Implementation of Arithmetic Operations over GF(2^8) and Multivariate Key Pre-distribution Scheme

NYIT Mobile Device and Wireless Network Security REU

August 18, 2013

Cory Pruce
cpruce@students.pitzer.edu
Pitzer College
Claremont Consortium

Farshid Delgosha
fdelgosh@nyit.edu
Electrical Engineering and Computing Sciences
New York Institute of Technology

1 Abstract:

Finite fields, or Galois fields, hold an important role in a number of areas including cryptography, error-correcting code, and number theory. For example, discrete-log based cryptosystems, Diffie-Hellman key exchange scheme, and Reed-Solomon codes are among some of the cryptographic and error-correcting constructions that utilize finite field arithmetic. The multivariate key pre-distribution scheme (MKPS) is designed as a practical security protocol for networks of resource-limited devices, particularly wireless sensor networks. The purpose of this research is to produce a resource-conscious implementation of MKPS using the finite field GF(2^8) as a foundation.

2 Introduction:

The advent of wireless networks has greatly facilitated the human race's ability to observe, monitor, and interpret data from multiple sources in different locations. Wireless sensor networks are used to record the physical world in several areas such as monitoring certain chemicals, temperature, illumination, and sound waves. However, often is the case that these sensor nodes will be vulnerable to destruction, capture, or obsolescence and therefore are required to be replaceable in order to minimize production costs and any loss. This, in effect, constrains the memory limit and computational power of each node. Since security and efficiency are inversely proportional, this produces a problem for wireless sensor networks which are located in a hostile environment or contain sensitive data. The multivariate key pre-distribution scheme remedies this problem by creating several sets of polynomials tailored to the size of the node field with the intention of forcing the possibility of recovering the necessary shares of said polynomials infeasible. Additionally, the version of MKPS produced from this research opportunity utilizes GF(256), allowing use of single byte elements to be conveniently used and thus further optimizing the memory requirement for the protocol.

3 GF(2^w):

Finite field arithmetic has efficacy in several subjects. In this instance, cryptographic use is the application. To understand the concept of a finite field, we must first clarify a few foundational definitions:

DEFINITION 3.1 *A **binary operator** is a given function Δ such that $X \ \Delta \ X \rightarrow Y$; the binary operator Δ takes two elements from set X and outputs an element from set Y.*[1]

BRANCHES (Binary Operator) 3.1
1. A binary operator Δ is a commutative function iff (if and only if) $x \ \Delta \ y = y \ \Delta \ x, \forall x, y \in X$.

EXAMPLES (Binary Operator) 3.1
1. Addition, +
2. Subtraction, -
3. Multiplication, ·
4. Division, ÷

Out of these, only addition and multiplication are commutative binary operators. Binary operators, in conjunction with sets, can contain special characteristics:

DEFINITION 3.2 *A **group**, denoted* $(\mathbb{G}, \Delta)$*, is a set* $\mathbb{G}$ *that, paired with a binary operator* Δ*, satisfies the following qualities:*[1]
Closure: For all x, y $\in \mathbb{G}$, x Δ y $\in \mathbb{G}$
Identity Element: A unique element $i \in \mathbb{G}$ exists that satisfies for all $x \in \mathbb{G}$, $x \Delta i = i \Delta x = x$.
Inverse Element: For a given element x in the group $\mathbb{G}$, there exists a unique element $x' \in \mathbb{G}$ such that $x \Delta x' = x' \Delta x = i$.
Associativity: For all $x, y, z \in \mathbb{G}$, $x \Delta (y \Delta z) = (x \Delta y) \Delta z$.

NOTATION (Group) 3.1
1. Given a group ($\mathbb{G}, \Delta$), an arbitrary element $x \in \mathbb{G}$ and a positive integer n, we define

$$x^n = \underbrace{x \Delta x \Delta \cdots \Delta x}_{\text{n times}}$$

By this definition, we allow $x^0 = i$, where i is the identity element.

BRANCHES (Group) 3.2
1. A group ($\mathbb{G}, \Delta$) that also holds the commutative property is called an *abelian* or *commutative group.*
2. A group ($\mathbb{G}, \Delta$) is called *cyclic* iff there exists $g \in \mathbb{G}$ such that $\mathbb{G} = \{g^0, g^1, ...\}$ (every element of the group is some power of g). When this quality is satisfied, g is said to be the $generator$ of the cyclic group $\mathbb{G}$.

EXAMPLES (Group) 3.2
1. The set of all integers $\mathbb{Z}$ with addition creates an abelian group ($\mathbb{Z}, +,$).
2. The rational numbers (those represented in fraction form) excluding 0 also make an abelian group with multiplication ($\mathbb{Q}/\{0\}, \cdot$).
3. The set of all Rubik's Cube moves $\mathbb{G}$ appended together also form a group ($\mathbb{G}, \cdot$).

