Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis

Andrea Continella, Yanick Fratantonio, Martina Lindorfer,
Alessandro Puccetti, Ali Zand, Christopher Kruegel, Giovanni Vigna

POLITECNICO
MILANO 1863

NDSS'17
02-28-2017

UCSB

Mobile Privacy Leak Detection

- Mobile apps are known to **leak private information over the network** (e.g., IMEI, Location, Contacts)
- Researchers developed approaches to detect them
  - Static taint analysis
  - Dynamic taint analysis

Mobile Privacy Leak Detection

- Mobile apps are known to **leak private information over the network** (e.g., IMEI, Location, Contacts)
- Researchers developed approaches to detect them
  - Static taint analysis
  - Dynamic taint analysis
- Recently, network-based detection
  - Leaked values **need** to flow through the network

```
http://i.w.inmobi.com/showad.asm?u-id-map
=iB7WTkCLJvNsaEQakKKXFhk8ZEIZlnL0jqbbYexc
BAXYHH4wSKyCDWVfp+q+FeLFTQV6jS2Xg97liEzDk
w+XNTghe9ekNyMnjypmgiu7xBS1TcwZmFxYOjJkgP
OzkI9j2lryBaLlAJBSDkEqZeMVvcjcNkx+Ps6SaTR
zBbYf8UY=&u-key-ver=2198564
```

Motivation

```
// get Android ID using the Java Reflection API
String aid = class.getDeclaredMethod("getAndroidId",
             Context.class).invoke(context);
MessageDigest sha1 = getInstance("SHA-1"); // hash
sha1.update(aid.getBytes());
byte[] digest = sha1.digest();

Random random = new Random(); // generate random key
int key = random.nextint();
// XOR Android ID with the randomly generated key
byte[] xored = customXOR(digest, key);

String encoded = Base64.encode(xored);

// send the encrypted value and key to ad server
HttpURLConnection conn = url.openConnection();
conn.write(Base64.encode(encoded).getBytes());
conn.write(("key=" + key).getBytes());
```

Motivation

```
// get Android ID using the Java Reflection API
String aid = class.getDeclaredMethod("getAndroidId",
             Context.class).invoke(context);
MessageDigest sha1 = getInstance("SHA-1"); // hash
sha1.update(aid.getBytes());
byte[] digest = sha1.digest();

Random random = new Random(); // generate random key
int key = random.nextint();
// XOR Android ID with the randomly generated key
byte[] xored = customXOR(digest, key);

String encoded = Base64.encode(xored);

// send the encrypted value and key to ad server
HttpURLConnection conn = url.openConnection();
conn.write(Base64.encode(encoded).getBytes());
conn.write(("key=" + key).getBytes());
```

Motivation

```
// get Android ID using the Java Reflection API
String aid = class.getDeclaredMethod("getAndroidId",
             Context.class).invoke(context);
MessageDigest sha1 = getInstance("SHA-1"); // hash
sha1.update(aid.getBytes());
byte[] digest = sha1.digest();

Random random = new Random(); // generate random key
int key = random.nextint();
// XOR Android ID with the randomly generated key
byte[] xored = customXOR(digest, key);

String encoded = Base64.encode(xored);

// send the encrypted value and key to ad server
HttpURLConnection conn = url.openConnection();
conn.write(Base64.encode(encoded).getBytes());
conn.write(("key=" + key).getBytes());
```

Motivation

```
// get Android ID using the Java Reflection API
String aid = class.getDeclaredMethod("getAndroidId",
             Context.class).invoke(context);
MessageDigest sha1 = getInstance("SHA-1"); // hash
sha1.update(aid.getBytes());
byte[] digest = sha1.digest();

Random random = new Random(); // generate random key
int key = random.nextint();
// XOR Android ID with the randomly generated key
byte[] xored = customXOR(digest, key);

String encoded = Base64.encode(xored);

// send the encrypted value and key to ad server
HttpURLConnection conn = url.openConnection();
conn.write(Base64.encode(encoded).getBytes());
conn.write(("key=" + key).getBytes());
```

Motivation

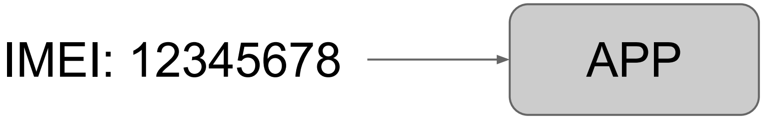
```
// get Android ID using the Java Reflection API
String aid = class.getDeclaredMethod("getAndroidId",
             Context.class).invoke(context);
MessageDigest sha1 = getInstance("SHA-1"); // hash
sha1.update(aid.getBytes());
byte[] digest = sha1.digest();

Random random = new Random(); // generate random key
int key = random.nextint();
// XOR Android ID with the randomly generated key
byte[] xored = customXOR(digest, key);

String encoded = Base64.encode(xored);

// send the encrypted value and key to ad server
HttpURLConnection conn = url.openConnection();
conn.write(Base64.encode(encoded).getBytes());
conn.write(("key=" + key).getBytes());
```

Our Approach

- Identify privacy leaks in a way that is resilient to obfuscation | encoding | encryption
- Perform **black-box differential analysis**
  1. Establish a **baseline** of the network behavior
  2. Modify sources of private information
  3. Detect leaks observing **differences** in network traffic

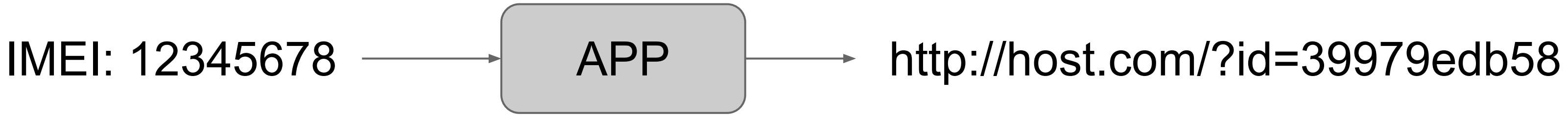
Our Approach

- Identify privacy leaks in a way that is resilient to obfuscation | encoding | encryption
- Perform **black-box differential analysis**
  1. Establish a **baseline** of the network behavior
  2. Modify sources of private information
  3. Detect leaks observing **differences** in network traffic



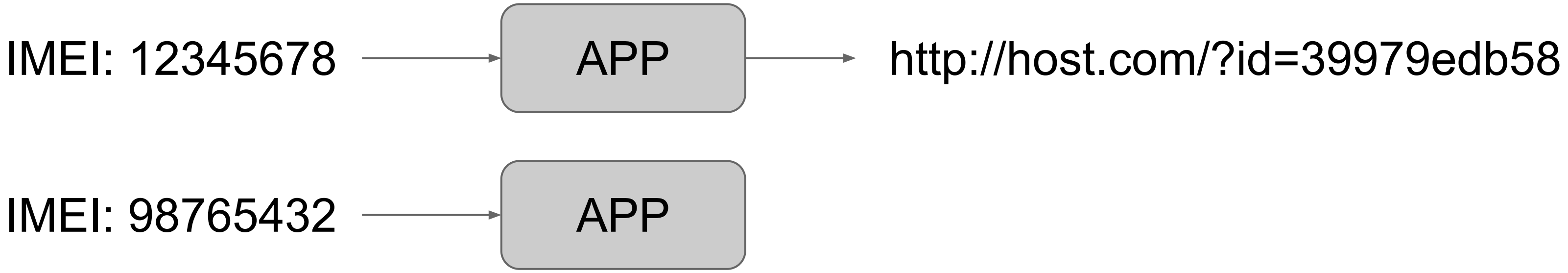
Our Approach

- Identify privacy leaks in a way that is resilient to obfuscation | encoding | encryption
- Perform **black-box differential analysis**
  1. Establish a **baseline** of the network behavior
  2. Modify sources of private information
  3. Detect leaks observing **differences** in network traffic



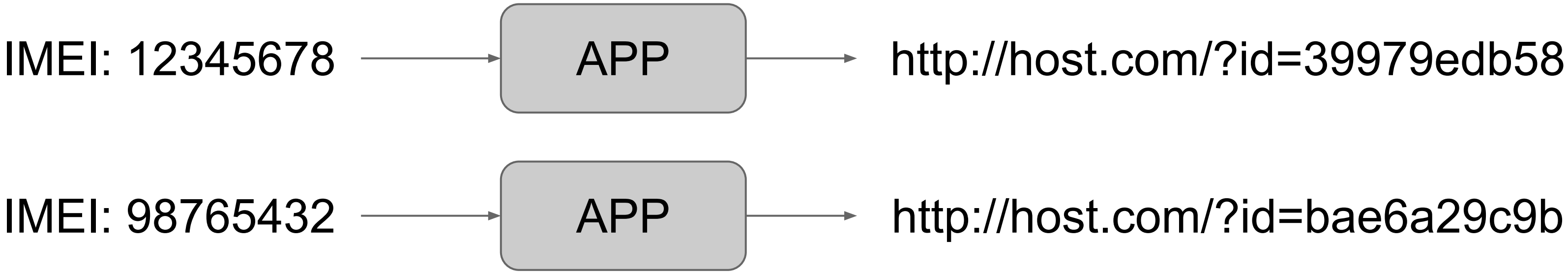
Our Approach

- Identify privacy leaks in a way that is resilient to obfuscation | encoding | encryption
- Perform **black-box differential analysis**
  1. Establish a **baseline** of the network behavior
  2. Modify sources of private information
  3. Detect leaks observing **differences** in network traffic



Our Approach

- Identify privacy leaks in a way that is resilient to obfuscation | encoding | encryption
- Perform **black-box differential analysis**
  1. Establish a **baseline** of the network behavior
  2. Modify sources of private information
  3. Detect leaks observing **differences** in network traffic



IMEI: 12345678 → APP → http://host.com/?id=39979edb58

IMEI: 98765432 → APP → http://host.com/?id=bae6a29c9b

Not so easy...

