: Split variable

Gauss eliminate monomials containing x_1 from the sets F and G producing $\langle F_{x_1}, G_{x_1}\rangle$ and $\langle F_{\overline{x_1}}, G_{\overline{x_1}}\rangle = \langle F, G\rangle \cap B[2, n]$.

B. Monomials of degree 3 are largest: Split deg $2/3$

- $\langle F \cup LG\rangle$ may contain more quadratic polynomials than just G.
- Produce a larger set of quadratic polynomials $G^{(2)}$ by Gaussian elimination on degree 3 monomials in order to try to produce some polynomials of degree 2.

3-*normal forms*: Normalizing cubics with respect to quadratics

- Eliminate particular monomials containing x_1 from F using G as basis.
- A polynomial $f \in B$ is said to be in *normal form* f^{Norm} with respect to G, if no monomial in f is divisible by the leading term of any polynomial in G → Achieve f^{Norm} by successively subtracting multiples of the polynomials in G.
- The effect of this procedure is that there is a rather large set of monomials containing x_1 that can not appear in the cubic polynomials output at the end.

The monomial orders

A. Monomials containing x_1 are largest: Split variable

Gauss eliminate monomials containing x_1 from the sets F and G producing $\langle F_{x_1}, G_{x_1}\rangle$ and $\langle F_{\overline{x_1}}, G_{\overline{x_1}}\rangle = \langle F, G\rangle \cap B[2, n]$.

B. Monomials of degree 3 are largest: Split deg $2/3$

- $\langle F \cup LG\rangle$ may contain more quadratic polynomials than just G.
- Produce a larger set of quadratic polynomials $G^{(2)}$ by Gaussian elimination on degree 3 monomials in order to try to produce some polynomials of degree 2.

3-*normal forms*: Normalizing cubics with respect to quadratics

- Eliminate particular monomials containing x_1 from F using G as basis.
- A polynomial $f \in B$ is said to be in *normal form* f^{Norm} with respect to G, if no monomial in f is divisible by the leading term of any polynomial in G → Achieve f^{Norm} by successively subtracting multiples of the polynomials in G.
- The effect of this procedure is that there is a rather large set of monomials containing x_1 that can not appear in the cubic polynomials output at the end.

The monomial orders

A. Monomials containing x_1 are largest: Split variable

Gauss eliminate monomials containing x_1 from the sets F and G producing $\langle F_{x_1}, G_{x_1}\rangle$ and $\langle F_{\overline{x_1}}, G_{\overline{x_1}}\rangle = \langle F, G\rangle \cap B[2, n]$.

B. Monomials of degree 3 are largest: Split deg $2/3$

- $\langle F \cup LG\rangle$ may contain more quadratic polynomials than just G.
- Produce a larger set of quadratic polynomials $G^{(2)}$ by Gaussian elimination on degree 3 monomials in order to try to produce some polynomials of degree 2.

3-*normal forms*: Normalizing cubics with respect to quadratics

- Eliminate particular monomials containing x_1 from F using G as basis.
- A polynomial $f \in B$ is said to be in *normal form* f^{Norm} with respect to G, if no monomial in f is divisible by the leading term of any polynomial in G → Achieve f^{Norm} by successively subtracting multiples of the polynomials in G.
- The effect of this procedure is that there is a rather large set of monomials containing x_1 that can not appear in the cubic polynomials output at the end.

The monomial orders

A. Monomials containing x_1 are largest: Split variable

Gauss eliminate monomials containing x_1 from the sets F and G producing $\langle F_{x_1}, G_{x_1}\rangle$ and $\langle F_{\overline{x_1}}, G_{\overline{x_1}}\rangle = \langle F, G\rangle \cap B[2, n]$.

B. Monomials of degree 3 are largest: Split deg $2/3$

- $\langle F \cup LG\rangle$ may contain more quadratic polynomials than just G.
- Produce a larger set of quadratic polynomials $G^{(2)}$ by Gaussian elimination on degree 3 monomials in order to try to produce some polynomials of degree 2.

3-*normal forms*: Normalizing cubics with respect to quadratics

- Eliminate particular monomials containing x_1 from F using G as basis.
- A polynomial $f \in B$ is said to be in *normal form* f^{Norm} with respect to G, if no monomial in f is divisible by the leading term of any polynomial in G $\rightarrow$ Achieve f^{Norm} by successively subtracting multiples of the polynomials in G.
- The effect of this procedure is that there is a rather large set of monomials containing x_1 that can not appear in the cubic polynomials output at the end.

What is the alternative to Gröbner bases?

- Resultants: Eliminate one variable from all monomials containing the targeted variable at the time.
- Let $f = a_0x_1 + a_1$ and $g = b_0x_1 + b_1$ be two polynomials in B, where the a_j and b_j are in $B[2, n]$. If f and g are quadratic, then a_0 and b_0 will be linear, a_1 and b_1 will (in general) be quadratic.
- The 2×2 Sylvester matrix of f and g with respect to x_1

$$\mathrm{Syl}(f, g, x_1) = \begin{pmatrix} a_0 & b_0 \\ a_1 & b_1 \end{pmatrix}$$

- The resultant of f and g with respect to x_1 is a polynomial in $B[2, n]$: $\mathrm{Res}(f, g, x_1) = \det(\mathrm{Syl}(f, g, x_1)) = a_0b_1 + a_1b_0 = b_0f + a_0g$. Also $\mathrm{Res}(f, g, x_1) \subset I' = (f, g) \cap B[2, n]$.

Good news

2×2 determinants are easy to compute, and cubic polynomials can be handled by a computer. Also the size of n we encounter in cryptanalysis of block ciphers are within tolerances.

What is the alternative to Gröbner bases?

- Resultants: Eliminate one variable from all monomials containing the targeted variable at the time.
- Let $f = a_0x_1 + a_1$ and $g = b_0x_1 + b_1$ be two polynomials in B, where the a_j and b_j are in $B[2, n]$. If f and g are quadratic, then a_0 and b_0 will be linear, a_1 and b_1 will (in general) be quadratic.
- The 2×2 Sylvester matrix of f and g with respect to x_1

$$\mathrm{Syl}(f, g, x_1) = \begin{pmatrix} a_0 & b_0 \\ a_1 & b_1 \end{pmatrix}$$

- The resultant of f and g with respect to x_1 is a polynomial in $B[2, n]$: $\mathrm{Res}(f, g, x_1) = \det(\mathrm{Syl}(f, g, x_1)) = a_0b_1 + a_1b_0 = b_0f + a_0g$. Also $\mathrm{Res}(f, g, x_1) \subset I' = (f, g) \cap B[2, n]$.

Good news

2×2 determinants are easy to compute, and cubic polynomials can be handled by a computer. Also the size of n we encounter in cryptanalysis of block ciphers are within tolerances.

What is the alternative to Gröbner bases?

- Resultants: Eliminate one variable from all monomials containing the targeted variable at the time.
- Let $f = a_0x_1 + a_1$ and $g = b_0x_1 + b_1$ be two polynomials in B, where the a_j and b_j are in $B[2, n]$. If f and g are quadratic, then a_0 and b_0 will be linear, a_1 and b_1 will (in general) be quadratic.
- The 2×2 Sylvester matrix of f and g with respect to x_1

$$\mathrm{Syl}(f, g, x_1) = \begin{pmatrix} a_0 & b_0 \\ a_1 & b_1 \end{pmatrix}$$

- The resultant of f and g with respect to x_1 is a polynomial in $B[2, n]$: $\mathrm{Res}(f, g, x_1) = \det(\mathrm{Syl}(f, g, x_1)) = a_0b_1 + a_1b_0 = b_0f + a_0g$. Also $\mathrm{Res}(f, g, x_1) \subset I' = (f, g) \cap B[2, n]$.

