Large Substitution Boxes with Efficient Combinational Implementations

Christopher A. Wood
Department of Computer Science
Rochester Institute of Technology

Now: `caw4567@rit.edu`

Soon: `woodc1@uci.edu`

Advisor: Stanisław Radziszowski
Reader: Marcin Lukowiak
Observer: Alan Kaminsky

Observer: Michael Kurdziel

August 9, 2013

Agenda

Security from an Information Theory Perspective

According to Shannon, cryptographic algorithms were secure if they yielded high measures of confusion and diffusion:

- *Confusion* - complex relationship between the secret key and ciphertext
- *Diffusion* - dissipation of plaintext bits throughout ciphertext bits

Rijndael - The Advanced Encryption Standard

Let's take a look at the inner workings of Rijndael (AES) from Shannon's perspective

- Diffusion (linear operations):
  - Shift rows
  - Mix columns
  - Add round key
- Confusion (nonlinear operations):
  - Substitute bytes

The Importance of the S-Box

Why bother with the S-box?

- In Substitution-Permutation algorithms, the S-box is typically the only nonlinear operation
  - Reasonably sized linear systems of equations are easy to solve
  - Nonlinearity provides resistance to many known attacks
- S-boxes are used in both the DES and AES
- They can be implemented “efficiently” in both hardware (as we will show) and software
  - Secure implementations are not a part of this work. We assume precautions are taken to mitigate side-channel attacks.

The Importance of the S-Box

Why bother with the S-box?

- In Substitution-Permutation algorithms, the S-box is typically the only nonlinear operation
    - Reasonably sized linear systems of equations are easy to solve
    - Nonlinearity provides resistance to many known attacks
- S-boxes are used in both the DES and AES
- They can be implemented “efficiently” in both hardware (as we will show) and software
    - Secure implementations are not a part of this work. We assume precautions are taken to mitigate side-channel attacks.

The Importance of the S-Box

Why bother with the S-box?

- In Substitution-Permutation algorithms, the S-box is typically the only nonlinear operation
  - Reasonably sized linear systems of equations are easy to solve
  - Nonlinearity provides resistance to many known attacks
- S-boxes are used in both the DES and AES
- They can be implemented “efficiently” in both hardware (as we will show) and software
  - Secure implementations are not a part of this work. We assume precautions are taken to mitigate side-channel attacks.

Why Bother Going Bigger?

Why study at 16-bit S-boxes?

- They might be useful in the future
    - Will the size of internal elements in the next block cipher standard bump up to 16-bits?
- To determine if such implementations are feasible or practical
    - LUTs for 16-bit S-boxes are out of the question.
- It may offer insight into new perspectives of S-box constructions and implementation techniques

Why Bother Going Bigger?

Why study at 16-bit S-boxes?

- They might be useful in the future
  - Will the size of internal elements in the next block cipher standard bump up to 16-bits?
- To determine if such implementations are feasible or practical
  - LUTs for 16-bit S-boxes are out of the question.
- It may offer insight into new perspectives of S-box constructions and implementation techniques

Why Bother Going Bigger?

Why study at 16-bit S-boxes?

- They might be useful in the future
  - Will the size of internal elements in the next block cipher standard bump up to 16-bits?
- To determine if such implementations are feasible or practical
  - LUTs for 16-bit S-boxes are out of the question.
- It may offer insight into new perspectives of S-box constructions and implementation techniques

Constructing Cryptographically Strong S-boxes

- Question 1: What constitutes a cryptographically strong S-box?
  - One that is not susceptible to known attacks
- Question 2: How do we build cryptographically strong S-boxes?
  - Choose constructions that have *known* and *measurable* security properties
- Question 3: How can we implement these S-boxes efficiently in hardware?
  1. Decompose the S-box calculation into bitwise operations
  2. Minimize the logic involved in these operations

Constructing Cryptographically Strong S-boxes

- Question 1: What constitutes a cryptographically strong S-box?
  - One that is not susceptible to known attacks
- Question 2: How do we build cryptographically strong S-boxes?
  - Choose constructions that have *known* and *measurable* security properties
- Question 3: How can we implement these S-boxes efficiently in hardware?
  1. Decompose the S-box calculation into bitwise operations
  2. Minimize the logic involved in these operations

Constructing Cryptographically Strong S-boxes

- Question 1: What constitutes a cryptographically strong S-box?
  - One that is not susceptible to known attacks
- Question 2: How do we build cryptographically strong S-boxes?
  - Choose constructions that have *known* and *measurable* security properties
- Question 3: How can we implement these S-boxes efficiently in hardware?
  1. Decompose the S-box calculation into bitwise operations
  2. Minimize the logic involved in these operations

Constructing Cryptographically Strong S-boxes

- Question 1: What constitutes a cryptographically strong S-box?
  - One that is not susceptible to known attacks
- Question 2: How do we build cryptographically strong S-boxes?
  - Choose constructions that have *known* and *measurable* security properties
- Question 3: How can we implement these S-boxes efficiently in hardware?
  1. Decompose the S-box calculation into bitwise operations
  2. Minimize the logic involved in these operations

Constructing Cryptographically Strong S-boxes

- Question 1: What constitutes a cryptographically strong S-box?
  - One that is not susceptible to known attacks
- Question 2: How do we build cryptographically strong S-boxes?
  - Choose constructions that have *known* and *measurable* security properties
- Question 3: How can we implement these S-boxes efficiently in hardware?
  1. Decompose the S-box calculation into bitwise operations
  2. Minimize the logic involved in these operations

Constructing Cryptographically Strong S-boxes

- Question 1: What constitutes a cryptographically strong S-box?
  - One that is not susceptible to known attacks
- Question 2: How do we build cryptographically strong S-boxes?
  - Choose constructions that have *known* and *measurable* security properties
- Question 3: How can we implement these S-boxes efficiently in hardware?
  1. Decompose the S-box calculation into bitwise operations
  2. Minimize the logic involved in these operations

Galois Fields

- A Galois field $GF(\cdot)$ is just a finite field of p ($GF(p)$) or p^n ($GF(p^n)$) elements, where p is prime
- The field's characteristic is the smallest non-negative integer k such that $\underbrace{a+a+\cdots+a}_{k \text{ times}}=0$
- We focus on Galois fields with characteristic 2 - $GF(2^n)$ - as elements are easily represented as bit strings in hardware and software
- We say that $GF(2^n)$ is an n degree extension of $GF(2)$ (the subfield)

Composite Galois Fields

Composite fields are simply fields composed of *more than one extension*!

Let $k = n \times m$

$GF((2^n)^m) \pmod{p(v), q(w)}$ is a composite field isomorphic to $GF(2^k) \pmod{r(x)}$ if the following are true:

- $GF(2^n)$ is an n degree extension of $GF(2)$ by an n degree irreducible polynomial $p(v)$ over $GF(2)$
- $GF((2^n)^m)$ is an m degree extension of $GF(2^n)$ by an m degree irreducible polynomial $q(w)$ over $GF(2^n)$
- $GF(2^k)$ is a k degree extension of $GF(2)$ by an k degree irreducible polynomial $r(x)$ over $GF(2)$

Bases of Galois Fields

It is natural to represent an element $\alpha \in GF(p^n)$ as a polynomial of the form

$$\alpha = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_2x^2 + a_1x + a_0,$$

where $a_i \in GF(p)$ are the coefficients of the polynomial.

This is the polynomial basis representation.

Bases of Galois Fields

It is natural to represent an element $\alpha \in GF(p^n)$ as a polynomial of the form

$$\alpha = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_2x^2 + a_1x + a_0,$$

where $a_i \in GF(p)$ are the coefficients of the polynomial.

This is the polynomial basis representation.

Bases of Galois Fields (continued)

Let θ be a root of $p(x)$, the irreducible polynomial defining $GF(p^n)$, and let $\alpha \in GF(p^n)$

- Polynomial basis: $[\theta^0, \theta^i, \ldots, \theta^{n-1}]$

$$\alpha = a_{n-1}\theta^{n-1} + a_{n-2}\theta^{n-2} + \cdots + a_1\theta + a_0$$

- Normal basis: $[\theta^{p^0}, \theta^{p^1}, \ldots, \theta^{p^{n-1}}]$

$$\alpha = a_{n-1}\theta^{p^{n-1}} + a_{n-2}\theta^{p^{n-2}} + \cdots + a_{p^1}\theta^p + a_0\theta$$

(Note: $a_i \in GF(p)$)

Bases of Galois Fields (continued)

Let θ be a root of $p(x)$, the irreducible polynomial defining $GF(p^n)$, and let $\alpha \in GF(p^n)$

- Polynomial basis: $[\theta^0, \theta^i, \ldots, \theta^{n-1}]$

$$\alpha = a_{n-1}\theta^{n-1} + a_{n-2}\theta^{n-2} + \cdots + a_1\theta + a_0$$

- Normal basis: $[\theta^{p^0}, \theta^{p^1}, \ldots, \theta^{p^{n-1}}]$

$$\alpha = a_{n-1}\theta^{p^{n-1}} + a_{n-2}\theta^{p^{n-2}} + \cdots + a_{p^1}\theta^p + a_0\theta$$

(Note: $a_i \in GF(p)$)

Bases of Galois Fields (continued)

Let θ be a root of $p(x)$, the irreducible polynomial defining $GF(p^n)$, and let $\alpha \in GF(p^n)$

- Polynomial basis: $[\theta^0, \theta^i, \ldots, \theta^{n-1}]$

$$\alpha = a_{n-1}\theta^{n-1} + a_{n-2}\theta^{n-2} + \cdots + a_1\theta + a_0$$

- Normal basis: $[\theta^{p^0}, \theta^{p^1}, \ldots, \theta^{p^{n-1}}]$

$$\alpha = a_{n-1}\theta^{p^{n-1}} + a_{n-2}\theta^{p^{n-2}} + \cdots + a_{p^1}\theta^p + a_0\theta$$

(Note: $a_i \in GF(p)$)

Boolean Functions

A Boolean function f is simply a function of the form:

$$f : \mathbb{F}_2^n \to \mathbb{F}_2$$

We may represent a Boolean function in the following ways:

