信息安全基础综合设计实验

Lecture 04

李经纬

电子科技大学

课程回顾

内容回顾

- 素性测试：Miller-Rabin算法
  - 确定k和q
  - 判定：**$a^q \bmod n ?= 1$或$a^{2\wedge(j-1)q} \bmod n ?= n-1$ $(j = 1, 2, \ldots, k)$**
- 乘法逆元
  - 存在条件：**互素**
  - 欧几里德算法：**辗转相除**，求最大公约数
  - 欧几里德扩展算法：求解k_1和k_2

素性测试 II

- 假设已将n分解为$2^k q + 1$
- 思路：实现判定式来进行素性测试**$a^q \bmod n \mathrel{?}= 1$和$a^{2^{(j-1)}q} \bmod n \mathrel{?}= n-1$**

```
// 判定a^q mod n ?= 1, 即temp ?= 1
for (i = 1; i < q; ++i) {
    temp = (unsigned long long)(temp * a) % n;
}
// 判定存在a^[(2^j-1)*q] mod n ?= n-1
for (i = 1; i < k && temp != (n-1); ++i) {
    temp = (unsigned long long)(temp * temp) % n;
}
```

乘法逆元

- 思路：通过**辅助函数**递归求得最大公约数以及对应x和y
  - 乘法逆元为x

```
int ex_gcd(int a, int m, int &x, int &y) {
    if (m == 0) {
        x = 1; y = 0;
        return a; // 到达递归边界返回上一层
    }
    int r = ex_gcd(m, a % m, x, y);
    int t = x; x = y; y = t - a / m * y;
    return r; // 得到最大公约数
}
```

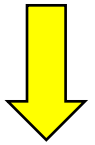
乘法逆元

➢思路：通过**辅助函数**递归求得最大公约数以及对应x和y

- 乘法逆元为x

```
int ex_gcd(int a, int m, int &x, int &y) {
    if (m == 0) {
        x = 1; y = 0;
        return a; // 到达递归边界返回上一层
    }
    int r = ex_gcd(m, a % m, x, y);
    int t = x; x = y; y = t - a / m * y;
    return r; // 得到最大公约数
}
```

x*m + y*(a%m) = r



x*m + y*(a - a/m * m) = r

y*a + (x - y*a/m)*m = r

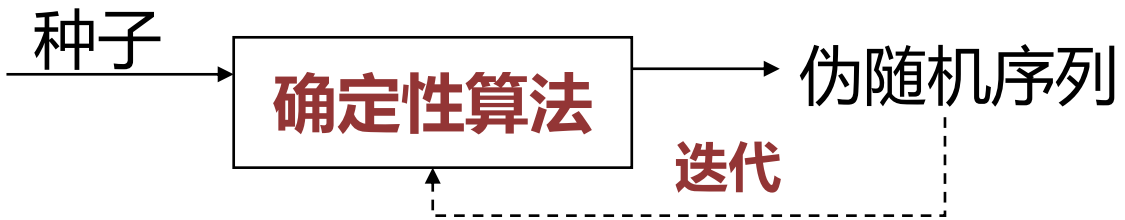
伪随机数生成器——线性同余

伪随机数

- 随机数在密码算法和协议中广泛应用：认证协议、产生会话密钥、流密码
  - 随机数具有随机性：**均匀分布、难以重现**
- 伪随机数通过伪随机数生成器产生，**近似随机数**

伪随机数生成器

➢框架：可以产生任意长度伪随机序列



➢性质：

- **伪随机性**：与随机数不可区分
- **可重现**：相同种子产生相同序列

实例：rand和srand

- **srand(s)**：设置种子
- **rand()**：基于种子产生伪随机数
- 结果：产生相同伪随机数序列
  - 原因：**种子固定**

```
#include<iostream>
// 包含srand和rand依赖的头文件
#include<cstdlib>
using namespace std;
int main () {
    int n = 10;
    srand(1);
    cout << '[';
    for (int i = 0; i < n; i++) {
        cout << rand();
        if (i < n-1) {
            cout << ', ';
        } else {
            cout << ']';
        }
    }
}
```

