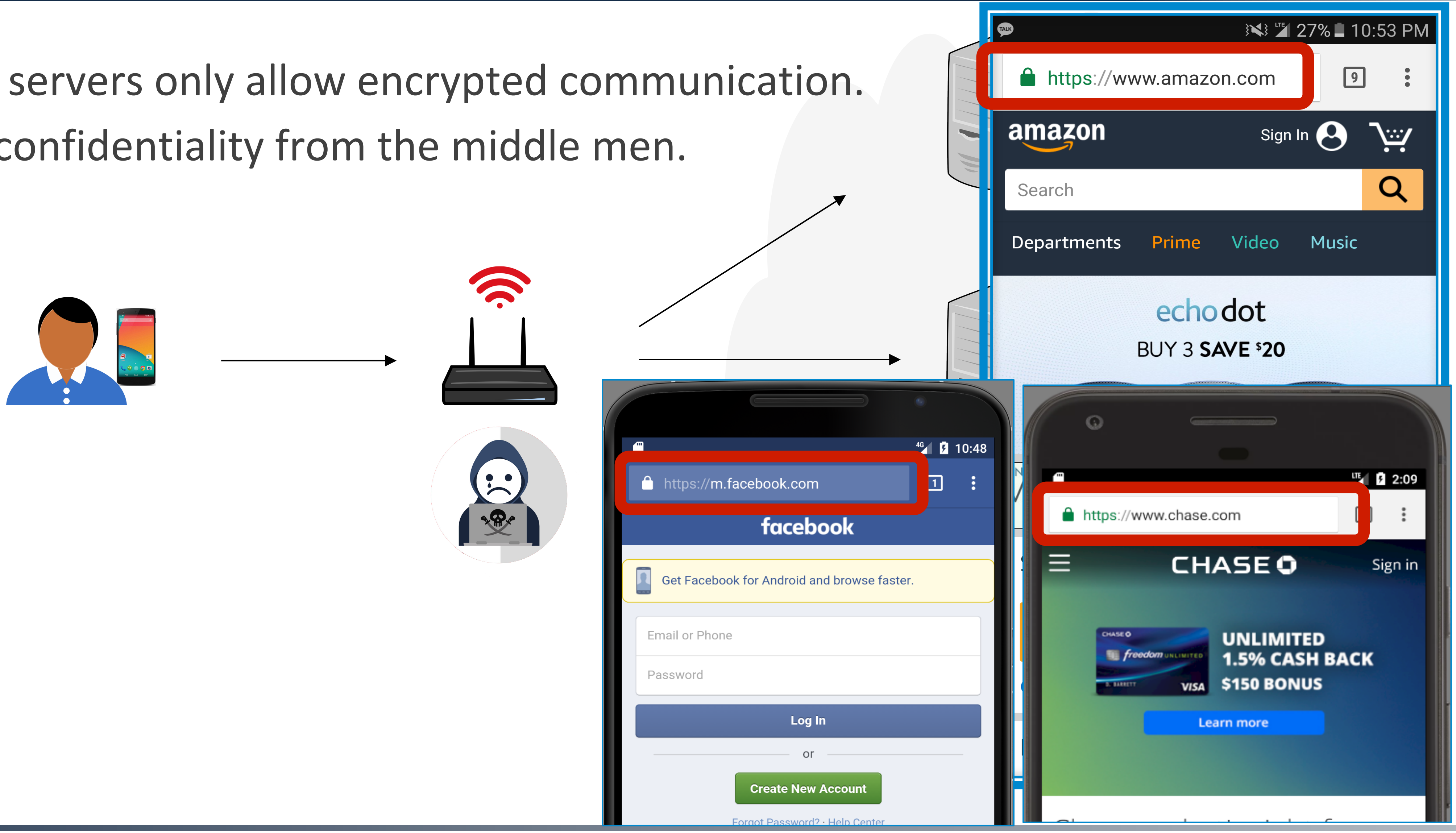
Phone users are easily exposed to insecure Wi-Fi.

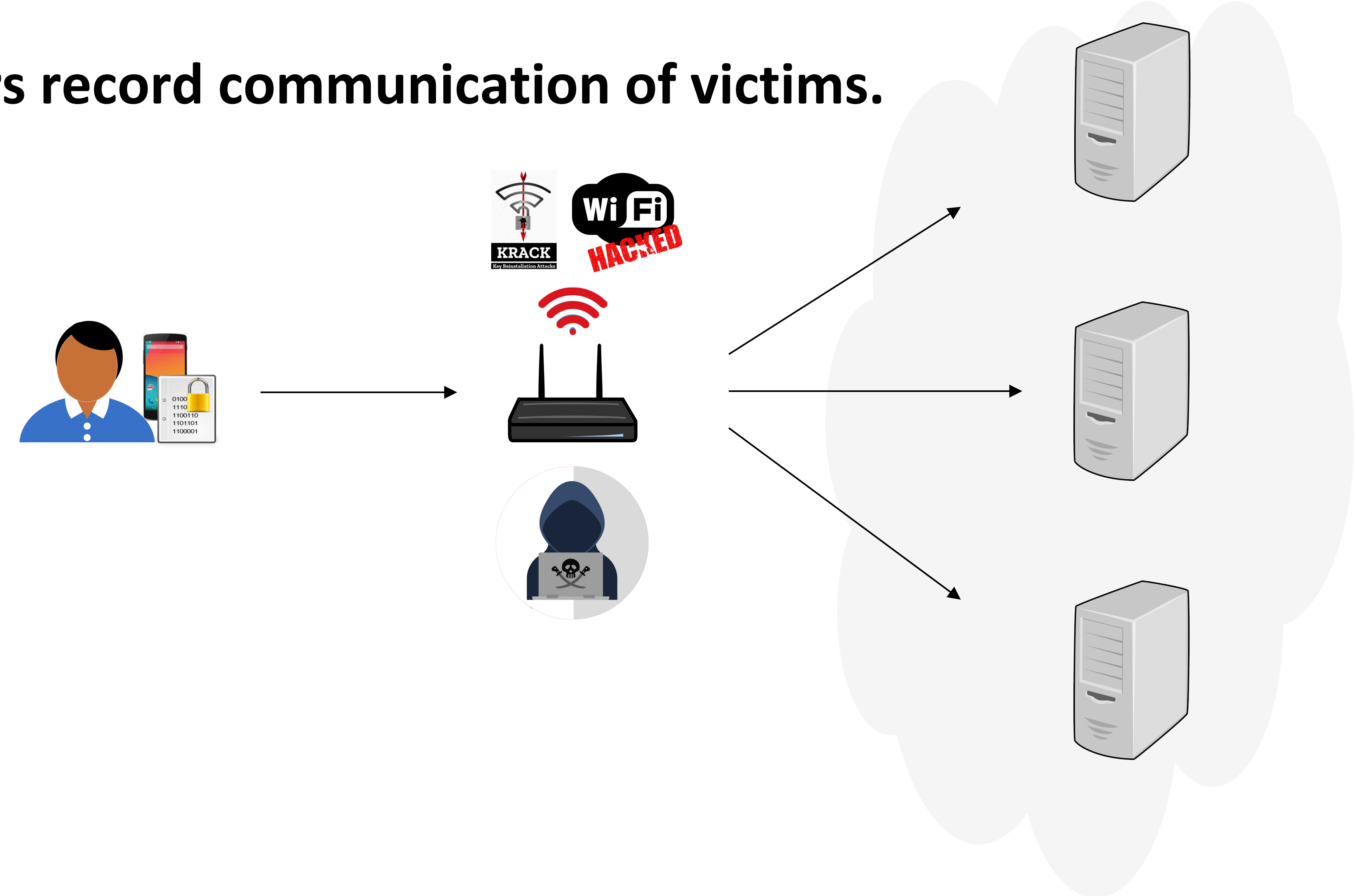
- Today major servers only allow encrypted communication.
- TLS ensures confidentiality from the middle men.