From here we can build the idea of a field:

DEFINITION 3.3: *A **field** is a set* $\mathbb{F}$ *that, coupled with the binary operators + and ·, satisfies the properties:* [1]
Zero Element: ($\mathbb{F}, +$) with 0 as the identity element forms an abelian group.
Unit Element: ($\mathbb{F}/\{0\}, \cdot$) with 1 as the identity element forms an abelian group.
Distributive Property: Multiplication · is distributive over addition + such that for all $x, y, z \in \mathbb{F}$, $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.

NOTATION (Field) 3.2:
1. Given a field ($\mathbb{F}, +, \cdot$), the additive inverse of an element $x \in \mathbb{F}$ is represented as -x.
2. In the same field, the multiplicative inverse of an arbitrary $x \in \mathbb{F}/\{0\}$ is labeled x^{-1}.
3. Subtraction and division are defined as x - y = x + $(-y)$ and $x \div y$ = $x \cdot y^{-1}$.
4. A finite field of order p and p^n is denoted as $\mathbb{Z}/p\mathbb{Z}$ or GF(p) and $\mathbb{Z}/p^n\mathbb{Z}$ or GF(p^n), respectfully.

BRANCHES (Field) 3.3:
1. A field $\mathbb{F}$ with a finite amount of elements is called a *finite field*, or *Galois field.* The cardinality, the number of elements, of the field is called the field's *order*.

EXAMPLES (Field) 3.3:
1. The complex numbers $\mathbb{C}$ along with regular addition and multiplication are associative, commutative, distributive, and contain the required unique elements; therefore, $\mathbb{C}$ is a field.
2. The rationals and the reals both form a field each along with ordinary addition and multiplication.
3. The field $\mathbb{Z}/5\mathbb{Z} = \{0, 1, 2, 3, 4\}$ is a finite field of order 5. The operations modulo-5 addition $\oplus$ and modulo-5 multiplication $\otimes$ make the field possible:

$$x \oplus y = (x + y) \bmod 5 \qquad x \otimes y = (x \cdot y) \bmod 5$$

The finite field $\mathbb{Z}/5\mathbb{Z}$ along with the operators $\oplus$ and $\otimes$ can be described as:

$\oplus$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

$\otimes$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

The order of a finite field cannot be any integer, however:

THEOREM 3.1.
Every order of a finite field is in the form p^n, where p is a prime integer and n is a positive integer. Moreover, for a given prime p and a positive integer n, a finite field of order p^n exists. [1]

In cryptography, the finite fields GF(p) and GF(2^n) are most useful. In this research project, GF(2^8) was the foundation for the elements due to its convenient representation in single bytes. Thus, we will focus on GF(2^n).

Elements of GF(2^n) are represented by binary polynomials of degree at most n - 1. The addition and subtraction operations in this type of field are equivalent to adding their polynomial representations, in which the coefficients are XORed. However, multiplication is not as straightforward using the polynomial representations. Optimizing the multiplication function is ideal because it is the root of two necessary functions in finding inverses, the division and Extended Euclidean algorithms. First, we must start with element representations:

DEFINITION 3.4: *A **binary polynomial** $f(x)$ of degree m is*

$$f(x) = x^m + f_{m-1}x^{m-1} + \cdots + f_1 x + f_0,$$

where $f_i \in \{0,1\} \forall i = 0, 1, ..., m-1$.[1]

Conveniently, we can also represent the polynomial in binary form using the coefficients:

$$1 f_{m-1} \cdots f_1 f_0$$

f_0 represents the least significant bit (LSB) whereas the most significant (MSB) will always be 1. On top of this, the amount of memory required to fit a binary polynomial f is its bit length, m + 1.

EXAMPLE (Polynomial Addition) 3.4: Consider the following two polynomials.

$$f(x) = x^2 + x + 1 \quad g(x) = x^3 + x$$

The addition, or subtraction, of these two polynomials is the XOR operation of their coefficients:

$f(x)$	0111
$g(x)$	1010
$f(x) + g(x)$	1101 +

The result 1101 is equal to $x^3 + x^2 + 1$, which is what the polynomial addition of $f(x)$ and $g(x)$ would have produced.