Good news

2×2 determinants are easy to compute, and cubic polynomials can be handled by a computer. Also the size of n we encounter in cryptanalysis of block ciphers are within tolerances.

What is the alternative to Gröbner bases?

- Resultants: Eliminate one variable from all monomials containing the targeted variable at the time.
- Let $f = a_0x_1 + a_1$ and $g = b_0x_1 + b_1$ be two polynomials in B, where the a_j and b_j are in $B[2, n]$. If f and g are quadratic, then a_0 and b_0 will be linear, a_1 and b_1 will (in general) be quadratic.
- The 2×2 Sylvester matrix of f and g with respect to x_1

$$\mathrm{Syl}(f, g, x_1) = \begin{pmatrix} a_0 & b_0 \\ a_1 & b_1 \end{pmatrix}$$

- The resultant of f and g with respect to x_1 is a polynomial in $B[2, n]$: $\mathrm{Res}(f, g, x_1) = \det(\mathrm{Syl}(f, g, x_1)) = a_0b_1 + a_1b_0 = b_0f + a_0g$. Also $\mathrm{Res}(f, g, x_1) \subset I' = (f, g) \cap B[2, n]$.

Good news

2×2 determinants are easy to compute, and cubic polynomials can be handled by a computer. Also the size of n we encounter in cryptanalysis of block ciphers are within tolerances.

What is the alternative to Gröbner bases?

- Resultants: Eliminate one variable from all monomials containing the targeted variable at the time.
- Let $f = a_0x_1 + a_1$ and $g = b_0x_1 + b_1$ be two polynomials in B, where the a_j and b_j are in $B[2, n]$. If f and g are quadratic, then a_0 and b_0 will be linear, a_1 and b_1 will (in general) be quadratic.
- The 2×2 Sylvester matrix of f and g with respect to x_1

$$\mathrm{Syl}(f, g, x_1) = \begin{pmatrix} a_0 & b_0 \\ a_1 & b_1 \end{pmatrix}$$

- The resultant of f and g with respect to x_1 is a polynomial in $B[2, n]$: $\mathrm{Res}(f, g, x_1) = \det(\mathrm{Syl}(f, g, x_1)) = a_0b_1 + a_1b_0 = b_0f + a_0g$. Also $\mathrm{Res}(f, g, x_1) \subset I' = (f, g) \cap B[2, n]$.

Good news

2×2 determinants are easy to compute, and cubic polynomials can be handled by a computer. Also the size of n we encounter in cryptanalysis of block ciphers are within tolerances.

Coefficient constraints and Resultant ideals

For $I(F) = (f_1, \ldots, f_s)$ where each f_i written as $f_i = a_i x_1 + b_i$:

- $\text{Res}_2(F) = (\text{Res}(f_i, f_j; x_1) | 1 \leq i < j \leq s)$.
- $\text{Co}_2(F) = (b_1(a_1 + 1), b_2(a_2 + 1), \ldots, b_s(a_s + 1))$.

Theorem

Let $F = \{f_1, \ldots, f_s\}$ be a set of Boolean polynomials in $B[1, n]$. Then

$$I(F) \cap B[2, n] = \text{Res}_2(F) + Co_2(F).$$

Note: IF f_i have degree $d \leftrightarrow \deg(\text{Res}_2(F) + Co_2(F)) = 2d - 1$.

Coefficient constraints and Resultant ideals

For $I(F) = (f_1, \ldots, f_s)$ where each f_i written as $f_i = a_i x_1 + b_i$:

- $\mathrm{Res}_2(F) = (\mathrm{Res}(f_i, f_j; x_1) | 1 \leq i < j \leq s)$.
- $\mathsf{Co}_2(F) = (b_1(a_1 + 1), b_2(a_2 + 1), \ldots, b_s(a_s + 1))$.

Theorem

Let $F = \{f_1, \ldots, f_s\}$ be a set of Boolean polynomials in $B[1, n]$. Then

$$I(F) \cap B[2, n] = \mathrm{Res}_2(F) + Co_2(F).$$

Note: IF f_i have degree $d \leftrightarrow \deg(\mathrm{Res}_2(F) + Co_2(F)) = 2d - 1$.

Coefficient constraints and Resultant ideals

For $I(F) = (f_1, \ldots, f_s)$ where each f_i written as $f_i = a_i x_1 + b_i$:

- $\mathrm{Res}_2(F) = (\mathrm{Res}(f_i, f_j; x_1) | 1 \leq i < j \leq s)$.
- $\mathsf{Co}_2(F) = (b_1(a_1 + 1), b_2(a_2 + 1), \ldots, b_s(a_s + 1))$.

Theorem

Let $F = \{f_1, \ldots, f_s\}$ be a set of Boolean polynomials in $B[1, n]$. Then

$$I(F) \cap B[2, n] = \mathrm{Res}_2(F) + Co_2(F).$$

Note: IF f_i have degree $d \leftrightarrow \deg(\mathrm{Res}_2(F) + Co_2(F)) = 2d - 1$.

The LG-$elim$ algorithm

- Replace F with F$\cup L \cdot G$.
- Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
- Split F^2 and F^3 into $F^2_{x_1}, F^3_{x_1}, F^2_{\overline{x_1}} F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
- Return $F^2_{\overline{x_1}} F^3_{\overline{x_1}}$.
- Repeat for F_j and G_j in smaller and smaller Boolean rings $B[j, n]$.

The LG-$elim$ algorithm

- Replace F with F$\cup L \cdot G$.
- Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
- Split F^2 and F^3 into $F^2_{x_1}, F^3_{x_1}, F^2_{\overline{x_1}} F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
- Return $F^2_{\overline{x_1}} F^3_{\overline{x_1}}$.
- Repeat for F_j and G_j in smaller and smaller Boolean rings $B[j, n]$.

The LG-$elim$ algorithm

- Replace F with F$\cup L \cdot G$.
- Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
- Split F^2 and F^3 into $F^2_{x_1}, F^3_{x_1}, F^2_{\overline{x_1}} F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
- Return $F^2_{\overline{x_1}} F^3_{\overline{x_1}}$.
- Repeat for F_j and G_j in smaller and smaller Boolean rings $B[j, n]$.

The LG-$elim$ algorithm

- Replace F with F$\cup L \cdot G$.
- Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
- Split F^2 and F^3 into $F^2_{x_1}, F^3_{x_1}, F^2_{\overline{x_1}} F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
- Return $F^2_{\overline{x_1}} F^3_{\overline{x_1}}$.
- Repeat for F_j and G_j in smaller and smaller Boolean rings $B[j, n]$.

The LG-$elim$ algorithm

- Replace F with F$\cup L \cdot G$.
- Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
- Split F^2 and F^3 into $F^2_{x_1}, F^3_{x_1}, F^2_{\overline{x_1}} F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
- Return $F^2_{\overline{x_1}} F^3_{\overline{x_1}}$.
- Repeat for F_j and G_j in smaller and smaller Boolean rings $B[j,n]$.