- Network traffic is **non-deterministic**
- The output **changes** even if you don't change the source
- Cannot pin a change in the output to a specific change in the input

We found that non-determinism can be often *explained* and *removed*, making differential analysis possible.

Sources of Non-Determinism

Sources of Non-Determinism

Random values

Sources of Non-Determinism

Random values

Timing values

Sources of Non-Determinism

Random values

Timing values

Network values

Sources of Non-Determinism

Random values

Timing values

Network values

System values

Sources of Non-Determinism

- Random values
- Timing values
- Network values
- System values
- Encryption

Sources of Non-Determinism

Random values

Timing values

Network values

System values

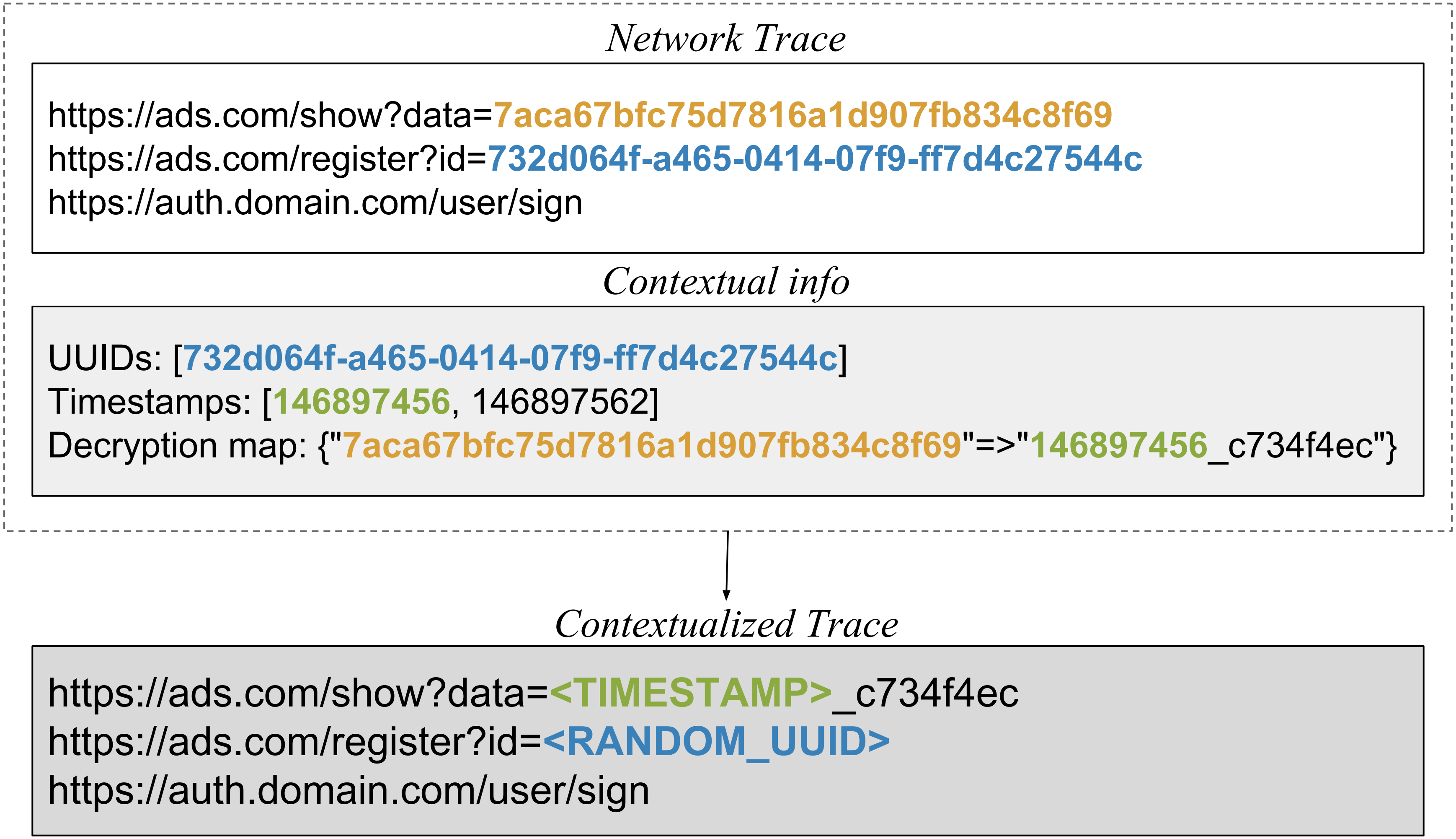
Encryption

Executions

Contextual Information

- Eliminate and explain non-determinism by **recording** and **replacing** non-deterministic values (either with previously seen or constant values)
    - Record and replay timestamps
    - Record random identifiers (UUID)
    - Record ptx and ctx during encryption
    - Set fixed seed for random num generation functions
    - Set values of performance measures to constants

Contextualized Trace



Network Trace

https://ads.com/show?data=7aca67bfc75d7816a1d907fb834c8f69
https://ads.com/register?id=732d064f-a465-0414-07f9-ff7d4c27544c
https://auth.domain.com/user/sign

Contextual info

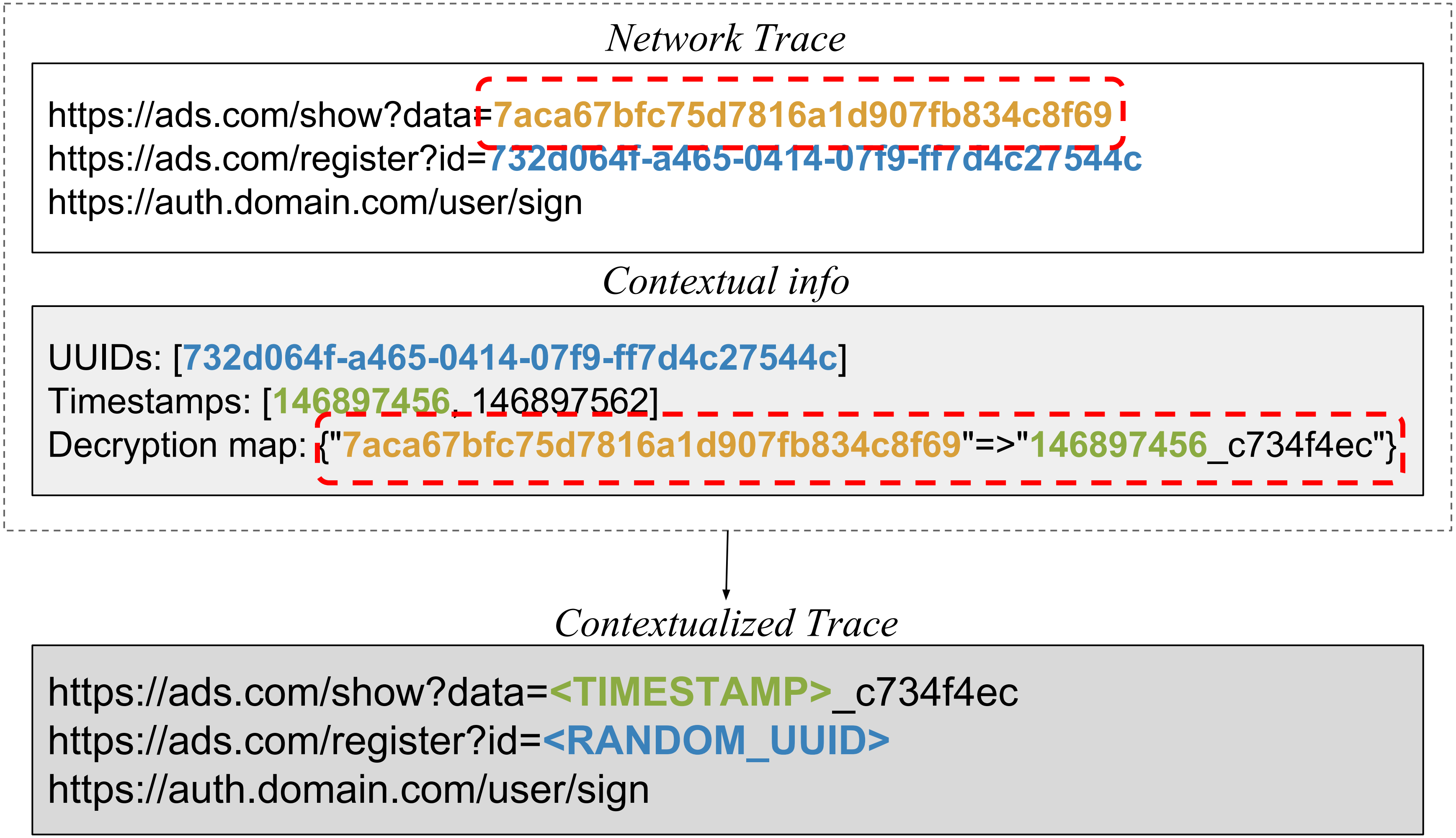
UUIDs: [732d064f-a465-0414-07f9-ff7d4c27544c]
Timestamps: [146897456, 146897562]
Decryption map: {"7aca67bfc75d7816a1d907fb834c8f69"=>"146897456_c734f4ec"}