EXAMPLE (Polynomial Multiplication) 3.5: Now consider their product:

$$\begin{aligned} f(x) \cdot g(x) &= x^5 + x^4 + (1+1)x^3 + x^2 + x \\ &= x^5 + x^4 + x^2 + x \end{aligned}$$

In order to understand polynomial multiplication, we must first understand monomial multiplication. For a given integer $l \geq 0$, the product of multiplying the monomial x^l by a polynomial $f(x)$ is

$$x^l \cdot f(x) = x^{m+l} + f_{m-1}x^{m+l-1} + \cdots + f_1 x^{l+1} + f_0 x^l.$$

Here, the binary representation of this product is

$$1 f_{m-1} \cdots f_1 f_0 \underbrace{0 \cdots 0}_{l \text{ zeros}}.$$

From this example, we can observe that multiplying a polynomial by a monomial merely shifts to the left by the power of the monomial. Polynomial multiplication is not too much different. Now, in addition to $f(x)$, consider the polynomial:

$$g(x) = x^k + g_{k-1}x^{l-1} + \cdots + g_1x + g_0.$$

Due to the distributive property of multiplication over addition, polynomial multiplication is no more than successive monomial multiplications as follows:

$$\begin{gathered} h(x) = f(x) \cdot g(x) \\ = x^k f(x) + g_{k-1}x^{k-1}f(x) + \cdots + g_1xf(x) + g_0f(x). \end{gathered}$$

However, in order to optimize efficiency, different implementations are chosen pertaining to the hardware.

4 Implementation of Operations:

As discussed earlier, the implementation of the multiplication function is one of the largest determinants of the program's efficiency. Addition is conveniently the inexpensive bitwise XOR operation. For machines that have more storage capacity, pre-computed lookup tables for multiplication is the desired method. For a field GF(2^k), the space required to store a full-multiplication table would be its Cartesian product, $2^k \cdot 2^k$.[3][1] Under this approach, a full-lookup table for GF(2^8) will take up approximately 64KB. GF(2^{16}) is the next possible field, though a full-lookup table would require about 8GB. Two alternatives to the full-lookup table exist that trade more computations for less memory: log-tables and composite fields. Especially relevant to resource-limited devices such as sensors, minimizing memory and power consumption is necessary. Both of these alternatives seek to solve this problem. The ability for the L1 and L2 caches[2] to hold a whole table or not is a large determinant of how efficient that method will be on that particular hardware setup. Holding a memory requirement of $2^k + 2^k$, log tables were most appropriate for this research because the order of the field is 2^8. It is important to note that we are able to represent a concatenation of a binary polynomial's coefficients as the decimal value of its binary representation due to the bijective map existing between the two fields containing 256 elements each.[2] This convenience gives the impression that we arithmetic is modulo the integer; in actuality, operations are computed modulo the binary representation of the irreducible polynomial. The implementation in this research used "bit-wise modulo 2 and word-wise modulo 285" to establish the field GF(256). In order to minimize the complexity of the GF(256) arithmetic, an irreducible polynomial with the lowest number of nonzero.[1]

Initialization:

INPUT: ( ) // for field of size 2^k and a prime polynomial P

```
    log[$2^k$], alog[$2^k$]; // setup tables[3]
    $log_0 \leftarrow 2^k - 1$; // initialize bases
    $alog_0 \leftarrow 1$; // initialize exponents

    for(i = 1 to $2^k$ - 1) {

        if($alog_{i-1} \geq (2^k \div 2)$)

            $alog_i \leftarrow XOR((alog_{i-1} * 2), P)$

        else

            $alog_i \leftarrow alog_{i-1} * 2$

        $log_{alog_i} = i$ // link corresponding values
    }
```

[1]When the symmetric table implementation is used, the memory requirement is reduced to $2^{k-1} \cdot (2^k + 1)$ because the table becomes a lower or left triangular matrix.

[2]The L1 and L2 caches are solid-state memory that (at least L1, if not both) are included inside the CPU and are the fastest memory depots in order of naming due to its locality to the processor.

[3]In our implementation, these two arrays are global.

Log and antilog tables are made possible by the existence of a generator α, two in this case, that allows the use of power representation to look up the product or quotient modulo 2^k - 1.[1]

Multiplication:
INPUT: $a, b \in GF(2^k)$

```
if (a == 0 or b == 0)
    return 0
else
    return $alog_{(log_a + log_b) mod (2^k - 1)}$
```

The division algorithm to lookup the resulting quotient of a given element a divided by another element b is similar to the product function, only looking up the corresponding exponent to the element that was need to produce a from b.

Division:
INPUT: $a, b \in GF(2^k)$

```
if (b == 0)
    return 1 - $2^k$
else if (a == 0)
    return 0
else
    return $alog_{(log_a - log_b + (2^k - 1)) mod (2^k - 1)}$
```

Here, the Extended Euclidean algorithm is not necessary. Looking up a multiplicative inverse b for a given a is simple with the quotient function and the inverse property $a \cdot b = 1$.