- Truth table:

$$f(x_0, \ldots, x_n) = (f(\bar{0}), \ldots, f(\bar{1}))$$

- Algebraic Normal Form (ANF):

$$f(x_0, \ldots, x_{n-1}) = \bigoplus_{(a_0,\ldots,a_{n-1}) \in \mathbb{F}_2^n} h(a_0, \ldots, a_{n-1}) x_0^{a_0} x_1^{a_1} \ldots x_{n-1}^{a_{n-1}}$$

S-boxes from $\mathbb{F}_2^n \to \mathbb{F}_2^m$ may be viewed as (n, m) Boolean functions

Boolean Functions

A Boolean function f is simply a function of the form:

$$f : \mathbb{F}_2^n \rightarrow \mathbb{F}_2$$

We may represent a Boolean function in the following ways:

- Truth table:

$$f(x_0, \ldots, x_n) = (f(\bar{0}), \ldots, f(\bar{1}))$$

- Algebraic Normal Form (ANF):

$$f(x_0, \ldots, x_{n-1}) = \bigoplus_{(a_0,\ldots,a_{n-1})\in\mathbb{F}_2^n} h(a_0, \ldots, a_{n-1}) x_0^{a_0} x_1^{a_1} \ldots x_{n-1}^{a_{n-1}}$$

S-boxes from $\mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$ may be viewed as (n, m) Boolean functions

Boolean Functions

A Boolean function f is simply a function of the form:

$$f : \mathbb{F}_2^n \to \mathbb{F}_2$$

We may represent a Boolean function in the following ways:

- Truth table:

$$f(x_0, \dots, x_n) = (f(\bar{0}), \dots, f(\bar{1}))$$

- Algebraic Normal Form (ANF):

$$f(x_0, \dots, x_{n-1}) = \bigoplus_{(a_0, \dots, a_{n-1}) \in \mathbb{F}_2^n} h(a_0, \dots, a_{n-1}) x_0^{a_0} x_1^{a_1} \dots x_{n-1}^{a_{n-1}}$$

S-boxes from $\mathbb{F}_2^n \to \mathbb{F}_2^m$ may be viewed as (n, m) Boolean functions

Common Attacks

- Linear cryptanalysis
- Differential cryptanalysis
- Algebraic attacks
- Interpolation attacks
- ... and more

Linear Cryptanalysis

Linear cryptanalysis exploits a high (or low) probability that some linear combination of input and output bits in the S-box is satisfied

What makes an S-box susceptible to this type of attack?

Low nonlinearity

Linear Cryptanalysis

Linear cryptanalysis exploits a high (or low) probability that some linear combination of input and output bits in the S-box is satisfied

What makes an S-box susceptible to this type of attack?

Low nonlinearity

Linear Cryptanalysis

Linear cryptanalysis exploits a high (or low) probability that some linear combination of input and output bits in the S-box is satisfied

What makes an S-box susceptible to this type of attack?

Low nonlinearity

S-Box Nonlinearity

The *nonlinearity* of an (n, m) S-box S, denoted as $\mathcal{N}_l(S)$, is defined as follows:

$$\mathcal{N}_l(S) = \min_{c \in \mathbb{F}_2^m} \{\mathcal{N}_l(c \cdot S)\} = \min_{c \in \mathbb{F}_2^m} \{\mathcal{N}_l(c_0 f_0 \oplus c_1 f_1 \oplus \cdots \oplus c_{m-1} f_{m-1})\},$$

where $f_0, \ldots, f_{m-1}$ are the m coordinate functions of S and

$$\begin{aligned} N_l(f) &= \min_{\phi \in \Omega_n} d(f, \phi) \\ &= 2^{n-1} - \frac{1}{2} \max_{u \in \mathbb{F}_2^n} |W_f(u)| \end{aligned}$$

Differential Cryptanalysis

- Differential cryptanalysis exploits output differences that occur with high probability for each round of a cipher
- Put another way: Frequently occurring differences mean there exists a value $\beta \in GF(2^k)$ such that for any two randomly selected input elements x and α, $S(x+\alpha)+S(x)=\beta$
- An ideal block cipher will have perfectly uniform differentials

What makes an S-box susceptible to this type of attack?

"High" differential uniformity

Differential Cryptanalysis

- Differential cryptanalysis exploits output differences that occur with high probability for each round of a cipher
- Put another way: Frequently occurring differences mean there exists a value $\beta \in GF(2^k)$ such that for any two randomly selected input elements x and α, $S(x+\alpha)+S(x)=\beta$
- An ideal block cipher will have perfectly uniform differentials

What makes an S-box susceptible to this type of attack?

"High" differential uniformity

Differential Cryptanalysis

- Differential cryptanalysis exploits output differences that occur with high probability for each round of a cipher
- Put another way: Frequently occurring differences mean there exists a value $\beta \in GF(2^k)$ such that for any two randomly selected input elements x and α, $S(x+\alpha)+S(x)=\beta$
- An ideal block cipher will have perfectly uniform differentials

What makes an S-box susceptible to this type of attack?

"High" differential uniformity

Differential Cryptanalysis

- Differential cryptanalysis exploits output differences that occur with high probability for each round of a cipher
- Put another way: Frequently occurring differences mean there exists a value $\beta \in GF(2^k)$ such that for any two randomly selected input elements x and α, $S(x+\alpha)+S(x)=\beta$
- An ideal block cipher will have perfectly uniform differentials

What makes an S-box susceptible to this type of attack?

"High" differential uniformity

Differential Cryptanalysis

- Differential cryptanalysis exploits output differences that occur with high probability for each round of a cipher
- Put another way: Frequently occurring differences mean there exists a value $\beta \in GF(2^k)$ such that for any two randomly selected input elements x and α, $S(x+\alpha)+S(x)=\beta$
- An ideal block cipher will have perfectly uniform differentials

What makes an S-box susceptible to this type of attack?

"High" differential uniformity

Differential Uniformity

An S-box $S : GF(2^n) \to GF(2^m)$ is δ-differentially uniform if for all $\alpha \in GF(2^n)$ and $\beta \in GF(2^m)$ we have

$$|\{x \in GF(2^n) | S(x+\alpha) + S(x) = \beta\}| \leq \delta,$$

- There exists many known (n, m) S-boxes with low values of δ (i.e. 1 - perfectly nonlinear - or 2 - almost perfectly nonlinear)
- They are *rarely* used! The inverse power mapping has $\delta = 4$, which is low enough for cryptographic purposes

Differential Uniformity

An S-box $S : GF(2^n) \to GF(2^m)$ is δ-differentially uniform if for all $\alpha \in GF(2^n)$ and $\beta \in GF(2^m)$ we have

$$|\{x \in GF(2^n) | S(x+\alpha) + S(x) = \beta\}| \leq \delta,$$

- There exists many known (n, m) S-boxes with low values of δ (i.e. 1 - perfectly nonlinear - or 2 - almost perfectly nonlinear)
- They are *rarely* used! The inverse power mapping has $\delta = 4$, which is low enough for cryptographic purposes

Differential Uniformity

An S-box $S : GF(2^n) \rightarrow GF(2^m)$ is δ-differentially uniform if for all $\alpha \in GF(2^n)$ and $\beta \in GF(2^m)$ we have

$$|\{x \in GF(2^n) | S(x+\alpha) + S(x) = \beta\}| \leq \delta,$$

- There exists many known (n, m) S-boxes with low values of δ (i.e. 1 - perfectly nonlinear - or 2 - almost perfectly nonlinear)
- They are *rarely* used! The inverse power mapping has $\delta = 4$, which is low enough for cryptographic purposes

Other Well Known Attacks

- Algebraic attacks
  - Goal: define the cipher as (large) system of *linearized* equations with key-dependent coefficients to solve
  - Relevant metric: algebraic immunity
    - Dependent on the number of affine equations and monomials present in the linearized system
    - Both can be calculated in $\mathcal{O}(n^2)$ time
- Interpolation attacks
  - Goal: model the cipher as a high-order polynomial with key-dependent coefficients and solve
  - Relevant metric: algebraic complexity
    - Proportional to the number of terms in the unique interpolation polynomial representing the S-box
    - Can be determined using Lagrangian interpolation

Other Well Known Attacks

- Algebraic attacks
  - Goal: define the cipher as (large) system of *linearized* equations with key-dependent coefficients to solve
  - Relevant metric: algebraic immunity
    - Dependent on the number of affine equations and monomials present in the linearized system
    - Both can be calculated in $\mathcal{O}(n^2)$ time
- Interpolation attacks
  - Goal: model the cipher as a high-order polynomial with key-dependent coefficients and solve
  - Relevant metric: algebraic complexity
    - Proportional to the number of terms in the unique interpolation polynomial representing the S-box
    - Can be determined using Lagrangian interpolation

Other Relevant Security Metrics

There are other important metrics by which we can assess the strength of cryptographic S-boxes:

- *t*-Resiliency
  - Balanced outputs (i.e. bijective) and *t*-CI (correlation immunity)
  - The output bit distribution remains unchanged when at most *t* input bits are fixed
- Branch number
  - Estimating the cipher's measure of diffusion
- Strict Avalanche Criteria (SAC)
  - Ensuring that $1/2$ of output bits change for every single input bit change

S-Box Constructions

- ~~Question 1: What constitutes a cryptographically strong S-box?~~
  - ~~One that is not susceptible to known attacks~~
- Question 2: How do we build cryptographically strong S-boxes?
  - Choose constructions that have *known* and *measurable* security properties

Possible S-Box Constructions

We enforce the constraint that S-boxes are bijective functions from $\mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n}$ (i.e. (n, n) Boolean functions).

Two very popular constructions are:

- Boolean functions
  - Pros: Superb cryptographic properties (nonlinearity, differential uniformity, correlation immunity, resiliency, etc)
  - Cons: Efficient implementations are difficult to obtain
- **Galois field power mappings**
  - Combined with affine transformations for increased algebraic complexity
  - Pros: simple algebraic expressions, well studied, *basis of the AES S-box*
  - Cons: Difficult to obtain ideal cryptographic properties that Boolean function constructions yield, but they are still good enough

Possible S-Box Constructions

We enforce the constraint that S-boxes are bijective functions from $\mathbb{F}_{2^n} \rightarrow \mathbb{F}_{2^n}$ (i.e. (n, n) Boolean functions).