Contextualized Trace

https://ads.com/show?data=<TIMESTAMP>_c734f4ec
https://ads.com/register?id=<RANDOM_UUID>
https://auth.domain.com/user/sign

A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Contextualized Trace



Network Trace

https://ads.com/show?data=7aca67bfc75d7816a1d907fb834c8f69
https://ads.com/register?id=732d064f-a465-0414-07f9-ff7d4c27544c
https://auth.domain.com/user/sign

Contextual info

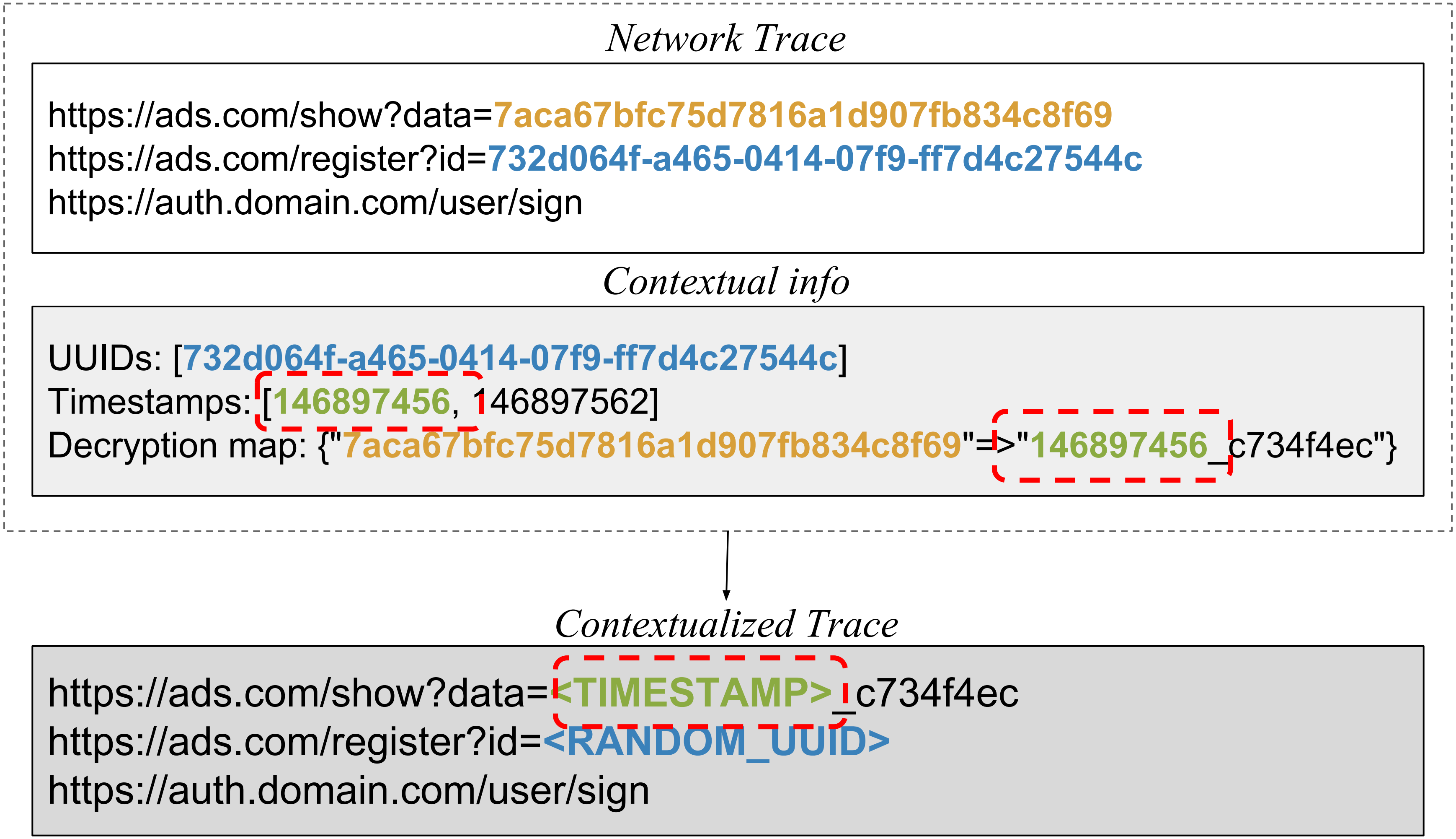
UUIDs: [732d064f-a465-0414-07f9-ff7d4c27544c]
Timestamps: [146897456, 146897562]
Decryption map: {"7aca67bfc75d7816a1d907fb834c8f69"=>"146897456_c734f4ec"}

Contextualized Trace

https://ads.com/show?data=<TIMESTAMP>_c734f4ec
https://ads.com/register?id=<RANDOM_UUID>
https://auth.domain.com/user/sign

A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Contextualized Trace



Network Trace

https://ads.com/show?data=7aca67bfc75d7816a1d907fb834c8f69

https://ads.com/register?id=732d064f-a465-0414-07f9-ff7d4c27544c

https://auth.domain.com/user/sign

Contextual info

UUIDs: [732d064f-a465-0414-07f9-ff7d4c27544c]

Timestamps: [146897456, 146897562]

Decryption map: {"7aca67bfc75d7816a1d907fb834c8f69"=>"146897456_c734f4ec"}

Contextualized Trace

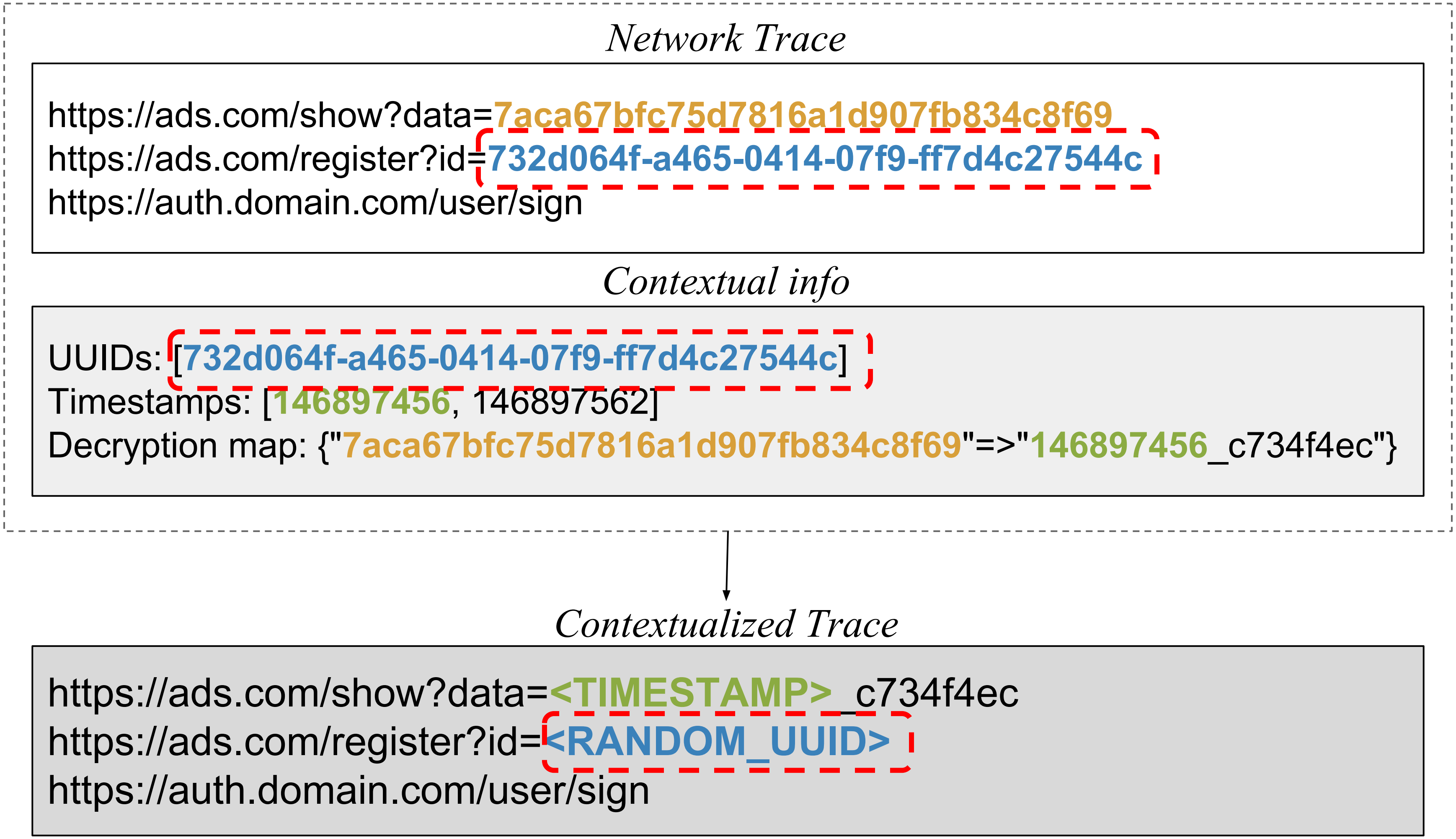
https://ads.com/show?data=<TIMESTAMP>_c734f4ec