RICE

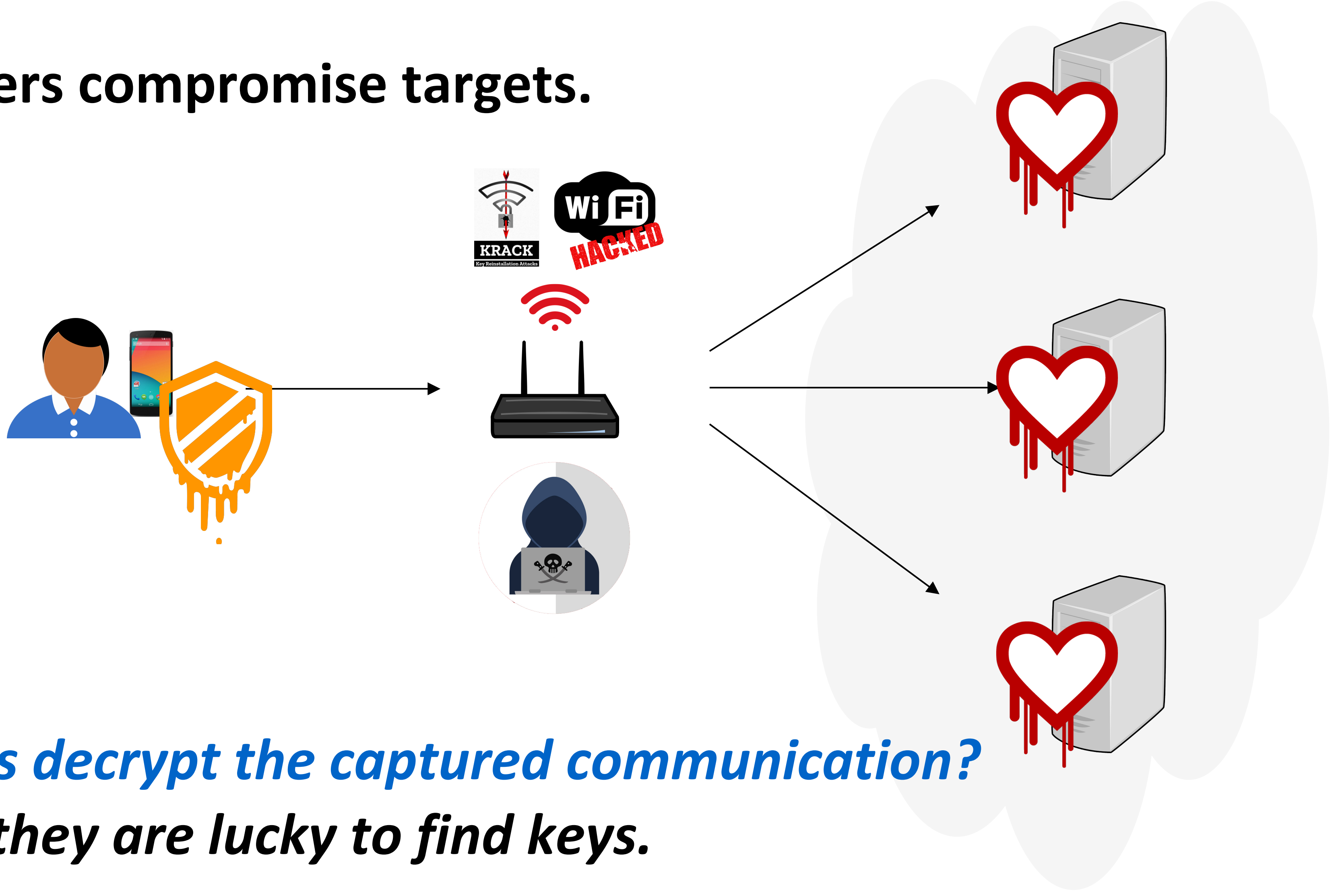
Threat Model

The attackers record communication of victims.



Threat Model

Later, attackers compromise targets.



Can attackers decrypt the captured communication?

Maybe, if they are lucky to find keys.

TLS Cryptosystem should resist this threat.

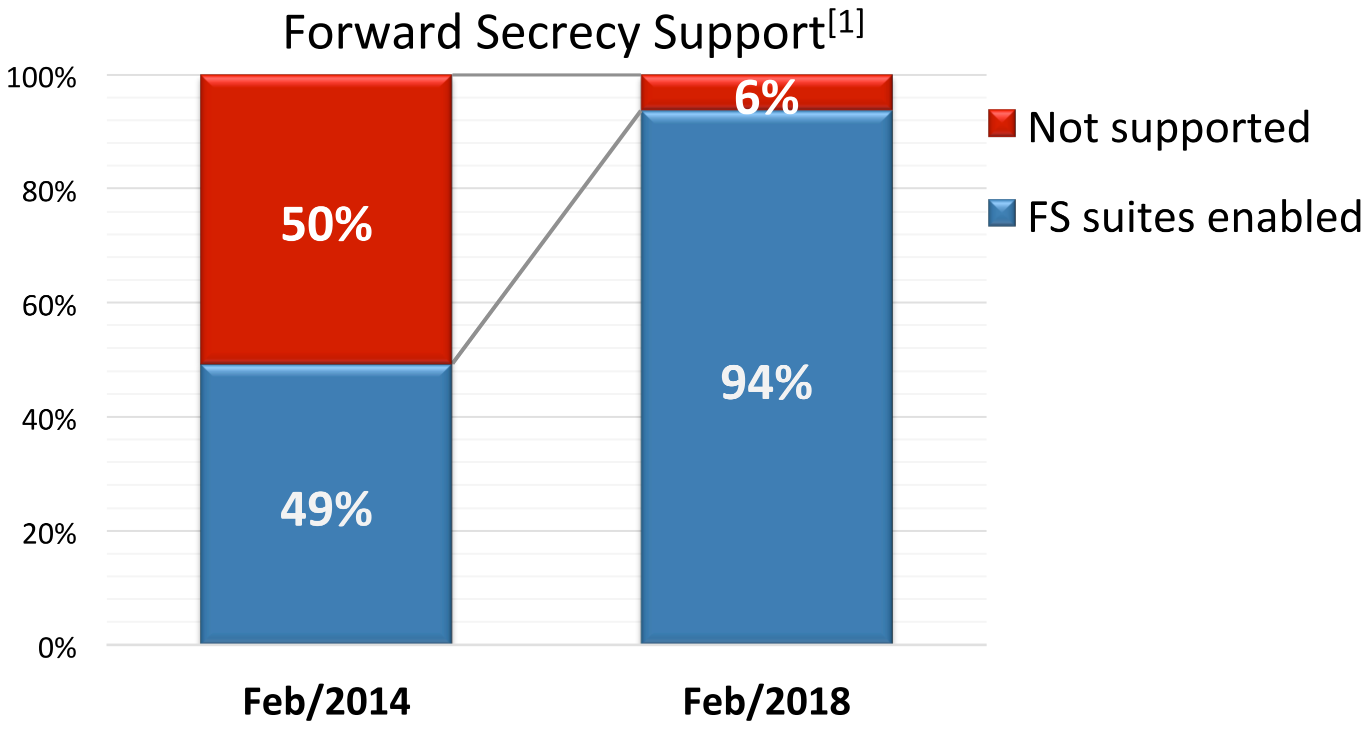
Various tactics are used to protect against future compromises.

- Long-term key material
- Short-term key material

TLS Cryptosystem should resist this threat.

Various tactics are used to protect against future compromises.

- Long-term key material: **Perfect forward secrecy**
- Short-term key material



[1] https://www.ssllabs.com/ssl-pulse/

TLS Cryptosystem should resist this threat.

Various tactics are used to protect against future compromises.

- Long-term key material: **Perfect forward secrecy**.
- Short-term key material: **TLS implementations** have responsibility.
  - OpenSSL goes to great length to clean up ephemeral keys rapidly.

```
 void *OPENSSL_clear_realloc(void *p, size_t old_len, size_t num)
 void OPENSSL_clear_free(void *str, size_t num)
 void OPENSSL_cleanse(void *ptr, size_t len);
 void *CRYPTO_clear_realloc(void *p, size_t old_len, size_t num, const char *file, int
line)
 void CRYPTO_clear_free(void *str, size_t num, const char *, int)
```

Research Question and Motivation

What about Android?

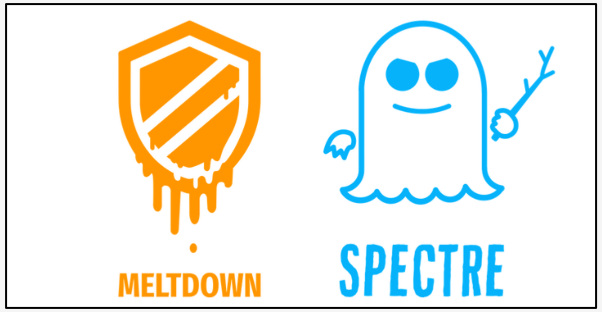
- Are previous communications safe under ***memory disclosure attack***?

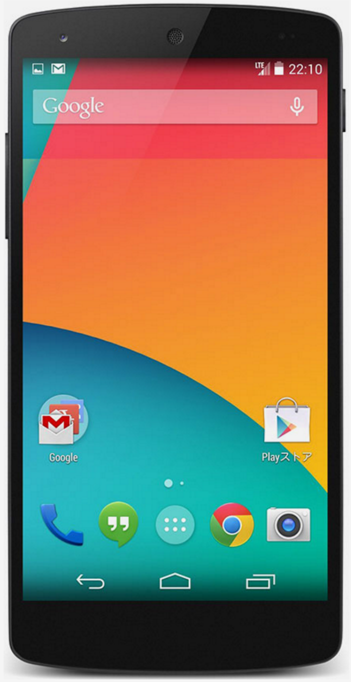
Motivation

1. Threat model is more practical.

Research Question and Motivation

By software exploitations

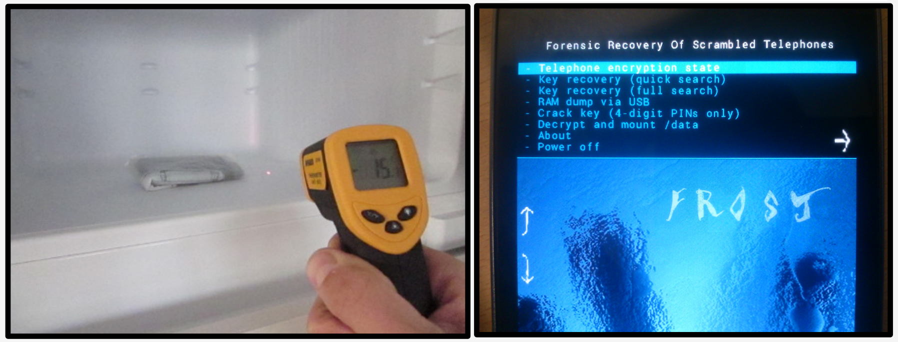




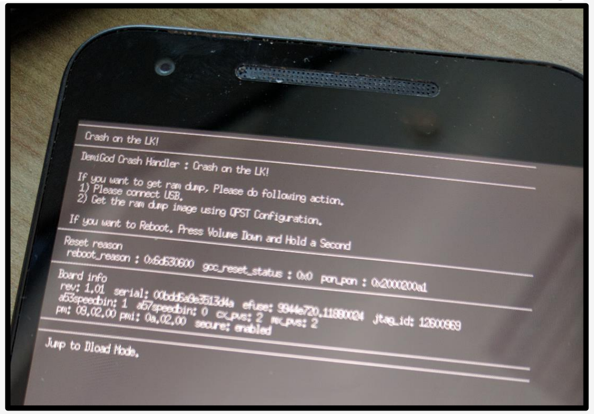
Android has various attack vectors.♪

By physical techniques

Cold-boot attack



Nexus 5X bootloader vulnerability



Research Question and Motivation

What about Android?