Main elimination algorithm: Eliminate

- Split G into $G_{x_1}, G_{\overline{x_1}} \subset B[2,n]$ by Gaussian elimination on monomials containing x_1
- If G_{x_1} or $G_{\overline{x_1}}$ changed in last iteration, then
  - Replace F with $(x_1+1)G_{x_1} \cup x_1 G_{\overline{x_1}} \cup F$ producing more cubic polynomials.
  - Normalize F with respect to G_{x_1} to eliminate particular monomials containing x_1.
  - Produce more degree 3 relations from resultants and coefficient constraints w.r.t x_1 of G_{x_1} and add to F.
  - Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
  - Split F^2 into $F^2_{x_1}, F^2_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
  - $G_{x_1} \leftarrow G_{x_1} \cup F^2_{x_1}$, G_{x_1} changes if $F^2_{x_1} \neq \emptyset$, causing new iteration
  - $G_{\overline{x_1}} \leftarrow G_{\overline{x_1}} \cup F^2_{\overline{x_1}}$, $G_{\overline{x_1}}$ changes if $F^2_{\overline{x_1}} \neq \emptyset$, causing new iteration
- Split F^3 into $F^3_{x_1}, F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1 and Return $F^3_{\overline{x_1}}, G_{\overline{x_1}}$

Main elimination algorithm: Eliminate

- Split G into $G_{x_1}, G_{\overline{x_1}} \subset B[2, n]$ by Gaussian elimination on monomials containing x_1
- If G_{x_1} or $G_{\overline{x_1}}$ changed in last iteration, then
  - Replace F with $(x_1 + 1)G_{x_1} \cup x_1 G_{\overline{x_1}} \cup F$ producing more cubic polynomials.
  - Normalize F with respect to G_{x_1} to eliminate particular monomials containing x_1.
  - Produce more degree 3 relations from resultants and coefficient constraints w.r.t x_1 of G_{x_1} and add to F.
  - Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
  - Split F^2 into $F^2_{x_1}, F^2_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
  - $G_{x_1} \leftarrow G_{x_1} \cup F^2_{x_1}$, G_{x_1} changes if $F^2_{x_1} \neq \emptyset$, causing new iteration
  - $G_{\overline{x_1}} \leftarrow G_{\overline{x_1}} \cup F^2_{\overline{x_1}}$, $G_{\overline{x_1}}$ changes if $F^2_{\overline{x_1}} \neq \emptyset$, causing new iteration
- Split F^3 into $F^3_{x_1}, F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1 and Return $F^3_{\overline{x_1}}, G_{\overline{x_1}}$

Main elimination algorithm: Eliminate

- Split G into $G_{x_1}, G_{\overline{x_1}} \subset B[2,n]$ by Gaussian elimination on monomials containing x_1
- If G_{x_1} or $G_{\overline{x_1}}$ changed in last iteration, then
  - Replace F with $(x_1+1)G_{x_1} \cup x_1 G_{\overline{x_1}} \cup F$ producing more cubic polynomials.
  - Normalize F with respect to G_{x_1} to eliminate particular monomials containing x_1.
  - Produce more degree 3 relations from resultants and coefficient constraints w.r.t x_1 of G_{x_1} and add to F.
  - Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
  - Split F^2 into $F^2_{x_1}, F^2_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
  - $G_{x_1} \leftarrow G_{x_1} \cup F^2_{x_1}$, G_{x_1} changes if $F^2_{x_1} \neq \emptyset$, causing new iteration
  - $G_{\overline{x_1}} \leftarrow G_{\overline{x_1}} \cup F^2_{\overline{x_1}}$, $G_{\overline{x_1}}$ changes if $F^2_{\overline{x_1}} \neq \emptyset$, causing new iteration
- Split F^3 into $F^3_{x_1}, F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1 and Return $F^3_{\overline{x_1}}, G_{\overline{x_1}}$

Main elimination algorithm: Eliminate

- Split G into $G_{x_1}, G_{\overline{x_1}} \subset B[2,n]$ by Gaussian elimination on monomials containing x_1
- If G_{x_1} or $G_{\overline{x_1}}$ changed in last iteration, then
  - Replace F with $(x_1+1)G_{x_1} \cup x_1 G_{\overline{x_1}} \cup F$ producing more cubic polynomials.
  - Normalize F with respect to G_{x_1} to eliminate particular monomials containing x_1.
  - Produce more degree 3 relations from resultants and coefficient constraints w.r.t x_1 of G_{x_1} and add to F.
  - Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
  - Split F^2 into $F^2_{x_1}, F^2_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
  - $G_{x_1} \leftarrow G_{x_1} \cup F^2_{x_1}$, G_{x_1} changes if $F^2_{x_1} \neq \emptyset$, causing new iteration
  - $G_{\overline{x_1}} \leftarrow G_{\overline{x_1}} \cup F^2_{\overline{x_1}}$, $G_{\overline{x_1}}$ changes if $F^2_{\overline{x_1}} \neq \emptyset$, causing new iteration
- Split F^3 into $F^3_{x_1}, F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1 and Return $F^3_{\overline{x_1}}, G_{\overline{x_1}}$

Main elimination algorithm: Eliminate

- Split G into $G_{x_1}, G_{\overline{x_1}} \subset B[2,n]$ by Gaussian elimination on monomials containing x_1
- If G_{x_1} or $G_{\overline{x_1}}$ changed in last iteration, then
  - Replace F with $(x_1+1)G_{x_1} \cup x_1 G_{\overline{x_1}} \cup F$ producing more cubic polynomials.
  - Normalize F with respect to G_{x_1} to eliminate particular monomials containing x_1.
  - Produce more degree 3 relations from resultants and coefficient constraints w.r.t x_1 of G_{x_1} and add to F.
  - Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
  - Split F^2 into $F^2_{x_1}, F^2_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
  - $G_{x_1} \leftarrow G_{x_1} \cup F^2_{x_1}$, G_{x_1} changes if $F^2_{x_1} \neq \emptyset$, causing new iteration
  - $G_{\overline{x_1}} \leftarrow G_{\overline{x_1}} \cup F^2_{\overline{x_1}}$, $G_{\overline{x_1}}$ changes if $F^2_{\overline{x_1}} \neq \emptyset$, causing new iteration
- Split F^3 into $F^3_{x_1}, F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1 and Return $F^3_{\overline{x_1}}, G_{\overline{x_1}}$

Main elimination algorithm: Eliminate

- Split G into $G_{x_1}, G_{\overline{x_1}} \subset B[2,n]$ by Gaussian elimination on monomials containing x_1
- If G_{x_1} or $G_{\overline{x_1}}$ changed in last iteration, then
  - Replace F with $(x_1+1)G_{x_1} \cup x_1 G_{\overline{x_1}} \cup F$ producing more cubic polynomials.
  - Normalize F with respect to G_{x_1} to eliminate particular monomials containing x_1.
  - Produce more degree 3 relations from resultants and coefficient constraints w.r.t x_1 of G_{x_1} and add to F.
  - Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
  - Split F^2 into $F^2_{x_1}, F^2_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
  - $G_{x_1} \leftarrow G_{x_1} \cup F^2_{x_1}$, G_{x_1} changes if $F^2_{x_1} \neq \emptyset$, causing new iteration
  - $G_{\overline{x_1}} \leftarrow G_{\overline{x_1}} \cup F^2_{\overline{x_1}}$, $G_{\overline{x_1}}$ changes if $F^2_{\overline{x_1}} \neq \emptyset$, causing new iteration
- Split F^3 into $F^3_{x_1}, F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1 and Return $F^3_{\overline{x_1}}, G_{\overline{x_1}}$

Main elimination algorithm: Eliminate

- Split G into $G_{x_1}, G_{\overline{x_1}} \subset B[2,n]$ by Gaussian elimination on monomials containing x_1
- If G_{x_1} or $G_{\overline{x_1}}$ changed in last iteration, then
  - Replace F with $(x_1+1)G_{x_1} \cup x_1 G_{\overline{x_1}} \cup F$ producing more cubic polynomials.
  - Normalize F with respect to G_{x_1} to eliminate particular monomials containing x_1.
  - Produce more degree 3 relations from resultants and coefficient constraints w.r.t x_1 of G_{x_1} and add to F.
  - Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
  - Split F^2 into $F^2_{x_1}, F^2_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
  - $G_{x_1} \leftarrow G_{x_1} \cup F^2_{x_1}$, G_{x_1} changes if $F^2_{x_1} \neq \emptyset$, causing new iteration
  - $G_{\overline{x_1}} \leftarrow G_{\overline{x_1}} \cup F^2_{\overline{x_1}}$, $G_{\overline{x_1}}$ changes if $F^2_{\overline{x_1}} \neq \emptyset$, causing new iteration
- Split F^3 into $F^3_{x_1}, F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1 and Return $F^3_{\overline{x_1}}, G_{\overline{x_1}}$