https://ads.com/register?id=<RANDOM_UUID>

https://auth.domain.com/user/sign

A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Contextualized Trace



Network Trace

https://ads.com/show?data=7aca67bfc75d7816a1d907fb834c8f69
https://ads.com/register?id=732d064f-a465-0414-07f9-ff7d4c27544c
https://auth.domain.com/user/sign

Contextual info

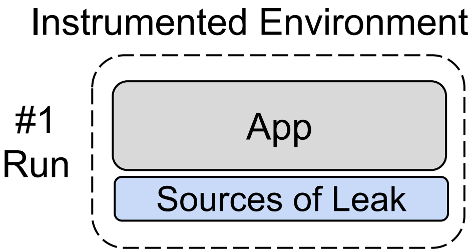
UUIDs: [732d064f-a465-0414-07f9-ff7d4c27544c]
Timestamps: [146897456, 146897562]
Decryption map: {"7aca67bfc75d7816a1d907fb834c8f69"=>"146897456_c734f4ec"}

Contextualized Trace

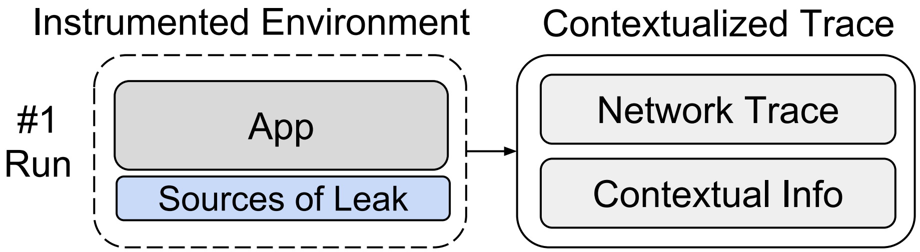
https://ads.com/show?data=<TIMESTAMP>_c734f4ec
https://ads.com/register?id=<RANDOM_UUID>
https://auth.domain.com/user/sign

