

# Note on a mobile security

... or How the Brave Permutation Rescued  
a Naughty Keyboard...



Petr Dvořák - @joshis\_tweets  
iOS Development Lead



Tomáš Rosa  
Senior Cryptologist, Raiffeisenbank

# Outline

- Mobile Security Landscape
- Typical Topics in Security
- The Perils of Jailbreaking
- The Tale of the Brave Permutation

# Mobile Security Landscape

# Mobile Security Landscape

- New Devices, New Problems
- New Devices, Old Problems
- The Murderer is always the Gardener



# Typical Topics

# Incorrect Logging

- NSLog is not harmless!
- Works with the system log, readable by anyone
- AppSwitch app
- Disable NSLog for the App Store build

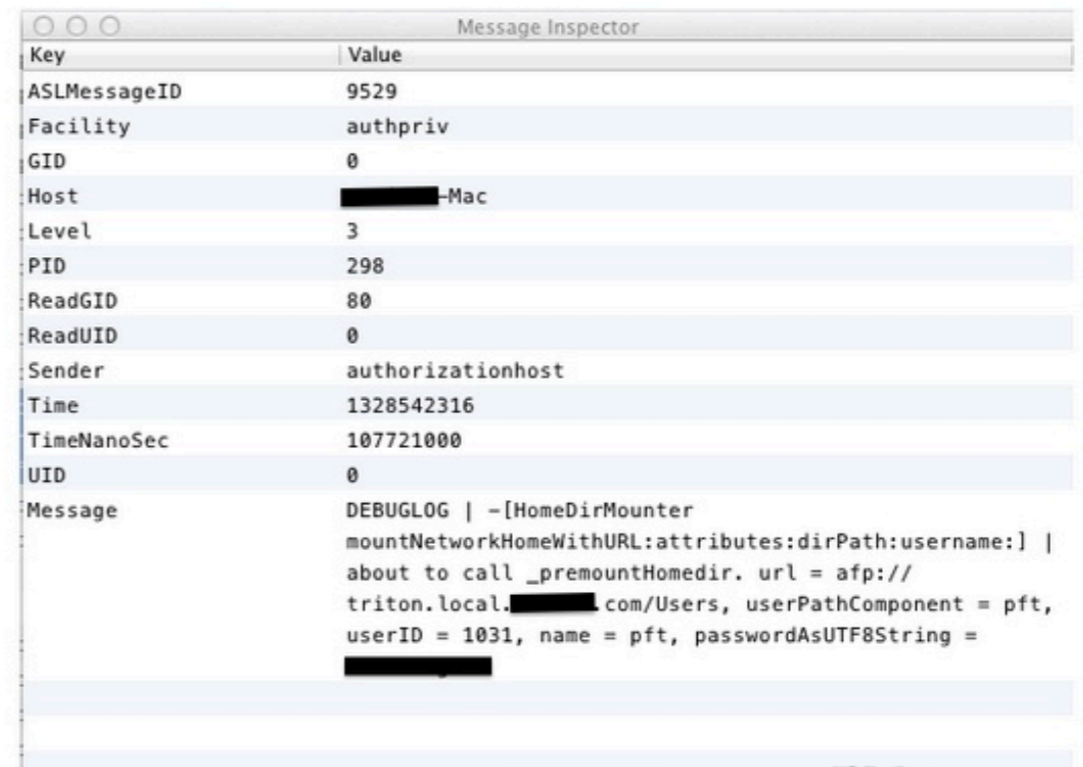
```
#define NSLog(...)
```

## Apple security blunder exposes Lion login passwords in clear text

By Emil Protalinski | May 6, 2012, 8:52am PDT

**Summary:** With the latest Lion security update, Mac OS X 10.7.3, Apple has accidentally turned on a debug log file outside of the encrypted area that stores the user's password in clear text.