Main elimination algorithm: Eliminate

- Split G into $G_{x_1}, G_{\overline{x_1}} \subset B[2,n]$ by Gaussian elimination on monomials containing x_1
- If G_{x_1} or $G_{\overline{x_1}}$ changed in last iteration, then
  - Replace F with $(x_1+1)G_{x_1} \cup x_1 G_{\overline{x_1}} \cup F$ producing more cubic polynomials.
  - Normalize F with respect to G_{x_1} to eliminate particular monomials containing x_1.
  - Produce more degree 3 relations from resultants and coefficient constraints w.r.t x_1 of G_{x_1} and add to F.
  - Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
  - Split F^2 into $F^2_{x_1}, F^2_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
  - $G_{x_1} \leftarrow G_{x_1} \cup F^2_{x_1}$, G_{x_1} changes if $F^2_{x_1} \neq \emptyset$, causing new iteration
  - $G_{\overline{x_1}} \leftarrow G_{\overline{x_1}} \cup F^2_{\overline{x_1}}$, $G_{\overline{x_1}}$ changes if $F^2_{\overline{x_1}} \neq \emptyset$, causing new iteration
- Split F^3 into $F^3_{x_1}, F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1 and Return $F^3_{\overline{x_1}}, G_{\overline{x_1}}$

Main elimination algorithm: Eliminate

- Split G into $G_{x_1}, G_{\overline{x_1}} \subset B[2,n]$ by Gaussian elimination on monomials containing x_1
- If G_{x_1} or $G_{\overline{x_1}}$ changed in last iteration, then
  - Replace F with $(x_1+1)G_{x_1} \cup x_1 G_{\overline{x_1}} \cup F$ producing more cubic polynomials.
  - Normalize F with respect to G_{x_1} to eliminate particular monomials containing x_1.
  - Produce more degree 3 relations from resultants and coefficient constraints w.r.t x_1 of G_{x_1} and add to F.
  - Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
  - Split F^2 into $F^2_{x_1}, F^2_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
  - $G_{x_1} \leftarrow G_{x_1} \cup F^2_{x_1}$, G_{x_1} changes if $F^2_{x_1} \neq \emptyset$, causing new iteration
  - $G_{\overline{x_1}} \leftarrow G_{\overline{x_1}} \cup F^2_{\overline{x_1}}$, $G_{\overline{x_1}}$ changes if $F^2_{\overline{x_1}} \neq \emptyset$, causing new iteration
- Split F^3 into $F^3_{x_1}, F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1 and Return $F^3_{\overline{x_1}}, G_{\overline{x_1}}$

Main elimination algorithm: Eliminate

- Split G into $G_{x_1}, G_{\overline{x_1}} \subset B[2,n]$ by Gaussian elimination on monomials containing x_1
- If G_{x_1} or $G_{\overline{x_1}}$ changed in last iteration, then
  - Replace F with $(x_1+1)G_{x_1} \cup x_1 G_{\overline{x_1}} \cup F$ producing more cubic polynomials.
  - Normalize F with respect to G_{x_1} to eliminate particular monomials containing x_1.
  - Produce more degree 3 relations from resultants and coefficient constraints w.r.t x_1 of G_{x_1} and add to F.
  - Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
  - Split F^2 into $F^2_{x_1}, F^2_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
  - $G_{x_1} \leftarrow G_{x_1} \cup F^2_{x_1}$, G_{x_1} changes if $F^2_{x_1} \neq \emptyset$, causing new iteration
  - $G_{\overline{x_1}} \leftarrow G_{\overline{x_1}} \cup F^2_{\overline{x_1}}$, $G_{\overline{x_1}}$ changes if $F^2_{\overline{x_1}} \neq \emptyset$, causing new iteration
- Split F^3 into $F^3_{x_1}, F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1 and Return $F^3_{\overline{x_1}}, G_{\overline{x_1}}$

Main elimination algorithm: Eliminate

- Split G into $G_{x_1}, G_{\overline{x_1}} \subset B[2,n]$ by Gaussian elimination on monomials containing x_1
- If G_{x_1} or $G_{\overline{x_1}}$ changed in last iteration, then
  - Replace F with $(x_1+1)G_{x_1} \cup x_1 G_{\overline{x_1}} \cup F$ producing more cubic polynomials.
  - Normalize F with respect to G_{x_1} to eliminate particular monomials containing x_1.
  - Produce more degree 3 relations from resultants and coefficient constraints w.r.t x_1 of G_{x_1} and add to F.
  - Gauss eliminate w.r.t degree to produce F^2, F^3 from F.
  - Split F^2 into $F^2_{x_1}, F^2_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1.
  - $G_{x_1} \leftarrow G_{x_1} \cup F^2_{x_1}$, G_{x_1} changes if $F^2_{x_1} \neq \emptyset$, causing new iteration
  - $G_{\overline{x_1}} \leftarrow G_{\overline{x_1}} \cup F^2_{\overline{x_1}}$, $G_{\overline{x_1}}$ changes if $F^2_{\overline{x_1}} \neq \emptyset$, causing new iteration
- Split F^3 into $F^3_{x_1}, F^3_{\overline{x_1}}$ by Gaussian elimination on monomials containing x_1 and Return $F^3_{\overline{x_1}}, G_{\overline{x_1}}$

Remarks and Complexity

- In general we have $\langle F \cup LG \rangle \cap B[2,n] \subseteq \langle F^3_{x_1} \cup L_2 G_{\overline{x_1}} \rangle$ *even* if we look for more quadratic polynomials in the LG-algorithm.
- $\binom{n-1}{\leq 3}$ and $\binom{n-1}{\leq 2}$ is the tight upper bound on the number of monomials and polynomials which can occur in F and G, respectively.
- Space complexity of the algorithm is storing $\mathcal{O}(n^6)$ monomials.
- The time complexity is dominated by the linear algebra done in SplitDeg2/3 and SplitVariable. In the worst case, we have input size $\mathcal{O}(n^3)$ in both polynomials and monomials, so the matrices constructed are of size $\mathcal{O}(n^3) \times \mathcal{O}(n^3)$. This leads to $\mathcal{O}(n^9)$ for the Gaussian reduction.

Remarks and Complexity

- In general we have $\langle F \cup LG \rangle \cap B[2,n] \subseteq \langle F^3_{x_1} \cup L_2 G_{\overline{x_1}} \rangle$ *even* if we look for more quadratic polynomials in the LG-algorithm.
- $\binom{n-1}{\leq 3}$ and $\binom{n-1}{\leq 2}$ is the tight upper bound on the number of monomials and polynomials which can occur in F and G, respectively.
- Space complexity of the algorithm is storing $\mathcal{O}(n^6)$ monomials.
- The time complexity is dominated by the linear algebra done in SplitDeg2/3 and SplitVariable. In the worst case, we have input size $\mathcal{O}(n^3)$ in both polynomials and monomials, so the matrices constructed are of size $\mathcal{O}(n^3) \times \mathcal{O}(n^3)$. This leads to $\mathcal{O}(n^9)$ for the Gaussian reduction.