A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

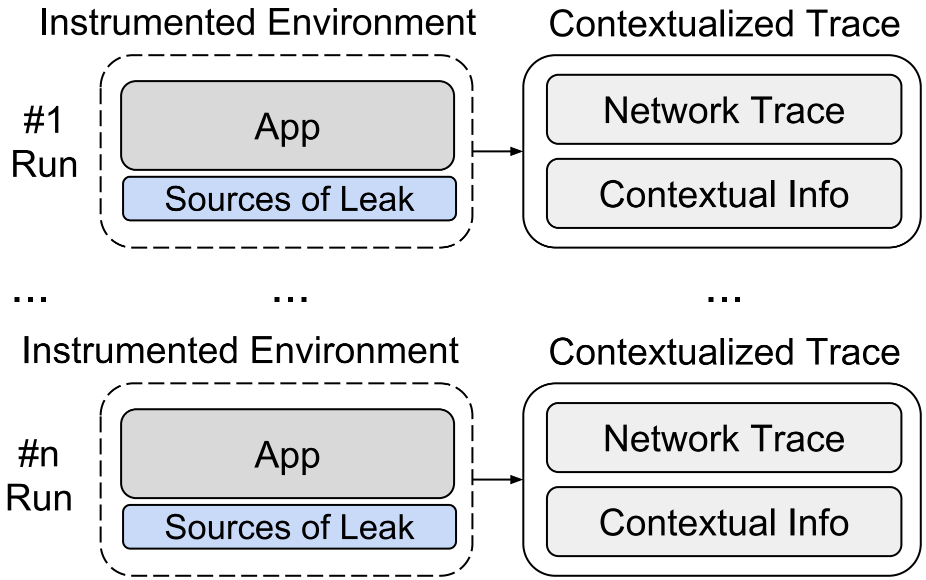
Agrigento: High-level Overview



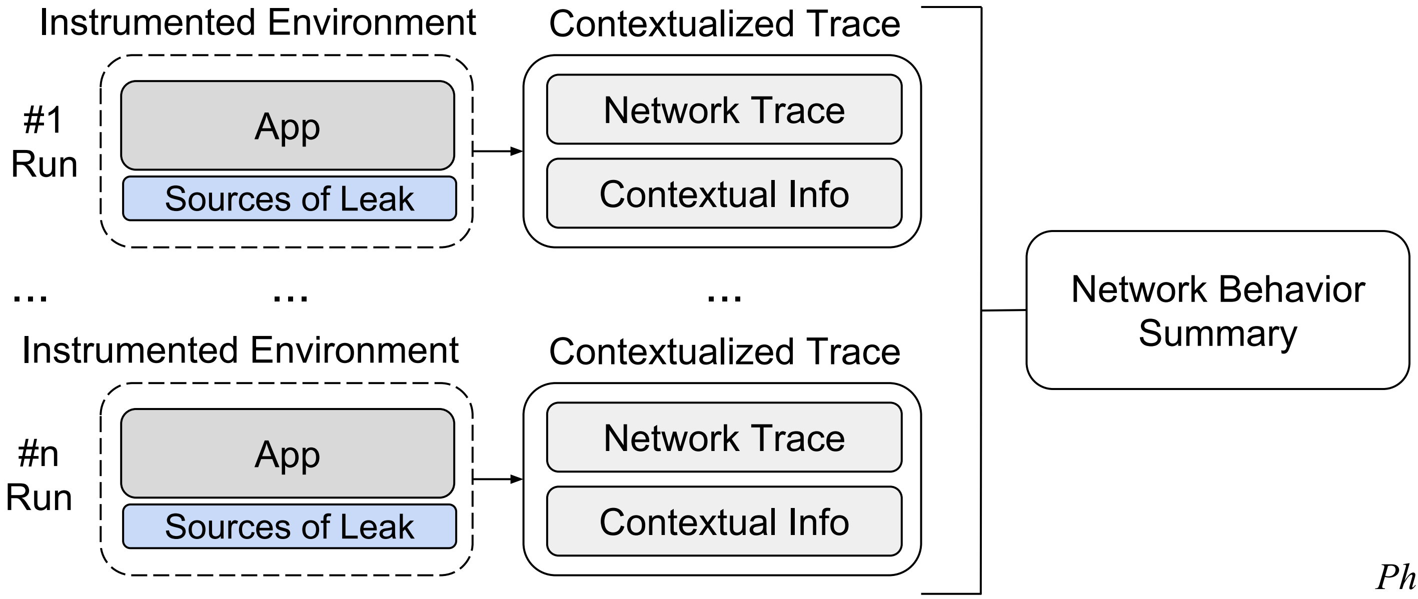
Agrigento: High-level Overview



Agrigento: High-level Overview

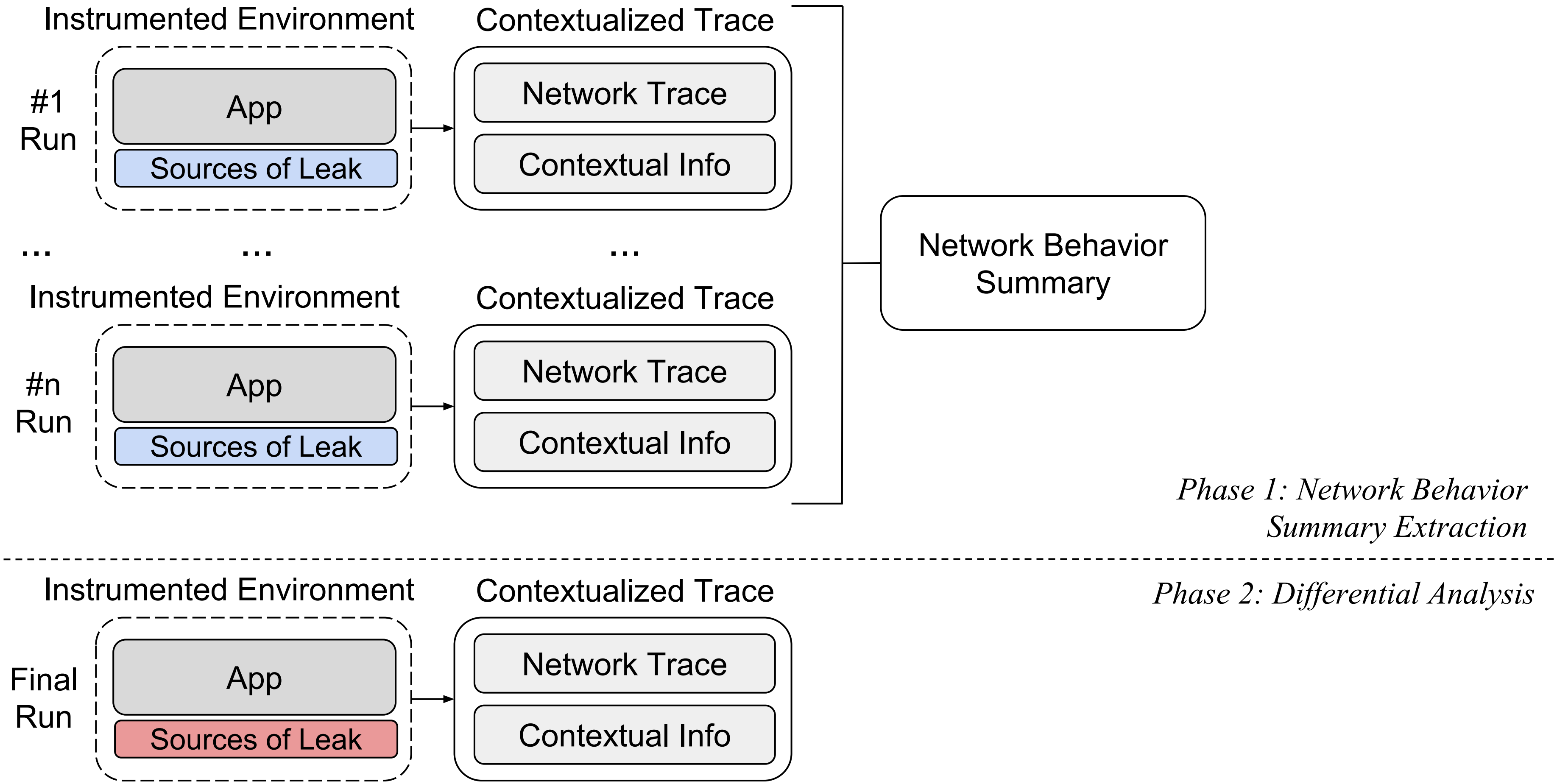


Agrigento: High-level Overview



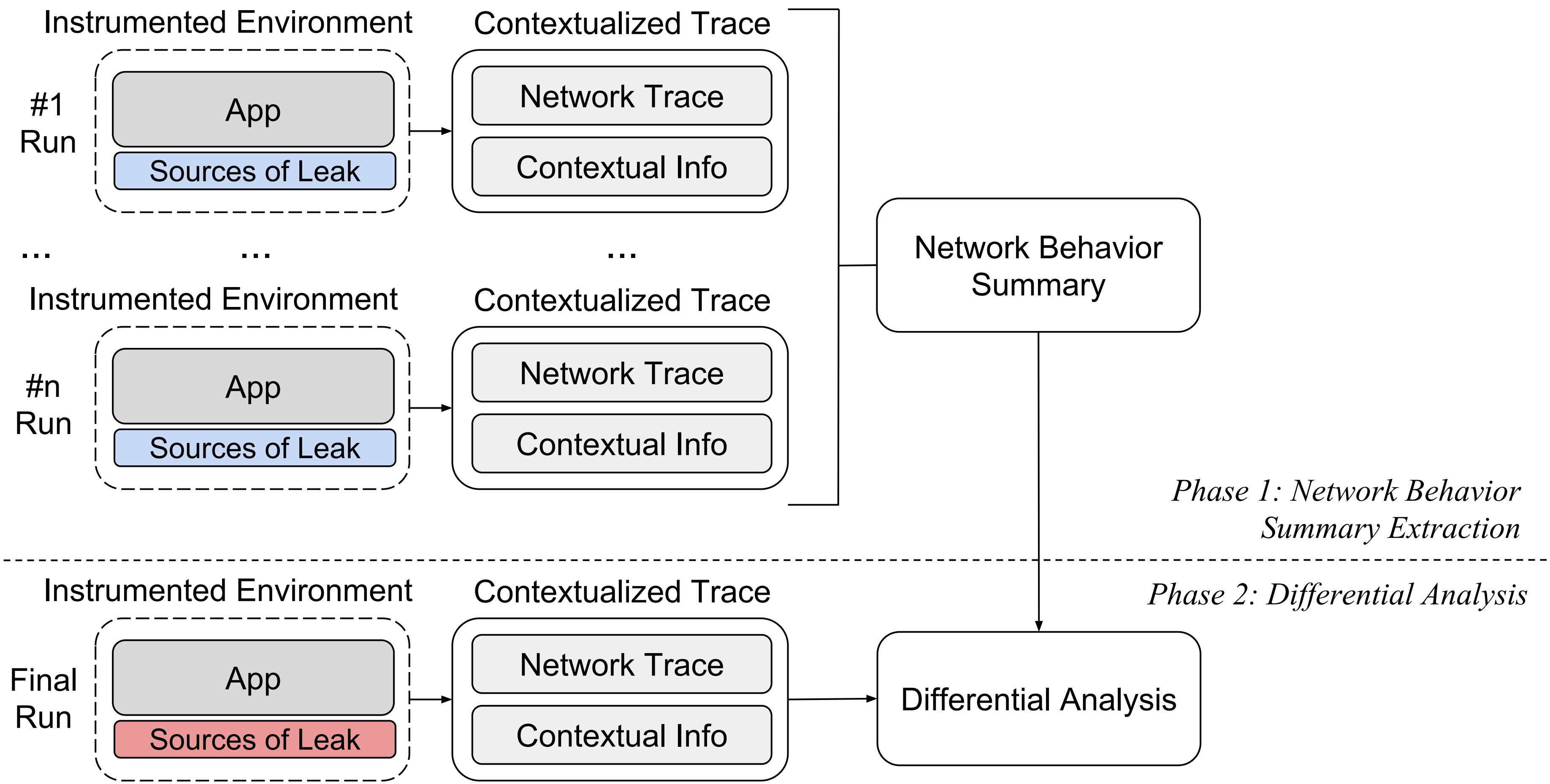
Phase 1: Network Behavior Summary Extraction