**Update on May 9:** Apple releases OS X Lion v10.7.4, fixes FileVault password bug



Key	Value
ASLMessageID	9529
Facility	authpriv
GID	0
Host	██████████-Mac
Level	3
PID	298
ReadGID	80
ReadUID	0
Sender	authorizationhost
Time	1328542316
TimeNanoSec	107721000
UID	0
Message	DEBUGLOG   -[HomeDirMounter mountNetworkHomeWithURL:attributes:dirPath:username:]   about to call _premountHomedir. url = afp:// triton.local.██████████.com/Users, userPathComponent = pft, userID = 1031, name = pft, passwordAsUTF8String = ██████████

# Incorrect SSL handling

- SSL != Super Secure Line
- iOS Checks if CA is trusted
- OCSP only for EV certificates, works best attempt
- <http://mitmproxy.org>

CNET News Geek Gestalt

14 comments

29 60 18 7

More +

## Path shares photos--oh, and uploads your contacts, too

The popular photo sharing app is rocked by news that it uploads contacts from iPhone users without permission.

by Daniel Terdiman | February 7, 2012 3:21 PM PST

Follow



Path founder Dave Morin, speaking at the 2011 Web 2.0 Summit.  
Credit: CNET James Martin

© 2012 CBS Interactive

# MITMProxy

```
petrdvorak — mitmproxy — Python — 80x24
mitmproxy
GET http://touch.www.linkedin.com/li/v1/pages/mailbox?nc=1341550926526
GET http://touch.www.linkedin.com/li/v1/people/34765372/connections?count=500
0&nc=1341550926526
<- 200 application/json 17.61kB
POST http://touch.www.linkedin.com/li/v2/metrics
GET http://touch.www.linkedin.com/li/v1/pages/home?start=0&count=60&nc=134155
0926526
GET http://touch.www.linkedin.com/li/v1/pages/init?nc=1341550926526
GET http://touch.www.linkedin.com/li/v1/people/person?nc=1341550926526
>> POST http://touch.www.linkedin.com/li/v2/metrics
GET http://media.linkedin.com/mpr/mpr/shrink_80_80/p/2/000/049/3e4/1c87df4.jp
g
GET http://static01.linkedin.com/scds/common/u/img/icon/icon_no_photo_no_bord
er_60x60.png
[28] [i:~q] ? :help [*:8080]
```



# NSURLConnection callback

```
- (BOOL)connection:(NSURLConnection *)connection
    canAuthenticateAgainstProtectionSpace:(NSURLProtectionSpace*)space
{
    SecTrustRef trust = [space serverTrust];
    SecCertificateRef cert =
        SecTrustGetCertificateAtIndex(trust, 0);

    NSData* serverCertificateData =
        (NSData*)SecCertificateCopyData(cert);

    NSString* description =
        (NSString*)SecCertificateCopySubjectSummary(cert);

    // check the data... "if (isOk(cert)) { phew(@"It's OK"); }"
}
```

# Insufficient design

- Too much weight on HTTPS
- Typical “session” is not always enough
  - Use HOTP / TOTP
- Study OAuth: Despite popular belief,  $2 < 1$

Jailbreak

# root || !(2\*root)

- You mustn't jailbreak!
- Jailbreaking = Full root access
- Attackers are tough & pretty smart!
- That sucks. Anything is possible
- The physics stops working



Saurik

#HITB



# root || !(2\*root)



Kelišová

- You must jailbreak!
- Users are uninformed + don't care
- JB can happen without users consent
- ... this is what exploits are about...
- Save them! Make your app ready for this

# Demo - Cycrypt

# root || !(2\*root)

- Considering Jailbreak makes things hard
- Dealing with security on application level
- One of many issues: How to protect the password?

