Information Security System
EC-415-F

Lecture 1

Topics covered

Cryptography II

- Number theory (groups and fields)
- Block cyphers
- Algorithms in the Real World

Cryptography Outline

- **Introduction:** terminology and background
- **Primitives:** one-way hash functions, trapdoors, …
- **Protocols:** digital signatures, key exchange, ..
- **Number Theory:** groups, fields, …
- **Private-Key Algorithms:** Rijndael**,** DES, RC4
- **Cryptanalysis:** Differential, Linear
- **Public-Key Algorithms:** Knapsack, RSA, El-Gamal, Blum-Goldwasser
- **Case Studies:** Kerberos, Digital Cash

Number Theory Outline

- **Groups**
  - Definitions, Examples, Properties
  - Multiplicative group modulo n
  - The Euler-phi function
- **Fields**
  - Definition, Examples
  - Polynomials
  - Galois Fields
- Why does number theory play such an important role?

It is **the** mathematics of finite sets of values.

Groups

- A **Group** is a set G with binary operator $*$ such that

1. **Closure**. For all $a,b \in G$, $a * b \in G$
2. **Associativity.** For all $a,b,c \in G$, $a*(b*c) = (a*b)*c$
3. **Identity.** There exists $I \in G$, such that for all a $\in$ G, $a*I=I*a=a$
4. **Inverse.** For every $a \in G$, there exist a unique element $b \in G$, such that $a*b=b*a=I$

- An **Abelian or Commutative Group** is a Group with the additional condition

5. **Commutativity.** For all $a,b \in G$, $a*b=b*a$

Examples of groups

- Integers, Reals or Rationals with Addition
- The nonzero Reals or Rationals with Multiplication
- Non-singular n x n real matrices with Matrix Multiplication
- Permutations over n elements with composition
  [0✹1, 1✹2, 2✹0] o [0✹1, 1✹0, 2✹2] = [0✹0, 1✹2, 2✹1]

- We will only be concerned with **<u>finite groups</u>**, I.e., ones with a finite number of elements.

Groups based on modular arithmetic

- The **multiplicative group modulo n**

  Z_n^* 🕓 {m : 1 ◉ m < n, gcd(n,m) = 1}

  * 🕓 multiplication modulo p

  Denoted as $(Z_n^*, *_n)$
- **Required properties**:
  - Closure. Yes.
  - Associativity. Yes.
  - Identity. 1.
  - Inverse. Yes.
- **Example:** $Z_{15}^* = \{1,2,4,7,8,11,13,14\}$
- $1^{-1} = 1,\ 2^{-1} = 8,\ 4^{-1} = 4,\ 7^{-1} = 13,\ 11^{-1} = 11,\ 14^{-1} = 14$

The Euler Phi Function

$$\phi(n) = \left|Z_n^*\right| = n\prod_{p|n}(1-1/p)$$

- If n is a product of two primes p and q, then

$$\phi(n) = pq(1-1/p)(1-1/q) = (p-1)(q-1)$$

Note that by Fermat's Little Theorem:

$$a^{\phi(n)} = 1 \pmod{n} \ \text{ for } \ a \in Z_n^*$$

Or for n = pq

$$a^{(p-1)(q-1)} = 1 \pmod{pq} \quad \text{for} \quad a \in Z_{pq}^*$$

This will be very important in RSA!

Generators

- Example of Z_{10}^*: {1, 3, 7, 9}

Generators

x	x^2	x^3	x^4
1	1	1	1
3	**9**	**7**	**1**
7	**9**	**3**	**1**
9	1	9	1

For all n the group is cyclic.

Operations we will need

- Multiplication
  - Can be done in $O(\log^2 n)$ bit operations
- Finding the inverse:
  - Euclids algorithm $O(\log n)$ steps
- Power:
  - The power method $O(\log n)$ steps

Discrete Logarithms

- If g is a generator of Z_n^*, then for all y there is a unique x such that
  - $y = g^x \bmod n$
- This is called the **discrete logarithm** of y and we use the notation
  - $x = \log_g(y)$
- In general finding the discrete logarithm is conjectured to be hard…as hard as factoring.

Euclid's Algorithm

- **Euclid's Algorithm**:
- gcd(a,b) = gcd(b,a mod b)
- gcd(a,0) = a
- **"Extended" Euclid's algorithm**:
  - Find **x** and **y** such that **ax + by = z = gcd(a,b)**
  - Can be calculated as a side-effect of Euclid's algorithm.
  - Note that **x** and **y** can be zero or negative.
- This allows us to find <u>**a^{-1} mod n**</u>, for **a** $\in Z_n^*$
- In particular return <u>**x**</u> in <u>**ax + ny = 1**</u>.