Agrigento: High-level Overview



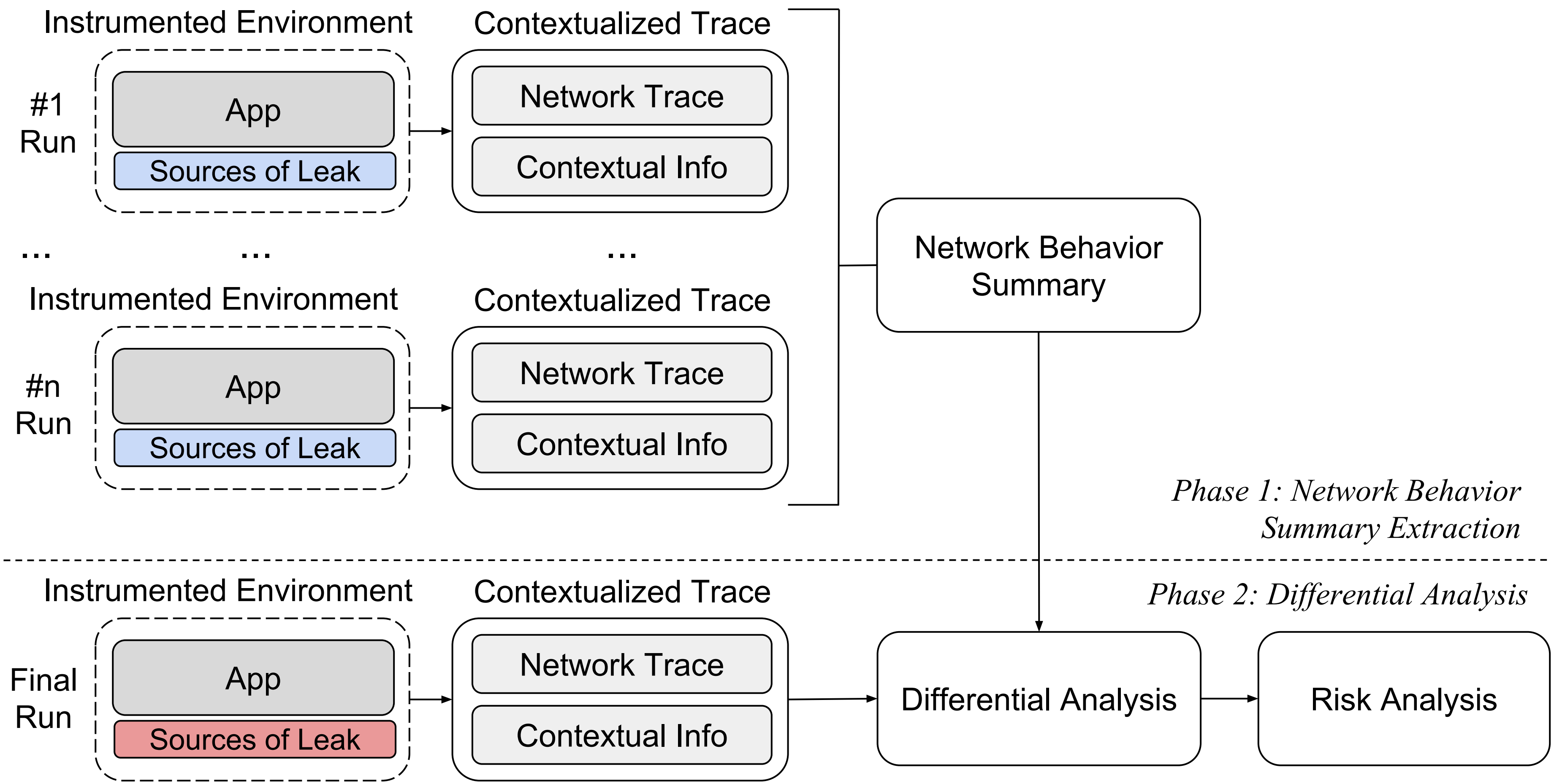
A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Agrigento: High-level Overview



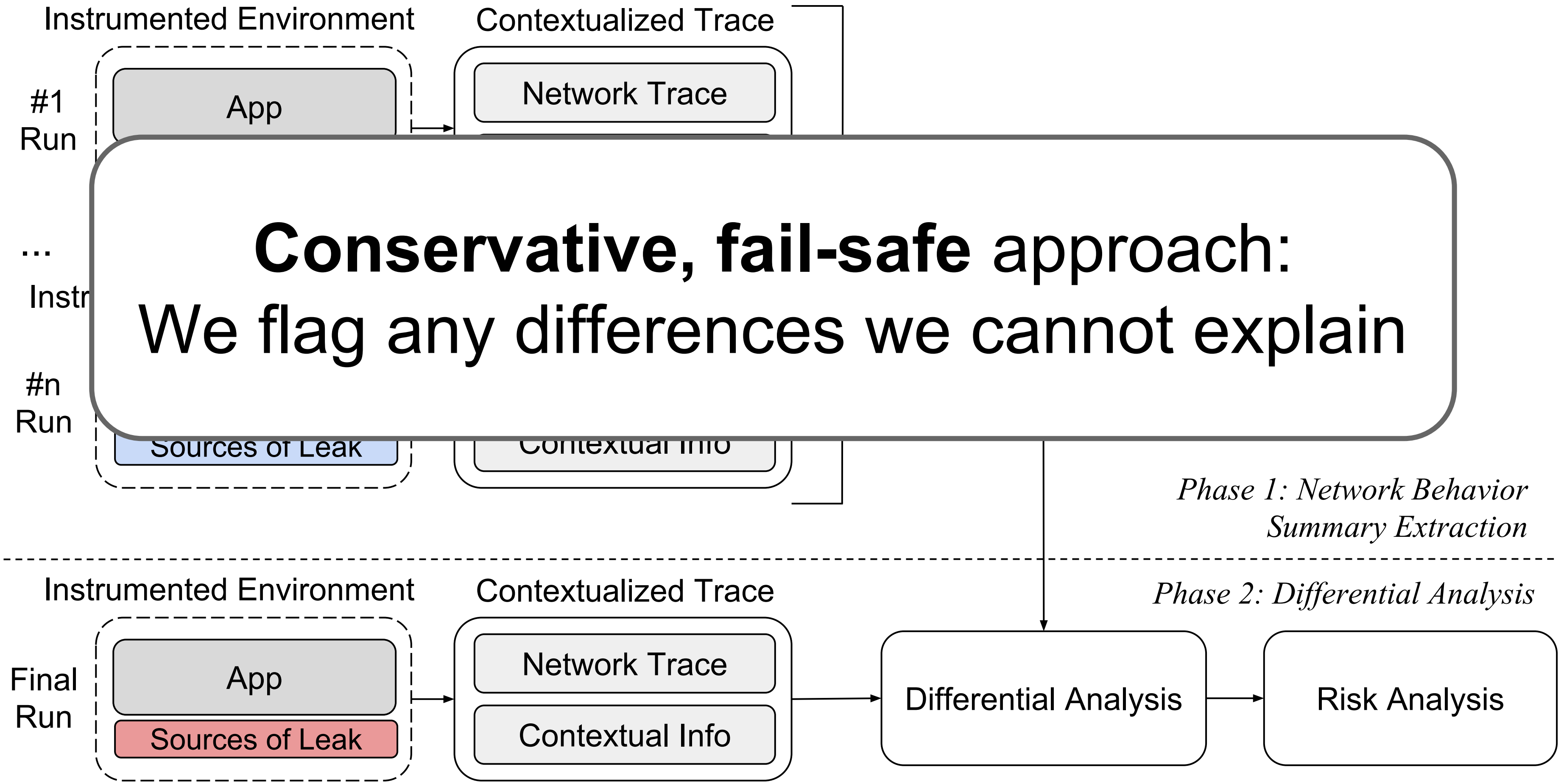
A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Agrigento: High-level Overview



A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Agrigento: High-level Overview



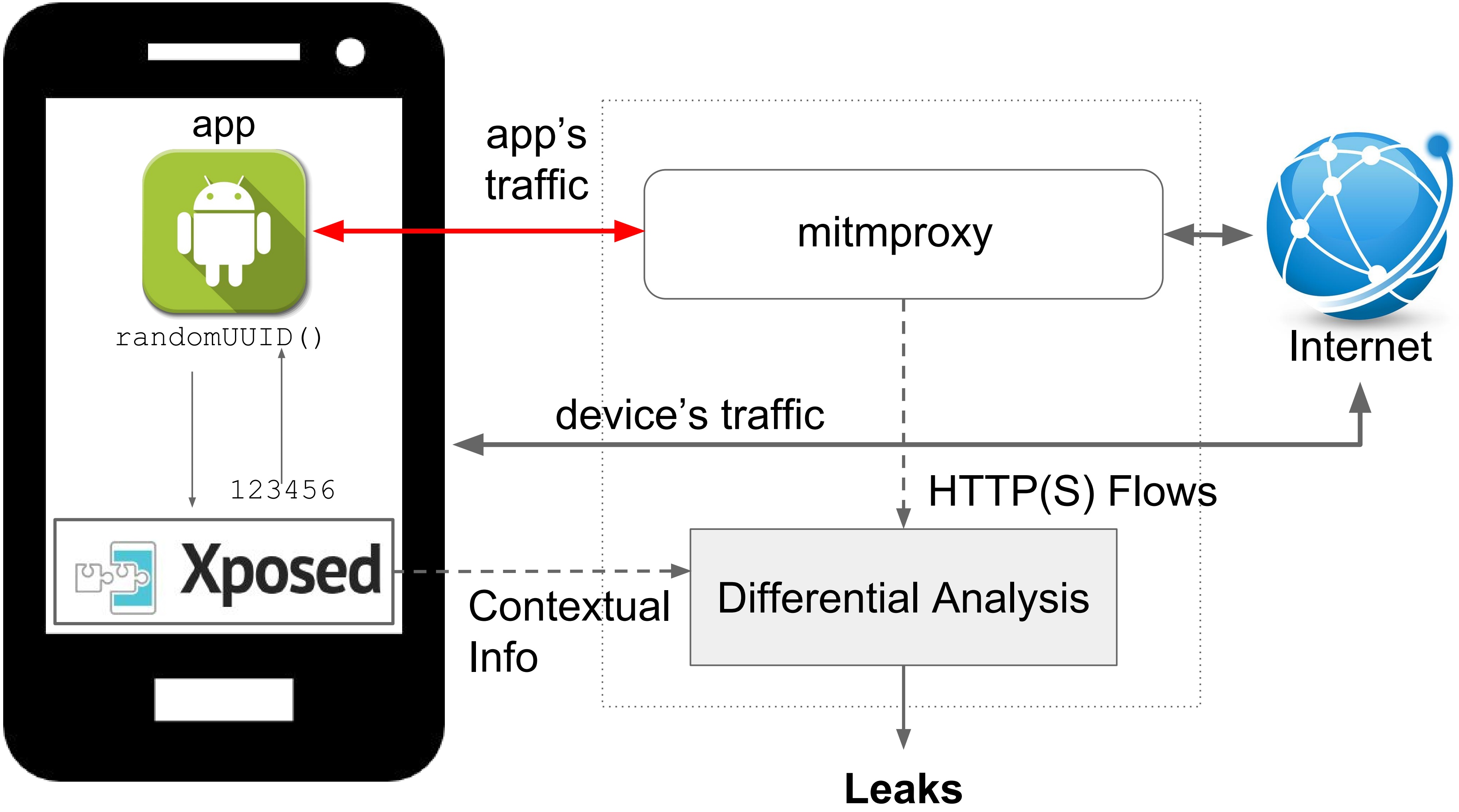
Conservative, fail-safe approach:
We flag any differences we cannot explain