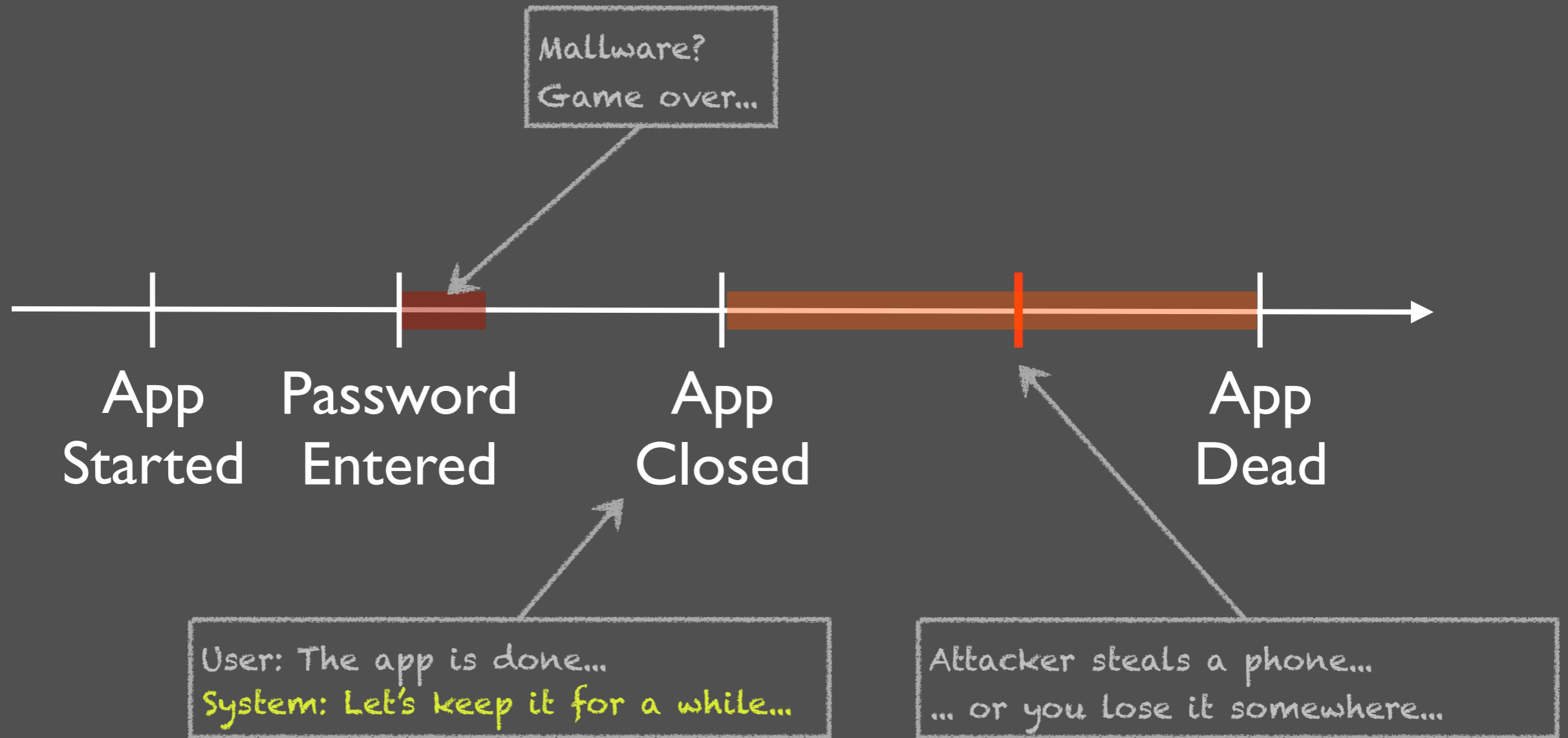


How to protect the  
password?

# How to protect password?

- Malware on the phone = game over
  - Password is stolen once you type it
- What about a stolen phone?
  - ... wait, why is it different from malware?