Inverse:
INPUT: $a \in GF(2^k)$

```
if (a == 0)
    return -1
return Quotient(1, a)
```

5 Multivariate Key Pre-distribution Scheme:

In this portion, the protocol multivariate key pre-distribution scheme and the reasoning for its use are introduced. With the growing demand for wireless sensor networks, cases where nodes may be compromised, out of communication range, or even destroyed may arise. This fact creates a huge incentive for the organization administering the network to build/purchase cheaper, disposable sensor nodes. However, these less expensive nodes have significant constraints in both memory and processing power. Thus, memory expensive and computationally intensive algorithms such as RSA and elliptic curve methods are impossible. Two phases compose MKPS: the setup and the link-key establishment phases. Before discussing the structure of MKPS, some frequently used notation would be useful:

5.1 Notation:

n	Number of nodes scattered
d	Dimension of hypercube
m	= $\lceil \sqrt[d]{n} \rceil$
I	Node ID (a d-tuple)
t	Degree of the multivariate polynomials in each variable
M	Maximum size of memory of each node
$k_{I,I',l}$	l-th common key between nodes I and I'
$k_{I,I'}$	Final link-key between I and I'
P_{lk}	Average probability of link-key establishment
β	Size of the connected component of the network normalized to the network size
λ	Security threshold of MKPS
P_{pr}	Probability of the polynomial recovery
P_{lkc}	Probability of the link-key compromise
p_r	Node-capture tolerance
d_{opt}	Optimal dimension
d_{wst}	Worst dimension
γ_1, γ_2	Thresholds used in the dimension-optimization procedure
$I\langle j \rangle$	A d - 1 tuple that is obtained from removing i_j (the (j + 1)-th element) from an ID I
$[m]$	The set of positive integers 0, 1, . . . , m - 1

5.2 Setup:

During this initialization phase that takes place prior to network deployment, the sink creates and distributes ID's to the sensor nodes. Each ID is a tuple $I = (i_0, ..., i_{d-1})$ of length d such that $i_0, ..., i_{d-1}$ are all nonnegative integers. $\mathbb{I}$ is the set of n ID's created by the hypercube. Due to the uniqueness of each node's ID, verification among nodes' IDs is implemented. Moreover, since m = $\lceil \sqrt[d]{n} \rceil$, $[m-1]^d \subset \mathbb{I} \subseteq [m]^d$, where $[m]^d$ is the Cartesian product of [m] with itself d times.

On top of the ID's, the sink pseudo-randomly generates dm symmetric d-variate polynomials represented by the notation $f_i^j(x_0, ..., x_{d-1})$ for all $i \in [m]$ and $j \in [d]$. These d-variate polynomials are of the form

$$f_i^j(x_0, ..., x_{d-1}) = \alpha(x_0^t x_1^{r_{1,1}} \cdots x_{d-1}^{r_{1,d-1}} + x_0^t x_1^{r_{1,d-1}} \cdots x_{d-1}^{r_{1,1}} + ... + x_0^{r_{1,1}} x_1^{r_{1,d-1}} \cdots x_{d-1}^{t}) + \\ \beta(x_0^{r_{1,0}} x_1^{r_{1,1}} \cdots x_{d-1}^{r_{1,d-1}} + x_0^{r_{1,0}} x_1^{r_{1,d-1}} \cdots x_{d-1}^{r_{1,1}} + ... + x_0^{r_{1,1}} x_1^{r_{1,d-1}} \cdots x_{d-1}^{r_{1,0}})$$

where each r is a random exponent greater than or equal to 0 and less than t. The symmetric characteristic of the polynomial is created from the existence of all the permutations of t with the d-1 pseudo-random, positive exponents less than t in the α component and all the permutations of the d pseudo-random, positive exponents less than t in the β component. The two coefficients associated with a symmetric d-variate, α and β, are members of the finite field $\mathbb{F}$[4] Every symmetric polynomial of this form contains degree t of every variable. Additionally, for each ID I, the sink builds the set

$$\mathbb{O}_I = \{f_{i_j}^j(I\langle j \rangle, x_{d-1}) \in \mathbb{F}[x_{d-1}] : \forall j \in [d]\}$$

containing d univariate polynomials. This set is called the key ring of a node ID I. The coefficients of all d polynomials are stored in the node with the corresponding ID.

[4]GF(2^8) in this implementation of MKPS.

The storage space required to save the d polynomials in $\mathbb{O}_I$ is $M = d(t+1)$, put in terms of $\mathbb{F}$. M is a predefined value decided by the amount of memory each sensor node contains, whereas d is arbitrarily chosen or is optimized. Given M and d,

$$t = \lfloor M/d - 1 \rfloor$$

EXAMPLE 5.2.1. Let d = 3. For a node with ID $I = (i_0, i_1, i_2)$, the set $\mathbb{O}_I$ is $\{f_0^0(i_1, i_2, x_2), f_1^1(i_0, i_2, x_2), f_2^2(i_0, i_2, x_2)$ $\}$[5]