Two very popular constructions are:

- Boolean functions
  - Pros: Superb cryptographic properties (nonlinearity, differential uniformity, correlation immunity, resiliency, etc)
  - Cons: Efficient implementations are difficult to obtain
- **Galois field power mappings**
  - Combined with affine transformations for increased algebraic complexity
  - Pros: simple algebraic expressions, well studied, *basis of the AES S-box*
  - Cons: Difficult to obtain ideal cryptographic properties that Boolean function constructions yield, but they are still good enough

Cryptographically Significant Power Mappings

Important properties of power mappings over $GF(2^n)$:

- Must be of the form $f(x) = x^d$
  - They are usually characterized by their exponent d
- Only bijective if $\gcd\{d, 2^n - 1\} = 1$

Cryptographically Significant Power Mappings (continued)

There are many known cryptographically significant power mappings with ideal nonlinearity and differential uniformity:

- Gold: $2^k + 1$, $\gcd\{k, n\} = 1$ for some $1 \leq k \leq 2^n - 1$
- Kasami: $2^{2k} - 2^k + 1$, $\gcd\{k, n\} = 1$ for some $1 \leq k \leq n/2$
- Dobbertin: $2^{4k+3k+2k+k} - 1$ over $GF(2^n)$ with $n = 5k$
- Niho: $2^m + 2^{m/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m even, $2^m + 2^{(3m+1)/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m odd
- Welch: $2^m + 3$ over $GF(2^n)$ with $n = 2m + 1$
- Inverse: $-1 \equiv 2^n - 2$

Choosing the Right Mapping

What if $n = 16$?

- ~~Gold: $2^k + 1, \gcd\{k, n\} = 1$ for some $1 \leq k \leq 2^n - 1$~~
- ~~Kasami: $2^{2k} - 2^k + 1, \gcd\{k, n\} = 1$ for some $1 \leq k \leq n/2$~~
- ~~Dobbertin: $2^{4k+3k+2k+k} - 1$ over $GF(2^n)$ with $n = 5k$~~
- ~~Niho: $2^m + 2^{m/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m even, $2^m + 2^{(3m+1)/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m odd~~
- ~~Welch: $2^m + 3$ over $GF(2^n)$ with $n = 2m + 1$~~

The inverse mapping is the only remaining candidate!

Choosing the Right Mapping

What if $n = 16$?

- ~~Gold: $2^k + 1$, $\gcd\{k, n\} = 1$ for some $1 \leq k \leq 2^n - 1$~~
- ~~Kasami: $2^{2k} - 2^k + 1$, $\gcd\{k, n\} = 1$ for some $1 \leq k \leq n/2$~~
- ~~Dobbertin: $2^{4k+3k+2k+k} - 1$ over $GF(2^n)$ with $n = 5k$~~
- ~~Niho: $2^m + 2^{m/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m even, $2^m + 2^{(3m+1)/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m odd~~
- ~~Welch: $2^m + 3$ over $GF(2^n)$ with $n = 2m + 1$~~

The inverse mapping is the only remaining candidate!

Choosing the Right Mapping

What if $n = 16$?

- ~~Gold: $2^k + 1$, $\gcd\{k, n\} = 1$ for some $1 \leq k \leq 2^n - 1$~~
- ~~Kasami: $2^{2k} - 2^k + 1$, $\gcd\{k, n\} = 1$ for some $1 \leq k \leq n/2$~~
- ~~Dobbertin: $2^{4k+3k+2k+k} - 1$ over $GF(2^n)$ with $n = 5k$~~
- ~~Niho: $2^m + 2^{m/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m even, $2^m + 2^{(3m+1)/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m odd~~
- ~~Welch: $2^m + 3$ over $GF(2^n)$ with $n = 2m + 1$~~

The inverse mapping is the only remaining candidate!

Choosing the Right Mapping

What if $n = 16$?

- ~~Gold: $2^k + 1, \gcd\{k, n\} = 1$ for some $1 \leq k \leq 2^n - 1$~~
- ~~Kasami: $2^{2k} - 2^k + 1, \gcd\{k, n\} = 1$ for some $1 \leq k \leq n/2$~~
- ~~Dobbertin: $2^{4k+3k+2k+k} - 1$ over $GF(2^n)$ with $n = 5k$~~
- ~~Niho: $2^m + 2^{m/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m even, $2^m + 2^{(3m+1)/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m odd~~
- ~~Welch: $2^m + 3$ over $GF(2^n)$ with $n = 2m + 1$~~

The inverse mapping is the only remaining candidate!

Choosing the Right Mapping

What if $n = 16$?

- ~~Gold: $2^k + 1, \gcd\{k, n\} = 1$ for some $1 \leq k \leq 2^n - 1$~~
- ~~Kasami: $2^{2k} - 2^k + 1, \gcd\{k, n\} = 1$ for some $1 \leq k \leq n/2$~~
- ~~Dobbertin: $2^{4k+3k+2k+k} - 1$ over $GF(2^n)$ with $n = 5k$~~
- ~~Niho: $2^m + 2^{m/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m even,~~
  ~~$2^m + 2^{(3m+1)/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m odd~~
- ~~Welch: $2^m + 3$ over $GF(2^n)$ with $n = 2m + 1$~~

The inverse mapping is the only remaining candidate!

Choosing the Right Mapping

What if $n = 16$?

- ~~Gold: $2^k + 1$, $\gcd\{k, n\} = 1$ for some $1 \leq k \leq 2^n - 1$~~
- ~~Kasami: $2^{2k} - 2^k + 1$, $\gcd\{k, n\} = 1$ for some $1 \leq k \leq n/2$~~
- ~~Dobbertin: $2^{4k+3k+2k+k} - 1$ over $GF(2^n)$ with $n = 5k$~~
- ~~Niho: $2^m + 2^{m/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m even, $2^m + 2^{(3m+1)/2} - 1$ over $GF(2^n)$ with $n = 2m + 1$ and m odd~~
- ~~Welch: $2^m + 3$ over $GF(2^n)$ with $n = 2m + 1$~~

The inverse mapping is the only remaining candidate!

Affine Transformation Criteria

- Affine transformations are used to increase the complexity of the algebraic representation of the S-box.
  - Without an affine transformation, the algebraic representation would simply be $S(x) = x^d$
- AES-like S-boxes $S(x)$ have the form

$$S(x) = A(P(x))$$
$$S^{-1}(x) = P^{-1}(A^{-1}(x))$$

  where $P(x)$ is the inverse power mapping and $A(x)$ is the affine transformation
- The maximum algebraic complexity (number of terms in the interpolation polynomial) of AES-like S-boxes over $GF(2^n)$ is $n+1$
  - This has proven sufficient for preventing interpolation attacks
- The final S-box should have no fixed points:

$$S(x) \oplus x = \{0\}^n$$
$$S(x) \oplus x = \{1\}^n$$

Affine Transformation Criteria

- Affine transformations are used to increase the complexity of the algebraic representation of the S-box.
  - Without an affine transformation, the algebraic representation would simply be $S(x) = x^d$
- AES-like S-boxes $S(x)$ have the form

$$S(x) = A(P(x))$$
$$S^{-1}(x) = P^{-1}(A^{-1}(x))$$

  where $P(x)$ is the inverse power mapping and $A(x)$ is the affine transformation
- The maximum algebraic complexity (number of terms in the interpolation polynomial) of AES-like S-boxes over $GF(2^n)$ is $n+1$
  - This has proven sufficient for preventing interpolation attacks
- The final S-box should have no fixed points:

$$S(x) \oplus x = \{0\}^n$$
$$S(x) \oplus x = \{1\}^n$$

Affine Transformation Criteria

- Affine transformations are used to increase the complexity of the algebraic representation of the S-box.
    - Without an affine transformation, the algebraic representation would simply be $S(x) = x^d$
- AES-like S-boxes $S(x)$ have the form

$$S(x) = A(P(x))$$
$$S^{-1}(x) = P^{-1}(A^{-1}(x))$$

  where $P(x)$ is the inverse power mapping and $A(x)$ is the affine transformation
- The maximum algebraic complexity (number of terms in the interpolation polynomial) of AES-like S-boxes over $GF(2^n)$ is $n+1$
    - This has proven sufficient for preventing interpolation attacks
- The final S-box should have no fixed points:

$$S(x) \oplus x = \{0\}^n$$
$$S(x) \oplus x = \{1\}^n$$

Affine Transformation Criteria

- Affine transformations are used to increase the complexity of the algebraic representation of the S-box.
  - Without an affine transformation, the algebraic representation would simply be $S(x) = x^d$
- AES-like S-boxes $S(x)$ have the form

$$S(x) = A(P(x))$$
$$S^{-1}(x) = P^{-1}(A^{-1}(x))$$

  where $P(x)$ is the inverse power mapping and $A(x)$ is the affine transformation
- The maximum algebraic complexity (number of terms in the interpolation polynomial) of AES-like S-boxes over $GF(2^n)$ is $n+1$
  - This has proven sufficient for preventing interpolation attacks
- The final S-box should have no fixed points:

$$S(x) \oplus x = \{0\}^n$$
$$S(x) \oplus x = \{1\}^n$$

Searching for Affine Transformations

Non-deterministic procedure for finding suitable affine transformations

Input: $GF(2^n), d, k$

1. Generate a random matrix $\mathbf{A}$ over $GF(2)$
2. If $det(\mathbf{A}) \neq 0$ go to step 1.
3. Generate a random element $c \in GF(2^n)$ with weight k
4. Iterate over every element $e \in GF(2^n)$ and compute $y = \mathbf{A}(e^d) + c$ and $y^{-1} = (\mathbf{A}^{-1}(y + c))^{d^{-1}}$. If e yields a fixed point with $\mathbf{A}$ and c, go to step 1. Store y and y^{-1} in maps S and S^{-1}.
5. Perform Lagrangian interpolation to obtain $p(y)$ and $p^{-1}(y)$ for S and S^{-1}.
6. If $\#p(y) > n$ and $\#p^{-1}(y) > n$, return $\mathbf{A}$ and c, else go to step 1.