- Are previous communications safe under ***memory disclosure attack***?

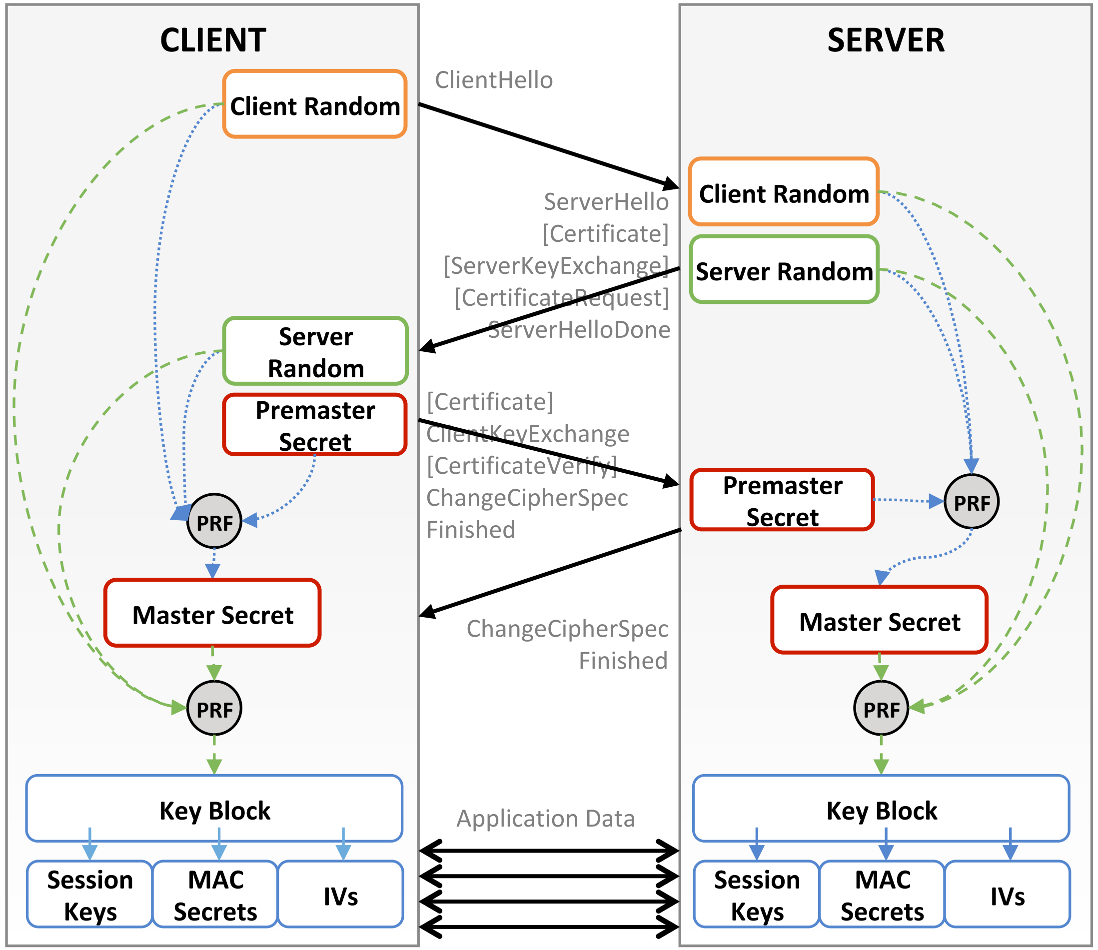
Motivation

1. Threat model is more practical.
2. Managing secrets on memory would be more challenging.
   - Multiple software layers
   - Complex application lifecycle

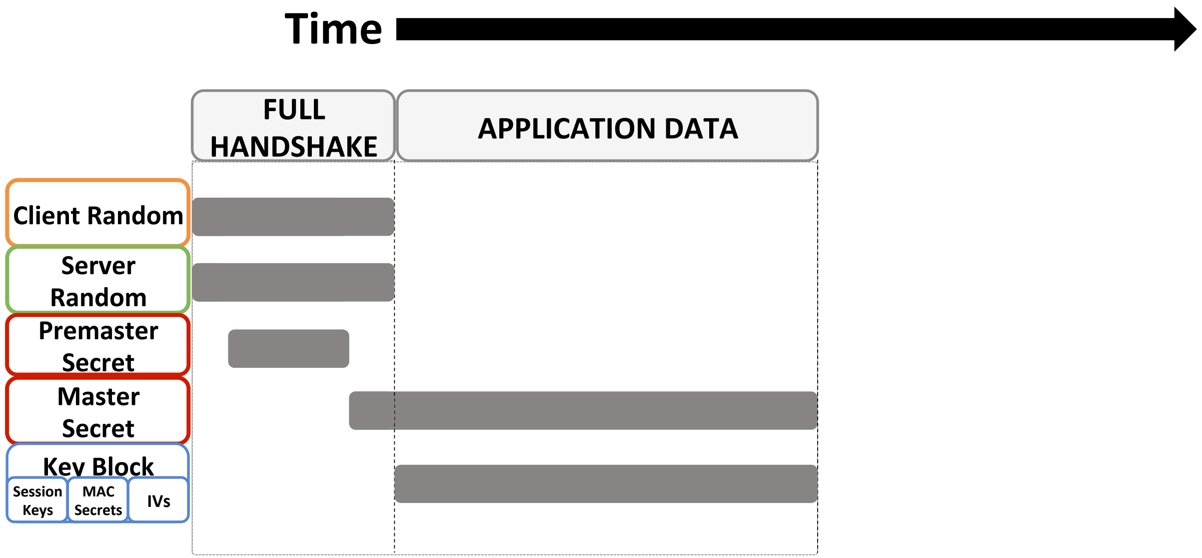
Let's see how Android TLS deals with those issues.

