Thesis Progress Report

Christopher A. Wood

April 15, 2013

Agenda

1. Generating Polynomials
2. Finite Field Arithmetic in Software
3. Measuring Boolean Functions

Testing for Irreducibility

Algorithm 1 Testing for polynomial irreducibility

Require: A prime p and *monic* polynomial $f(x)$.
Ensure: YES if $f(x)$ is irreducible, NO otherwise.

$g(x) \leftarrow x$ ▷ The smallest possible factor
for $i = 1 \to \lfloor m/2 \rfloor$ **do**
    $g(x) \leftarrow g(x)^p \mod f(x)$
    $d(x) \leftarrow \gcd\{f(x), g(x) - x\}$
    if $d(x) \neq 1$ **then**
        NO
    end if
end forreturn YES

Order-Based Test for Irreducibility

Algorithm 2 Testing for polynomial irreducibility

Require: An integer k and *monic* polynomial $f(x)$.

Ensure: YES if $f(x)$ is irreducible, NO otherwise.

1: **for** $u(x) = x \to x^{k-1} + x^{k-2} + \cdots + x + 1$ **do**

2: **if** $u(x)$ is a generator of $GF(2^k)$ defined by $f(x)$ **then return** YES

3: **end if**

4: **end forreturn** NO

Extending to Primitive Polynomials

- A monic irreducible polynomial $f(x)$ is primitive in $GF(2^k)$ if its order is equal to $2^k - 1$.
- Alternatively, a monic irreducible polynomial is primitive if x is a generator of $GF(2^k)$ defined by $f(x)$.
    - Check to see that x is in the list of generators.
- Verify the correctness by counting:
    - Primitive polynomial count: $a_q(n) = \frac{\phi(q^n - 1)}{n}$
    - Irreducible polynomial count: $L_q(n) = \frac{1}{n}\sum_{d|n} \mu(n/d) q^d$

Extending the algorithms to composite fields

Algorithm 3 Testing for polynomial irreducibility

Require:

Require: Positive integers n and m, an irreducible polynomial $p(x)$ in $GF(2^n)$, and candidate polynomial $q(y)$ in $GF((2^n)^m)$.

Ensure: YES if $f(x)$ is irreducible, NO otherwise.

1: **for** $u(y) = y \rightarrow y^{m-1} + (x^{n-1} + \cdots + x + 1)y^{m-2} + \cdots + (x^{n-1} + \cdots + x + 1)y + (x^{n-1} + \cdots + x + 1)$ **do**

2: **if** $u(y)$ is a generator of $GF((2^n)^m)$ defined by $p(x)$ and $q(y)$ **then return** YES

3: **end if**

4: **end forreturn** NO

A similar test for primitivity holds here (check to see if y is a generator of $GF((2^n)^m)$)

Multiplicative Inverse Calculations

- $GF(2^{16})$ is relatively small compared to other fields (e.g. ECC)
- Yet, we should strive for two types of software implementations under two assumptions:
  1. Constant time computations are required
  2. No memory constraints and no possibility for side-channel attacks

Software Review

- Composite field library
- Polynomial generation
- Isomorphic function generation
- 16-bit inverse calculation
- Boolean function analysis

Cryptographically Significant S-box Metrics

- Linear and differential approximations (tables will be large)
- Boolean differential uniformity
- Boolean function nonlinearity
- Algebraic immunity
- Correlation immunity
- Resiliency

Questions to Answer

- How many irreducible and primitive polynomials exist for extension fields?
- Can we optimize the isomorphic function generation algorithm?
- Which cryptographic properties are most important? Should nonlinearity be more critical than correlation immunity?
- How can Boolean functions be efficiently implemented in software? They generally have no algebraic constructions. Are large sequences of bitwise operations or LUTS the only ways?
- Without using an affine transformation, how can we make the S-box expression *algebraically complex*?

Action Items

- Literature survey of software optimizations for Galois field arithmetic and AES-specific operations
- Write the introduction chapter for thesis (S-box motivation, attacks, etc)
- Exhaustive list of all polynomials $P(x)$, $Q(y)$, and $R(z)$ and the corresponding list of all transformation matrices (using OSG!)
- Composite-field implementation of 16-bit inverse in software and hardware
- Preliminary Boolean function construction code

Next meeting: **4/29/13**