时间函数

- **`time()`**：返回当前时间相对1970年1月1日0:00经过的秒数
  - 依赖头文件**`ctime`**
- 修改：**`srand(1)→srand(time(NULL))`**
  - **每次运行产生不同种子**
- 其他时间函数：**`gettimeofday`**

线性同余

- 早期rand函数基于**线性同余**算法实现
- 迭代式：$X_{i+1} = aX_i + c \bmod m$
  - 参数：a（乘数）、c（增量）、m（模数）
  - 种子：X_0

Source	modulus *m*	multiplier *a*	increment *c*	output bits of seed in *rand()* or *Random(L)*
Numerical Recipes	2^{32}	1664525	1013904223	
Borland C/C++	2^{32}	22695477	1	bits 30..16 in *rand()*, 30..0 in *lrand()*
glibc (used by GCC)[15]	2^{31}	1103515245	12345	bits 30..0
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++ [16] C90, C99, C11: Suggestion in the ISO/IEC 9899,[17] C18	2^{31}	1103515245	12345	bits 30..16
Borland Delphi, Virtual Pascal	2^{32}	134775813	1	bits 63..32 of *(seed × L)*
Turbo Pascal	2^{32}	134775813 ($0x8088405_{16}$)	1	
Microsoft Visual/Quick C/C++	2^{32}	214013 ($343FD_{16}$)	2531011 ($269EC3_{16}$)	bits 30..16
Microsoft Visual Basic (6 and earlier)[18]	2^{24}	1140671485 ($43FD43FD_{16}$)	12820163 ($C39EC3_{16}$)	
RtlUniform from Native API[19]	$2^{31} - 1$	2147483629 ($7FFFFFED_{16}$)	2147483587 ($7FFFFFC3_{16}$)	
Apple CarbonLib, C++11's `minstd_rand0` [20]	$2^{31} - 1$	16807	0	see MINSTD
C++11's `minstd_rand` [20]	$2^{31} - 1$	48271	0	see MINSTD
MMIX by Donald Knuth	2^{64}	6364136223846793005	1442695040888963407	
Newlib, Musl	2^{64}	6364136223846793005	1	bits 63..32
VMS's **MTH$RANDOM**,[21] old versions of glibc	2^{32}	69069 ($10DCD_{16}$)	1	
Java's java.util.Random, POSIX [lnrand48, glibc [lnrand48[_r]	2^{48}	25214903917 ($5DEECE66D_{16}$)	11	bits 47..16

安全性

- 评价标准
  - **全周期**：{0, 1,..., m-1}中任意数都可能被生成
  - **不可预测**：无法基于X_0, X_1,..., X_{i-1}推断X_i
- 对于任意参数配置，线性同余生成器无法满足全周期
  - 例如：m = 32，a = 7，c = 0，产生序列{7, 17, 23, 1, 7,...}
- 参数(a, c, m) + 伪随机数X_i → 后续伪随机数序列{X_{i+1}, X_{i+2},...}
  - 增强方法：使用系统时钟修正增量