RICE

Background: Secrets on TLS

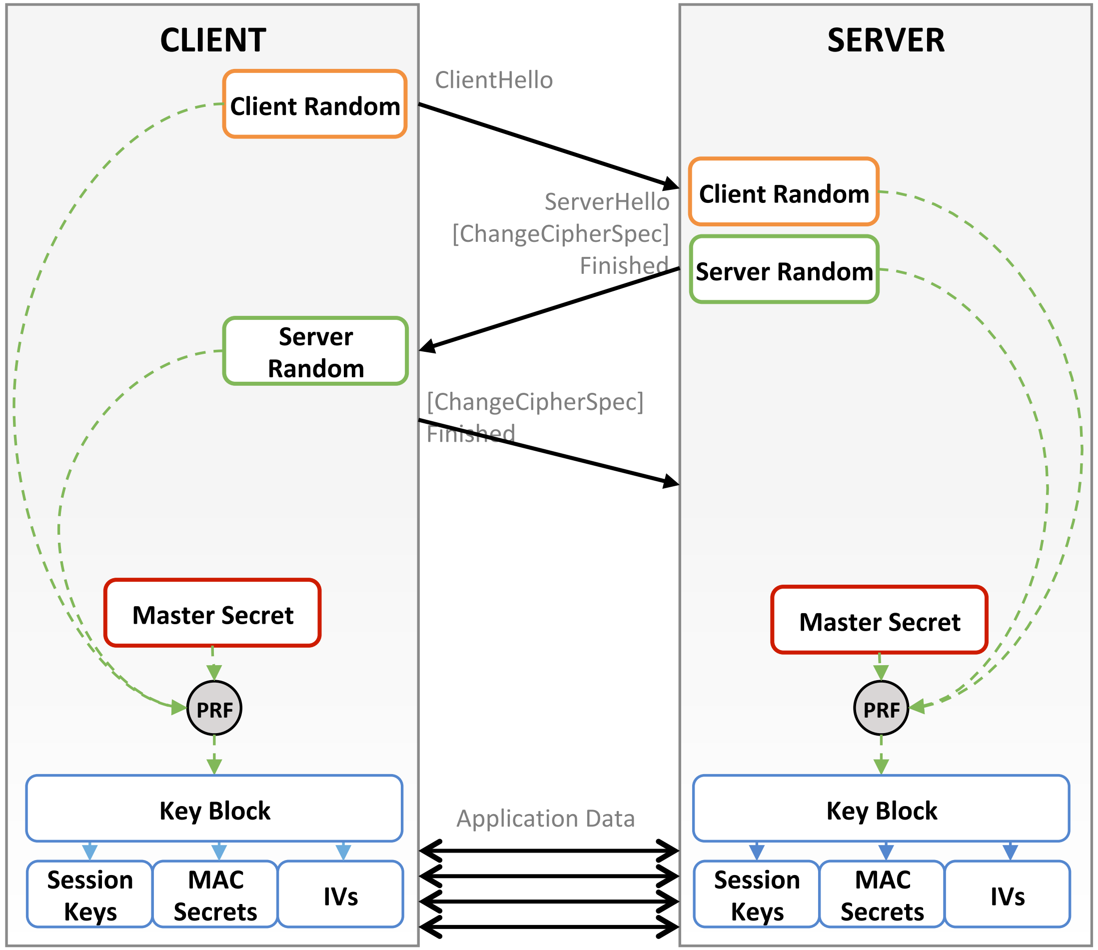


TLS Full Handshake

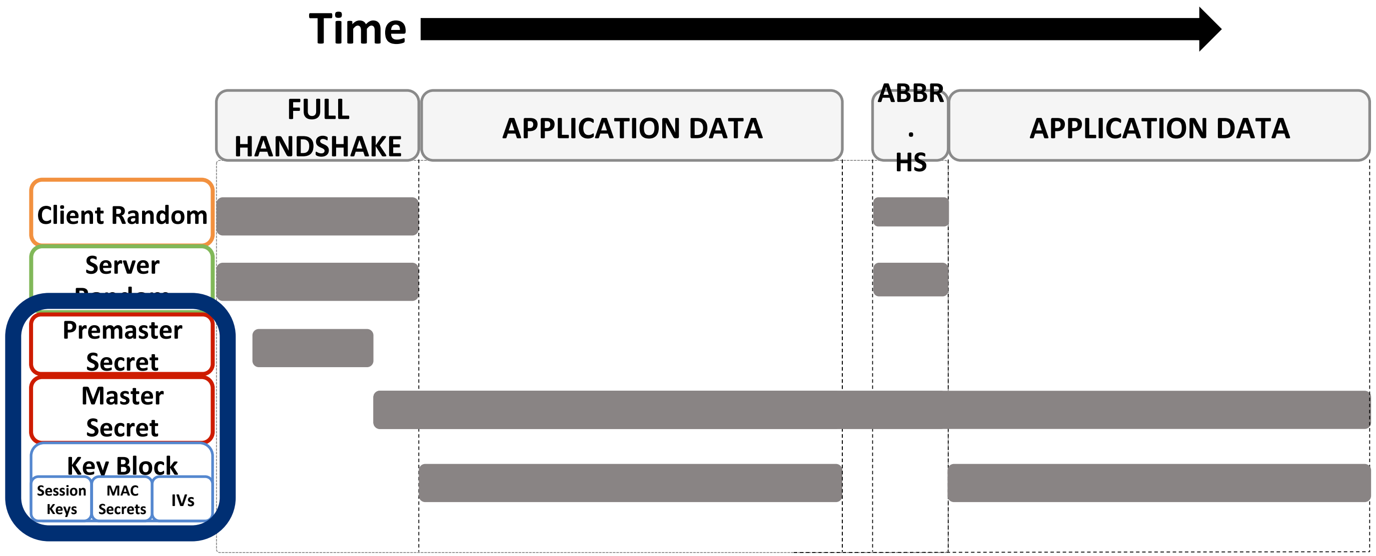


Lifetime of Secrets

Background: Secrets on TLS



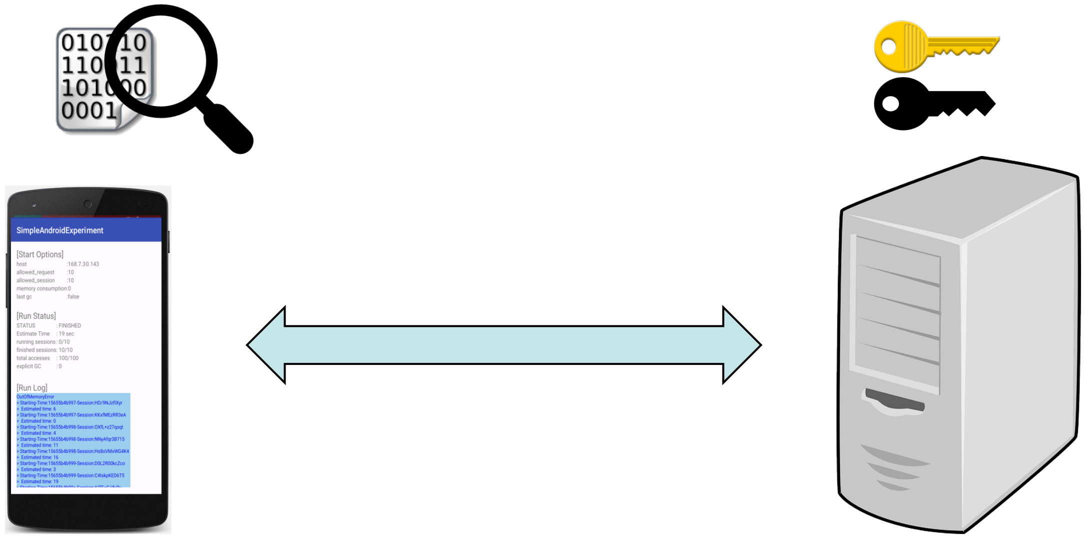
TLS Abbreviated Handshake



Lifetime of Secrets

Black-Box Security Analysis

1. Establishing TLS Connections
2. Logging the keys during the handshake
3. Dumping Android’s memory
4. Searching keys from the memory dump

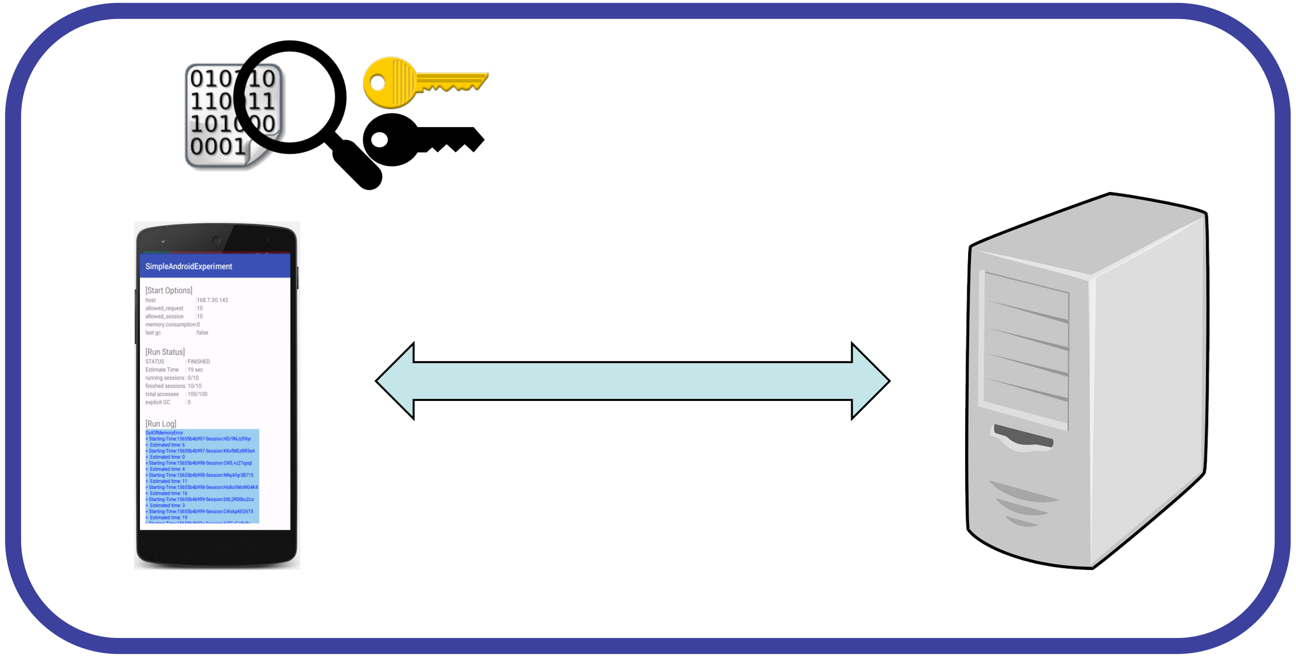


Black-Box Security Analysis
Experiment

Repeating

- Different version: Emulators (Ver 4, Ver 5, Ver 6, Ver 8) and Nexus 5
- Performing additional actions

Test Framework *supporting automation♪*



Black-Box Security Analysis
Key Result of Experiment

The results are almost same for all the cases regardless of versions.

Premaster Secret	✗
Master Secret	✓
Key Block (Session Key)	✗

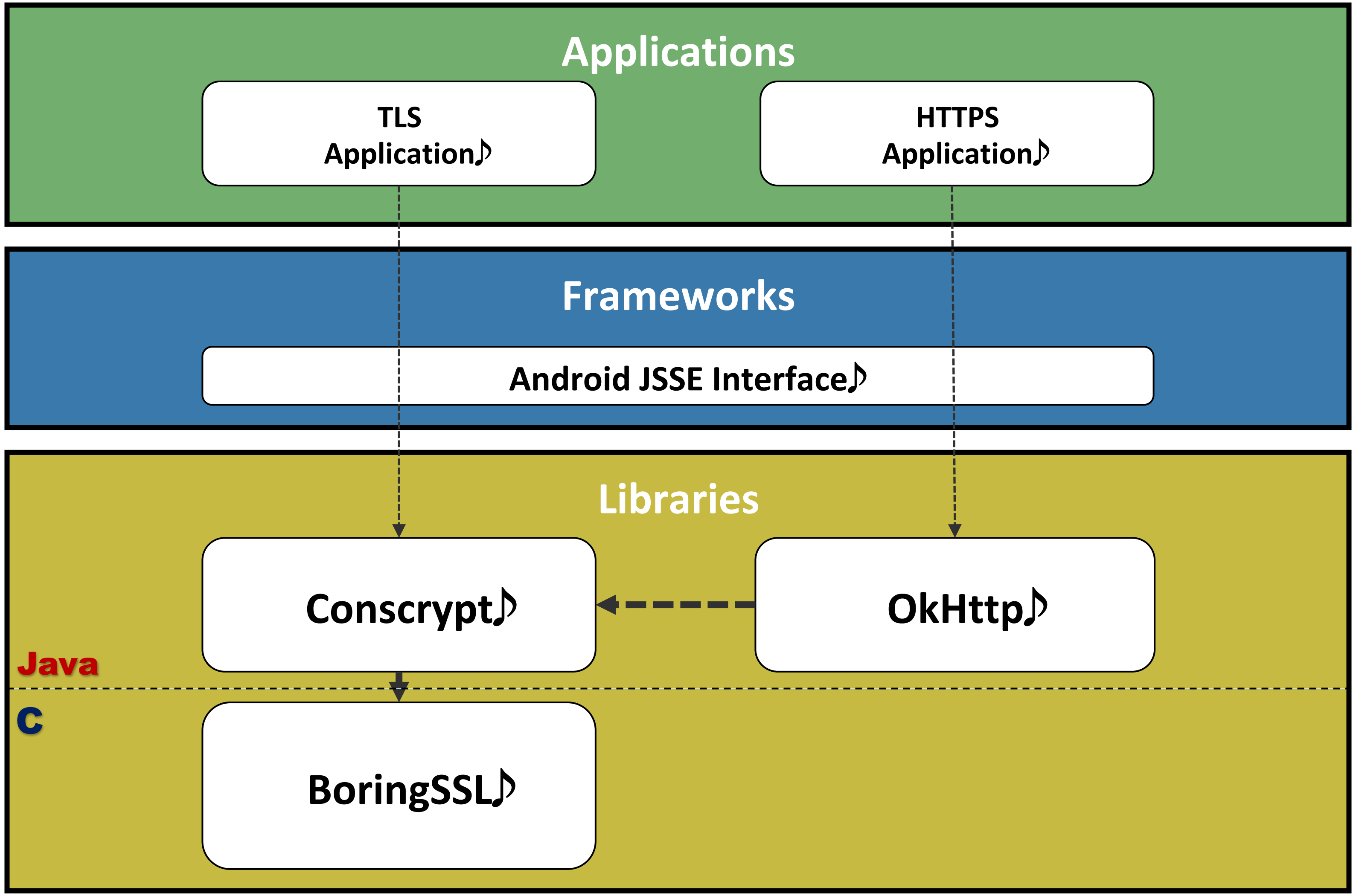
But, Why?
Is this a bug or intended?