Efficient S-Box Computations

- ~~Question 1: What constitutes a cryptographically strong S-box?~~
  - ~~One that is not susceptible to known attacks~~
- ~~Question 2: How do we build cryptographically strong S-boxes?~~
  - ~~Choose constructions that have *known* and *measurable* security properties~~
- Question 3: How can we implement these S-boxes efficiently in hardware?
  1. Decompose the S-box calculation into bitwise operations
  2. Minimize the logic involved in these operations

Computing the Multiplicative Inverse

- Extended Euclidean algorithm
- Fermat's Little Theorem: $x^{-1} \equiv x^{2^n - 2}$ in $GF(2^n)$
- **Reduction to subfield inversion using composite fields**
  - Decomposition to arithmetic over $GF(2)$
  - Itoh-Tsujii inversion algorithm

Computing the Multiplicative Inverse

- Extended Euclidean algorithm
- Fermat's Little Theorem: $x^{-1} \equiv x^{2^n-2}$ in $GF(2^n)$
- **Reduction to subfield inversion using composite fields**
  - Decomposition to arithmetic over $GF(2)$
  - Itoh-Tsujii inversion algorithm

Computing the Multiplicative Inverse

- Extended Euclidean algorithm
- Fermat's Little Theorem: $x^{-1} \equiv x^{2^n-2}$ in $GF(2^n)$
- **Reduction to subfield inversion using composite fields**
  - Decomposition to arithmetic over $GF(2)$
  - Itoh-Tsujii inversion algorithm

Computing the Multiplicative Inverse

- Extended Euclidean algorithm
- Fermat's Little Theorem: $x^{-1} \equiv x^{2^n-2}$ in $GF(2^n)$
- **Reduction to subfield inversion using composite fields**
  - Decomposition to arithmetic over $GF(2)$
  - Itoh-Tsujii inversion algorithm

Computing the Multiplicative Inverse

- Extended Euclidean algorithm
- Fermat's Little Theorem: $x^{-1} \equiv x^{2^n-2}$ in $GF(2^n)$
- **Reduction to subfield inversion using composite fields**
  - Decomposition to arithmetic over $GF(2)$
  - Itoh-Tsujii inversion algorithm

Isomorphic Composite Fields

$$\begin{tikzcd}
 & & & GF(2^{16}) \arrow[dll] \arrow[d] \arrow[drr] & & \\
 & GF((2^8)^2) \arrow[dl] \arrow[dr] & & GF((2^2)^8) & & GF((2^4)^4) \arrow[dl] \\
GF(((2^4)^2)^2) \arrow[dr] & & GF(((2^2)^4)^2) & & GF(((2^2)^2)^4) & \\
 & \mathbf{GF((((2^2)^2)^2)^2)} & & & &
\end{tikzcd}$$

We focus on $GF((((2^2)^2)^2)^2)$.

Isomorphic Composite Fields

$$\begin{tikzcd}
 & & & GF(2^{16}) \arrow[dll] \arrow[d] \arrow[dr] & \\
 & GF((2^8)^2) \arrow[dl] \arrow[dr] & & GF((2^2)^8) & GF((2^4)^4) \arrow[d] \\
GF(((2^4)^2)^2) \arrow[dr] & & GF(((2^2)^4)^2) & & GF(((2^2)^2)^4) \\
 & \mathbf{GF}((((2^2)^2)^2)^2) & & &
\end{tikzcd}$$

We focus on $GF((((2^2)^2)^2)^2)$.

Galois Field Arithmetic Decompositions

With composite fields, coefficient arithmetic is performed over the subfield.

How do we go about determining the complexity of all relevant arithmetic operations?

Define arithmetic in the first extension of $GF(2)$ - $GF(2^2)$ - and then walk up the tower to $GF((((2^2)^2)^2)^2)$, constructed as follows:

- $GF(2^2)/p(v) = v^2 + v + 1$
- $GF((2^2)^2)/q(w) = w^2 + w + \Sigma$
- $GF(((2^2)^2)^2)/r(x) = x^2 + x + \Pi$
- $GF((((2^2)^2)^2)^2)/s(y) = y^2 + y + \Lambda$

Galois Field Arithmetic Decompositions

With composite fields, coefficient arithmetic is performed over the subfield.

How do we go about determining the complexity of all relevant arithmetic operations?

Define arithmetic in the first extension of $GF(2)$ - $GF(2^2)$ - and then walk up the tower to $GF((((2^2)^2)^2)^2)$, constructed as follows:

- $GF(2^2)/p(v) = v^2 + v + 1$
- $GF((2^2)^2)/q(w) = w^2 + w + \Sigma$
- $GF(((2^2)^2)^2)/r(x) = x^2 + x + \Pi$
- $GF((((2^2)^2)^2)^2)/s(y) = y^2 + y + \Lambda$

Galois Field Arithmetic Decompositions

With composite fields, coefficient arithmetic is performed over the subfield.

How do we go about determining the complexity of all relevant arithmetic operations?

Define arithmetic in the first extension of $GF(2)$ - $GF(2^2)$ - and then walk up the tower to $GF((((2^2)^2)^2)^2)$, constructed as follows:

- $GF(2^2)/p(v) = v^2 + v + 1$
- $GF((2^2)^2)/q(w) = w^2 + w + \Sigma$
- $GF(((2^2)^2)^2)/r(x) = x^2 + x + \Pi$
- $GF((((2^2)^2)^2)^2)/s(y) = y^2 + y + \Lambda$

Inversion in $GF(2^2)$

Inversion in $GF(2)$ is trivial, so we start with the first extension:

$$GF(2^2)/p(v) = v^2 + v + 1$$

For $\delta \in GF(2^2)$ and $\gamma \in GF(2)$. We may compute δ^{-1} as follows:

- Polynomial basis $[1, V]$

$$\delta = \gamma_1 v + \gamma_2$$
$$\delta^{-1} = \gamma_1 v + (\gamma_1 + \gamma_2)$$

- Normal basis $[V, V^2]$ (by Fermat's Little Theorem)

$$\delta = \gamma_1 v^2 + \gamma_2 v$$
$$\delta^{-1} = \gamma_2 v^2 + \gamma_1 v$$

Overall: Polynomial basis requires one XOR and normal basis is "free" (bit swap)

Inversion in $GF(2^2)$

Inversion in $GF(2)$ is trivial, so we start with the first extension:

$$GF(2^2)/p(v) = v^2 + v + 1$$

For $\delta \in GF(2^2)$ and $\gamma \in GF(2)$. We may compute δ^{-1} as follows:

- Polynomial basis $[1, V]$

$$\delta = \gamma_1 v + \gamma_2$$
$$\delta^{-1} = \gamma_1 v + (\gamma_1 + \gamma_2)$$

- Normal basis $[V, V^2]$ (by Fermat's Little Theorem)

$$\delta = \gamma_1 v^2 + \gamma_2 v$$
$$\delta^{-1} = \gamma_2 v^2 + \gamma_1 v$$

Overall: Polynomial basis requires one XOR and normal basis is “free” (bit swap)

Inversion in $GF(2^2)$

Inversion in $GF(2)$ is trivial, so we start with the first extension:

$$GF(2^2)/p(v) = v^2 + v + 1$$

For $\delta \in GF(2^2)$ and $\gamma \in GF(2)$. We may compute δ^{-1} as follows:

- Polynomial basis $[1, V]$

$$\delta = \gamma_1 v + \gamma_2$$
$$\delta^{-1} = \gamma_1 v + (\gamma_1 + \gamma_2)$$

- Normal basis $[V, V^2]$ (by Fermat's Little Theorem)

$$\delta = \gamma_1 v^2 + \gamma_2 v$$
$$\delta^{-1} = \gamma_2 v^2 + \gamma_1 v$$

Overall: Polynomial basis requires one XOR and normal basis is “free” (bit swap)

Inversion in $GF(2^2)$

Inversion in $GF(2)$ is trivial, so we start with the first extension:

$$GF(2^2)/p(v) = v^2 + v + 1$$

For $\delta \in GF(2^2)$ and $\gamma \in GF(2)$. We may compute δ^{-1} as follows:

- Polynomial basis $[1, V]$

$$\delta = \gamma_1 v + \gamma_2$$
$$\delta^{-1} = \gamma_1 v + (\gamma_1 + \gamma_2)$$

- Normal basis $[V, V^2]$ (by Fermat's Little Theorem)

$$\delta = \gamma_1 v^2 + \gamma_2 v$$
$$\delta^{-1} = \gamma_2 v^2 + \gamma_1 v$$

Overall: Polynomial basis requires one XOR and normal basis is “free” (bit swap)

Inversion With More Extensions

Things become more complicated when $GF(2^2)$ is extended to larger fields. Let $\varepsilon \in GF((2^2)^2)/q(w) = w^2 + w + \Sigma$, $\delta \in GF(2^2)$.

- Polynomial basis $[1, W]$

$$\varepsilon^{-1} = \delta_1(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\Sigma)^{-1}w + (\delta_1 + \delta_2)(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\Sigma)^{-1}$$

- Normal basis $[W, W^4]$

$$\varepsilon^{-1} = ((\delta_1\delta_2 + (\delta_1 + \delta_2)^2\Sigma)^{-1}\delta_2)w^4 + ((\delta_1\delta_2 + (\delta_1 + \delta_2)^2\Sigma)^{-1}\delta_1)w$$

Inversion With More Extensions

Things become more complicated when $GF(2^2)$ is extended to larger fields. Let $\varepsilon \in GF((2^2)^2)/q(w) = w^2 + w + \Sigma$, $\delta \in GF(2^2)$.

- Polynomial basis $[1, W]$

$$\varepsilon^{-1} = \delta_1(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\Sigma)^{-1}w + (\delta_1 + \delta_2)(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\Sigma)^{-1}$$

- Normal basis $[W, W^4]$

$$\varepsilon^{-1} = ((\delta_1\delta_2 + (\delta_1 + \delta_2)^2\Sigma)^{-1}\delta_2)w^4 + ((\delta_1\delta_2 + (\delta_1 + \delta_2)^2\Sigma)^{-1}\delta_1)w$$

Inversion With More Extensions

Things become more complicated when $GF(2^2)$ is extended to larger fields. Let $\varepsilon \in GF((2^2)^2)/q(w) = w^2 + w + \Sigma$, $\delta \in GF(2^2)$.

