Implementing Capabilities

Capabilities are implemented using:
1. Cryptography
2. Unforgeable pointers/handles (UNIX file descriptors)

Unforgeable Handles
- capabilities are non-transferable (*per process* tables)
- revocation --> easy (by simply flipping valid bit)

<table>
<thead>
<tr>
<th>App Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>fd 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kernel Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>FD</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

Application Source

```c
read(fdid,...);
```
- statement identifies objects
- represents application's right to return on that object
- can only create entry in table via syscall that does some permission check.

Cryptography

Message Authentication Code --> MAC
- similar to Error Correcting Code
- but you need to know the secret (key) to compute MAC

<table>
<thead>
<tr>
<th>Message (M)</th>
<th>ECC(M)</th>
</tr>
</thead>
</table>

ECC(M) --> Checksum

What if ECC is computed on M along with a secret? --> ECC(M,K) --> MAC!

Alice  
1) MAC(K,M)

Bob  
2) verifies that MAC(K,M) is still correct

Examples for MAC: HMAC, SHA256 (usually some timestamps and counters are used to avoid replays)

Application

<table>
<thead>
<tr>
<th>open(“foo”)</th>
<th>read(M, ...)</th>
</tr>
</thead>
</table>

Kernel

Key K

M' = (“foo”, R)

M = M'||MAC(K, M')

Capability Transfer
- capability can be copied without OS interaction
- capability is forever => Revocation is impossible
- can be transferred even over network

Alternate Scheme (Revocation and Non-Transferable)
- Capabilities can expire
  - applications must renew capabilities (M' = (foo, R, Exp))
- Blacklist capabilities --> maintain a list of revoked capabilities
whoops : state
Add Process ID to capability (M' = (foo, R, Exp, PID))
e.g. Kerberos

**Authentication**
How does a computer know 'who you are', given 'who you say you are'?
- password } something that you know
- biometrics } something you are
- secure token } something you have

Biometrics are a very simple password scheme.
- Each reading differs slightly so we can only do appropriate match.
  => cannot use fingerprints as keys

**Iris Scan**
- Iris scan converted to 2048 bit string
- Two scans of same iris agree on >= 1600 bits
- Two scans of different iris agree on < 1200 bits

**Biometric Attacks**
- Iris photograph
  - Solution -> liveness detector
- Thumb prints – gummy-bears
  - Solution -> liveness detector

**Iris Scan Implementation**
There is an ECC that can correct 400 bit errors in a 2048 bit message and no more.

I --> [ECC] --> (I CRCi)
I, if I’ is valid } <-- [ ECC$^1$ ]
Fail, otherwise

Note: In our threat model CRC$_1$ is assumed public.