# iOS App in Action

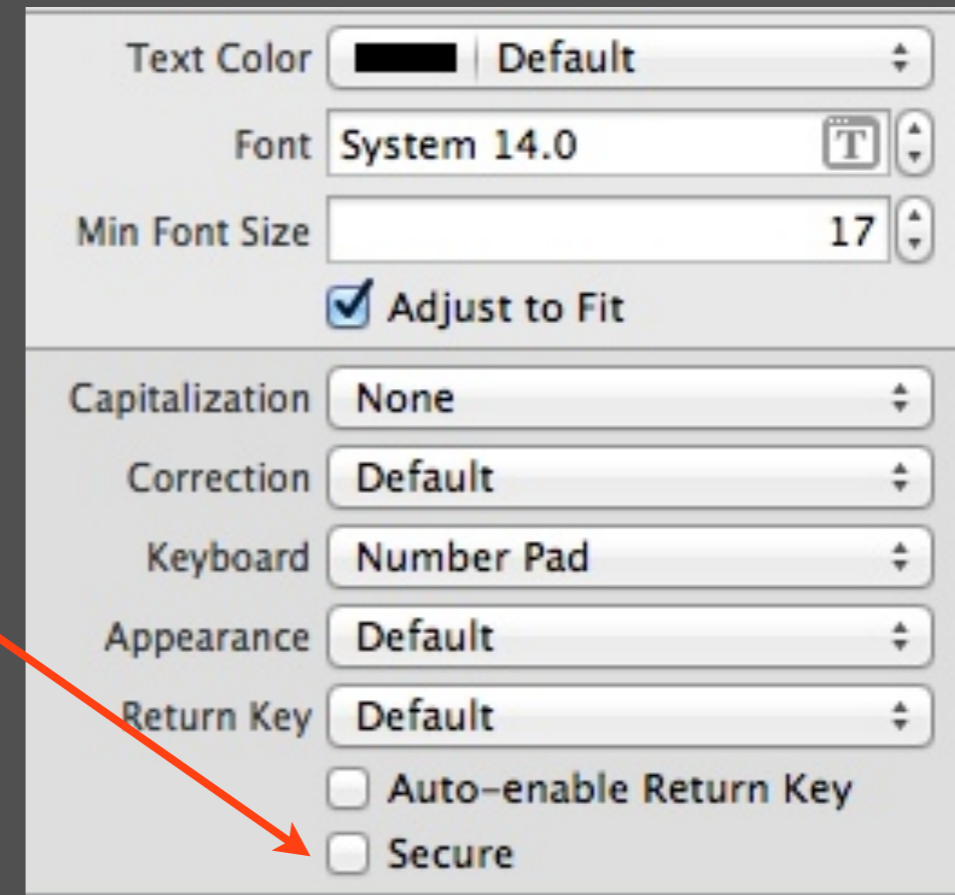


iOS Docs: “The system [iOS] keeps suspended apps in memory for as long as possible, removing them only when the amount of free memory gets low.”

# Tale of a Brave Permutation

# The Problem

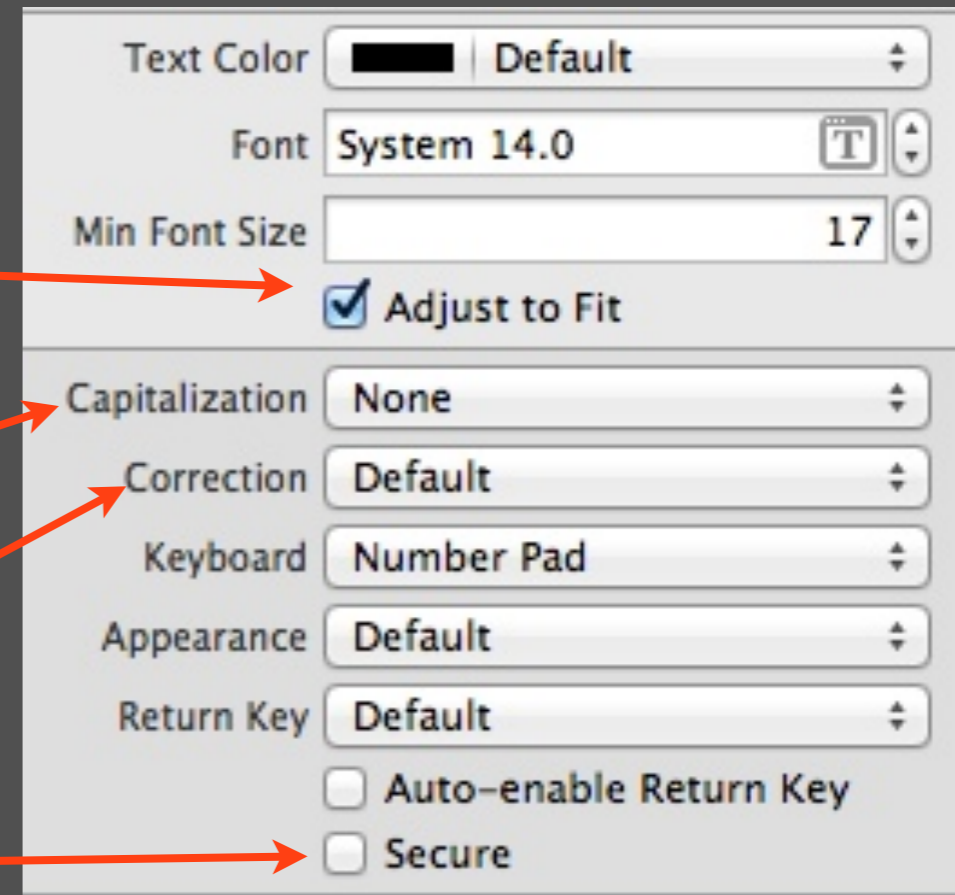
- UITextField is very, very naughty
- Even when it's "Secure", it's not secure...
- How to eliminate password footprint?



