CS255 Programming Project 1

Programming Project 1

- Due: Friday Feb 9th (11:59pm)
  - Can use extension days
- Can work in pairs
  - One solution per pair
- Test and submit on Leland machines
  - SCPD students: get SUNet ID!

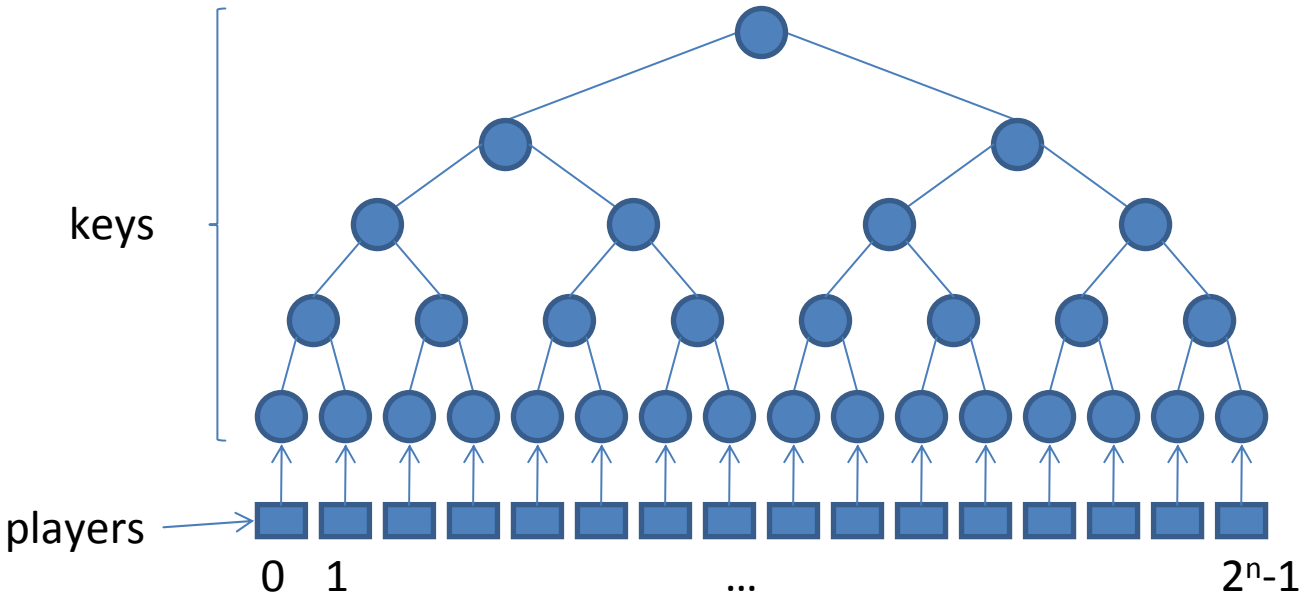
    sunetid.stanford.edu

Overview

- Build an AACS (HD-DVD) like DRM system
- Modeled after problem 2 in PS 1
- Three main components
  - Generate keys and issue to players
  - Encrypt content, accounting for revocation
  - Content “playback” (decryption)
- Written in Java using JCE