5.3 Link-key Establishment:

After all nodes receive their ID's and key-rings, the link-key establishment phase can commence after the deployment of the nodes in the field. When this occurs, all nodes broadcast their ID's within their communication range. If any of the neighboring nodes' ID's in that area have a Hamming-distance of one with that node's ID, the two nodes create a link-key. Sharing a Hamming-distance of one means that one string, or tuple in this case, can be obtained from the other by substituting only one element. Consider the two node ID's I and I' that share a Hamming-distance one. These nodes differ only at the j-th coordinate. Due to the symmetry of the polynomials, the two nodes can form the following set of $d-1$ common keys.

$$k_{I,I',l} = f_{i_l}^l(I\langle j,l\rangle, i_j, i'_j) = f_{i_l}^l(I\langle j,l\rangle, i'_j, i_j) \quad \forall l \in [d]/\{j\}$$

As will be shown, the $d-1$ common keys of each link-key establishment significantly lowers the probability of link-key compromise. The protocol is thus deterministic and any established link between two nodes is guaranteed to share $d-1$. The final link-key $k_{I,I'}$ is established via a symmetric function $\psi : \mathbb{F}^{d-1} \to \mathbb{F}$ by evaluating all $d-1$ common keys, producing

$$k_{I,I'} = \psi(k_{I,I',l} : \forall l \in [d]/\{j\}).$$

This function is visible to all nodes as well as the opponent. The function ψ must be efficient due to the constraints of the sensor nodes. This function must also produce the same result regardless of the chronology of the inputs in order the create equal importance in obtaining each key for the link-key compromise and to optimize the link-key establishment between two nodes. The ψ function[5] could be the bitwise XOR of all $d-1$ keys or the hash of the concatenation of the keys.

EXAMPLE 5.3.1. Let $d = 3$ and consider the two node ID's $I = (i_0, i_1, i_2)$ and $I' = (i_0, i_1, i'_2)$. Sharing a Hamming-distance of one, the two ID's establish two common keys $k_{I,I',0} = f_0^0(i_1, i_2, i'_2) = f_0^0(i_1, i'_2, i_2)$ and $k_{I,I',1} = f_1^1(i_0, i_2, i'_2) = f_1^1(i_0, i'_2, i_2)$. The resulting link-key is $k_{I,I'} = \psi(k_{I,I',0}, k_{I,I',1})$.[5]

5.4 Network Connectivity:

As mentioned before, all of the nodes n scattered in the field may not be connected in the final network β. Two nodes share an edge if they can form a link-key and in turn form a path. The final network all possible paths connected to the sink. Thus, the size of β is directly related to the probability of link-key establishment P_{lk}.[5] To refresh our memory, two node ID's can establish a link-key if they share a Hamming-distance of one. The probability of link-key establishment P_{lk} is denoted[6]

$$P_{lk} \cong \frac{([d(m-2)+v](m-1)^d)(\theta m^{d+1}[d-2+v\theta^{v-1}+(2-d-v)\theta^v])}{n(n-1)} \leq \frac{d(m-1)}{n-1}$$

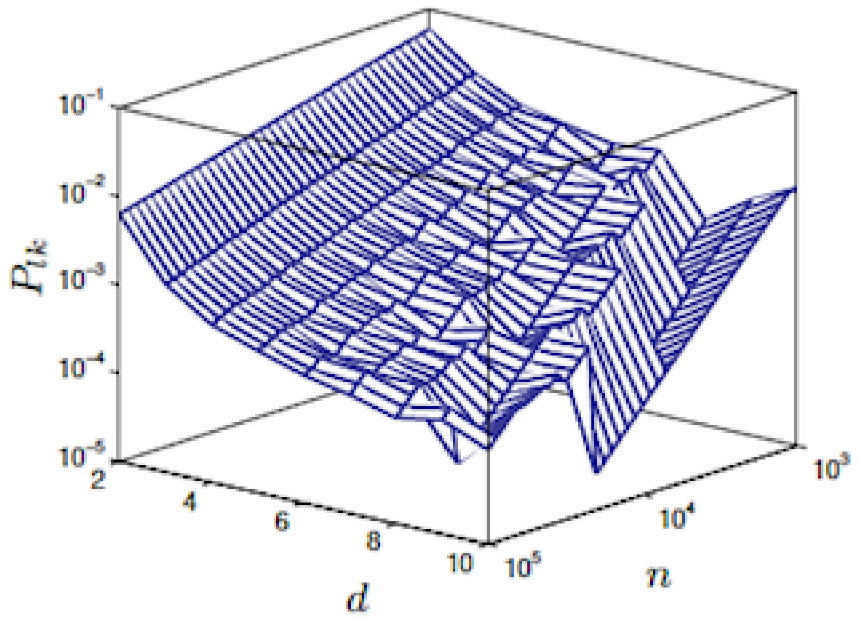
where

$$\theta := 1 - \frac{1}{m} \qquad v := \lfloor \frac{log(1+\theta^d - nm^{-d})}{log\theta} \rfloor \qquad [5]$$