Remarks and Complexity

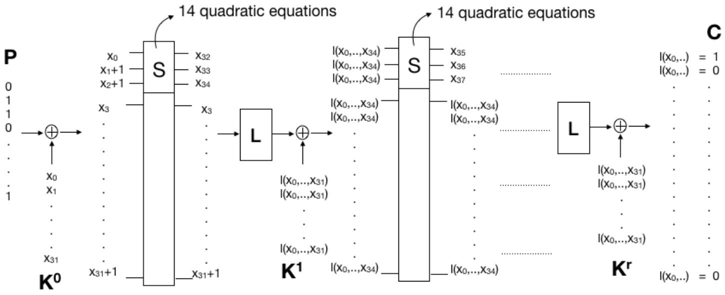
- In general we have $\langle F \cup LG \rangle \cap B[2,n] \subseteq \langle F^3_{\overline{x_1}} \cup L_2 G_{\overline{x_1}} \rangle$ *even* if we look for more quadratic polynomials in the LG-algorithm.
- $\binom{n-1}{\leq 3}$ and $\binom{n-1}{\leq 2}$ is the tight upper bound on the number of monomials and polynomials which can occur in F and G, respectively.
- Space complexity of the algorithm is storing $\mathcal{O}(n^6)$ monomials.
- The time complexity is dominated by the linear algebra done in SplitDeg2/3 and SplitVariable. In the worst case, we have input size $\mathcal{O}(n^3)$ in both polynomials and monomials, so the matrices constructed are of size $\mathcal{O}(n^3) \times \mathcal{O}(n^3)$. This leads to $\mathcal{O}(n^9)$ for the Gaussian reduction.

Remarks and Complexity

- In general we have $\langle F \cup LG \rangle \cap B[2,n] \subseteq \langle F^3_{\overline{x_1}} \cup L_2 G_{\overline{x_1}} \rangle$ *even* if we look for more quadratic polynomials in the LG-algorithm.
- $\binom{n-1}{\leq 3}$ and $\binom{n-1}{\leq 2}$ is the tight upper bound on the number of monomials and polynomials which can occur in F and G, respectively.
- Space complexity of the algorithm is storing $\mathcal{O}(n^6)$ monomials.
- The time complexity is dominated by the linear algebra done in SplitDeg2/3 and SplitVariable. In the worst case, we have input size $\mathcal{O}(n^3)$ in both polynomials and monomials, so the matrices constructed are of size $\mathcal{O}(n^3) \times \mathcal{O}(n^3)$. This leads to $\mathcal{O}(n^9)$ for the Gaussian reduction.

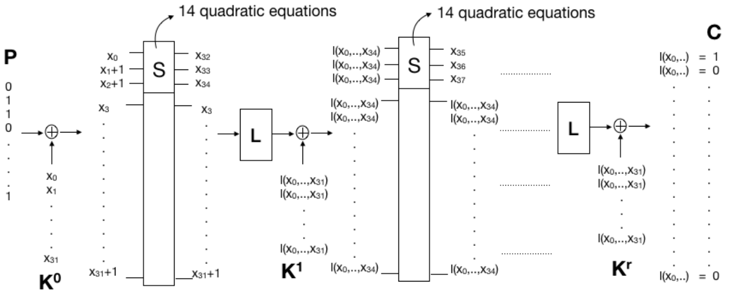
Remarks and Complexity

- In general we have $\langle F \cup LG \rangle \cap B[2,n] \subseteq \langle F^3_{\overline{x_1}} \cup L_2 G_{\overline{x_1}} \rangle$ *even* if we look for more quadratic polynomials in the LG-algorithm.
- $\binom{n-1}{\leq 3}$ and $\binom{n-1}{\leq 2}$ is the tight upper bound on the number of monomials and polynomials which can occur in F and G, respectively.
- Space complexity of the algorithm is storing $\mathcal{O}(n^6)$ monomials.
- The time complexity is dominated by the linear algebra done in SplitDeg2/3 and SplitVariable. In the worst case, we have input size $\mathcal{O}(n^3)$ in both polynomials and monomials, so the matrices constructed are of size $\mathcal{O}(n^3) \times \mathcal{O}(n^3)$. This leads to $\mathcal{O}(n^9)$ for the Gaussian reduction.



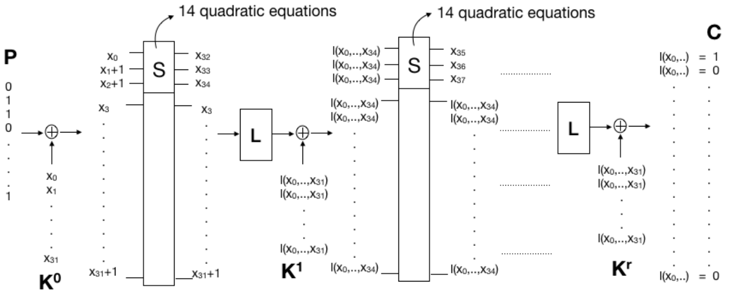
The (Reduced) LowMC cipher

- Uses a 3×3 S-box $\rightarrow$ 14 quadratic polynomials describe S-box $\rightarrow$ S-boxes do not cover the whole state $\rightarrow$ part of the cipher block is not affected by the S-box layer.
- Cipher parameters used: Block size: 24 bits, Key size: 32 bits, 1 S-box per round, $12/13$ rounds.



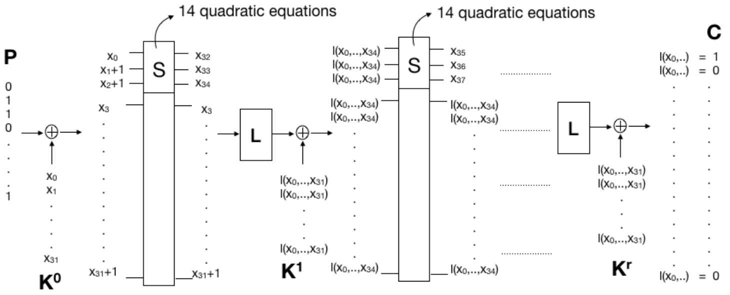
The (Reduced) LowMC cipher

- Uses a 3×3 S-box $\rightarrow$ 14 quadratic polynomials describe S-box $\rightarrow$ S-boxes do not cover the whole state $\rightarrow$ part of the cipher block is not affected by the S-box layer.
- Cipher parameters used: Block size: 24 bits, Key size: 32 bits, 1 S-box per round, $12/13$ rounds.



The (Reduced) LowMC cipher

- Uses a 3×3 S-box $\rightarrow$ 14 quadratic polynomials describe S-box $\rightarrow$ S-boxes do not cover the whole state $\rightarrow$ part of the cipher block is not affected by the S-box layer.
- Cipher parameters used: Block size: 24 bits, Key size: 32 bits, 1 S-box per round, $12/13$ rounds.



The (Reduced) LowMC cipher

- Uses a 3×3 S-box $\rightarrow$ 14 quadratic polynomials describe S-box $\rightarrow$ S-boxes do not cover the whole state $\rightarrow$ part of the cipher block is not affected by the S-box layer.
- Cipher parameters used: Block size: 24 bits, Key size: 32 bits, 1 S-box per round, $12/13$ rounds.

Experimental results

- Eliminate all variables x_i for $i \geq 32 \rightarrow$ Find some polynomials of degree at most 3, only in $x_0, \ldots, x_{31}$.
- **12 rounds:** 44 variables, $F = \emptyset$, $|G| = 168$.
  - $LG - elim$: Produces 1-2 cubic polynomial(s) only in key variables. Memory requirement: Store 7560 polynomials from $G \cdot L$.
  - $eliminate$: Produce same polynomials as $LG - elim$. Size of F never above 2000 polynomials $\leftrightarrow$ $eliminate$ has less space complexity than $LG - elim$. Running time: Roughly the same.
- 15 different systems using different p/c-pairs $\rightarrow$ 20 cubic polynomials in only key bits $\rightarrow$ Seems that we can produce many independent polynomials from different p/c-pairs.

Other results

- Checking for linear dependencies among 20 cubic polynomials we produced five *linear* polynomials in only key bits $\leftrightarrow$ Need much fewer polynomials than expected to find the values of $x_0, \ldots, x_{31}$.
- **13 rounds:** 47 variables, $F = \emptyset$, $|G| = 182$. For the 13-round systems we tried, neither $LG - elim$ or $eliminate$ found any cubic polynomials in only key variables $\rightarrow$ Only up to 12 rounds may be attacked using techniques.