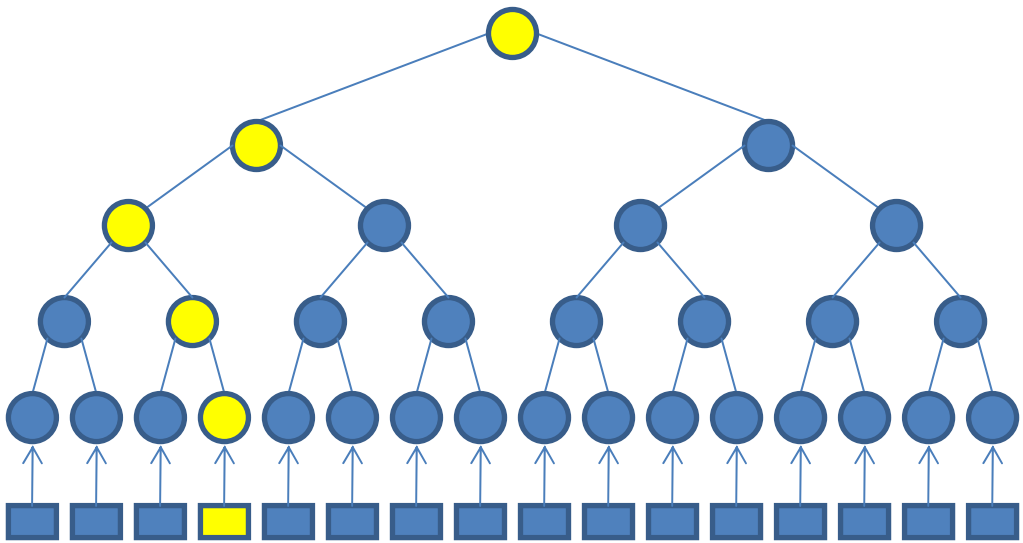
Review of Problem 2

- How to encrypt content so players can be efficiently revoked?
  - Place keys in a binary tree
  - Each player is associated with a leaf of the tree



Issuing Keys

- Each player of the 2^n players issued the n+1 keys on the path from the root to its leaf



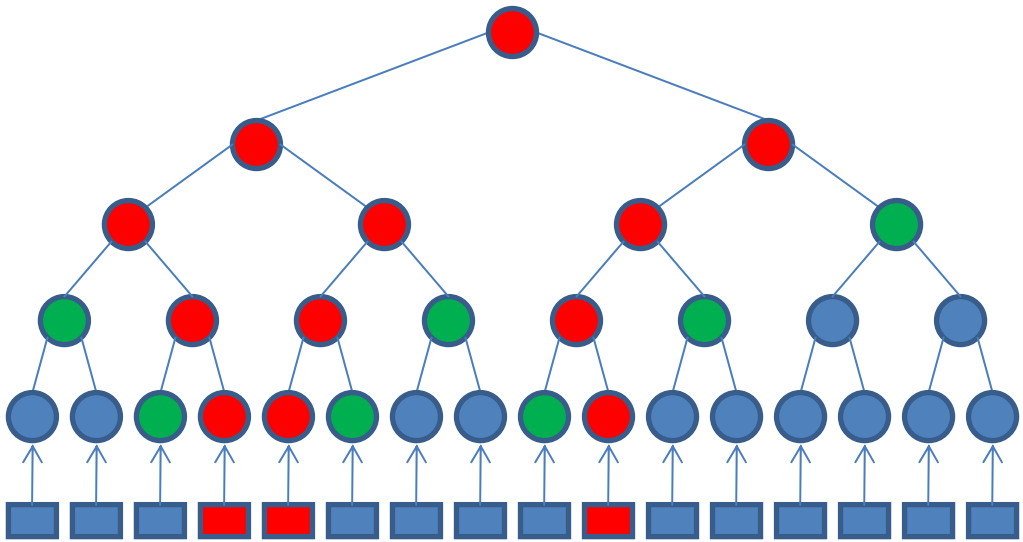
Encrypting Content

- Need to encrypt content so that active players can decrypt, revoked ones cannot
- For each new title, choose a random title key K_{title}
- Encrypt content with K_{title}, then encrypt K_{title} with keys from the tree

$$E[K_{i1}, K_{title}] \,||\, \ldots \,||\, E[K_{im}, K_{title}] \,||\, E[K_{title}, \text{content}]$$

Which Keys to Encrypt K_{title}?

- Need a set of keys which form an exact cover of the non-revoked players
  - Non-revoked players can decrypt
  - Revoked players cannot decrypt



Security Features

- Secure generation and storage (password protected) of player keys
- Encryption of all content with AES in counter mode
- Revocation of compromised players
- Integrity checking using MACs

What is provided?

