SchrodinText: Strong Protection of Sensitive Textual Content of Mobile Applications

Ardalan Amiri Sani

Motivation

- Textual content can contain extremely sensitive and private information
- Such content must only be displayed to the user but must be otherwise protected against unauthorized access malware
- This solution is to leverage novel hardware features in ARM processors to create a security monitor for showing and protecting the text

Challenges

- Operating system needs to perform the layout of text
- Once character glyphs are rasterized and their locations are determined, they must be composited on top[of other layers in the framebuffer and displayed to the user.

Overview

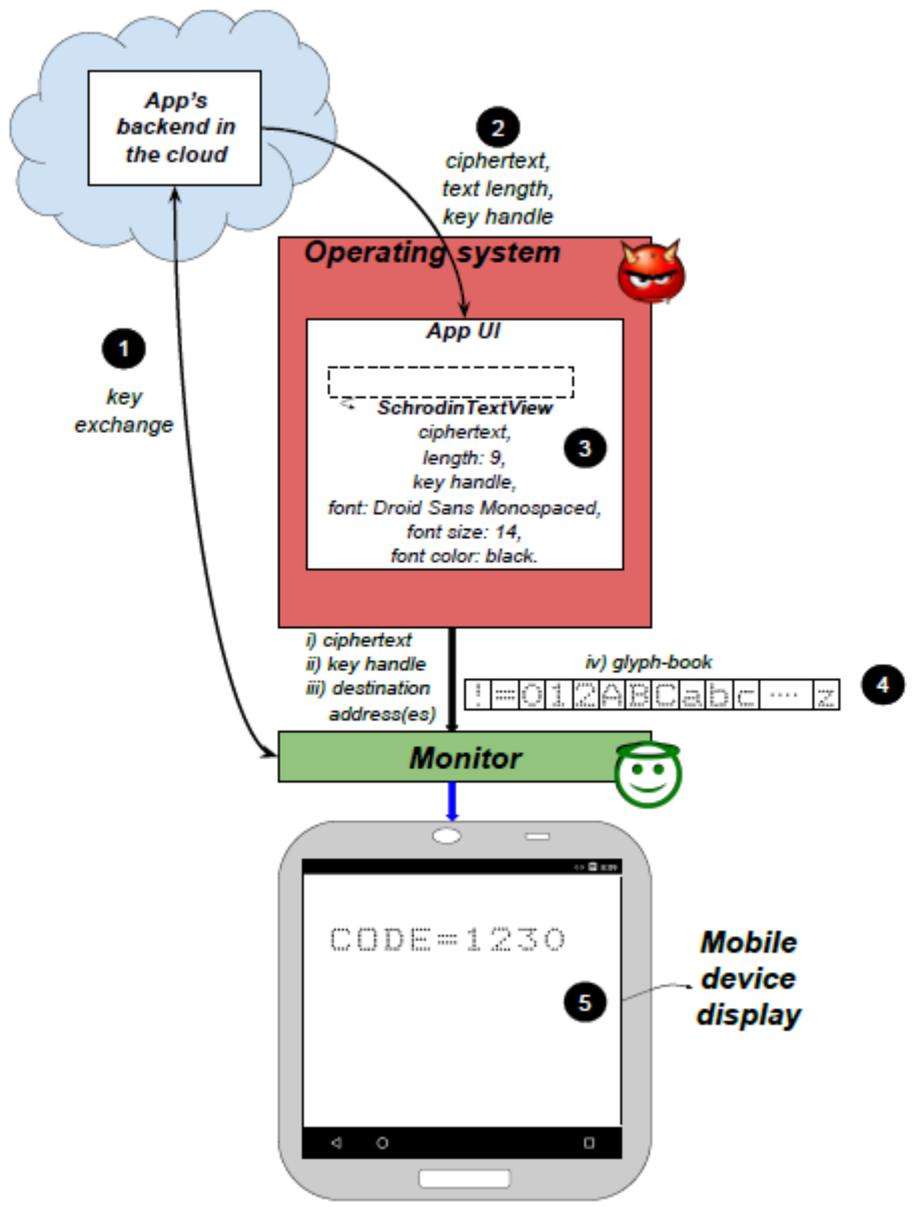
- Key Idea and Design
  - Operating system should not have access to the protected content
  - Virtualization and TrustZone
- SchrodinText

```java
void drawText(byte[] ciphertext,
              int keyHandle)
{
   String text =
      decrypt(ciphertext, keyHandle);
   TextView view = (TextView)
      findViewById(R.id.textWidget);
   view.setText(text);
}
```

```java
void drawText(byte[] ciphertext,
              int textLen,
              int keyHandle)
{
   SchrodinTextView view = (SchrodinTextView)
      findViewById(R.id.textWidget);
   view.setCiphertext(ciphertext, textLen,
                      keyHandle);
}
```