Experimental results

- Eliminate all variables x_i for $i \geq 32 \rightarrow$ Find some polynomials of degree at most 3, only in $x_0, \ldots, x_{31}$.
- **12 rounds:** 44 variables, $F = \emptyset$, $|G| = 168$.
  - $LG - elim$: Produces 1-2 cubic polynomial(s) only in key variables. Memory requirement: Store 7560 polynomials from $G \cdot L$.
  - $eliminate$: Produce same polynomials as $LG - elim$. Size of F never above 2000 polynomials $\leftrightarrow$ $eliminate$ has less space complexity than $LG - elim$. Running time: Roughly the same.
- 15 different systems using different p/c-pairs $\rightarrow$ 20 cubic polynomials in only key bits $\rightarrow$ Seems that we can produce many independent polynomials from different p/c-pairs.

Other results

- Checking for linear dependencies among 20 cubic polynomials we produced five *linear* polynomials in only key bits $\leftrightarrow$ Need much fewer polynomials than expected to find the values of $x_0, \ldots, x_{31}$.
- **13 rounds:** 47 variables, $F = \emptyset$, $|G| = 182$. For the 13-round systems we tried, neither $LG - elim$ or $eliminate$ found any cubic polynomials in only key variables $\rightarrow$ Only up to 12 rounds may be attacked using techniques.

Experimental results

- Eliminate all variables x_i for $i \geq 32 \rightarrow$ Find some polynomials of degree at most 3, only in $x_0, \ldots, x_{31}$.
- **12 rounds:** 44 variables, $F = \emptyset$, $|G| = 168$.
  - $LG - elim$: Produces 1-2 cubic polynomial(s) only in key variables. Memory requirement: Store 7560 polynomials from $G \cdot L$.
  - $eliminate$: Produce same polynomials as $LG - elim$. Size of F never above 2000 polynomials $\leftrightarrow$ $eliminate$ has less space complexity than $LG - elim$. Running time: Roughly the same.
- 15 different systems using different p/c-pairs $\rightarrow$ 20 cubic polynomials in only key bits $\rightarrow$ Seems that we can produce many independent polynomials from different p/c-pairs.

Other results

- Checking for linear dependencies among 20 cubic polynomials we produced five *linear* polynomials in only key bits $\leftrightarrow$ Need much fewer polynomials than expected to find the values of $x_0, \ldots, x_{31}$.
- **13 rounds:** 47 variables, $F = \emptyset$, $|G| = 182$. For the 13-round systems we tried, neither $LG - elim$ or $eliminate$ found any cubic polynomials in only key variables $\rightarrow$ Only up to 12 rounds may be attacked using techniques.

Experimental results

- Eliminate all variables x_i for $i \geq 32 \rightarrow$ Find some polynomials of degree at most 3, only in $x_0, \ldots, x_{31}$.
- **12 rounds:** 44 variables, $F = \emptyset$, $|G| = 168$.
  - $LG - elim$: Produces 1-2 cubic polynomial(s) only in key variables. Memory requirement: Store 7560 polynomials from $G \cdot L$.
  - $eliminate$: Produce same polynomials as $LG - elim$. Size of F never above 2000 polynomials $\leftrightarrow$ $eliminate$ has less space complexity than $LG - elim$. Running time: Roughly the same.
- 15 different systems using different p/c-pairs $\rightarrow$ 20 cubic polynomials in only key bits $\rightarrow$ Seems that we can produce many independent polynomials from different p/c-pairs.

Other results

- Checking for linear dependencies among 20 cubic polynomials we produced five *linear* polynomials in only key bits $\leftrightarrow$ Need much fewer polynomials than expected to find the values of $x_0, \ldots, x_{31}$.
- **13 rounds:** 47 variables, $F = \emptyset$, $|G| = 182$. For the 13-round systems we tried, neither $LG - elim$ or $eliminate$ found any cubic polynomials in only key variables $\rightarrow$ Only up to 12 rounds may be attacked using techniques.

Experimental results

- Eliminate all variables x_i for $i \geq 32 \rightarrow$ Find some polynomials of degree at most 3, only in $x_0, \ldots, x_{31}$.
- **12 rounds:** 44 variables, $F = \emptyset$, $|G| = 168$.
  - $LG - elim$: Produces 1-2 cubic polynomial(s) only in key variables. Memory requirement: Store 7560 polynomials from $G \cdot L$.
  - $eliminate$: Produce same polynomials as $LG - elim$. Size of F never above 2000 polynomials $\leftrightarrow$ $eliminate$ has less space complexity than $LG - elim$. Running time: Roughly the same.
- 15 different systems using different p/c-pairs $\rightarrow$ 20 cubic polynomials in only key bits $\rightarrow$ Seems that we can produce many independent polynomials from different p/c-pairs.

Other results

- Checking for linear dependencies among 20 cubic polynomials we produced five *linear* polynomials in only key bits $\leftrightarrow$ Need much fewer polynomials than expected to find the values of $x_0, \ldots, x_{31}$.
- **13 rounds:** 47 variables, $F = \emptyset$, $|G| = 182$. For the 13-round systems we tried, neither $LG - elim$ or $eliminate$ found any cubic polynomials in only key variables $\rightarrow$ Only up to 12 rounds may be attacked using techniques.

Experimental results

- Eliminate all variables x_i for $i \geq 32 \rightarrow$ Find some polynomials of degree at most 3, only in $x_0, \ldots, x_{31}$.
- **12 rounds:** 44 variables, $F = \emptyset$, $|G| = 168$.
  - $LG - elim$: Produces 1-2 cubic polynomial(s) only in key variables. Memory requirement: Store 7560 polynomials from $G \cdot L$.
  - $eliminate$: Produce same polynomials as $LG - elim$. Size of F never above 2000 polynomials $\leftrightarrow$ $eliminate$ has less space complexity than $LG - elim$. Running time: Roughly the same.
- 15 different systems using different p/c-pairs $\rightarrow$ 20 cubic polynomials in only key bits $\rightarrow$ Seems that we can produce many independent polynomials from different p/c-pairs.

Other results

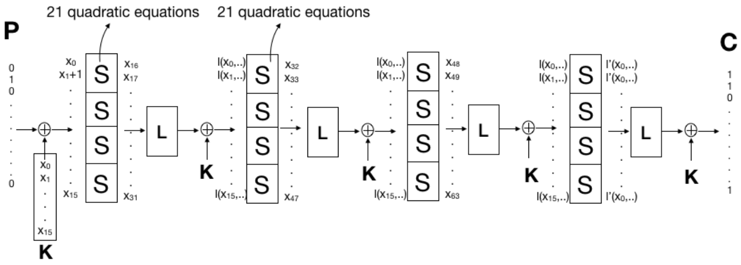
- Checking for linear dependencies among 20 cubic polynomials we produced five *linear* polynomials in only key bits $\leftrightarrow$ Need much fewer polynomials than expected to find the values of $x_0, \ldots, x_{31}$.
- **13 rounds:** 47 variables, $F = \emptyset$, $|G| = 182$. For the 13-round systems we tried, neither $LG - elim$ or $eliminate$ found any cubic polynomials in only key variables $\rightarrow$ Only up to 12 rounds may be attacked using techniques.

Experimental results

- Eliminate all variables x_i for $i \geq 32 \rightarrow$ Find some polynomials of degree at most 3, only in $x_0, \ldots, x_{31}$.
- **12 rounds:** 44 variables, $F = \emptyset$, $|G| = 168$.
  - $LG - elim$: Produces 1-2 cubic polynomial(s) only in key variables. Memory requirement: Store 7560 polynomials from $G \cdot L$.
  - $eliminate$: Produce same polynomials as $LG - elim$. Size of F never above 2000 polynomials $\leftrightarrow$ $eliminate$ has less space complexity than $LG - elim$. Running time: Roughly the same.
- 15 different systems using different p/c-pairs $\rightarrow$ 20 cubic polynomials in only key bits $\rightarrow$ Seems that we can produce many independent polynomials from different p/c-pairs.