Master secrets are found regardless of different actions.

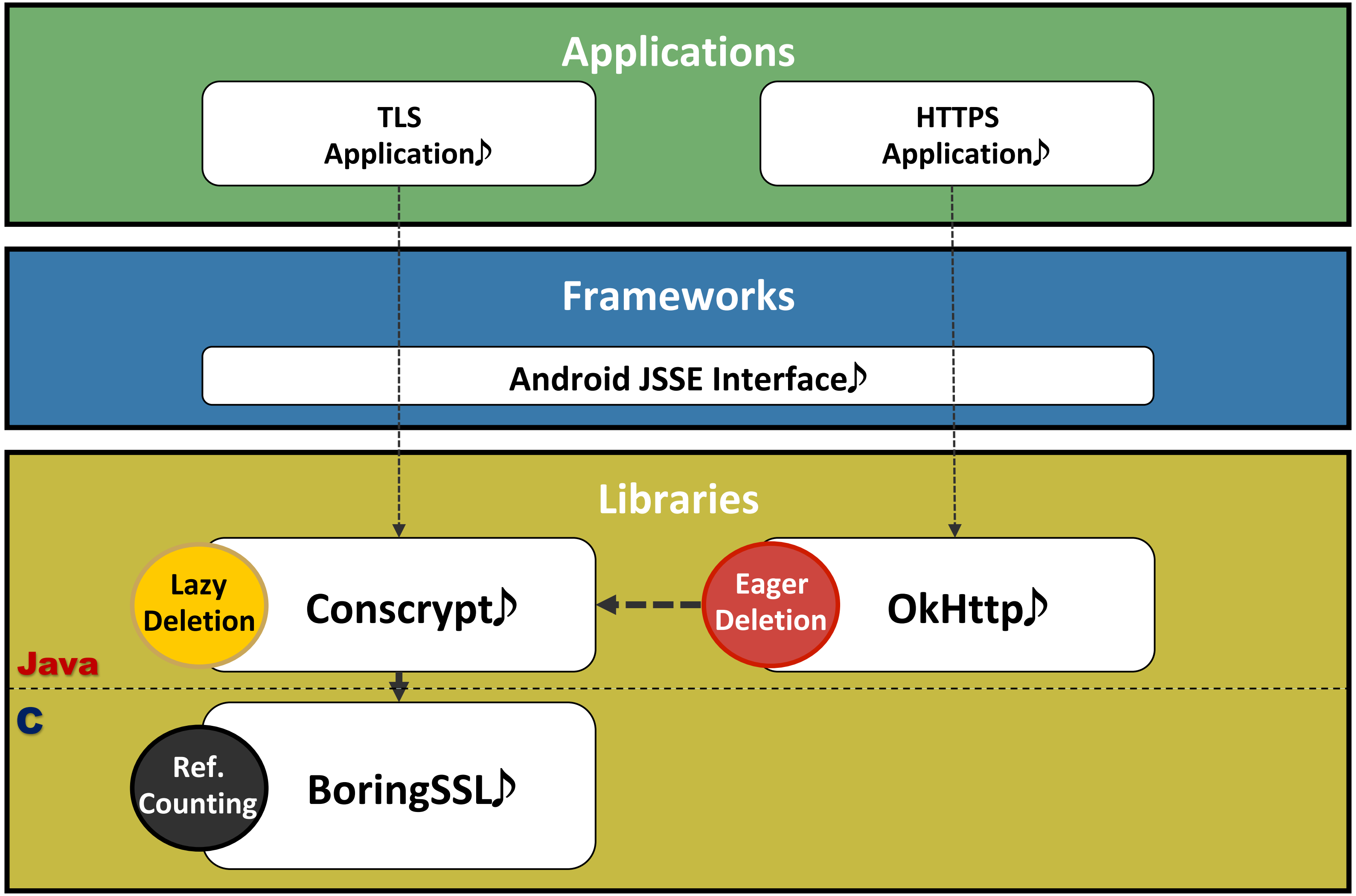
- Moving apps to background.
- Forcing garbage collection.
- Killing apps.

Developers cannot control this retention.

In-depth Analysis
Android TLS Stack

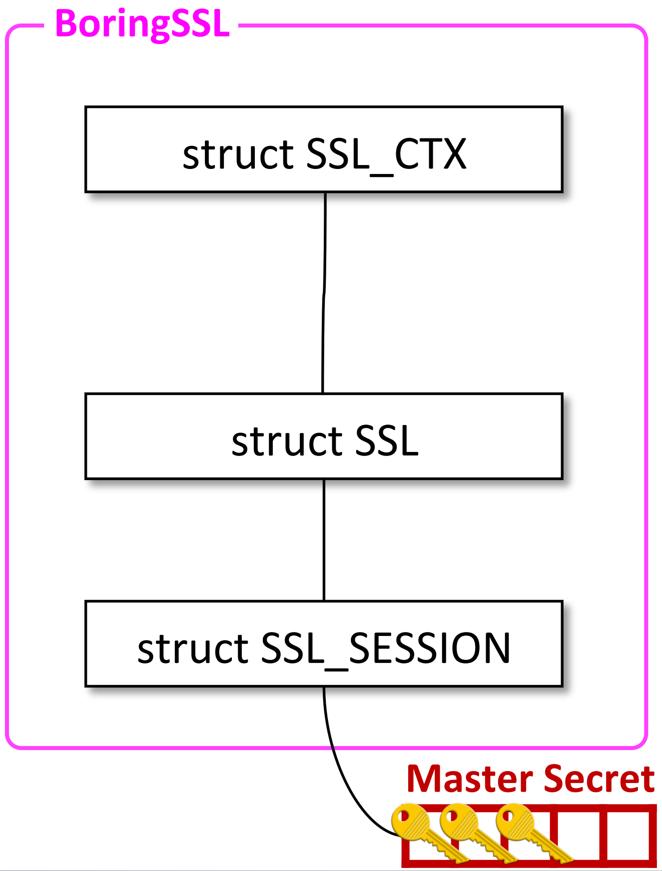


Problem: Inconsistency in object management



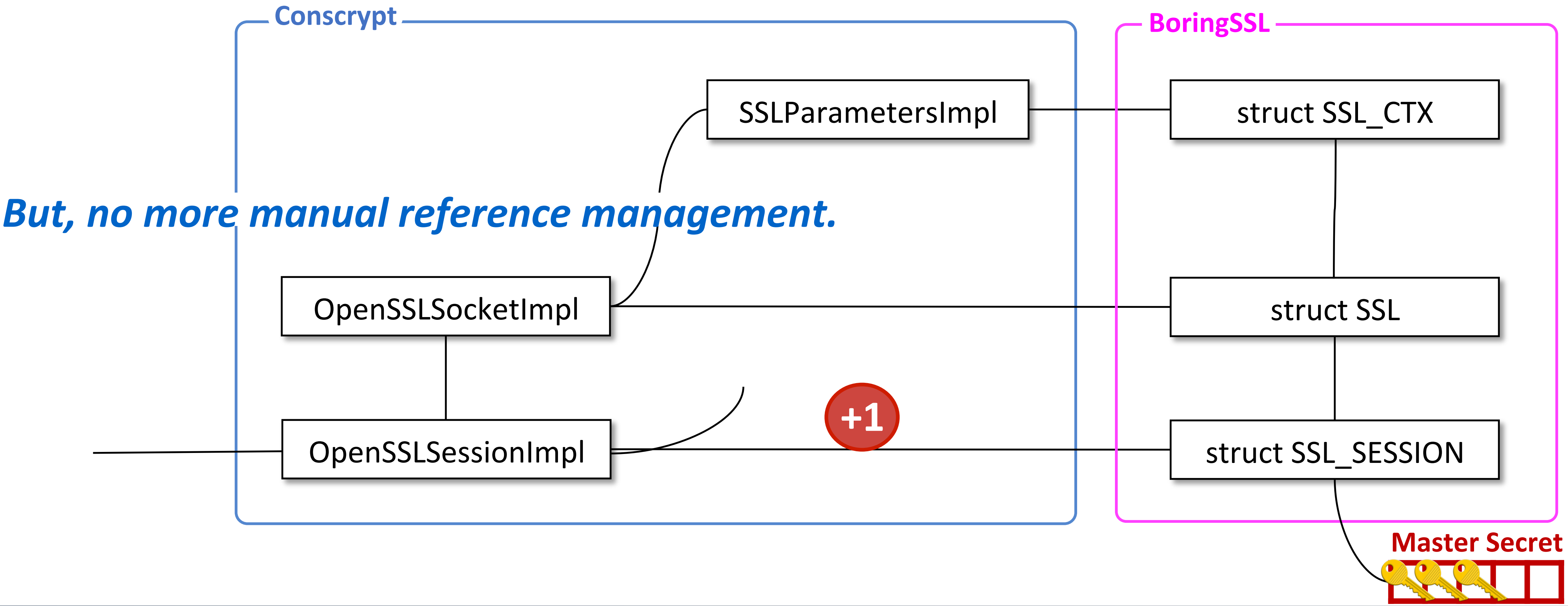
BoringSSL/OpenSSL: Reference Counting

- Each structure has reference count field.
- Objects are correctly freed when their reference count is zero.
- All key materials are managed within BoringSSL.