- Polynomial basis $[1, W]$

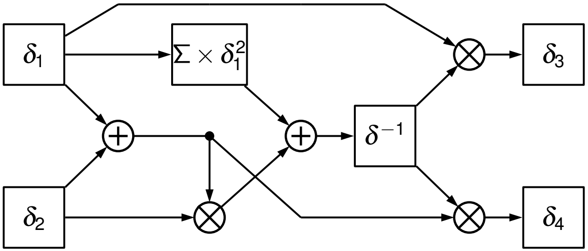
$$\varepsilon^{-1} = \delta_1(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\Sigma)^{-1}w + (\delta_1 + \delta_2)(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\Sigma)^{-1}$$

- Normal basis $[W, W^4]$

$$\varepsilon^{-1} = ((\delta_1\delta_2 + (\delta_1 + \delta_2)^2\Sigma)^{-1}\delta_2)w^4 + ((\delta_1\delta_2 + (\delta_1 + \delta_2)^2\Sigma)^{-1}\delta_1)w$$

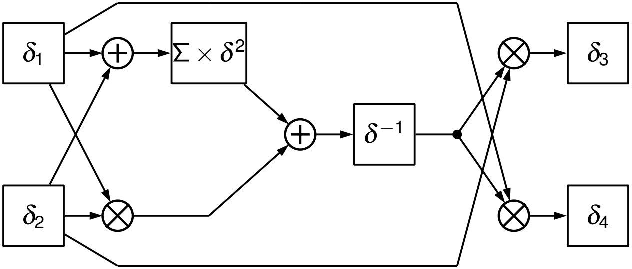
Corresponding Inversion Circuits - Polynomial Basis

$$\varepsilon^{-1} = \delta_1(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\Sigma)^{-1}w + (\delta_1 + \delta_2)(\delta_2^2 + \delta_1\delta_2 + \delta_1^2\Sigma)^{-1}$$



Corresponding Inversion Circuits - Normal Basis

$$\varepsilon^{-1} = ((\delta_1\delta_2 + (\delta_1+\delta_2)^2\Sigma)^{-1}\delta_2)w^4 + ((\delta_1\delta_2 + (\delta_1+\delta_2)^2\Sigma)^{-1}\delta_1)w$$



Other Relevant Arithmetic

The previous inverse expressions require subfield multiplication, squaring, scaling, addition, and inversion.

We can derive similar expressions for these operations to count the number of required logic gates.

Other Relevant Arithmetic

The previous inverse expressions require subfield multiplication, squaring, scaling, addition, and inversion.

We can derive similar expressions for these operations to count the number of required logic gates.

Combinational Arithmetic Complexity

Required XOR gates for $GF(2^2)$ arithmetic operations using polynomial and normal bases. Note that each multiplication operation requires three AND gates.

Operation	*Polynomial Basis*	*Normal Basis*
Inverse	1	0
Add	2	2
Multiply	4	4
Square	1	0
Scale	1	1
Square-Scale	0 ($\Sigma = v$) or 1 ($\Sigma = v + 1$)	1

Arithmetic in Higher Extension Fields

Arithmetic in higher fields is relatively similar, except:

- Coefficient arithmetic is not in the base field $GF(2)$
- The norm is no longer unity (e.g. $q(w) = w^2 + w + \Sigma$), so all arithmetic must take Σ into account

$GF((2^2)^2)$ General Arithmetic Results

Subfield arithmetic costs ((A)dditions, (M)ultiplications, (Sq)uares, (I)nversions, (SS)quare-scales, (Sc)ales) for finite field arithmetic operations in $GF((2^2)^2)$ using polynomial and normal bases.

Operation	*Polynomial Basis*	*Normal Basis*
Inverse	$3M + 2A + I + 1SS$	$3M + 2A + I + 1SS$
Add	$2A$	$2A$
Multiply	$3M + 4A + 1Sc$	$3M + 4A + Sc$
Square	$2Sq + Sc + A$	$3A + 2Sq + Sc$

Scaling (and square-scaling) can be further optimized to account for known constants

Arithmetic in $GF((2^2)^2)$ - Polynomial Scaling

Optimized costs of polynomial scaling in $GF((2^2)^2)$.

Coefficients for Polynomial $GF((2^2)^2)$ Basis			XOR Gate Counts		
$\Pi =$ $\delta_3 w + \delta_4$		$\varepsilon_1 \times \Pi =$ $(\delta_2\delta_4 + (\delta_1+\delta_2)(\delta_3+\delta_4))w + (\delta_2\delta_4 + \delta_1\delta_3\Sigma)$	Pol. $GF(2^2)$ $\Sigma = v$	$\Sigma = v^2$	Norm. $GF(2^2)$
Σ	0	$(\Sigma(\delta_1+\delta_2))w + (\delta_1\Sigma^2)$	4	4	4
Σ^2	0	$(\Sigma^2(\delta_1+\delta_2))w + (\delta_1)$	3	3	3
Σ	Σ	$(\delta_2\Sigma)w + (\delta_2\Sigma + \delta_1\Sigma^2)$	4	4	4
Σ^2	Σ^2	$(\delta_2\Sigma^2)w + (\delta_2\Sigma^2 + \delta_1)$	3	3	3
Σ	1	$(\delta_2 + \Sigma^2\delta_1 + \Sigma^2\delta_2)w + (\delta_2 + \Sigma^2\delta_1)$	6	6	6
Σ^2	Σ	$(\delta_2\Sigma + \delta_1 + \delta_2)w + (\delta_2\Sigma + \delta_1)$	5	5	5
Σ	Σ^2	$(\delta_2\Sigma^2 + \delta_1 + \delta_2)w + (\Sigma^2(\delta_1+\delta_2))$	6	6	6
Σ^2	1	$(\delta_2 + \Sigma(\delta_1+\delta_2))w + (\delta_2 + \delta_1)$	5	5	5

Arithmetic in $GF((2^2)^2)$ - Polynomial Square-Scaling

Optimized costs of polynomial square-scaling in $GF((2^2)^2)$.

Coefficients for Polynomial $GF((2^2)^2)$ Basis			XOR Gate Counts		
$\Pi = \delta_3 w + \delta_4$		$\varepsilon_1^2 \times \Pi = (\delta_1^2(\delta_3\Sigma^2+\delta_4)+\delta_2^2\delta_3)w+(\delta_1^2\Sigma(\delta_3+\delta_4)+\delta_2^2\delta_4)$	Pol. $GF(2^2)$		Norm. $GF(2^2)$
			$\Sigma = v$	$\Sigma = v^2$	
Σ	0	$(\delta_1^2+\delta_2^2\Sigma)w+(\delta_1^2\Sigma^2)$	4	4	4
Σ^2	0	$(\delta_1^2\Sigma+\delta_2^2\Sigma^2)w+(\delta_1^2)$	4	4	4
Σ	Σ	$(\delta_1^2\Sigma^2+\delta_2^2\Sigma)w+(\delta_2^2\Sigma)$	3	3	4
Σ^2	Σ^2	$(\delta_1^2+\delta_2^2\Sigma^2)w+(\delta_2^2\Sigma^2)$	4	3	3
Σ	1	$(\delta_2^2\Sigma)w+((\delta_1+\delta_2)^2)$	3	4	3
Σ^2	Σ	$(\delta_2^2\Sigma^2)w+(\Sigma(\delta_1+\delta_2)^2)$	3	3	4
Σ	Σ^2	$(\Sigma(\delta_1+\delta_2)^2)w+(\Sigma(\delta_1+\delta_2)^2+\delta_2^2)$	5	6	5
Σ^2	1	$(\Sigma^2(\delta_1+\delta_2)^2)w+(\Sigma^2(\delta_1+\delta_2)^2+\delta_2^2\Sigma)$	5	5	6

Arithmetic in $GF(((2^2)^2)^2)$

- We did not perform such low-level optimizations for $GF(((2^2)^2)^2)$
  - There are 128 possible polynomials $s(y)$ with trace of unity
  - Equivalently, there are 128 distinct values for Λ to consider
- Details omitted for the sake of time
  - See the full thesis or talk to me offline, I'll buy the coffee :-)

Counting Gates for $GF((((2^2)^2)^2)^2)$ Inversion

Assume a basis $[1, V]$, $[1, W]$, $[1, X]$, and $[1, Y]$ and coefficients $\Sigma = v$, $\Pi = (v+1)w + v$, $\Lambda = vwx + (w+v)$

- Inversion: 78 gates
- Multiplication: 81 gates
- Square-scale: 81 gates (none of our optimizations apply)

```
> F2 := GF(2);
> Poly2<V> := PolynomialRing(F2);
> P := V^2 + V + 1;
> F4<v> := ext<F2 | P>;
> Poly4<W> := PolynomialRing(F4);
> Q := W^2 + W + v;
> F16<w> := ext<F4 | Q>;
> Poly16<X> := PolynomialRing(F16);
> R := X^2 + X + ((v + 1)*w + v);
> F256<x> := ext<F16 | R>;
> Poly256<Y> := PolynomialRing(F256);
> S := Y^2 + Y + (v*w*x + (w + v));
> F6K<y> := ext<F256 | S>;
> gatesInv16(P, Q, R, S, v, ((v + 1)*w + v), \
(v*w*x + (w + v)), 1, v, 1, w, 1, x, 1, y);
398
```

Itoh-Tsujii Inversion

Theorem

Let $\alpha \in GF(q^k) \setminus \{0\}$ and $r = (q^k - 1)/(q - 1)$. Then, the multiplicative inverse of α can be computed as

$$\alpha^{-1} = (\alpha^r)^{-1} \alpha^{r-1},$$

where $\alpha^r, (\alpha^r)^{-1} \in GF(q)$.

- If $k = n \times m$, the number of multiplications and exponentiations in $GF((2^n)^m)$ and $GF(2^n)$ required to compute the inverse in $GF(2^k)$ is well defined.
- If $n = 8$ and $m = 2$ a combinational circuit for computing this inverse requires 5 multiplication, 2 squaring, and 1 scaling operation in $GF(2^8)$ (assuming a normal basis)
- This results in *at least* 552 XOR gates using the best known polynomial multiplication circuit for $GF(2^8)$

Itoh-Tsujii Inversion

Theorem

Let $\alpha \in GF(q^k) \setminus \{0\}$ and $r = (q^k - 1)/(q - 1)$. Then, the multiplicative inverse of α can be computed as

$$\alpha^{-1} = (\alpha^r)^{-1} \alpha^{r-1},$$

where $\alpha^r, (\alpha^r)^{-1} \in GF(q)$.