# Demo - GDB

# UITextField Properties

- !!! You need to set
  - Adjust to Fit
  - Auto-capitalization
  - Auto-correction
  - Secure
- Not Apple-like. And is it really enough?

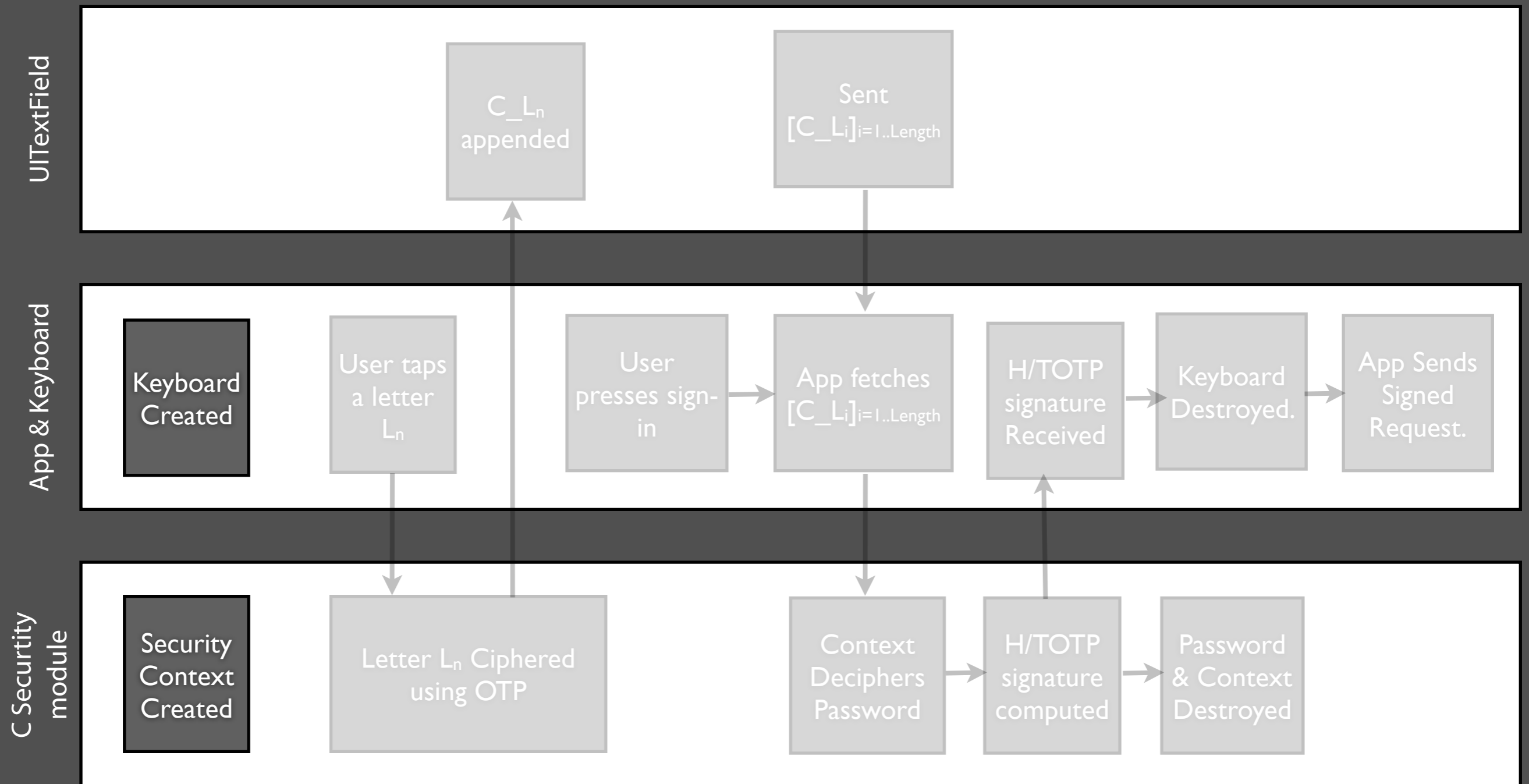