Conscrypt: Lazy Deletion

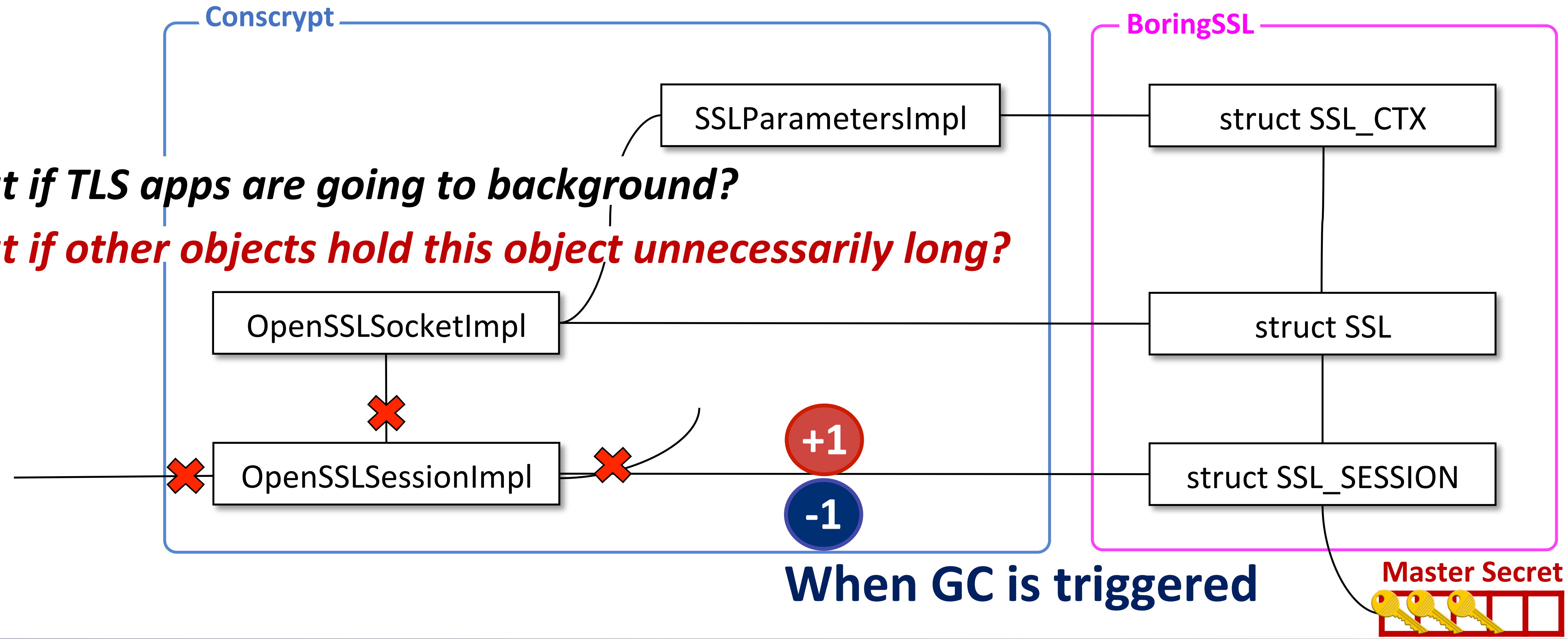
- Corresponding classes one-to-one mapped with the BoringSSL structures.
- On creation, OpenSSLSessionImpl increasing the ref. count of its underlying object.



Conscrypt: Lazy Deletion

Problem1: Dependence on JVM's Automatic Memory Management.

- Clean-up timing is undefined.



What if TLS apps are going to background?

What if other objects hold this object unnecessarily long?

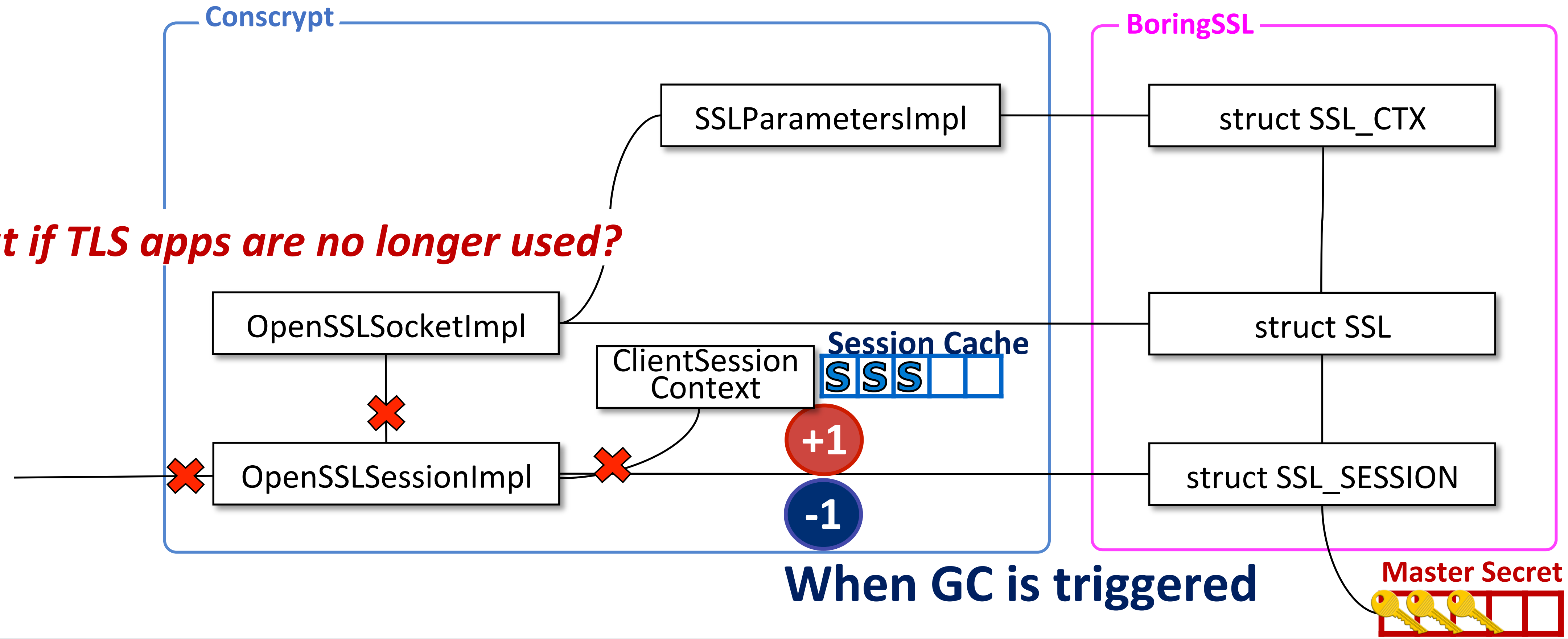
When GC is triggered

Conscrypt: Lazy Deletion

Problem2: Session Cache's LRU replacement policy

- No explicit eviction routine. Expired OpenSSLSessions are still in the cache.

What if TLS apps are no longer used?



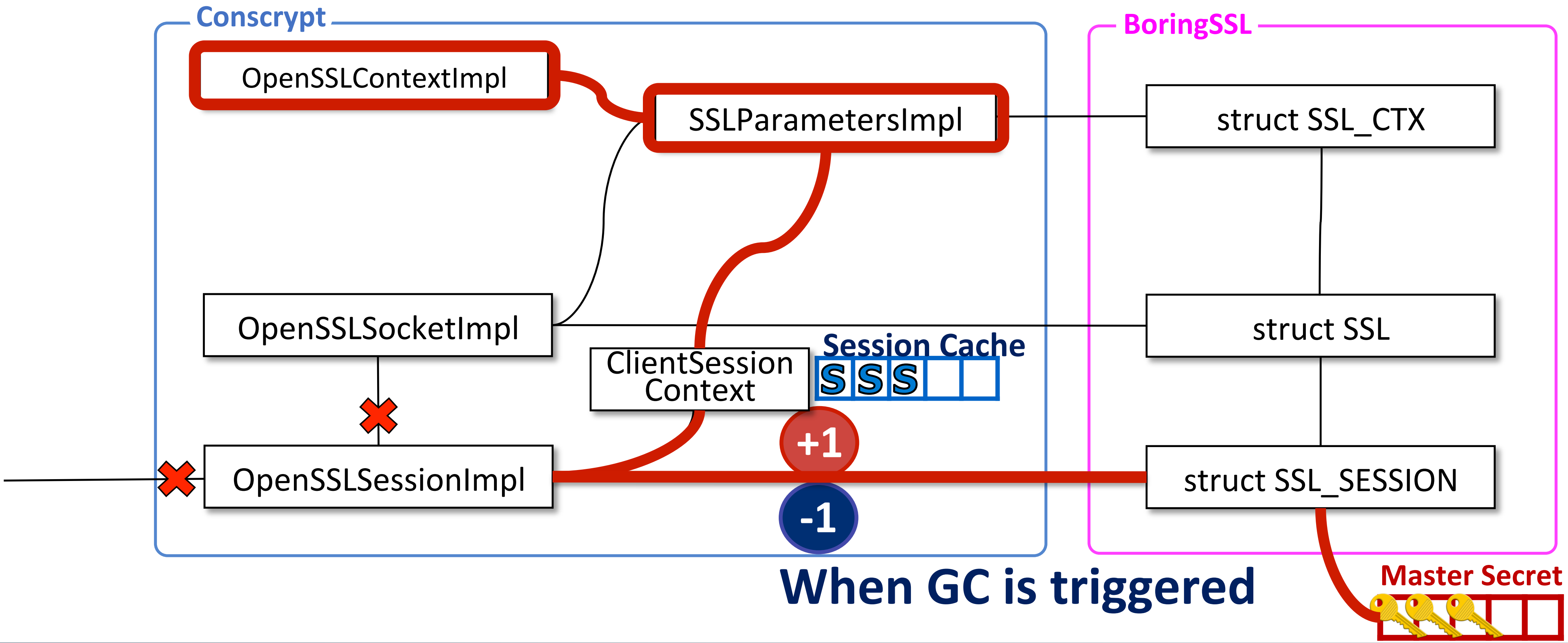
When GC is triggered

RICE

Conscrypt: Lazy Deletion

Problem3: Static Singleton objects are connected to them.

