Network Security and Cryptography
Symmetric-key cryptography

Lecture 7: AES and finite fields of polynomials

Mark Ryan

AES and finite fields of polynomials.

Certain operations and constants in AES were not properly defined in the slides so far:

- The operation $\otimes$ on bytes;
- The substitution table;
- The bytes $RC_1, \ldots, RC_{10}$ used in the key schedule algorithm.

These are defined using finite fields of polynomials.

Definition

- A *polynomial* is an expression of the form

$$a_n x^n + \ldots + a_2 x^2 + a_1 x + a_0,$$

  where x is a variable symbol, and $a_0, \ldots, a_n$ are values chosen from some set.

  We say this is a "polynomial in x".
  Often $a_i \in \mathbb{Z}$ (each i), and we say it's a "polynomial over $\mathbb{Z}$".

- The a_i's are called **coefficients**, and n is called the **degree** of the polynomial.

- We denote the set of all polynomials in x over $\mathbb{Z}$ as $\mathbb{Z}[x]$.

Polynomials with bit coefficients

Instead of having the coefficients in $\mathbb{Z}$, we can use *bits*.
So the coefficients will be 0 or 1.
Because the set of bits $\{0, 1\}$ is written $\mathbb{F}_2$, we write the set of polynomials over bits as $\mathbb{F}_2[x]$.

Operations on polynomials in $\mathbb{F}_2[x]$

1. You can add them. Just add the respective coefficients, remembering that the coefficients are *bits* (thus, $1 \oplus 1 = 0$).
2. You can multiply them. You might remember how to multiply polynomials from school. Polynomials in $\mathbb{F}_2[x]$ work the same way, but again, you need to remember that the coefficients are bits.
3. You can divide one polynomial by another, yielding a quotient and remainder.
4. Combining these ideas, you can multiply two polynomials *modulo a third one*.

Irreducible polynomials

Definition

An integer n is called *prime* if its only divisors are 1 and n

The same notion for polynomials is called irreducible:

Definition

A polynomial $p(x) \in \mathbb{F}_2[x]$ is called *irreducible* if its only divisors are $p(x)$ and the constant polynomial $1 \in \mathbb{F}_2[x]$.

If $p(x)$ is an irreducible polyomial in $\mathbb{F}_2[x]$, then we write $\mathbb{F}_2[x]/p(x)$ for the set of polynomials in $\mathbb{F}_2[x]$ considered modulo $p(x)$.

Using polynomials to define a new operation on bitstrings

We identify polynomial of degree 7 with a bitstring length 8:

$$\begin{array}{cccccccc} x^7 & +x^6 & & +x^4 & +x^3 & & & +1 \\ 1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{array}$$

Multiplication of polys as a new bitstring op

Consider multiplication in $\mathbb{F}_{2^3} = \mathbb{F}_2[x]/p(x)$, with $p(x) = x^3 + x + 1$ as irreducible polynomial.
This is an operation on 3-bit strings. Example:

$$\begin{array}{ccccc} (x^2+x+1) & \cdot & (x^2+1) & \equiv & x^2+x \pmod{x^3+x+1} \\ 111 & \otimes & 101 & = & 110 \end{array}$$

Observe that the choice of irreducible polynomial really matters in the definition of $\otimes$. For instance, if our choice of irreducible polynomial were $x^3 + x^2 + 1$ then the example would look like this:

$$\begin{array}{ccccc} (x^2+x+1) & \cdot & (x^2+1) & \equiv & 1 \pmod{x^3+x^2+1} \\ 111 & \otimes & 101 & = & 001 \end{array}$$

Two operations over bitstrings length 3

The previous example defined $\otimes$. So now we have two operations over bitstrings of length 3:

$\oplus$	000	001	010	011	100	101	110	111
000	000	001	010	011	100	101	110	111
001	001	000	011	010	101	100	111	110
010	010	011	000	001	110	111	100	101
011	011	010	001	000	111	110	101	100
100	100	101	110	111	000	001	010	011
101	101	100	111	110	001	000	011	010
110	110	111	100	101	010	011	000	001
111	111	110	101	100	011	010	001	000

$\otimes$	000	001	010	011	100	101	110	111
000	000	000	000	000	000	000	000	000
001	000	001	010	011	100	101	110	111
010	000	010	100	110	011	001	111	101
011	000	011	110	101	111	100	001	010
100	000	100	011	111	110	010	101	001
101	000	101	001	100	010	111	011	110
110	000	110	111	001	101	011	010	100
111	000	111	101	010	001	110	100	011

Note that 000 is a special element. It is the identity element for $\oplus$, and it is a "destructor" for $\otimes$, i.e. $000 \otimes b_2 b_1 b_0 = 000$. Each element in the table for $\oplus$ has an inverse w.r.t. 000. Also, 001 is the identity element for $\otimes$, and each element in the table for $\otimes$ has an inverse w.r.t. 001. All this means that we have defined a mathematical structure called a *field*.

Bitstring operation, continued

The $\otimes$ operation is computed as follows:

- Each bitstring $a_2 a_1 a_0$ is interpreted as a polynomial $a_2 x^2 + a_1 x + a_0$.
- The two polymomials are multiplied together, and reduced modulo our chosen polynomial, which is this one: $x^3 + x + 1$.
- The result is converted back into a 3-bit string.

You can check that the field properties are satisfied.

Connection with AES

In AES, we use the field

$$\mathbb{F}_2[x] \; / \; (x^8 + x^4 + x^3 + x + 1)$$

This gives us two operations $\oplus$ and $\otimes$ on bytes. For example:

$$0x53 \otimes 0xCA = 0x01$$

These two operations is used to define the MixColumns operation and the S-boxes of AES.

Substitution in AES

The substitution operation for a byte B is defined as follows.

1. First compute the multiplicative inverse of B in the AES field, to obtain $B' = [x_7, \ldots, x_0]$. In this step, the zero element is mapped to $[0, \ldots, 0]$.
2. Then compute a new bit vector $B'' = [y_7, \ldots, y_0]$ with the following transformation in $\mathbb{F}_2$ (observe that the vector addition is the same as an xor $\oplus$):

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

The result of the substitution is B''.

Key schedule in AES

Recall that the key schedule algorithm in AES used some constants, $RC_1, \ldots, RC_{10}$. We didn't define these, but we can do so now.

RC_i is defined as the 4 byte (32 bit) value $rc_i\,00\,00\,00$, where the first byte is rc_i is defined as the byte corresponding to the polynomial:

$$x^{i-1} \mod x^8 + x^4 + x^3 + x + 1$$

and the remaining three bytes are zeros. Thus, rc_1 is the byte 00000001 in binary, or 01 in hex, and therefore RC_i is 01 00 00 00 in hex. The byte rc_3 is the byte 00000100 in binary, or 04 in hex, and (a bit harder to calculate!) rc_{10} is the byte 00110110 in binary, or 36 in hex.

AES security

AES has been subjected to a huge amount of analysis and attempted attacks, and has proved very resilient. So far, there are only very small "erosions" of AES:

- There is a meet-in-the-middle key recovery attack for AES-128. It requires 2^{126} operations, so it is only about four times faster than brute-force.
- There is a "related key" attack on AES-192 and AES-256. This means that if you use two keys that are related in a certain way, the security may be reduced. But this is an "invalid" attack, since correct use of AES means you will always choose random keys.

People have also studied simplified versions of AES, e.g. by considering a reduced number of rounds. A large number of small erosions exist in this situation too.
The Snowden documents revealed that the NSA has teams working on breaking AES, but there is no evidence that they have achieved much beyond what is publicly known.