# Framework / Application

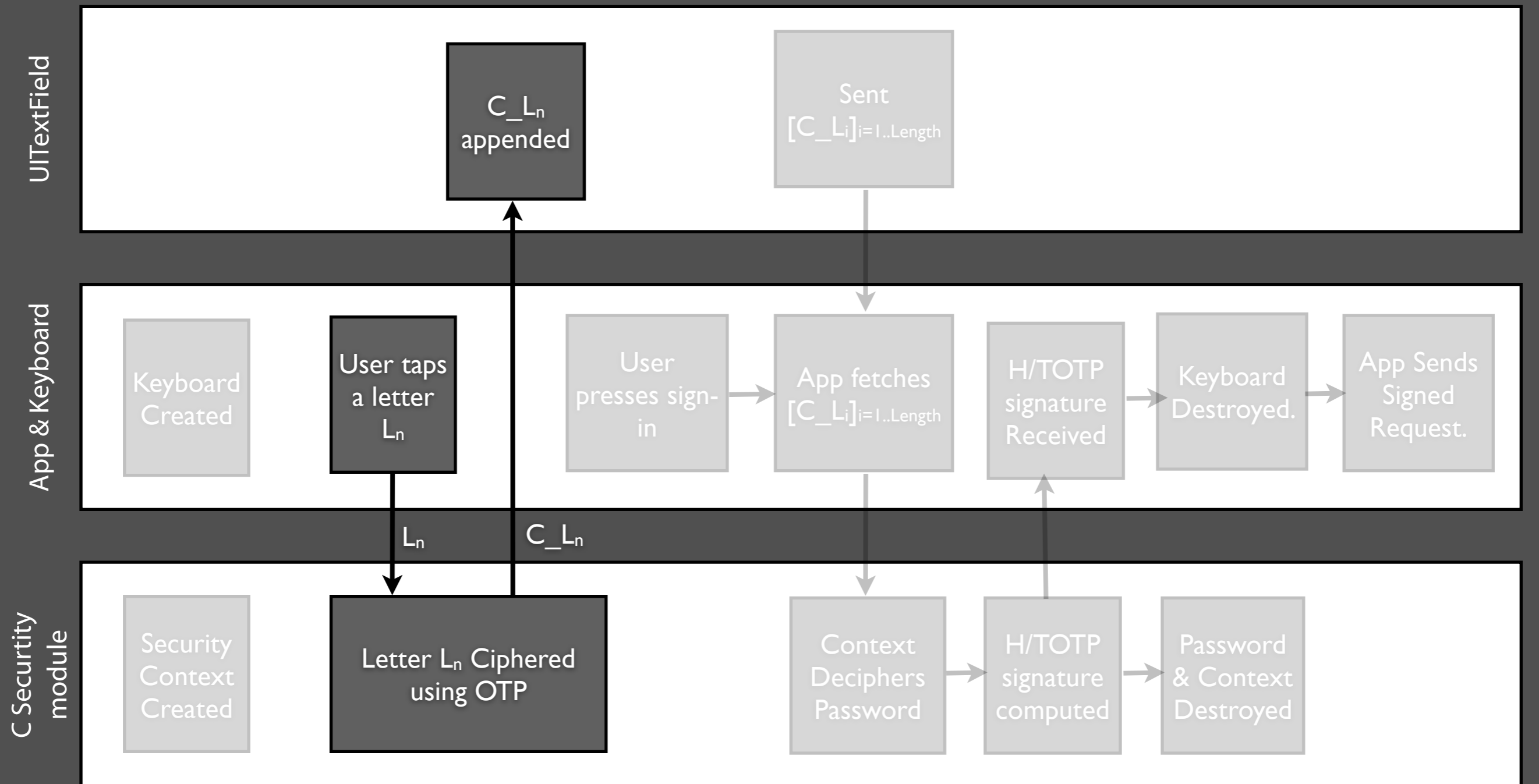
- Let's do better!
- Idea
  - Custom keyboard
  - One-Time Pad (Vernam cipher)
  - Security context under strict control
  - C implementation