- Their lifetime is same as the application. No way to release them.

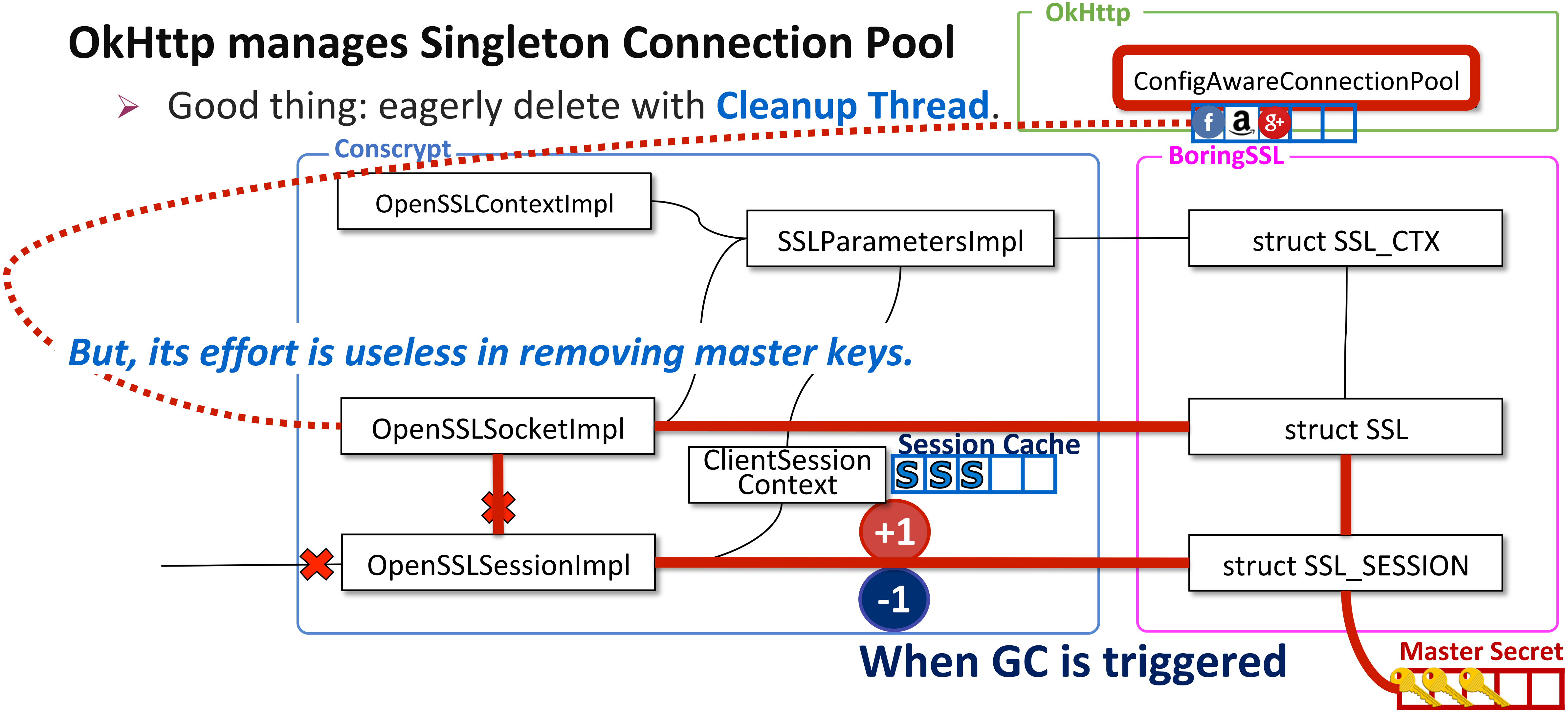


OkHttp: Eager Deletion

OkHttp manages Singleton Connection Pool

- Good thing: eagerly delete with **Cleanup Thread**.

But, its effort is useless in removing master keys.



When GC is triggered

RICE

What is the consequence of the problem?

- Each TLS application holds some number of master secrets whether they are expired or not.

Evaluation of Attack Feasibility

Can attackers exploit this problem in practice?

1. Is an attacker able to find 48 bytes of keys in a reasonable time?

- Yes. We found the pattern.
- Simple tool finds master secrets in several seconds.

2. How long does master keys live in memory with real-world apps?

- Additional experiment with Chrome application.

Evaluation of Attack Feasibility

How long does master key live in memory?

Result with Chrome application

Time (Hour)	Event	# of Found Keys

Evaluation of Attack Feasibility
How long does master key live in memory?

Result with Chrome application

Time (Hour)	Event	# of Found Keys
0	Access five web sites	51
1	Move the app to background	42
3	Run YouTube application	42
…	Keep playing movies	…
51	**After 2 days**	**38**

Most of master secrets are preserved as long as the app is alive.

Demo

What if attackers access Android memory of the targeted victim?

```
14:52:08:jl128@securitylab: ~/NDSS18_Demo
$ ls -l
total 1049172
drwxrwxr-x 2 jl128 jl128       4096 Feb 16 14:34 forensic_tools/
-rw-rw-r-- 1 jl128 jl128 1073741824 Feb 16 14:43 memory_dump.dmp
-rw-r--r-- 1 jl128 jl128     601247 Feb 16 14:43 packet_capture.pcap

14:52:19:jl128@securitylab: ~/NDSS18_Demo
$ 
```

Solutions

We implemented two solutions.

1. Hooking Android lifecycle

- Clean up expired keys when applications are going to background.

2. Eager Deletion: Sync with OkHttp

- Run secondary thread to evict expired TLS sessions.

Two modest patches can mitigate this problem.

RICE

Reporting to Google

- Reported the issue with the patches in Nov 2017.
- Recently, we received the feedback.

status: Assigned → Infeasible
ASR Severity: Moderate → NSBC

...

we don't consider deleting information from the application's memory fast enough to be a security issue ...

But, we believe expired master secrets should be deleted.

RICE

Conclusion

We first investigate Android TLS in terms of managing ephemeral keys.

Android retains master secrets because of conflicting memory models.

- Impact on all applications using standard TLS APIs.
- Impact on all Android versions we examined from Android 4 to 8.
- Our forensics tools show that it is exploitable practically.

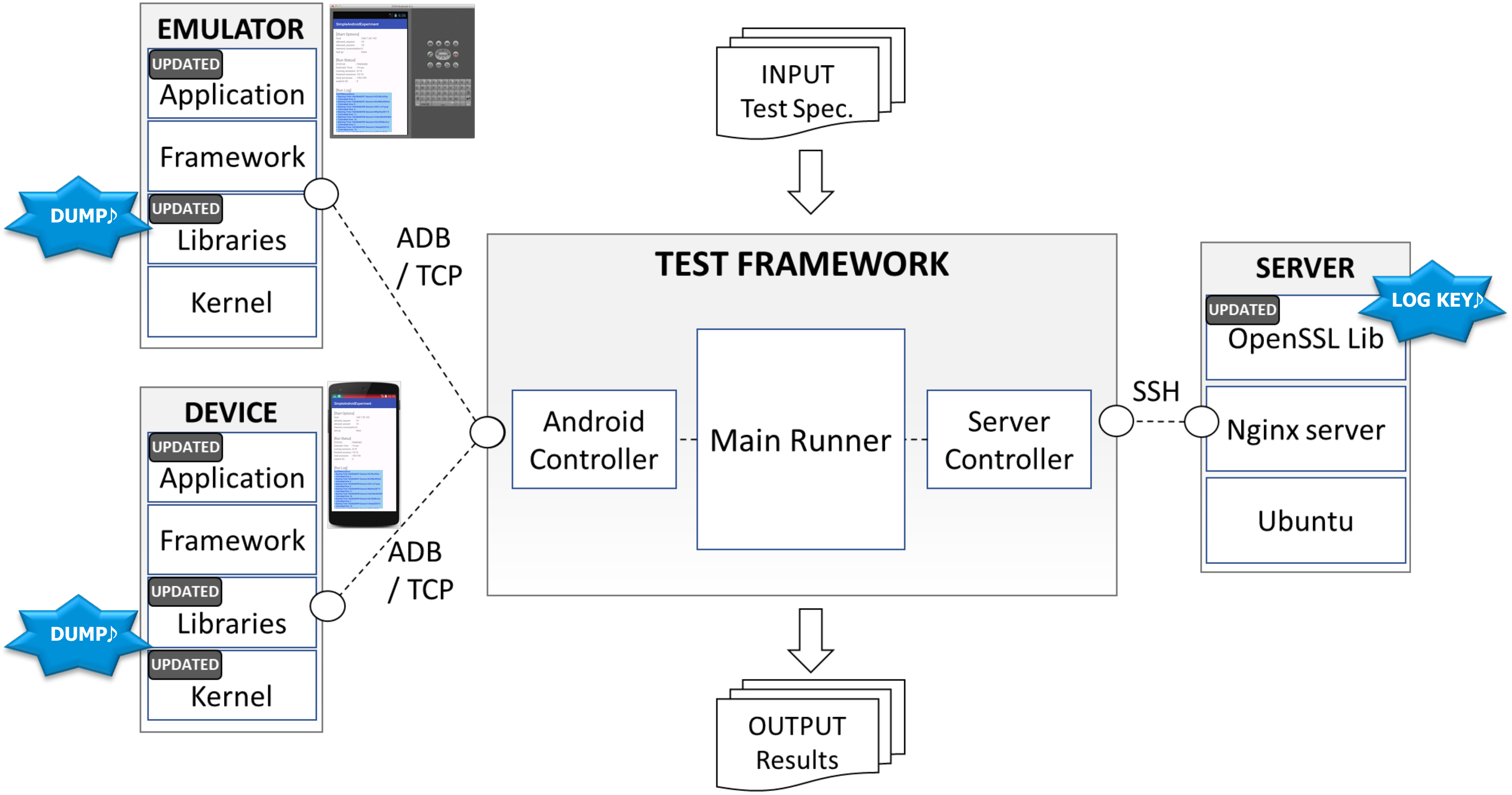
We suggest the practical solutions.