Fields

- A **<u>Field</u>** is a set of elements F with binary operators * and + such that
  1. (F, +) is an abelian group
  2. (F \ I_+, *) is an abelian group
  3. Distribution. a*(b+c) = a*b + a*c
  4. Cancellation. a*I_+ = I_+
- The **order** of a field is the number of elements.
- A field of finite order is a **finite field**.

- The reals and rationals with + and * are fields.
- Z_p (p prime) with + and * mod p, is a finite field.

- Long division on polynomials ($\mathbb{Z}_5[x]$):

Division and Modulus

$$\boxed{1x + 4}$$

$$x^2 + 1 \;\big)\; \overline{x^3 + 4x^2 + 0x + 3}$$

$$\underline{x^3 + 0x^2 + 1x + 0}$$

$$4x^2 + 4x + 3$$

$$\underline{4x^2 + 0x + 4}$$

$$\boxed{4x + 4}$$

$$(x^3 + 4x^2 + 3)/(x^2 + 1) = (x + 4)$$

$$(x^3 + 4x^2 + 3) \bmod (x^2 + 1) = (4x + 4)$$

$$(x^2 + 1)(x + 4) + (4x + 4) = (x^3 + 4x^2 + 3)$$

Polynomials modulo Polynomials

- How about making a field of polynomials modulo another polynomial? This is analogous to $\mathbb{Z}_p$ (i.e., integers modulo another integer).
- e.g. $\mathbb{Z}_5[x]$ mod (x^2+2x+1)
- Does this work?
- Does $(x + 1)$ have an inverse?

Definition: An **irreducible polynomial** is one that is not a product of two other polynomials both of degree greater than 0.

e.g. $(x^2 + 2)$ for $\mathbb{Z}_5[x]$

Analogous to a prime number.

Galois Fields

- The polynomials
- $\mathbb{Z}_p[x] \bmod p(x)$
- where
  $p(x) \in \mathbb{Z}_p[x]$,
  $p(x)$ is irreducible,
  and $\deg(p(x)) = n$
- form a finite field. Such a field has p^n elements.
- These fields are called **Galois Fields** or **$GF(p^n)$.**
- The special case $n = 1$ reduces to the fields $\mathbb{Z}_p$
- The multiplicative group of $GF(p^n)/\{0\}$ is cyclic (this will be important later).

GF(2^n)

- **Hugely practical!**
- The coefficients are **bits** {0,1}.
- For example, the elements of GF(2^8) can be represented as **a byte**, one bit for each term, and GF(2^{64}) as **a 64-bit word**.
  - *e.g.*, $x^6 + x^4 + x + 1 = 01010011$
- How do we do addition?

Addition over $\mathbb{F}_2$ corresponds to xor.

- Just take the xor of the bit-strings (bytes or words in practice). This is dirt cheap

Multiplication over GF(2^n)

- If n is small enough can use a table of all combinations.
- The size will be 2^n x 2^n (e.g. 64K for GF(2^8)).
- Otherwise, use standard shift and add (xor)

- **Note**: dividing through by the irreducible polynomial on an overflow by 1 term is simply a test and an xor.
- e.g. 0111 / 1001 = 0111
- 1011 / 1001 = 1011 xor 1001 = 0010
- ^ just look at this bit for GF(2^3)

Multiplication over GF(2^n)

```
typedef unsigned char uc;

uc mult(uc a, uc b) {
  int p = a;
  uc r = 0;
  while(b) {
    if (b & 1) r = r ^ p;
    b = b >> 1;
    p = p << 1;
    if (p & 0x10) p = p ^ 0x11B;
  }
  return r;
}
```

Finding inverses over $GF(2^n)$

- Again, if n is small just store in a table.
  - Table size is just 2^n.
- For larger n, use Euclid's algorithm.
  - This is again easy to do with shift and xors.

Polynomials with coefficients in GF(p^n)

- Recall that GF(p^n) were defined in terms of coefficients that were themselves fields (*i.e.*, Z_n).
- We can apply this **recursively** and define GF(p^n)[x]
- e.g. for coefficients GF(2^3)
-   $f(x) = 001x^2 + 101x + 010$
- Where 101 is shorthand for x^2+1.
- We can make a finite field by using an irreducible polynomial M(x) selected from GF(p^n)[x].
- For an order m polynomial and by abuse of notation we can write: **GF(GF(p^n)^m)**, which has p^{nm} elements.
- Used in **Reed-Solomon codes** and **Rijndael**.