Workflow

- Backend server exchanges a key with the monitor
- Backend uses the key to encrypt the text
- Application use SchrodinTextView
- Operating system rasterizes all the Character glyphs and shares them with monitor
- Decrypt and resolve the right glyph


App's backend in the cloud
1
key exchange
2
ciphertext, text length, key handle
Operating system
App UI
SchrodinTextView
ciphertext, length: 9, key handle, font: Droid Sans Monospaced, font size: 14, font color: black.
3
i) ciphertext
ii) key handle
iii) destination address(es)
iv) glyph-book
! = 0 1 2 A B C a b c ···· z
4
Monitor
CODE=1230
5
Mobile device display

Security Monitor

- Trustzone
  - Trustzone divides the execution into two worlds
  - Secure world has access to a device-unique key, which is not accessible to the normal world
  - Cryptographic key management and operations
- Virtualization
  - It adds a new privilege mode in the normal world(hyp)
  - It can display the protected text on the framebuffer, while preventing the operating system from accessing it.

Oblivious Rendering

- Rasterization
  - Normal android's rasterization simply contain the alpha channel information
  - SchrodinText apply the color in the rasterization phase
- Layout
  - Operating system knows the number of the characters
  - Limit the SchrodinTextView to monospaced fonts

Secure Compositing

- Multi-View Pages
  - CPU MMU and various IOMMUs in the ARM SoC to provide different views of the framebuffer pages containing protected text pixels
- Two-Stage Compositing
  - Operating system composites the unprotected text using the GPU
  - Monitor composites the protected text using the CPU

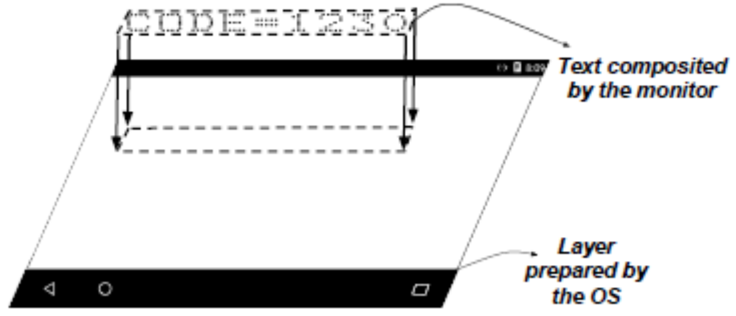
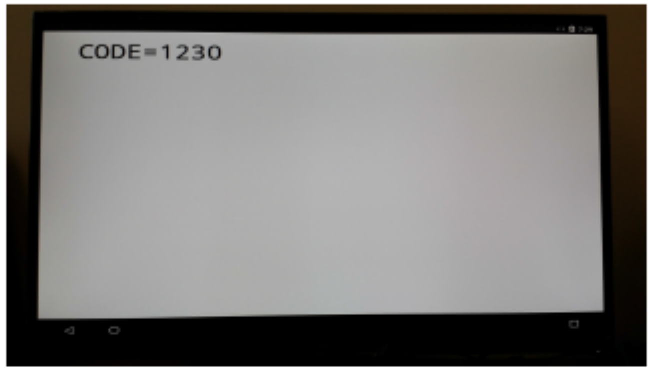
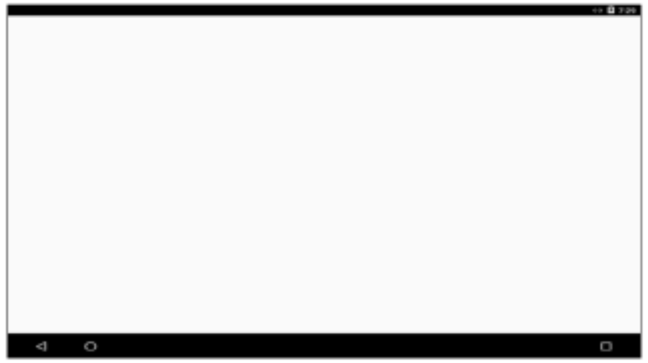


Figure 6: Secure two-stage compositing.

Evaluation



(a) What the user sees on the display



(b) What the operating system sees in the framebuffer

Figure 7: SchrodinText protects the text against framebuffer read-back attacks.