Thank you!

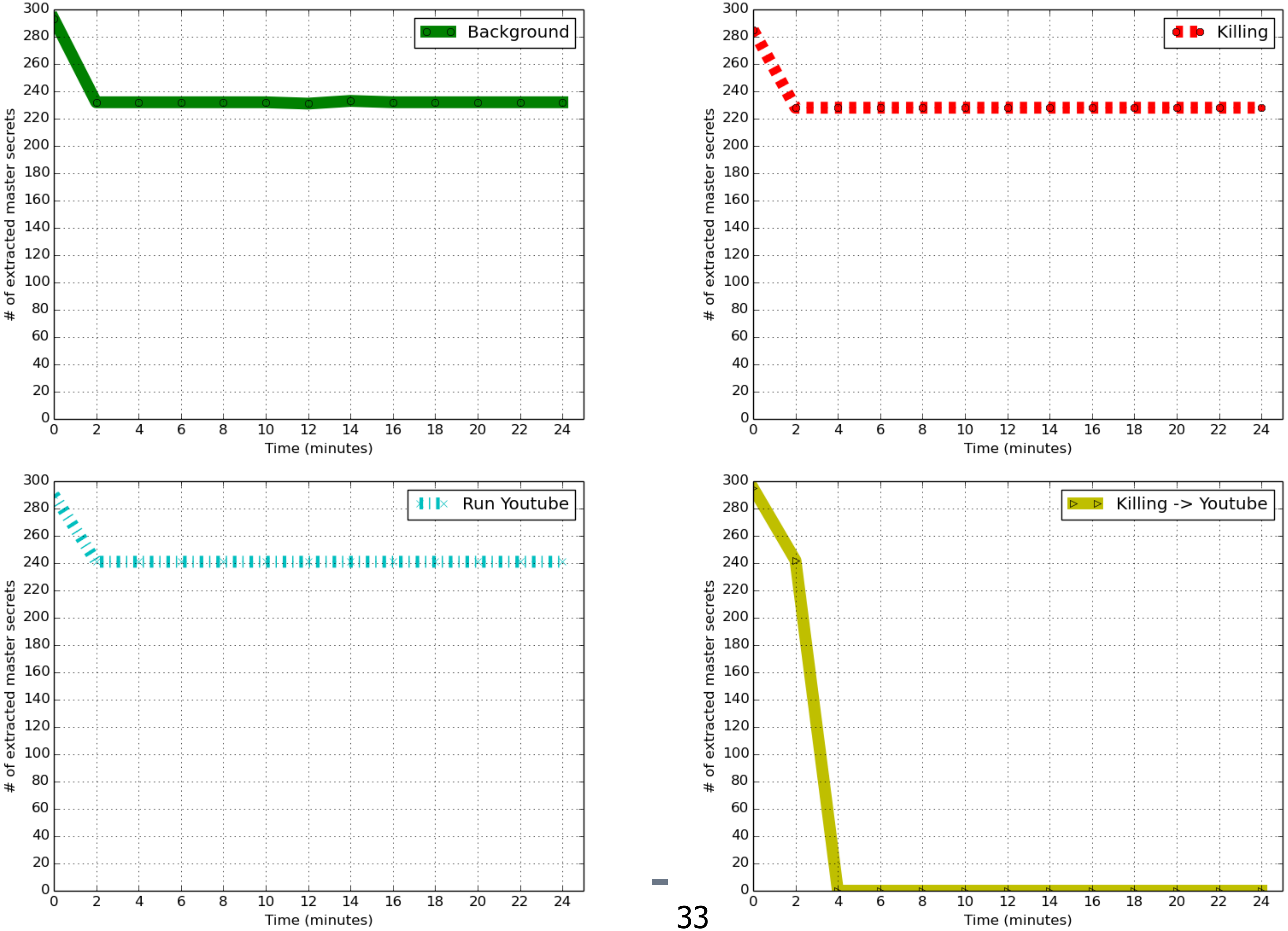
Jaeho Lee
PhD student, Rice University

Contact: Jaeho.Lee@rice.edu
Web: https://cs.rice.edu/~jl128

Analysis Framework



Results Detail



SSL_SESSION Structure

```
struct ssl_session_st {
  int ssl_version;
  int master_key_length;
  uint8_t master_key[SSL_MAX_MASTER_KEY_LENGTH];
  unsigned int session_id_length;
  uint8_t session_id[SSL_MAX_SSL_SESSION_ID_LENGTH];
  ...
}
```

0x0301~0303♪

0x30♪

0x20♪

Discussion

Conscrypt (Java) vs BoringSSL (C)

- Conscrypt: effective Java coding
- BoringSSL: isolated secret management

Conscrypt (TLS Session Cache) vs OkHttp (HTTP Connection Pool)

- Different perspective dealing with underlying objects
  - OkHttp: Eagerly eviction with Timer
  - Conscrypt: No explicit eviction

Bad Programming Pattern: Singleton object + Dependence on GC

- Singleton object + Dependence on GC for critical routines

Methodology

Black-Box

Security

Analysis

White-Box

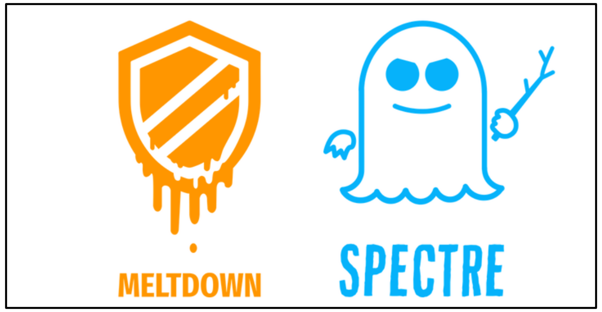
Security

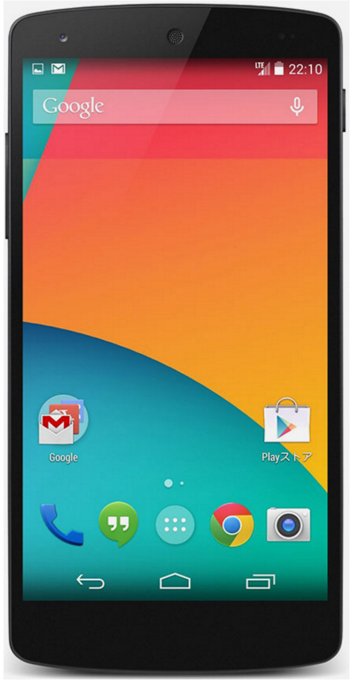
Analysis♪

Research Question and Motivation

Android has various attack vectors.

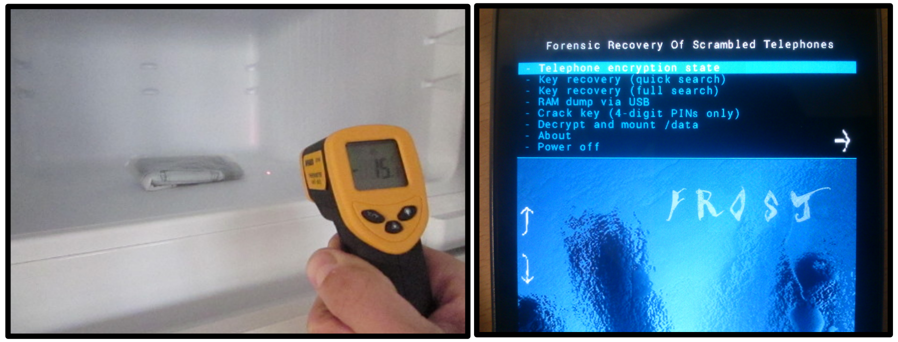
By software exploitations



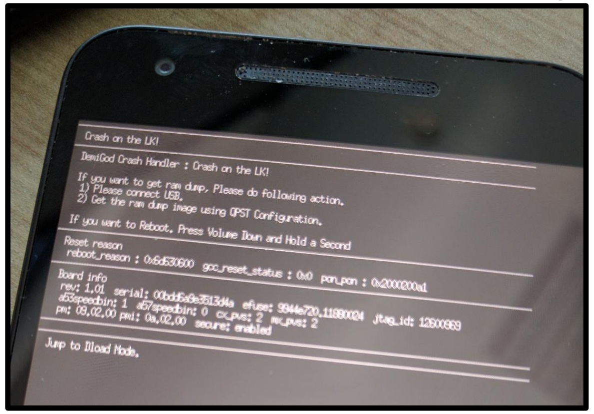


By physical techniques

Cold-boot attack



Nexus 5X bootloader vulnerability



RICE