Cryptography Outline

- **Introduction:** terminology and background
- **Primitives:** one-way hash functions, trapdoors, …
- **Protocols:** digital signatures, key exchange, ..
- **Number Theory:** groups, fields, …
- **Private-Key Algorithms:** Rijndael, DES, RC4
- **Cryptanalysis:** Differential, Linear
- **Public-Key Algorithms:** Knapsack, RSA, El-Gamal, Blum-Goldwasser
- **Case Studies:** Kerberos, Digital Cash

Private Key Algorithms

Plaintext

$Key_1 \rightarrow$ **Encryption** $E_k(M) = C$

Cyphertext

$Key_1 \rightarrow$ **Decryption** $D_k(C) = M$

Original Plaintext

What granularity of the message does E_k encrypt

Private Key: Block Ciphers

- Encrypt one block at a time (e.g. 64 bits)
- $c_i = f(k,m_i) \quad m_i = f'(k,c_i)$
- Keys and blocks are often about the same size.
- Equal message blocks will encrypt to equal codeblocks
  - Why is this a problem?
- Various ways to avoid this:
  - E.g. $c_i = f(k,c_{i-1} \oplus m_i)$
    "**Cipher block chaining**" (CBC)
- Why could this still be a problem?

Solution: attach random block to the front of the message

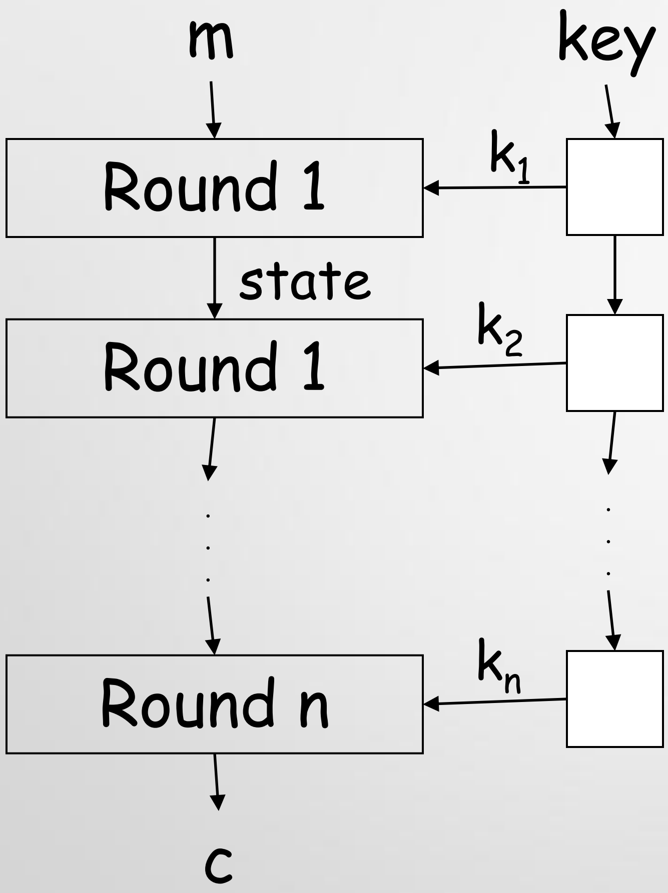
Security of block ciphers

- **Ideal**:
  - k-bit -> k-bit key-dependent subsitution
    (i.e. "random permutation")
  - If keys and blocks are k-bits, can be implemented with 2^{2k} entry table.

Product Ciphers

- Multiple rounds each with
  - **Substitution** on smaller blocks
    Decorrelate input and output: “confusion”
  - **Permutation** across the smaller blocks
    Mix the bits: “diffusion”
- **Substitution-Permutation Product Cipher**
- **Avalanch Effect**: 1 bit of input should affect all output bits, ideally evenly, and for all settings of other in bits

Iterated Block Ciphers



- Each round is the same and typically involves substitutions and permutations
- Decryption works with the same number of rounds either by running them backwards, or using a Feistel network.

Blocks and Keys

$$\begin{pmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{pmatrix} \quad \begin{pmatrix} k_0 & k_4 & k_8 & k_{12} \\ k_1 & k_5 & k_9 & k_{13} \\ k_2 & k_6 & k_{10} & k_{14} \\ k_3 & k_7 & k_{11} & k_{15} \end{pmatrix}$$

- The blocks and keys are organized as matrices of bytes. For the 128-bit case, it is a 4x4 matrix.

Data block

b_0, b_1, …, b_{15} is the order of the bytes in the stream.