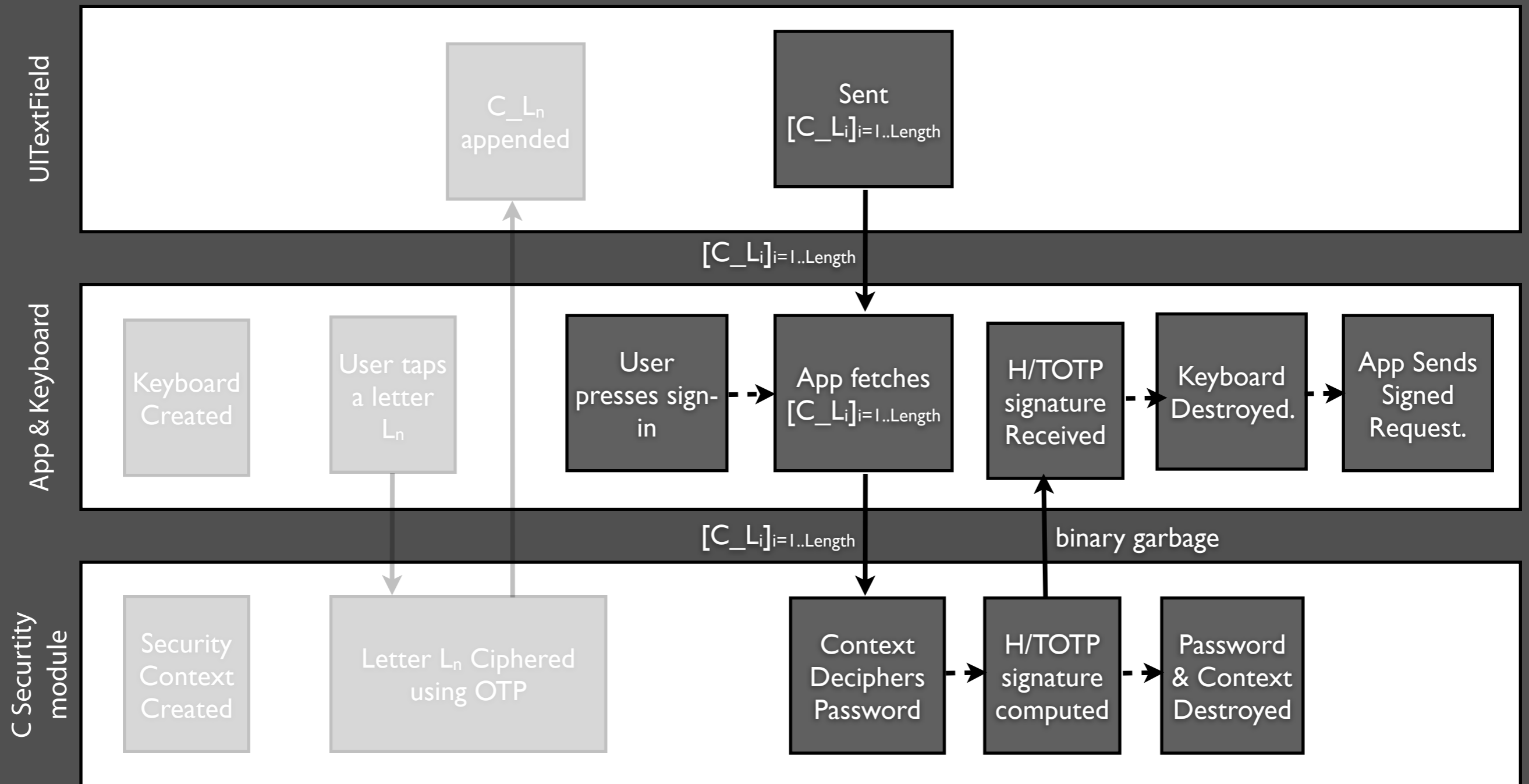
# Mechanism illustration



# Mechanism illustration



# Mechanism illustration



How to (de)cipher the  
text?

# Preconditions

- Decimal PIN of 4 to 8 digits.
- Unpredictable cursor shifts are allowed.
- UITextField must be able to process the crypto-chars.
- The encryption/decryption as well as the setup phase shall be pretty fast.

# Permutation tables

- To encrypt a PIN digit, we use a particular permutation table  $r_i : \{0, \dots, 9\} \rightarrow \{0, \dots, 9\}$ .
- Each permutation table is chosen randomly from the set of all possible  $10!$  ( $=3\,628\,800$ ) bijective mappings.

# Table Generator

- There is an algorithm that for each permutation on  $n$ -element set computes a unique number  $k$ , such that:
  - $0 \leq k < n!$
- It was already noted in [1] that we can obtain a fast permutation generator by running this algorithm backwards.
- So called shuffling, cf. [1], algorithms 3.3.2P and 3.4.2P.

# Compact Key For Tables

- Instead of generating random nonces for each generator cycle (as suggested in [1]), we generate just one random key  $k$  with uniform distribution on  $\langle 0, \dots, n! \rangle$ .
- According to the factorial number system [1], such  $k$  uniquely describes the particular permutation on  $n$ -element set.
- We then run alg. 3.3.2P in the simple reverse order.



# Generator Properties

- It can be easily shown that our approach is equivalent to generating random tweets for each pass through the main cycle of the reversed alg. 3.3.2P.
- We just collect all these nonces in one number using the wonderful factorial number system.
- Of course, there is an independent fresh  $k$  for each table generated.