- If $k = n \times m$, the number of multiplications and exponentiations in $GF((2^n)^m)$ and $GF(2^n)$ required to compute the inverse in $GF(2^k)$ is well defined.
- If $n = 8$ and $m = 2$ a combinational circuit for computing this inverse requires 5 multiplication, 2 squaring, and 1 scaling operation in $GF(2^8)$ (assuming a normal basis)
- This results in *at least* 552 XOR gates using the best known polynomial multiplication circuit for $GF(2^8)$

Itoh-Tsujii Inversion

Theorem

Let $\alpha \in GF(q^k) \setminus \{0\}$ and $r = (q^k - 1)/(q - 1)$. Then, the multiplicative inverse of α can be computed as

$$\alpha^{-1} = (\alpha^r)^{-1} \alpha^{r-1},$$

where $\alpha^r, (\alpha^r)^{-1} \in GF(q)$.

- If $k = n \times m$, the number of multiplications and exponentiations in $GF((2^n)^m)$ and $GF(2^n)$ required to compute the inverse in $GF(2^k)$ is well defined.
- If $n = 8$ and $m = 2$ a combinational circuit for computing this inverse requires 5 multiplication, 2 squaring, and 1 scaling operation in $GF(2^8)$ (assuming a normal basis)
- This results in *at least* 552 XOR gates using the best known polynomial multiplication circuit for $GF(2^8)$

Itoh-Tsujii Inversion

Theorem

Let $\alpha \in GF(q^k) \setminus \{0\}$ and $r = (q^k - 1)/(q - 1)$. Then, the multiplicative inverse of α can be computed as

$$\alpha^{-1} = (\alpha^r)^{-1} \alpha^{r-1},$$

where $\alpha^r, (\alpha^r)^{-1} \in GF(q)$.

- If $k = n \times m$, the number of multiplications and exponentiations in $GF((2^n)^m)$ and $GF(2^n)$ required to compute the inverse in $GF(2^k)$ is well defined.
- If $n = 8$ and $m = 2$ a combinational circuit for computing this inverse requires 5 multiplication, 2 squaring, and 1 scaling operation in $GF(2^8)$ (assuming a normal basis)
- This results in *at least* 552 XOR gates using the best known polynomial multiplication circuit for $GF(2^8)$

Change Between Different Bases

Changing from a polynomial basis in $\mathbb{F}_1 = GF(2^{16})$ to a mixed basis in $\mathbb{F}_2 = GF((((2^2)^2)^2)^2)$ can be done in two steps:

1. Map from $\mathbb{F}_1$ to $\mathbb{F}_2$ using a homomorphism obtained by embedding $\mathbb{F}_2$ into $\mathbb{F}_1$
2. Change from a polynomial basis in $\mathbb{F}_2$ to the desired mixed basis in $\mathbb{F}_2$

Both transformations can be done with matrix multiplications **A** and **T**.

- We let Magma find **A** and $\mathbf{A}^{-1}$ by embedding $\mathbb{F}_2$ into $\mathbb{F}_1$
  - Future work will entail exhaustively finding all matrices **A** and $\mathbf{A}^{-1}$ (details omitted).
- We exhaustively considered all possible basis change matrices **T** and $\mathbf{T}^{-1}$

Change Between Different Bases

Changing from a polynomial basis in $\mathbb{F}_1 = GF(2^{16})$ to a mixed basis in $\mathbb{F}_2 = GF((((2^2)^2)^2)^2)$ can be done in two steps:

1. Map from $\mathbb{F}_1$ to $\mathbb{F}_2$ using a homomorphism obtained by embedding $\mathbb{F}_2$ into $\mathbb{F}_1$
2. Change from a polynomial basis in $\mathbb{F}_2$ to the desired mixed basis in $\mathbb{F}_2$

Both transformations can be done with matrix multiplications $\mathbf{A}$ and $\mathbf{T}$.

- We let Magma find $\mathbf{A}$ and $\mathbf{A}^{-1}$ by embedding $\mathbb{F}_2$ into $\mathbb{F}_1$
  - Future work will entail exhaustively finding all matrices $\mathbf{A}$ and $\mathbf{A}^{-1}$ (details omitted).
- We exhaustively considered all possible basis change matrices $\mathbf{T}$ and $\mathbf{T}^{-1}$

Choosing the Right Basis Representation

- We want to select the basis that minimizes both *inversion* and *changing bases*!
- For $GF(2^{16})$, there are $3 \times (2 \times 3) \times (8 \times 3) \times (128 \times 3) = 165888$ different basis combinations to choose from for $GF((((2^2)^2)^2)^2)$
- There are 4080 degree 16 irreducible polynomials over $GF(2)$
- We must consider a total of $165888 \times 4080 = 676823040$ possibilities to be complete
- Each output file for a degree 16 polynomial used to count the S-box gates and perform optimization is approximately 0.5GB in size, leading to about 2TB of data required to perform an exhaustive search

... we were time, storage space, and compute bound, so we only analyzed the smallest 21 degree 16 polynomials without performing optimizations

Choosing the Right Basis Representation

- We want to select the basis that minimizes both *inversion* and *changing bases*!
- For $GF(2^{16})$, there are $3 \times (2 \times 3) \times (8 \times 3) \times (128 \times 3) = 165888$ different basis combinations to choose from for $GF((((2^2)^2)^2)^2)$
- There are 4080 degree 16 irreducible polynomials over $GF(2)$
- We must consider a total of $165888 \times 4080 = 676823040$ possibilities to be complete
- Each output file for a degree 16 polynomial used to count the S-box gates and perform optimization is approximately 0.5GB in size, leading to about 2TB of data required to perform an exhaustive search

... we were time, storage space, and compute bound, so we only analyzed the smallest 21 degree 16 polynomials without performing optimizations

Choosing the Right Basis Representation

- We want to select the basis that minimizes both *inversion* and *changing bases*!
- For $GF(2^{16})$, there are $3 \times (2 \times 3) \times (8 \times 3) \times (128 \times 3) = 165888$ different basis combinations to choose from for $GF((((2^2)^2)^2)^2)$
- There are 4080 degree 16 irreducible polynomials over $GF(2)$
- We must consider a total of $165888 \times 4080 = 676823040$ possibilities to be complete
- Each output file for a degree 16 polynomial used to count the S-box gates and perform optimization is approximately 0.5GB in size, leading to about 2TB of data required to perform an exhaustive search

... we were time, storage space, and compute bound, so we only analyzed the smallest 21 degree 16 polynomials without performing optimizations

ASIC S-Box Implementations

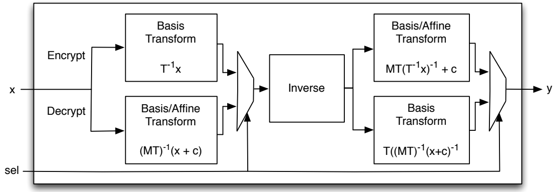
There are really two options for implementing an S-box circuit:

1. Implement the forward and inverse S-boxes separately
   - Redundant inversion circuits
2. Merge the forward and inverse S-box circuits together
   - Share an inverter at the cost of some extra multiplexors

ASIC S-Box Implementations

There are really two options for implementing an S-box circuit:

1. Implement the forward and inverse S-boxes separately
   - Redundant inversion circuits
2. Merge the forward and inverse S-box circuits together
   - Share an inverter at the cost of some extra multiplexors



Changing Gears...

- To compute the multiplicative inverse of an element in $GF(2^{16})$ using the isomorphic field $GF((((2^2)^2)^2)^2)$.
  1. Change from polynomial basis in $GF(2^{16})$ to mixed basis in $GF((((2^2)^2)^2)^2)$
  2. Compute the inverse
  3. Change from mixed basis in $GF((((2^2)^2)^2)^2)$ to polynomial basis in $GF(2^{16})$
- There are at least two more opportunities for area improvement
  1. Reduce the logic required to change bases (and perform the affine transformation) - these are both linear operations
  2. Reduce the logic required to compute the multiplicative inverse - this is a nonlinear operation

Changing Gears...

- To compute the multiplicative inverse of an element in $GF(2^{16})$ using the isomorphic field $GF((((2^2)^2)^2)^2)$.
  1. Change from polynomial basis in $GF(2^{16})$ to mixed basis in $GF((((2^2)^2)^2)^2)$
  2. Compute the inverse
  3. Change from mixed basis in $GF((((2^2)^2)^2)^2)$ to polynomial basis in $GF(2^{16})$
- There are at least two more opportunities for area improvement
  1. Reduce the logic required to change bases (and perform the affine transformation) - these are both linear operations
  2. Reduce the logic required to compute the multiplicative inverse - this is a nonlinear operation

Linear Circuits as Straight Line Programs

- Consider the following system of linear equations over $GF(2)$:

$$
\begin{aligned}
y_1 &= \alpha_{1,1}x_1 + \alpha_{1,2}x_2 + \cdots + \alpha_{1,n} \\
y_2 &= \alpha_{2,1}x_1 + \alpha_{2,2}x_2 + \cdots + \alpha_{2,n} \\
\ldots & \\
y_m &= \alpha_{m,1}x_1 + \alpha_{m,2}x_2 + \cdots + \alpha_{m,n},
\end{aligned}
$$

- An SLP is a series of lines of the form $u := v + w$ where v and w are *input variables* x and some u are *output variables* y.
- Minimizing linear circuits (i.e. circuits with only XOR gates) is equivalent to finding the shortest equivalent *straight line program* (SLP)

Optimal Straight Line Programs

- An SLP is *cancellation-free* (i.e. completely factored) if for every line $u := v + w$, v and w do not share any common variables
- Optimal cancellation-free SLPs do not always yield optimal SLPs!

$$\begin{aligned} y_0 &:= x_1 + x_2 \\ y_1 &:= x_1 + x_2 + x_3 \\ y_2 &:= x_1 + x_2 + x_3 + x_4 \\ y_3 &:= x_2 + x_3 + x_4 \end{aligned}$$