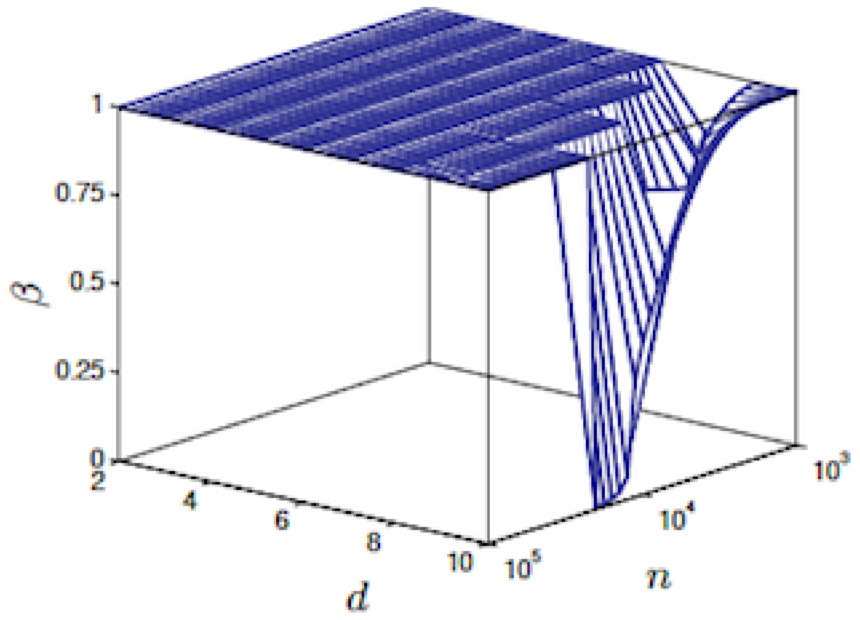
As given in [5], graph (a) in Figure 1 maps P_{lk} to the total number of nodes n and the dimensionality d. The sudden changes in the 3-dimensional graph are a cause of the ceiling function's integer approximation in the definition of m.[5] Shown in the figure, given a fixed dimension d, the average probability of link-key establishment P_{lk} decreases

[5]Because of time limitations, the XOR operation will be ψ in this implementation.

[6]To keep the paper concise, the derivation of the link-key establishment probability function won't be discussed here.



(a) Average probability of the link-key establishment.



(b) Size of the largest component in the network normalized to the network size.

Figure 1: The average probability of link-key establishment versus dimension d and number of nodes n and the corresponding normalized size of the connected component.

as a result of adding more nodes to the field n. On the other hand, in fixing n and increasing d, the probability P_{lk} generally decreases; however, some linearly increasing moments exist.[5] The global decrease is due to m exponentially decreasing with the decrease of d, even though a linear coefficient of d exists in the first term of of P_{lk}. The linearly increasing sections of the graph are due to the range of n in which m is constant and therefore, the linear coefficient of d is the dominant factor.[5]

These graphs were formed using the random graph model $G(n, P_{lk})$ in deriving P_{lk}. The radius of each node's signal is assumed unlimited in the model. However, when the nodes are deployed, the communication radius will be a constant. Previously proved in [6], the minimum radius that is necessary to have a connected network is $R = \sqrt{(ln\ \ n + \zeta)/(\pi n P_{lk})}$, where $\zeta > 0$ is a constant.[5]

5.5 Resilience Against Node Capture:

As discussed in [5], an adversary may try to compromise the whole connected component by splitting the network into subdivisions. To accomplish this, the adversary must compromise the link-keys established between nodes without physical capture. However, as stated in the link-key establishment phase, a link-key is a symmetric function of all d - 1 common keys between the two nodes. Therefore, in order to compromise the link-key, the adversary must compromise all d - 1 common keys. Since these common keys are gathered by evaluation of the shares of multivariate polynomials at the nodes IDs, the adversary must recover, without physical capture, some number of variables in the multivariate polynomials through capturing other nodes in the network that store those shares. In effect, a threshold of a minimum number of nodes captured exists to compromise the protocol. Before reaching this threshold, the adversary is unable to compromise any link-keys. First, we must find the MKPS security threshold:

Consider the node ID's

$$I = (i_0, ..., i_{j-1}, i_j, i_{j+1}, ..., i_{d-1})$$

$$I' = (i_0, ..., i_{j-1}, i'_j, i_{j+1}, ..., i_{d-1})$$

for some $j \in [d]$. The Hamming-distance of these two ID's is equal to 1 and therefore can form the link-key $k_{I,I'}$, which is the symmetric function ψ of all d - 1 common keys between the two nodes. Thus, in order to compromise the link-key, the adversary needs to compromise all d - 1 keys created by the polynomials $f^l_{i_l}(I\langle l\rangle, x_{d-1})$ and $f^l_{i_l}(I'\langle l\rangle, x_{d-1})$ as in $k_{I,I',l}$. That said, after deployment, these polynomials only exist in the memories of the nodes I and I', which are inaccessible to the adversary. Hence, for each $l \in [d]/\{j\}$, the adversary must recover the polynomial