Evaluation

- Memory Usage
- CPU Usage
- Latency

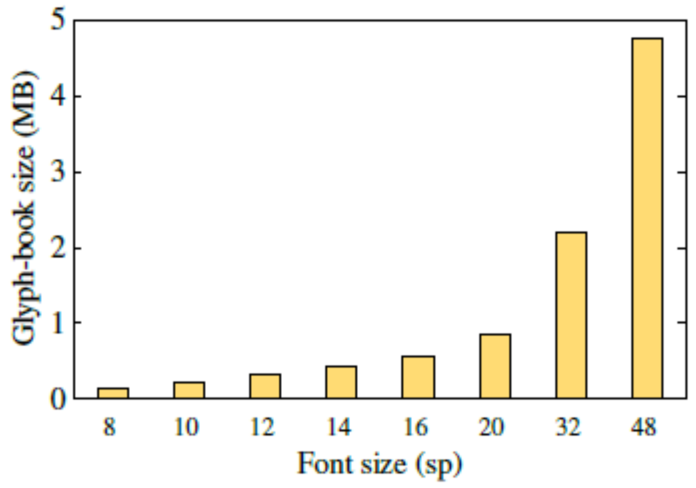
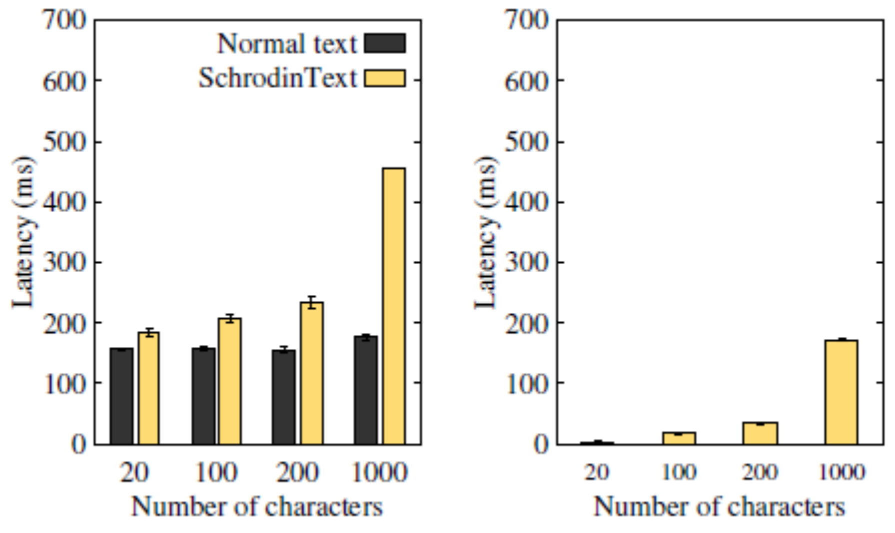
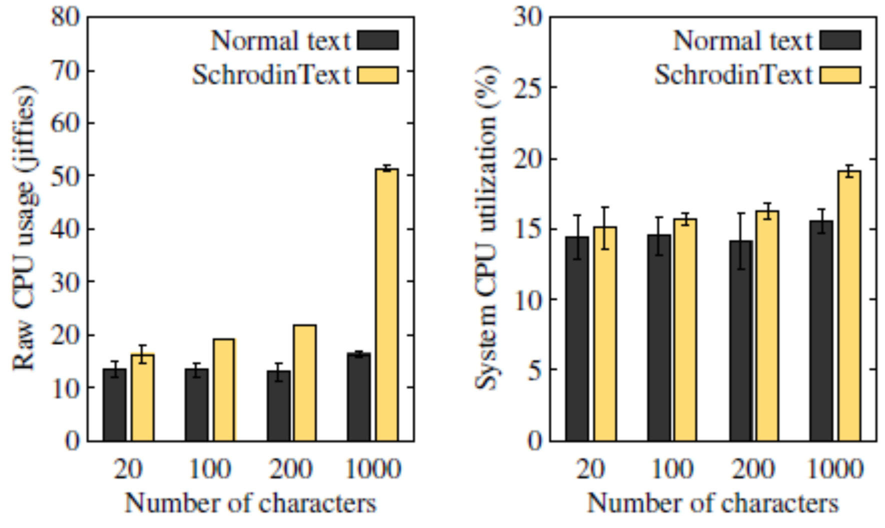


Figure 8: Size of glyph book for varying font sizes. The font size unit is in sp (scale-independent pixel).



(a) (b)



(a) (b)