A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Number of Runs

- **Automatically** determine number of executions
- After each run, differential analysis **without** any source modification
- An app reaches **convergence** when there are no diffs in the network for *K* consecutive runs

System Architecture



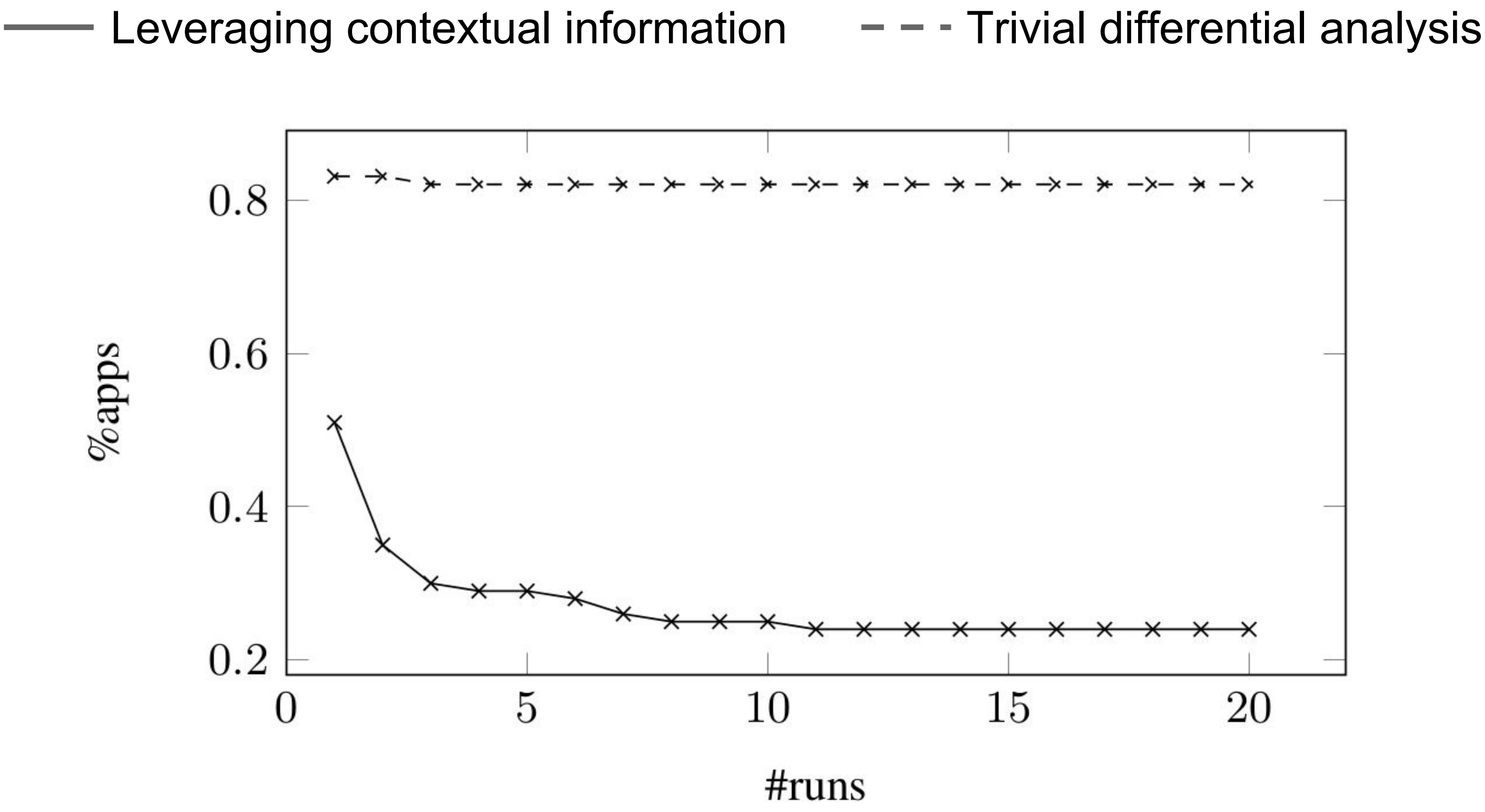
A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Experimental Setup & Datasets

- Setup
  - Six Nexus 5 running Android 4.4.4
  - 10 mins execution per app, Monkey for UI stimulation (fixed seed)
- Datasets
  - 100 most popular free apps across all the categories from the Google Play Store in June 2016
  - 100 randomly selected less popular apps
  - 750 apps from ReCon dataset
  - 54 apps from BayesDroid dataset

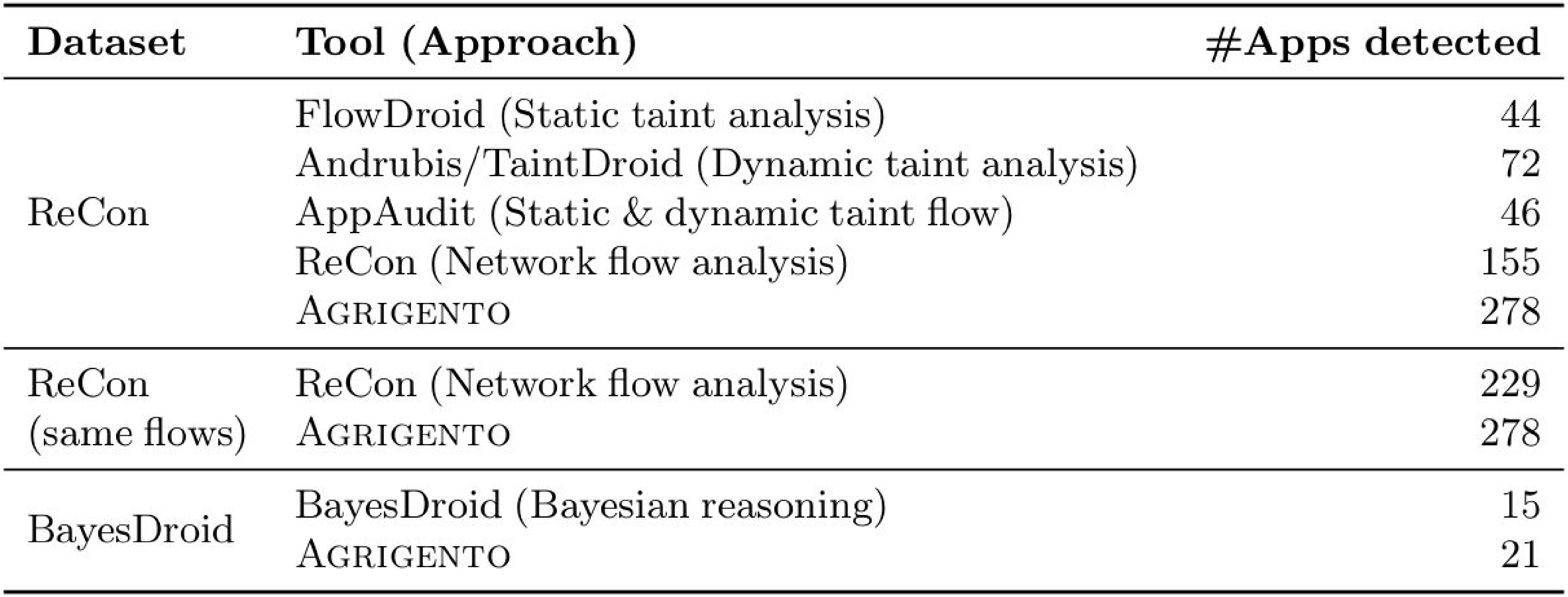
Non-Determinism in Network Traffic

- Top 100 Google Play apps from the ReCon dataset
- % of apps with non-deterministic network traffic



A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Comparison with Existing Tools