$$f_l(x_{d-r-1}, ..., x_{d-1}) = f^l_{i_l}(\hat{i}_0, ..., \hat{i}_{d-r-2}, x_{d-r-1}, ..., x_{d-1})$$

from its shares which are distributed throughout the network for some integer $1 \le r \le d-1$ and where $(\hat{i}_0, ..., \hat{i}_{d-2}) = I\langle l\rangle$. These shares can be represented as $f_l(\hat{i}_{d-r-1}, ..., \hat{i}_{d-2}, x_{d-1})$, where $\hat{i}_{d-r-1}, ..., \hat{i}_{d-2} \in [m]$.[5] These shares are obtainable to the adversary by capturing other sensor nodes, which at most m^r possible shares of this polynomial are available in the network. The following lemma shows the minimum number of shares required to build the intended polynomial:

LEMMA 5.5.1. *To recover the polynomial $f_l(x_{d-r-1}, ..., x_{d-1})$ from its scattered shares, the required number of shares is*

$$\lambda(r,t) = \binom{t+r}{r}, \text{for } 1 \le r \le d-1$$

Proof. Since

$$f_l(x_{d-r-1}, ..., x_{d-1}) = \sum_{i=0}^{t} f_{l_i}(x_{d-r-1}, ..., x_{d-2})x^i_{d-1},$$

where each $f_{l_i}(x_{d-r-1}, ..., x_{d-2})$ is an r-variate symmetric polynomial of degree t in each variable that has $\binom{t+r}{r}$ coefficients. Thus, $\lambda(r,t)$ shares are required to recover the polynomial $f_l(x_{d-r-1}, ..., x_{d-1})$.[5] ■

If, for some r, $m^r < \lambda(r,t)$, then fewer shares of the polynomial exist in the network than are needed to recover r variables. As a result, the adversary will be unable to reconstruct the polynomial $f_l(x_{d-r-1}, ..., x_{d-1})$ for the given value of r. To clarify, the inequality $m^r < \lambda(r,t)$ does not imply $m^{r+1} < \lambda(r+1,t)$. This means that the situation of $m^{r+1} \geq \lambda(r+1,t)$ may exist in which the network holds enough shares of f to recover $r+1$ variables. Therefore, the security threshold is decided by the amount of variables in the network that have enough shares in the network to recover that number of variables. The following corollary recapitulates the security threshold:

COROLLARY 5.5.1. *If there exists an integer 1 $\leq$ r $\leq$ d - 1 such that $m^i < \lambda(i,t)$ for all integers 1 $\leq$ i $\leq$ r - 1, but $m^r \geq \lambda(r,t)$, then the instance of MKPS is $(\lambda(r,t)-1)$-secure in the network.*

As shown by the corollary, the first positive integer r that satisfies the condition $m^r \geq \lambda(r,t)$, is the amount of variables for which enough shares in the network exist to recover the target polynomial. In order to make future references to security parameters more convenient, we define the security level of an implementation of MKPS:

DEFINITION 5.5.1. *If an occurrence of MKPS fits COROLLARY 5.1, that version of MKPS is said to be **r-Variate secure.***

In review, an adversary must capture at least $\lambda(r,t)$ nodes to recover the polynomial f_l in an r-Variate secure scheme. By constructing this polynomial, the adversary can compromise the d - 1 common keys that are needed to compromise the link-key between two node ID's, I and I', that have not been captured already. Put consciously, MKPS is completely secure until $\lambda(r,t)-1$ nodes are captured, which gives the protocols threshold. The security threshold r, m, and $\lambda(r,t)$ are all global parameters of the network.

With the security threshold of the protocol, the probability of link-key compromise can be calculated. Let p_{nc} be the fraction of captured nodes in the network and the implementation of MKPS be r-variate secure. By Lemma 5.5.1, at least $\lambda(r,t)$ shares of any polynomial is needed to obtain r variables of that polynomial. In accordance with the number of shares of a polynomial being a binomially-distributed random variable, the probability of polynomial recover is

$$P_{pr} = \sum_{i=\lambda(r,t)}^{m^r} \binom{m^r}{i} p_{nc}^i (1-p_{nc})^{m^r-i} \qquad [5]$$

Since all d - 1 common keys must be recovered to compromise a particular link-key, the probability of link-key compromise is

$$P_{lkc} = P_{pr}^{d-1}.$$

Figure 2 maps the probability of link-key compromise against the fraction of captured nodes p_{nc} under different values of d. In this model, we fix the memory used in each node M to M = 50 elements from $\mathbb{F}^7$ for practicality. With M = 50, the polynomial degree in each variable is $t = \lfloor 50 \div (d-1) \rfloor$. For every test, the total number of nodes n is 10,000. Observing the figure, the probability P_{lkc} generally decreases when increasing d. However, this trend is not true for all values of d. These irregularities can be accounted for from the interaction of the different parameters in the probability of polynomial recovery function, which are all dependent on d, and the ceiling function in the definition of m.