# Setup Phase

- We subdivide the 7-bit ASCII set to 9 code pages by 10 characters each:
  - $\{32, \dots, 41\}, \{42, \dots, 51\}, \dots, \{112, \dots, 121\}$ .
- We also generate 9 independent keys and their corresponding permutation tables:
  - $(k_1, \dots, k_9) \rightarrow (r_0, \dots, r_8)$ .

# Encryption

- To encrypt  $j$ -th character typed  $p_j$ , we choose the permutation  $r_i$ , where  $i = j \bmod 9$ , and compute:
  - $c_j = r_i ( p_j ) + 10*i + 32$ .
- The counter  $j$  is incremented with each character encrypted regardless possible cursor shifts, etc.

# Decryption

- To decrypt a crypto-char  $c$ , we first decide which table was used for its encryption:
  - $i = (c - 32) \text{ div } 10$ .
- Then we use the inverse permutation to obtain the original plaintext char:
  - $p = r_i^{-1} (c - 10*i - 32)$ .
- We prepare both  $r_i$  and  $r_i^{-1}$  tables in setup.

# Why This Way?

- To allow unpredictable cursor shifts
- we use the code page offset to encode the keystream index  $i$  within each crypto-char.
- To eliminates the risk of compromising the whole table when the keystream index  $i$  accidentally repeats
- we use the general permutation tables instead of a simple finite group operation like xor, add, mul, etc.

# Cautionary note

- This was pretty clever, right?
- Don't spoil it by doing something stupid.
- Wipe out all the keys and permutation tables after having finished!

Thank you!

# References

- [1] Knuth, D.-E.: The Art of Computer Programming / Vol. 2 - Seminumerical Algorithms, 3rd ed., Addison-Wesley, 1998.
- [2] <http://developer.apple.com>
- [3] <http://theiphonewiki.com>
- [4] <http://www.cycrypt.org>