Original system

$$\begin{aligned} u_1 &:= x_1 + x_2 \ (y_0) \\ u_2 &:= u_1 + x_3 \ (y_1) \\ u_3 &:= u_2 + x_4 \ (y_2) \\ v_1 &:= x_2 + x_3 \\ u_4 &:= v_1 + x_4 \ (y_3) \end{aligned}$$

Factored SLP

$$\begin{aligned} u_1 &:= x_1 + x_2 \ (y_0) \\ u_2 &:= u_1 + x_3 \ (y_1) \\ u_3 &:= u_2 + x_4 \ (y_2) \\ u_4 &:= u_3 + x_1 \ (y_3) \end{aligned}$$

Unfactored SLP

Finding an optimal length SLP is an NP-hard problem

Optimal Straight Line Programs

- An SLP is *cancellation-free* (i.e. completely factored) if for every line $u := v + w$, v and w do not share any common variables
- Optimal cancellation-free SLPs do not always yield optimal SLPs!

$$\begin{aligned} y_0 &:= x_1 + x_2 \\ y_1 &:= x_1 + x_2 + x_3 \\ y_2 &:= x_1 + x_2 + x_3 + x_4 \\ y_3 &:= x_2 + x_3 + x_4 \end{aligned}$$

Original system

$$\begin{aligned} u_1 &:= x_1 + x_2 \,(y_0) \\ u_2 &:= u_1 + x_3 \,(y_1) \\ u_3 &:= u_2 + x_4 \,(y_2) \\ v_1 &:= x_2 + x_3 \\ u_4 &:= v_1 + x_4 \,(y_3) \end{aligned}$$

Factored SLP

$$\begin{aligned} u_1 &:= x_1 + x_2 \,(y_0) \\ u_2 &:= u_1 + x_3 \,(y_1) \\ u_3 &:= u_2 + x_4 \,(y_2) \\ u_4 &:= u_3 + x_1 \,(y_3) \end{aligned}$$

Unfactored SLP

Finding an optimal length SLP is an NP-hard problem

Optimal Straight Line Programs

- An SLP is *cancellation-free* (i.e. completely factored) if for every line $u := v + w$, v and w do not share any common variables
- Optimal cancellation-free SLPs do not always yield optimal SLPs!

$$\begin{aligned} y_0 &:= x_1 + x_2 \\ y_1 &:= x_1 + x_2 + x_3 \\ y_2 &:= x_1 + x_2 + x_3 + x_4 \\ y_3 &:= x_2 + x_3 + x_4 \end{aligned}$$

Original system

$$\begin{aligned} u_1 &:= x_1 + x_2 \,(y_0) \\ u_2 &:= u_1 + x_3 \,(y_1) \\ u_3 &:= u_2 + x_4 \,(y_2) \\ v_1 &:= x_2 + x_3 \\ u_4 &:= v_1 + x_4 \,(y_3) \end{aligned}$$

Factored SLP

$$\begin{aligned} u_1 &:= x_1 + x_2 \,(y_0) \\ u_2 &:= u_1 + x_3 \,(y_1) \\ u_3 &:= u_2 + x_4 \,(y_2) \\ u_4 &:= u_3 + x_1 \,(y_3) \end{aligned}$$

Unfactored SLP

Finding an optimal length SLP is an NP-hard problem

Heuristic-Based Optimizations

To date, the most effective heuristic-based techniques are:

- Christof Paar's greedy factorization technique ✓
- David Canright's exhaustive factorization ✓
- Daniel Bernstein's Bos-Coster approach ✗(generated longer SLPs)
- Rene Peralta and Joan Boyar's predicate distance-based heuristic ✓
- *Exact* optimizations with SAT-based proofs ✗(not in scope)

Heuristic-Based Optimizations

To date, the most effective heuristic-based techniques are:

- Christof Paar's greedy factorization technique ✓
- David Canright's exhaustive factorization ✓
- Daniel Bernstein's Bos-Coster approach ✗(generated longer SLPs)
- Rene Peralta and Joan Boyar's predicate distance-based heuristic ✓
- *Exact* optimizations with SAT-based proofs ✗(not in scope)

Heuristic-Based Optimizations

To date, the most effective heuristic-based techniques are:

- Christof Paar’s greedy factorization technique ✓
- David Canright’s exhaustive factorization ✓
- Daniel Bernstein’s Bos-Coster approach ✗(generated longer SLPs)
- Rene Peralta and Joan Boyar’s predicate distance-based heuristic ✓
- *Exact* optimizations with SAT-based proofs ✗(not in scope)

Heuristic-Based Optimizations

To date, the most effective heuristic-based techniques are:

- Christof Paar’s greedy factorization technique ✓
- David Canright’s exhaustive factorization ✓
- Daniel Bernstein’s Bos-Coster approach ✗(generated longer SLPs)
- Rene Peralta and Joan Boyar’s predicate distance-based heuristic ✓
- *Exact* optimizations with SAT-based proofs ✗(not in scope)

Heuristic-Based Optimizations

To date, the most effective heuristic-based techniques are:

- Christof Paar's greedy factorization technique ✓
- David Canright's exhaustive factorization ✓
- Daniel Bernstein's Bos-Coster approach ✗(generated longer SLPs)
- Rene Peralta and Joan Boyar's predicate distance-based heuristic ✓
- *Exact* optimizations with SAT-based proofs ✗(not in scope)

Heuristic-Based Optimizations

To date, the most effective heuristic-based techniques are:

- Christof Paar's greedy factorization technique ✓
- David Canright's exhaustive factorization ✓
- Daniel Bernstein's Bos-Coster approach ✗(generated longer SLPs)
- Rene Peralta and Joan Boyar's predicate distance-based heuristic ✓
- *Exact* optimizations with SAT-based proofs ✗(not in scope)

Performance Comparison

Matrix Size	*Algorithm*	*Average XOR Count*
16×16	Paar Greedy Factorization	58.14
16×16	Boyar-Peralta Optimization #1	50.09
16×16	Boyar-Peralta Optimization #2	50.11
16×16	Boyar-Peralta Optimization #3	50.09
16×16	Boyar-Peralta Optimization #4	53.1
16×16	Bernstein Optimization	85.0
32×16	Paar Greedy Factorization	103.89
32×16	Boyar-Peralta Optimization #1	83.22
32×16	Boyar-Peralta Optimization #2	83.27
32×16	Boyar-Peralta Optimization #3	83.22
32×16	Boyar-Peralta Optimization #4	88.15
32×16	Bernstein Optimization	146.66

Parallel Implementations of Linear Optimization Techniques

The most appealing candidates for improvement are the exhaustive factorization and Boyar-Peralta techniques:

- The factorization search tree can grow exponentially with the input matrix dimensions
- Computing new predicate distances in Boyar and Peralta's technique is the bottleneck

Parallel Exhaustive Factorization Performance

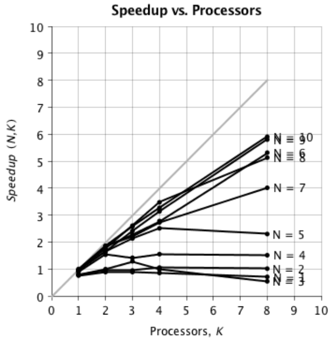
Pattern: recursive fork/join to distribute the search tree traversal among all available cores.

$m \times n$	*Sequential Time (msec)*	*Parallel Time (msec)*
7×7	16	15
8×8	16	17
9×9	188	79
10×10	4625	1703
12×6	9	9
14×7	247	93

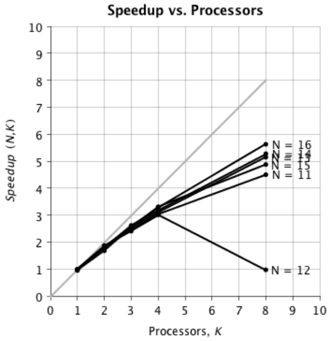
Note: difficult to collect more data

Parallel Boyar-Peralta Performance

Pattern: result-parallelism to evenly distribute the predicate distance computations among all possible cores.



(a) $n \times n$ matrices



(b) $2n \times n$ matrices

Putting Everything Together

Before applying all of these techniques together, let’s revisit the AES S-box

A Brief History of the AES S-Box

- (2001) Rudra et al. use the composite field $GF((2^4)^2)$ to reduce the area of the AES S-Box at CHES 2001.
- (2001) Satoh et al. build upon this result by using the composite field $GF(((2^2)^2)^2)$ with a tower of polynomial bases.
- (2005) Mentens et al. show that the previous work can be improved with a different selection of irreducible polynomial $r(x)$.
- (2005) Canright provides an exhaustive search of all 432 basis representations of $GF(((2^2)^2)^2)$ with low-level arithmetic optimizations.
- (2008) Nikova et al. use the composite field $GF((2^4)^2)$ with a tower of normal bases to show that the selection of an optimal normal basis in $GF(2^4)$ can yield lower GEs than Canright's construction (in some cases).
- (2011-2013) Boyar and Peralta used their novel combinational logic minimization techniques for linear and nonlinear circuits to improve Canright's construction to 32 AND and 83 XOR/XNOR gates.
- ... and our work

A Brief History of the AES S-Box

- (2001) Rudra et al. use the composite field $GF((2^4)^2)$ to reduce the area of the AES S-Box at CHES 2001.
- (2001) Satoh et al. build upon this result by using the composite field $GF(((2^2)^2)^2)$ with a tower of polynomial bases.
- (2005) Mentens et al. show that the previous work can be improved with a different selection of irreducible polynomial $r(x)$.
- (2005) Canright provides an exhaustive search of all 432 basis representations of $GF(((2^2)^2)^2)$ with low-level arithmetic optimizations.
- (2008) Nikova et al. use the composite field $GF((2^4)^2)$ with a tower of normal bases to show that the selection of an optimal normal basis in $GF(2^4)$ can yield lower GEs than Canright's construction (in some cases).
- (2011-2013) Boyar and Peralta used their novel combinational logic minimization techniques for linear and nonlinear circuits to improve Canright's construction to 32 AND and 83 XOR/XNOR gates.
- ... and our work