6 Conclusions:

Earlier, the problem of sensor node resource limitations constraining the level of security from traditional methods was introduced. Large bit size encryption is simply not possible for the disposable sensor node today. However, sensor fields will certainly be deployed over hostile environments. MKPS can supply the security that is needed for a network without creating too much power or memory consumption for each node. Moreover, combining the already memory and power conscious multi-variate key pre-distribution scheme with the base field GF(256) will optimize a secure protocol for a network of resource-limited devices. A simulation and further analysis of this particular MKPS implementation is on the way.

[7] Since our implementation uses GF(256), M = 50 bytes.

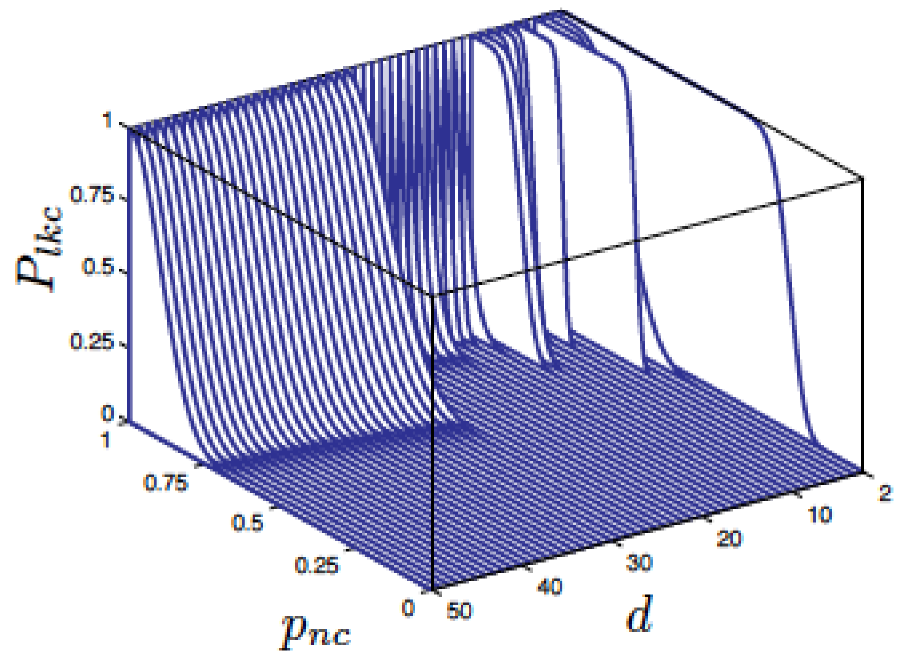


Figure 2: Probability of link-key compromise with $n = 10000$ and $t = \lfloor 50 \div (d-1) \rfloor$.

7 Bibliography:

[1] F. Delgosha. ”Finite-field Arithmetic.” Electrical Engineering and Computing Sciences New York Institute of Technology, New York NY. 2009.

[2] AIM Inc. ”Arithmetic Operations in a Power-of-Two Galois Field.” 634 Alpha Drive Pittsburgh, PA, 15238. Sep 22, 2001.

[3] K. Greenan, E. Miler, T. Schwartz. ”Analysis and Construction of Galois Fields for Efficient Storage Reliability.” Technical Report UCSC-SSRC-07-09. Storage Systems Research Center Department of Computer Engineering Jack Baskin School of Engineering Santa Clara University Computer Science Department, Santa Clara, CA 95053 University of California, Santa Cruz, August 21, 2007.

[4] J. Luo, K. Bowers, A. Oprea, L. Xu. ”Efficient Software Implementations of Large Finite Fields GF(2^n) for Secure Storage Applications.” Wayne State University. RSA Laboratories.

[5] F. Delgosha, F. Fekri. ”A Multivariate Key-Establishment Scheme for Wireless Sensor Networks.” IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, VOL. 8, NO. 4, APRIL 2009.

[6] H. Pishro-Nik, K. Chan, and F. Fekri. ”On connectivity properties of large-scale wireless sensor networks.” in Proc. First Ann. IEEE Commun. Society Conf. on Sensor Commun. and Net., Santa Clara, CA, 4-7 October 2004, CD-ROM

8 Acknowledgements:

Thanks to Dr. Ziqian Dong, Dr. Farshid Delgosha, REU Program Manager Dr. Marta Panero, NYIT Graduate Student Randy Espejo, and fellow REU participants for their helpful guidance and support. The project is funded by National Science Foundation Grant No.1263283 and New York Institute of Technology