Random Number Generation and Electronic Cash

Dawn Song
dawnsong@cs.berkeley.edu

Random Number Generation

• Many crypto protocols require parties to generate random numbers
  – Key generation
  – Generating nonces
• How to generate random numbers?
  – Step 1: how to generate truly random bits?
  – Step 2: crypto methods to stretch a little bit of true randomness into a large stream of pseudorandom values that are indistinguishable from true random bits (PRNG)

Case Study

• Random number generation is easy to get wrong
• Can you spot the problems in this example?

```c
unsigned char key[16];
srand(time(NULL));
for (i=0; i<16; i++)
  key[i] = rand() & 0xFF;
where

static unsigned int next = 0;
void srand(unsigned int seed)
{
  next = seed;
}

int rand(void)
{
  next = next * 1103515245 + 12345;
  return next % 32768;
}
```
Real-world Examples

• X Windows “magic cookie” was generated using rand()
• Netscape browsers generated SSL session keys using time & process ID as seed (1995)
• Kerberos
  – First discover to be similarly flawed
  – 4 yrs later, discovered flaw with memset()
• PGP used return value from read() to seed its PRNG, rather than the contents of buffer
• On-line poker site used insecure PRNG to shuffle cards

Lessons Learned

• Seeds must be unpredictable
• Algorithm for generating pseudorandom bits must be secure

Generating Pseudorandom Numbers

• True random number generator (TRNG)
  – Generates bits that are distributed uniformly at random, so that all outputs are equally likely, with no patterns, correlations, etc.
• Cryptographically secure pseudorandom number generator (CS-PRNG)
  – Taking a short true-random seed, and generates long sequence of bits that is computationally indistinguishable from true random bits
CS-PRNG

- CS-PRNG: cryptographically secure pseudorandom number generator
  - G: maps a seed to an output G(S)
    - E.g., G: \(\{0,1\}^{128} \rightarrow \{0,1\}^{3000000}\)
  - Let K denote a random variable distributed uniformly at random in domain of G
  - Let U denote a random variable distributed uniformly at random in range of G
  - G is secure if output G(K) is computationally indistinguishable from U

Sample construction
- Use the seed as a key k, and compute AES-CBC(k, 0^n)

TRNG (I)

- TRNG should be random and unpredictable
- Good or bad choices?
  - IP addresses
  - Contents of network packets
  - Process IDs
  - High-speed clock
  - Soundcard
  - Keyboard input
  - Disk timings

TRNG (II)

- How to convert non-uniform sources of randomness into TRNG?
  - Use a cryptographic hash function, such as SHA1
  - Suppose x is a value from an imperfect source, or a concatenation of values from multiple sources, and it is impossible for an attacker to predict the exact value x except with probability 1/2^n
  - Then hash(x) truncated to n bits should provide a n-bit value that is uniformly distributed, if hash() is secure
Administrative Matters

• HW2 graded
  Mean: 41.7
  Standard deviation: 13.2
  1st quartile: 39.8
  2nd quartile (median): 44.5
  3rd quartile: 50.0
  Maximum: 57.0

Ecash

• Example for how crypto helps e-commerce
• Traditional cash

1. Withdraw cash
2. Spend cash for goods
3. Deposit cash

Ecash

• Digital form of cash
• First attempt, what’s the problem?

1. Withdraw ecash: 
   Signature($1, serial #)
2. Spend cash for goods:
   Send Signature($1, serial #)
3. Deposit cash:
   Send Signature($1, serial #)
Desired Properties for Ecash

• Anonymous: bank should not know how Alice spends her money
• Prevent forging
• Prevent double spending

Building Block: Blind Signatures

• Blind signature: achieve anonymity
  – How can Alice get a signature from the Bank without the Bank knowing what message is being signed?
• Protocol:
  – generating blind signature on message m in RSA setting
  – Bank’s private key (d, p, q), public key (e, N)

     Alice (Customer) ---- s = r^m mod N ---- Bank
     t = s^d mod N

     t = sd mod N
     s = r^m mod N

     Alice computes t/r mod N = m^d mod N

Ecash Using Blind Signature

• How to use blind signature to build ecash?
  A valid $1 bill is a pair (x, y), where
  y = hash(x)^d mod N, hash() is one-way function
• How does the ecash protocol work?
• Why do we need hash()?
• How to prevent double spending?
• What to do for different denominations?
  – Nickles, dimes, dollars
Other Methods for Ecash

- Use zero-knowledge proofs (out of scope)
  - More building blocks of ZKP
  - Support many properties
    - Identifying double spenders

Conclusion

- Random number generator
  - CS-PRNG
    - Definition
    - How to construct it?
  - TRNG
- Ecash
  - Example of the power of crypto
  - Blind signatures