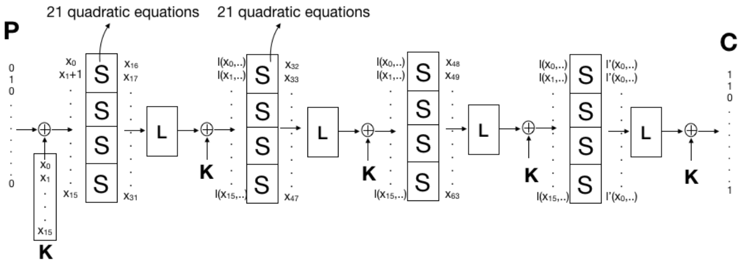
Other results

- Checking for linear dependencies among 20 cubic polynomials we produced five *linear* polynomials in only key bits $\leftrightarrow$ Need much fewer polynomials than expected to find the values of $x_0, \ldots, x_{31}$.
- **13 rounds:** 47 variables, $F = \emptyset$, $|G| = 182$. For the 13-round systems we tried, neither $LG - elim$ or $eliminate$ found any cubic polynomials in only key variables $\rightarrow$ Only up to 12 rounds may be attacked using techniques.



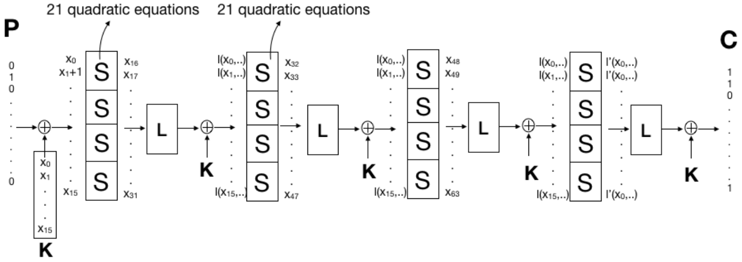
The toy cipher

- Uses four 4×4 S-boxes (the same S-box as used in PRINCE) → Use same key in every round.
- Cipher parameters used: Block size: 16-bit, key size: 16-bit → Used a 4-round version of Cipher.



The toy cipher

- Uses four 4×4 S-boxes (the same S-box as used in PRINCE) $\rightarrow$ Use same key in every round.
- Cipher parameters used: Block size: 16-bit, key size: 16-bit $\rightarrow$ Used a 4-round version of Cipher.



The toy cipher

- Uses four 4×4 S-boxes (the same S-box as used in PRINCE) $\rightarrow$ Use same key in every round.
- Cipher parameters used: Block size: 16-bit, key size: 16-bit $\rightarrow$ Used a 4-round version of Cipher.

Experimental results

- Eliminate all non-key variables $x_{16}, \ldots, x_{63}$ from the system → Find some polynomials of degree at most 3 only in $x_0, \ldots, x_{15}$.
- **4 rounds:** 64 variables, $F = \emptyset$, $|G| = 336$
  - None of $LG-elim$ or $eliminate$ were able to find any cubic polynomial in only key variables.

.

Information loss

- Running $LG-elim/eliminate$ → Throw away polynomials giving constraints on the solution space Introduce false solutions.
- $F = \emptyset$ and $G = \emptyset$ → all solutions are valid → "Lost all information about the possible solutions to the original equation system".
- Measure how fast the information about the solutions we seek disappear for the toy cipher.
- With only a 16-bit key it is possible to do exhaustive search → Check which key values that fit in any of the equation systems we get after eliminating some variables.

Experimental results

- Eliminate all non-key variables $x_{16}, \ldots, x_{63}$ from the system $\rightarrow$ Find some polynomials of degree at most 3 only in $x_0, \ldots, x_{15}$.
- **4 rounds:** 64 variables, $F = \emptyset$, $|G| = 336$
  - None of $LG - elim$ or $eliminate$ were able to find any cubic polynomial in only key variables.

.

Information loss

- Running $LG - elim/eliminate$ $\rightarrow$ Throw away polynomials giving constraints on the solution space Introduce false solutions.
- $F = \emptyset$ and $G = \emptyset$ $\rightarrow$ all solutions are valid $\rightarrow$ "Lost all information about the possible solutions to the original equation system".
- Measure how fast the information about the solutions we seek disappear for the toy cipher.
- With only a 16-bit key it is possible to do exhaustive search $\rightarrow$ Check which key values that fit in any of the equation systems we get after eliminating some variables.

Experimental results

- Eliminate all non-key variables $x_{16}, \ldots, x_{63}$ from the system → Find some polynomials of degree at most 3 only in $x_0, \ldots, x_{15}$.
- **4 rounds:** 64 variables, $F = \emptyset$, $|G| = 336$
  - None of $LG - elim$ or $eliminate$ were able to find any cubic polynomial in only key variables.

.

Information loss

- Running $LG - elim/eliminate$ → Throw away polynomials giving constraints on the solution space Introduce false solutions.
- $F = \emptyset$ and $G = \emptyset$ → all solutions are valid → "Lost all information about the possible solutions to the original equation system".
- Measure how fast the information about the solutions we seek disappear for the toy cipher.
- With only a 16-bit key it is possible to do exhaustive search → Check which key values that fit in any of the equation systems we get after eliminating some variables.

Experimental results

- Eliminate all non-key variables $x_{16}, \ldots, x_{63}$ from the system $\rightarrow$ Find some polynomials of degree at most 3 only in $x_0, \ldots, x_{15}$.
- **4 rounds:** 64 variables, $F = \emptyset$, $|G| = 336$
  - None of $LG - elim$ or $eliminate$ were able to find any cubic polynomial in only key variables.

.

Information loss

- Running $LG - elim/eliminate \rightarrow$ Throw away polynomials giving constraints on the solution space Introduce false solutions.
- $F = \emptyset$ and $G = \emptyset \rightarrow$ all solutions are valid $\rightarrow$ "Lost all information about the possible solutions to the original equation system".
- Measure how fast the information about the solutions we seek disappear for the toy cipher.
- With only a 16-bit key it is possible to do exhaustive search $\rightarrow$ Check which key values that fit in any of the equation systems we get after eliminating some variables.

Experimental results

- Eliminate all non-key variables $x_{16}, \ldots, x_{63}$ from the system → Find some polynomials of degree at most 3 only in $x_0, \ldots, x_{15}$.
- **4 rounds:** 64 variables, $F = \emptyset$, $|G| = 336$
  - None of $LG - elim$ or $eliminate$ were able to find any cubic polynomial in only key variables.

.

Information loss

- Running $LG - elim/eliminate$ → Throw away polynomials giving constraints on the solution space Introduce false solutions.
- $F = \emptyset$ and $G = \emptyset$ → all solutions are valid → "Lost all information about the possible solutions to the original equation system".
- Measure how fast the information about the solutions we seek disappear for the toy cipher.
- With only a 16-bit key it is possible to do exhaustive search → Check which key values that fit in any of the equation systems we get after eliminating some variables.

Experimental results

- Eliminate all non-key variables $x_{16}, \ldots, x_{63}$ from the system $\rightarrow$ Find some polynomials of degree at most 3 only in $x_0, \ldots, x_{15}$.
- **4 rounds:** 64 variables, $F = \emptyset$, $|G| = 336$
  - None of $LG - elim$ or $eliminate$ were able to find any cubic polynomial in only key variables.

.