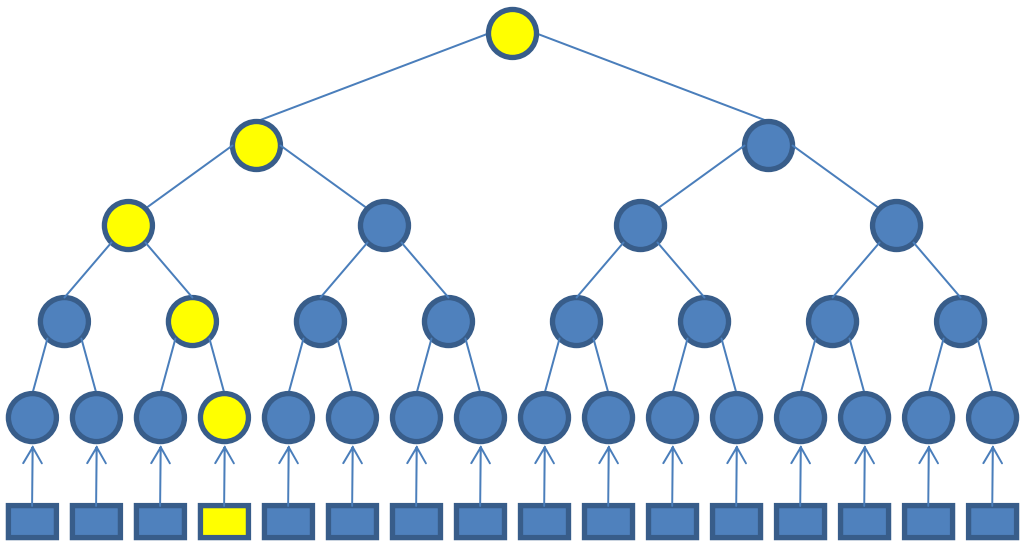
- KeyTree API
  - KeyTree.java
  - Computes player key set and covering set
- Skeleton Code
  - PlayerKeys.java (issues a player's keyfile)
  - DVDManufacturer.java (encrypts content)
  - DVDPlayer.java (verifies and decrypts content)

KeyTree API

- Tree never explicitly represented
- Actual keys derived from a master Key, $K_{aacs,}$ and a unique node ID (you implement derivation)
- Two types of data
  - Player IDs (serial number)
  - Node IDs
  - Both represented as `long`

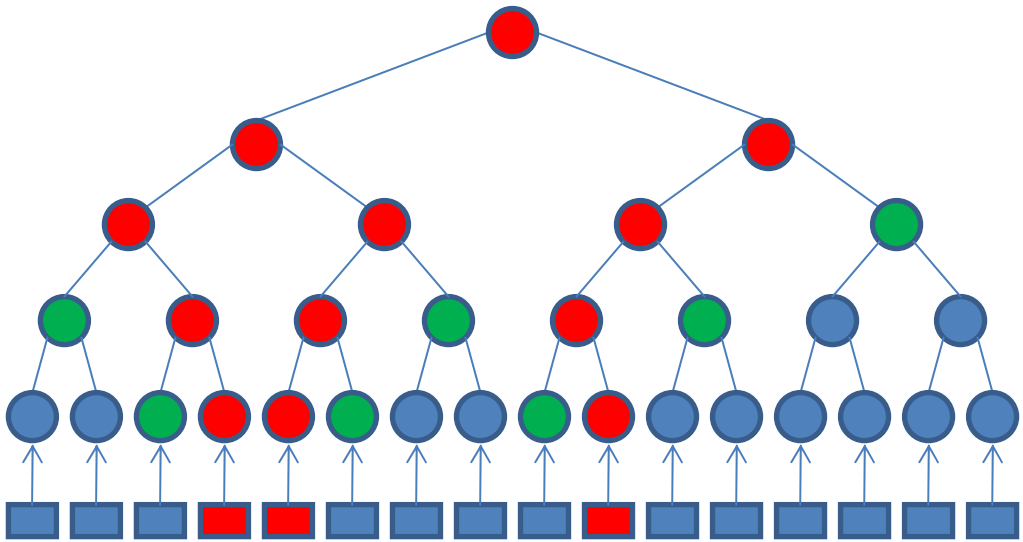
KeyTree API

- `long[] getPathNodes(long playerID)`
  - Returns Node IDs associated with a given player