A Brief History of the AES S-Box

- (2001) Rudra et al. use the composite field $GF((2^4)^2)$ to reduce the area of the AES S-Box at CHES 2001.
- (2001) Satoh et al. build upon this result by using the composite field $GF(((2^2)^2)^2)$ with a tower of polynomial bases.
- (2005) Mentens et al. show that the previous work can be improved with a different selection of irreducible polynomial $r(x)$.
- (2005) Canright provides an exhaustive search of all 432 basis representations of $GF(((2^2)^2)^2)$ with low-level arithmetic optimizations.
- (2008) Nikova et al. use the composite field $GF((2^4)^2)$ with a tower of normal bases to show that the selection of an optimal normal basis in $GF(2^4)$ can yield lower GEs than Canright's construction (in some cases).
- (2011-2013) Boyar and Peralta used their novel combinational logic minimization techniques for linear and nonlinear circuits to improve Canright's construction to 32 AND and 83 XOR/XNOR gates.
- ... and our work

A Brief History of the AES S-Box

- (2001) Rudra et al. use the composite field $GF((2^4)^2)$ to reduce the area of the AES S-Box at CHES 2001.
- (2001) Satoh et al. build upon this result by using the composite field $GF(((2^2)^2)^2)$ with a tower of polynomial bases.
- (2005) Mentens et al. show that the previous work can be improved with a different selection of irreducible polynomial $r(x)$.
- (2005) Canright provides an exhaustive search of all 432 basis representations of $GF(((2^2)^2)^2)$ with low-level arithmetic optimizations.
- (2008) Nikova et al. use the composite field $GF((2^4)^2)$ with a tower of normal bases to show that the selection of an optimal normal basis in $GF(2^4)$ can yield lower GEs than Canright's construction (in some cases).
- (2011-2013) Boyar and Peralta used their novel combinational logic minimization techniques for linear and nonlinear circuits to improve Canright's construction to 32 AND and 83 XOR/XNOR gates.
- ... and our work

A Brief History of the AES S-Box

- (2001) Rudra et al. use the composite field $GF((2^4)^2)$ to reduce the area of the AES S-Box at CHES 2001.
- (2001) Satoh et al. build upon this result by using the composite field $GF(((2^2)^2)^2)$ with a tower of polynomial bases.
- (2005) Mentens et al. show that the previous work can be improved with a different selection of irreducible polynomial $r(x)$.
- (2005) Canright provides an exhaustive search of all 432 basis representations of $GF(((2^2)^2)^2)$ with low-level arithmetic optimizations.
- (2008) Nikova et al. use the composite field $GF((2^4)^2)$ with a tower of normal bases to show that the selection of an optimal normal basis in $GF(2^4)$ can yield lower GEs than Canright's construction (in some cases).
- (2011-2013) Boyar and Peralta used their novel combinational logic minimization techniques for linear and nonlinear circuits to improve Canright's construction to 32 AND and 83 XOR/XNOR gates.
- ... and our work

A Brief History of the AES S-Box

- (2001) Rudra et al. use the composite field $GF((2^4)^2)$ to reduce the area of the AES S-Box at CHES 2001.
- (2001) Satoh et al. build upon this result by using the composite field $GF(((2^2)^2)^2)$ with a tower of polynomial bases.
- (2005) Mentens et al. show that the previous work can be improved with a different selection of irreducible polynomial $r(x)$.
- (2005) Canright provides an exhaustive search of all 432 basis representations of $GF(((2^2)^2)^2)$ with low-level arithmetic optimizations.
- (2008) Nikova et al. use the composite field $GF((2^4)^2)$ with a tower of normal bases to show that the selection of an optimal normal basis in $GF(2^4)$ can yield lower GEs than Canright's construction (in some cases).
- (2011-2013) Boyar and Peralta used their novel combinational logic minimization techniques for linear and nonlinear circuits to improve Canright's construction to 32 AND and 83 XOR/XNOR gates.
- ... and our work

A New Look at the AES S-Box

- A set of basis change matrices different from the ones used by Canright
- Parameters:
  - $GF(2^8)$ polynomial: $p(v) = v^8 + v^4 + v^3 + v + 1$ (AES field polynomial)
  - Tower of normal bases with the same coefficients Σ and Π as Canright: $\Sigma = v^2$ and $\Pi = vw$

$$\mathbf{T} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \quad \mathbf{T}^{-1} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Implication: Canright didn't explore all possible isomorphisms between standard AES representation and mixed basis composite representation - can we still do better for the AES S-box?

AES S-Box Alternative

If we use the $GF(2^8)$ field polynomial $s(v) = v^8 + v^6 + v^5 + v^4 + v^2 + v + 1$, a possible S-box with low area requirements (103 XOR and 36 AND gates without logic optimizations similar to Canright) is as follows:

$$S(x) = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{pmatrix}^{-1} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Other Suitable Power Mappings

- What if we don't want to use the inverse mapping?
- There are other power mapping exponents d (and inverses d^{-1}) with high nonlinearity (112) and low differential uniformity (4):
  - $d = 127$, $d^{-1} = 253$
  - $d = 191$, $d^{-1} = 251$
  - $d = 223$, $d^{-1} = 247$
  - $d = 239$, $d^{-1} = 239$
  - $d = 247$, $d^{-1} = 223$
  - $d = 251$, $d^{-1} = 191$
  - $d = 253$, $d^{-1} = 127$
  - $d = 254$, $d^{-1} = 254$
- We did not explore implementations of S-boxes based on these power mappings

16-Bit S-Box Results

Our best candidate has the following parameters:

- Field polynomial: $t(v) = v^{16} + v^5 + v^3 + v + 1$
- Basis sets: $[1, V]$, $[1, W]$, $[1, X]$, and $[Y^{256}, Y]$
- Coefficients: $\Sigma = v$, $\Pi = vw + v$, and $\Lambda = (vw + v)x + w$

- Our (unoptimized) version of this S-box requires 1238 XOR gates and 144 AND gates
- This can clearly be improved by applying Boyar and Peralta's optimization techniques

16-Bit S-Box Definition

$$S(x) = \begin{pmatrix}
0&0&1&0&0&0&0&1&0&0&1&1&1&1&1&0\\
1&1&0&0&0&0&0&1&0&1&1&0&1&0&1&0\\
1&1&0&0&1&0&1&1&0&1&0&1&0&0&1&1\\
1&1&1&0&0&0&1&0&0&1&1&0&0&0&0&0\\
1&1&0&0&0&1&1&0&0&1&1&1&1&0&1&1\\
0&1&0&0&0&0&1&1&0&1&1&1&1&1&0&1\\
0&0&1&0&1&0&1&0&1&1&0&0&1&1&0&0\\
1&0&1&1&1&0&1&1&0&0&0&1&0&1&1&1\\
0&1&0&0&0&0&0&0&1&0&0&1&1&1&0&1\\
1&0&1&1&0&0&0&1&0&0&1&0&1&0&0&0\\
1&0&1&0&0&1&1&1&0&0&1&1&0&1&0&0\\
1&0&1&1&1&0&1&1&1&1&0&1&1&0&0&1\\
1&0&1&0&0&1&0&1&1&0&0&1&0&0&0&1\\
0&1&0&0&0&1&1&1&1&0&0&0&0&0&0&1\\
1&0&0&0&1&1&0&1&0&1&1&1&1&0&0&0\\
1&1&0&1&0&1&1&0&1&0&0&1&1&0&0&0
\end{pmatrix}
\begin{pmatrix}
x_{15}\\ x_{14}\\ x_{13}\\ x_{12}\\ x_{11}\\ x_{10}\\ x_9\\ x_8\\ x_7\\ x_6\\ x_5\\ x_4\\ x_3\\ x_2\\ x_1\\ x_0
\end{pmatrix}^{-1}
+
\begin{pmatrix}
0\\1\\0\\0\\0\\1\\0\\1\\1\\0\\1\\1\\0\\1\\1\\1
\end{pmatrix}$$

16-Bit S-Box Basis Change Matrices

$$\mathbf{T}^{-1} = \begin{pmatrix}
1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1
\end{pmatrix}$$

Future Work

1. AES S-box
   - Exhaustively generate all basis change matrices
   - Use Canright's inversion circuit counts to determine the total cost of the S-box unoptimized
   - Apply Boyar and Peralta's technique to see if the area and depth can be reduced below the current best-known result
2. Apply combinational logic minimization techniques to other cryptographic problems:
   - Low weight and depth binary polynomial multiplication
   - Efficient 2-party secure computation protocols
   - NAND/NOT round modifications for CMOS-efficient nonlinear circuits
3. Galois field arithmetic library
   - Support flexible polynomial and normal basis arithmetic for arbitrary composite fields and any number of extensions (already started!)
   - Add symbolic evaluation to programmatically search for and eliminate common subexpressions in all relevant arithmetic expressions

Future Work

1. AES S-box
   - Exhaustively generate all basis change matrices
   - Use Canright's inversion circuit counts to determine the total cost of the S-box unoptimized
   - Apply Boyar and Peralta's technique to see if the area and depth can be reduced below the current best-known result
2. Apply combinational logic minimization techniques to other cryptographic problems:
   - Low weight and depth binary polynomial multiplication
   - Efficient 2-party secure computation protocols
   - NAND/NOT round modifications for CMOS-efficient nonlinear circuits
3. Galois field arithmetic library
   - Support flexible polynomial and normal basis arithmetic for arbitrary composite fields and any number of extensions (already started!)
   - Add symbolic evaluation to programmatically search for and eliminate common subexpressions in all relevant arithmetic expressions

Future Work

1. AES S-box
   - Exhaustively generate all basis change matrices
   - Use Canright's inversion circuit counts to determine the total cost of the S-box unoptimized
   - Apply Boyar and Peralta's technique to see if the area and depth can be reduced below the current best-known result
2. Apply combinational logic minimization techniques to other cryptographic problems:
   - Low weight and depth binary polynomial multiplication
   - Efficient 2-party secure computation protocols
   - NAND/NOT round modifications for CMOS-efficient nonlinear circuits
3. Galois field arithmetic library
   - Support flexible polynomial and normal basis arithmetic for arbitrary composite fields and any number of extensions (already started!)
   - Add symbolic evaluation to programmatically search for and eliminate common subexpressions in all relevant arithmetic expressions

Wrapping Up

Special thanks to:

- Staszek, Marcin, Alan, and Michael
- RIT, GCCIS, and Harris RF
- Mats Rynge and the OSG
- David Canright, Rene Peralta (NIST), Christof Paar, Nele Mentens
- Family and friends

Questions?

Fire away!