Information loss

- Running $LG - elim/eliminate$ $\rightarrow$ Throw away polynomials giving constraints on the solution space Introduce false solutions.
- $F = \emptyset$ and $G = \emptyset$ $\rightarrow$ all solutions are valid $\rightarrow$ "Lost all information about the possible solutions to the original equation system".
- Measure how fast the information about the solutions we seek disappear for the toy cipher.
- With only a 16-bit key it is possible to do exhaustive search $\rightarrow$ Check which key values that fit in any of the equation systems we get after eliminating some variables.

Experimental results

- Eliminate all non-key variables $x_{16}, \dots, x_{63}$ from the system → Find some polynomials of degree at most 3 only in $x_0, \dots, x_{15}$.
- **4 rounds:** 64 variables, $F = \emptyset$, $|G| = 336$
  - None of $LG - elim$ or $eliminate$ were able to find any cubic polynomial in only key variables.

.

Information loss

- Running $LG - elim/eliminate$ → Throw away polynomials giving constraints on the solution space Introduce false solutions.
- $F = \emptyset$ and $G = \emptyset$ → all solutions are valid → "Lost all information about the possible solutions to the original equation system".
- Measure how fast the information about the solutions we seek disappear for the toy cipher.
- With only a 16-bit key it is possible to do exhaustive search → Check which key values that fit in any of the equation systems we get after eliminating some variables.

The information loss experiment

- Eliminate variables distributed evenly throughout the system → Check how many keys fits in the given system after each elimination → Gives information on how much information the system has about the unknown secret key.
- The amount of information a system S has about the key: $i(S) = 16 - log_2(\#$ of keys that fit in $S)$. S_v is the system after eliminating v variables.
- For the plaintext/ciphertext pair we used there were three keys that fit in the initial system ↔ $i(S_0) \approx 14.42$.
- What is the rate of information loss during elimination?

The information loss experiment

- Eliminate variables distributed evenly throughout the system $\rightarrow$ Check how many keys fits in the given system after each elimination $\rightarrow$ Gives information on how much information the system has about the unknown secret key.
- The amount of information a system S has about the key: $i(S) = 16 - log_2(\#$ of keys that fit in $S)$. S_v is the system after eliminating v variables.
- For the plaintext/ciphertext pair we used there were three keys that fit in the initial system $\leftrightarrow i(S_0) \approx 14.42$.
- What is the rate of information loss during elimination?

The information loss experiment

- Eliminate variables distributed evenly throughout the system → Check how many keys fits in the given system after each elimination → Gives information on how much information the system has about the unknown secret key.
- The amount of information a system S has about the key: $i(S) = 16 - log_2(\#$ of keys that fit in $S)$. S_v is the system after eliminating v variables.
- For the plaintext/ciphertext pair we used there were three keys that fit in the initial system $\leftrightarrow i(S_0) \approx 14.42$.
- What is the rate of information loss during elimination?

The information loss experiment

- Eliminate variables distributed evenly throughout the system $\rightarrow$ Check how many keys fits in the given system after each elimination $\rightarrow$ Gives information on how much information the system has about the unknown secret key.
- The amount of information a system S has about the key: $i(S) = 16 - log_2(\#$ of keys that fit in $S)$. S_v is the system after eliminating v variables.
- For the plaintext/ciphertext pair we used there were three keys that fit in the initial system $\leftrightarrow i(S_0) \approx 14.42$.
- What is the rate of information loss during elimination?

The information loss experiment

- Eliminate variables distributed evenly throughout the system $\rightarrow$ Check how many keys fits in the given system after each elimination $\rightarrow$ Gives information on how much information the system has about the unknown secret key.
- The amount of information a system S has about the key: $i(S) = 16 - log_2(\#$ of keys that fit in $S)$. S_v is the system after eliminating v variables.
- For the plaintext/ciphertext pair we used there were three keys that fit in the initial system $\leftrightarrow i(S_0) \approx 14.42$.
- What is the rate of information loss during elimination?

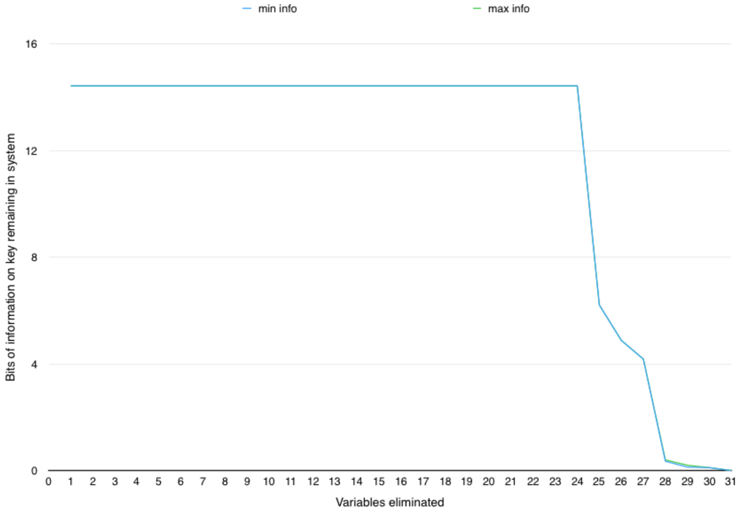


Figure: $i(S_v)$ for $0 \leq v \leq 31$

What this tells us

- For the Toy cipher it is possible to construct a cubic equation system, with the same information on the key, with only $k + (n-k)/2$ variables where k is the number of key bits $\rightarrow$ Trade-off between degree and number of variables needed to describe a cipher.
- I.e: For the toy cipher, increasing the degree by one allows to cut the number of non-key variables in half to describe the same cipher.

Open questions

- Attacks on other ciphers? When does the algorithm work and not?
- Generalizations of elimination algorithm?

What this tells us

- For the Toy cipher it is possible to construct a cubic equation system, with the same information on the key, with only $k + (n - k)/2$ variables where k is the number of key bits $\rightarrow$ Trade-off between degree and number of variables needed to describe a cipher.
- I.e: For the toy cipher, increasing the degree by one allows to cut the number of non-key variables in half to describe the same cipher.

Open questions

- Attacks on other ciphers? When does the algorithm work and not?
- Generalizations of elimination algorithm?

What this tells us

- For the Toy cipher it is possible to construct a cubic equation system, with the same information on the key, with only $k + (n - k)/2$ variables where k is the number of key bits $\rightarrow$ Trade-off between degree and number of variables needed to describe a cipher.
- I.e: For the toy cipher, increasing the degree by one allows to cut the number of non-key variables in half to describe the same cipher.

Open questions

- Attacks on other ciphers? When does the algorithm work and not?
- Generalizations of elimination algorithm?

What this tells us

- For the Toy cipher it is possible to construct a cubic equation system, with the same information on the key, with only $k + (n - k)/2$ variables where k is the number of key bits → Trade-off between degree and number of variables needed to describe a cipher.
- I.e: For the toy cipher, increasing the degree by one allows to cut the number of non-key variables in half to describe the same cipher.

Open questions

- Attacks on other ciphers? When does the algorithm work and not?
- Generalizations of elimination algorithm?

What this tells us

- For the Toy cipher it is possible to construct a cubic equation system, with the same information on the key, with only $k + (n - k)/2$ variables where k is the number of key bits $\rightarrow$ Trade-off between degree and number of variables needed to describe a cipher.
- I.e: For the toy cipher, increasing the degree by one allows to cut the number of non-key variables in half to describe the same cipher.

Open questions

- Attacks on other ciphers? When does the algorithm work and not?
- Generalizations of elimination algorithm?

What this tells us

- For the Toy cipher it is possible to construct a cubic equation system, with the same information on the key, with only $k + (n - k)/2$ variables where k is the number of key bits $\rightarrow$ Trade-off between degree and number of variables needed to describe a cipher.
- I.e: For the toy cipher, increasing the degree by one allows to cut the number of non-key variables in half to describe the same cipher.

Open questions

- Attacks on other ciphers? When does the algorithm work and not?
- Generalizations of elimination algorithm?