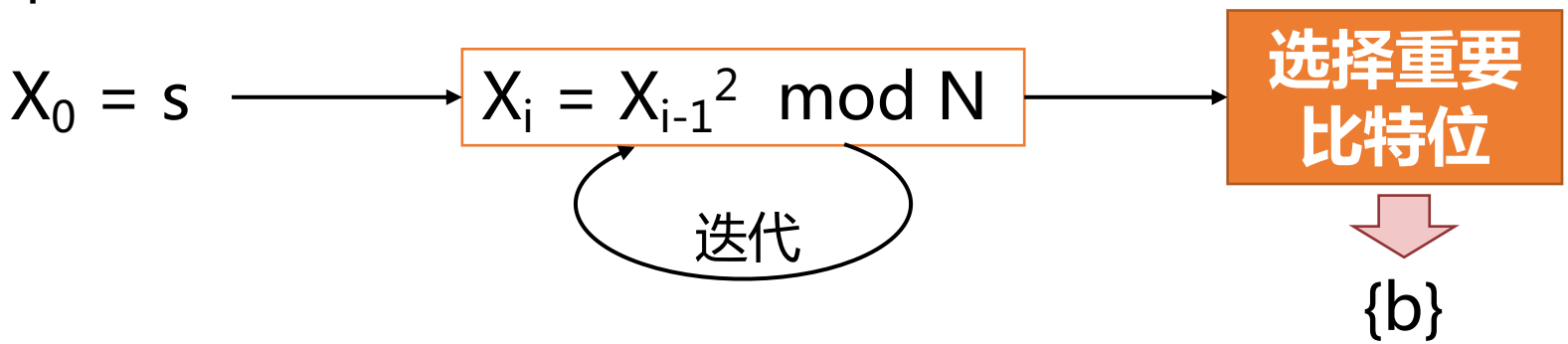
伪随机数生成器——BBS

BBS伪随机数生成器

- 线性同余位随机数生成器不具备**可证明的安全性**
  - 可证明安全性：可以将区分伪随机数和随机数规约为解决数学难题
- 一种可证明安全的伪随机数生成器：BBS (Blum Blum Shub) 伪随机数生成器

原理

- 参数选择：
  - 选择素数p和q，满足p mod 4 = q mod 4 = 3，模数N = p*q
  - 选择种子s，满足s与N互素
- 迭代计算：



安全性

- BBS伪随机数生成器安全性基于**大数难分解**困难问题
  - 大数难分解困难问题：给定n，无法确定n的因子

示例

- 参数选择：p = 11，q = 19，s = 3
  - N = 209
- 迭代计算：设定$X_0 = s$
  - $X_1 = (X_0)^2 \bmod N = 9$；$X_2 = (X_1)^2 \bmod N = 81$；
  - $X_3 = (X_2)^2 \bmod N = 82$；$X_4 = (X_3)^2 \bmod N = 36$；
  - $X_5 = (X_4)^2 \bmod N = 42$；……

示例

- 选择重要位
- 输出伪随机数
  - 最低位：11000 = 24
  - 奇校验位：10010 = 18
  - 偶校验位：01101 = 13

X_i	二进制表示	最低位	奇校验位	偶校验位
$X_1 = 9$	1001	1	1	0
$X_2 = 81$	1010001	1	0	1
$X_3 = 82$	1010010	0	0	1
$X_4 = 36$	100100	0	1	0
$X_5 = 42$	101010	0	0	1

课程作业

线性同余伪随机数生成器

- 基于线性同余算法，产生**下一个**伪随机数

- 函数头：

```
void lcg_srand(unsigned int seed); // 设置线性同余种子

unsigned int lcg_rand();
// 实现线性同余伪随机数生成器算法，采用固定参数配置：
//    a = 1103515245; c = 12345; m = 2^31
// 返回线性同余算法产生的下一个unsigned int伪随机数
// 注：该函数运算将基于lcg_srand设置的种子；
//    如果未通过lcg_srand设置种子，默认种子为1
```

BBS伪随机数生成器

- 基于BBS算法，产生**下一个**伪随机数

- 函数头：

```
unsigned int bbs_rand(int flag);
// 实现BBS伪随机数生成器算法，采用固定参数配置：
//    p = 11; q = 19; s（种子） = 3
// 返回BBS算法产生的下一个unsigned int伪随机数
// 参数flag：标识选择重要比特位的方式
//    flag = 0 - 选择最低比特位
//    flag = 1 - 选择奇校验位
//    flag = 2 - 选择偶校验位
// 提示：执行32轮迭代，将产生的32伪随机比特转换为伪随机数
```

运行时间

- 比较线性同余算法和BBS算法运行时间

- 函数头

```
void rand_time();
// 通过调用lcg_rand()和bbs_rand()，分别产生10个unsigned int
// 伪随机数，比较两种算法的运行时间（精确至微秒级），并在屏幕输出
```