KeyTree API

- `long[] getCoverSet(long[] excludedPlayers)`
  - Returns a list of Node IDs that represents a “cover set”, that covers all players EXCEPT those whose player ID is listed in excludedPlayers



Skeleton Code

- Provides a basis for each program you must implement
- Reads and parses command line arguments
- Reads revocation list (newline separated integer player IDs)
- Example file IO
  - You must change to add encryption, fit your format, etc
- You may add any additional classes, files needed to facilitate a well decomposed implementation

Components: PlayerKeys

- For a given player ID, generates a password encrypted keyfile
  - Can use the given APIs to
    - Get a list of nodeIDs associated with a player
    - Get key bytes from a password
  - You need to
    - Generate keys from a master AACS key (password)
    - Choose a file format
    - Encrypt using a player specific password (CTR mode)
    - Provide integrity of file (use a MAC)

Components: DVDManufacturer

- Takes content, content title (metadata), and a revocation list and encrypts the content
- Can use given API for computing “cover set”
- You must
  - Generate random title key
  - Generate keys for cover set and encrypt title key
  - Encrypt content
  - Provide integrity for the entire file

Components: DVDPlayer

- For a given player, reads an encrypted content file and tries to decrypt it.
- You must
  - Detect revocation (no associated keys in the header) – O(player_keys + header_keys) time
  - Detect integrity (MAC) failure
  - Decrypt the content, otherwise

Security

- Don’t use the same key to encrypt and MAC !!!
- Use a common key, K, and derive encryption and MAC keys, K_{enc}, K_{mac} using a PRF
  - K_{enc} = HMAC(K, “encrypt”);
  - K_{mac} = HMAC(K, ”integrity”);

Counter Mode

- You must implement it.
- To get a “plain” cipher use ECB mode with no padding
  - Warning! CBC mode used by default
  - Need to specify “AES/ECB/NoPadding”
- Need a counter (try BigInteger)

Java Cryptography Extension

- Implementations of crypto primitives

Cipher	**`Cipher`**
Pseudo-random Generator	**`SecureRandom`**
Message Authentication Code	**`Mac`**
Cryptographic Hash	**`MessageDigest`**

JCE: Generating Random Keys

1. Start the PRG (random seed set by default)
2. Initialize KeyGenerator with the PRG
3. Generate the key

```
// Generate a random encryption key
SecureRandom prng = SecureRandom.getInstance("SHA1PRNG");
KeyGenerator enckeygen = KeyGenerator.getInstance("AES");
enckeygen.init(prng);
SecretKey enckey = enckeygen.generateKey();
```

JCE: Keys From Byte Data

- Use SecretKeySpec
  - Extends SecretKey

```
// Use KeyTree API to get key bytes from password
byte[] keyBytes = KeyTree.createAESKeyMaterial(passwd);

// Use the bytes to create a new SecretKey
SecretKeySpec keySpec = new SecretKeySpec(keyBytes, “AES”);
```

JCE: Using Ciphers

1. Select the algorithm
2. Initialize with desired mode and key
3. Encrypt/Decrypt

```
// Create and initialize the cipher
Cipher cipher = Cipher.getInstance("AES/ECB/NoPadding");
cipher.init(Cipher.ENCRYPT_MODE, enckey);

// Encrypt the message
byte[] msg = "Content is here.".getBytes();
byte[] enc = cipher.doFinal(msg);
```

- *Mac class has a similar API*

Grading

- Security comes first
  - Design choices
  - Correctness of the implementation
- Did you implement all required parts?
- Secondary
  - Cosmetics
  - Coding style
  - Efficiency

Submitting

- README file
  - Names, student IDs
  - Describe your design choices
  - Answer to discussion question
- Your sources
- Use `/usr/class/cs255/bin/submit` from a Leland machine

Stuck?

- Use the newsgroup (su.class.cs255)
  - Best way to have your questions answered quickly
- TAs cannot:
  - Debug your code
  - Troubleshoot your local Java installation