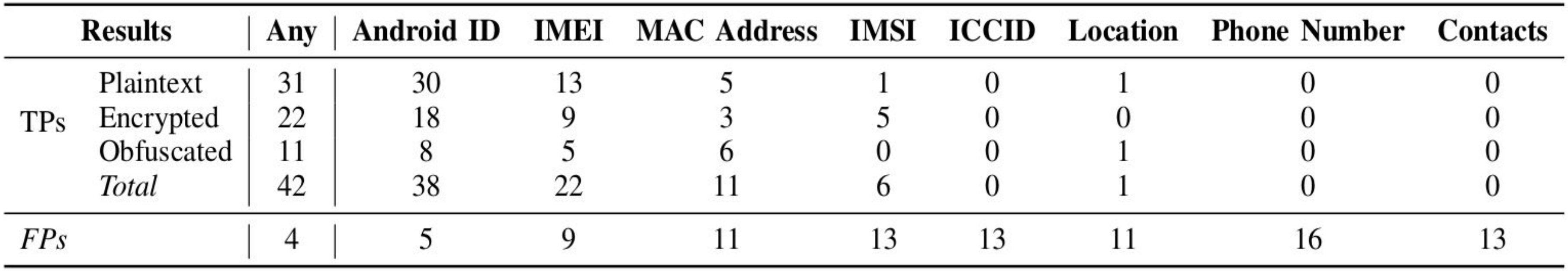
Dataset	Tool (Approach)	#Apps detected
ReCon	FlowDroid (Static taint analysis)	44
	Andrubis/TaintDroid (Dynamic taint analysis)	72
	AppAudit (Static & dynamic taint flow)	46
	ReCon (Network flow analysis)	155
	AGRIGENTO	278
ReCon (same flows)	ReCon (Network flow analysis)	229
	AGRIGENTO	278
BayesDroid	BayesDroid (Bayesian reasoning)	15
	AGRIGENTO	21

Agrigento detected many **more** apps &&
we manually verified most of them were true positives!

A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Privacy Leaks in Popular Apps

- Top 100 apps from the Google Play Store (July 2016)
- We classified the type of leak in three groups:
  - plaintext, encrypted, obfuscated
- Agrigento identified privacy leaks in **46** of the 100 apps
  - **42** true positives, **4** false positives



Results		Any	Android ID	IMEI	MAC Address	IMSI	ICCID	Location	Phone Number	Contacts
TPs	Plaintext	31	30	13	5	1	0	1	0	0
	Encrypted	22	18	9	3	5	0	0	0	0
	Obfuscated	11	8	5	6	0	0	1	0	0
	Total	42	38	22	11	6	0	1	0	0
FPs		4	5	9	11	13	13	11	16	13

A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Case Study: ThreatMetrix

```
https://h.online-metrix.net/fp/clear.png?ja=333034
26773f3a3930643667663b33383831303d343526613f2d3638
30247a3f363026663d333539347a31323838266c603d687c76
72253163253066253066616f6e74656e762f6a732c74637062
6f7926636f652466723f6a747670253161273266253266616d
6d2e65616f656b69726b757326727069786763 6e617730266a
683d65616437613732316431353c65613a31386e6760656330
3736363936343433632 66d64643f6561633336303b64336a39
353166633036666361373261363a61616335636761266d6673
3f353b32306d383230613230643b6534643934383a31663636
623b32323767616126616d65613d3139333331333331333131
3331333126617 43d6365656e765f6f6f6a696c6d26617e3f76
72777174666566676e6665722b6d6f606b6c652733632b392e
3226342d3b...
```

Case Study: ThreatMetrix

1. IMEI, Location, MAC address ~> HashMap
2. **XOR** HashMap with a **randomly** generated key
3. Hex-encode HashMap
4. Send obfuscated HashMap & random key

```
https://h.online-metrix.net/fp/clear.png?ja=3330342677 3f3a3930
643667663b33383831303d343526613f2d363830247a3f363026663d333539
347a3132383826 6c603d687c76722531632530662530666 16f6e74656e762f
6a732c746370626f7926636f652466723f6a74767025316127326625326661
6d6d2e65616f656b69726b757326727069786763 6e617730266a683d656164
37613732316431353c65613a31386e67606563303736363936343433 63266d
64643f656163333630 3b64336a39353166633036666361373261363a616163
35636761266d66733f353b32306d383230613230643b653464393438 3a3166
3636623b323237676161266 16d65613d31393333313333313331313331 3331
2661743d6365656e765f6f6f6a696c6d26617e3f767277717466656667 6e66
65722b6d6f606b6c652733632b392e3226342d3b...
```

Limitations & Future Work

- Limited code coverage
- Covert channels
- No native code instrumentation
  - We use a conservative approach: FP in worst case
- Only HTTP(S) GET and POST
- Investigate malicious intents behind obfuscation

Conclusions

- Non-Determinism in network traffic can be **often explained** and **removed**
- Agrigento can detect privacy leaks using a black-box, **obfuscation-resilient** approach
- Apps and ad libraries **hide** their information leaks using different types of **encoding and encryption**

https://github.com/ucsb-seclab/agrigento

Thank you!
Questions?

andrea.continella@polimi.it

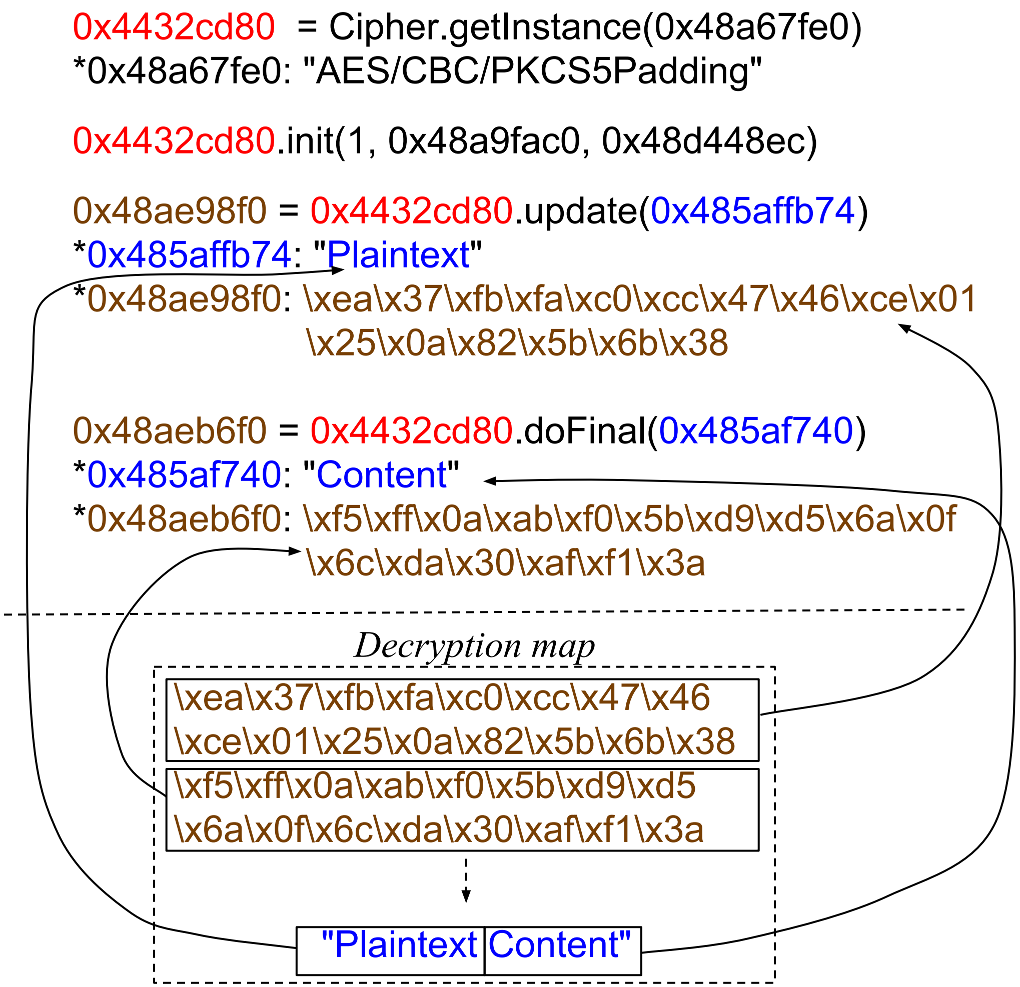
🐦 @_conand

seclab

THE COMPUTER SECURITY GROUP AT UC SANTA BARBARA

https://github.com/ucsb-seclab/agrigento

Decryption Map



```
0x4432cd80  = Cipher.getInstance(0x48a67fe0)
*0x48a67fe0: "AES/CBC/PKCS5Padding"

0x4432cd80.init(1, 0x48a9fac0, 0x48d448ec)

0x48ae98f0 = 0x4432cd80.update(0x485affb74)
*0x485affb74: "Plaintext"
*0x48ae98f0: \xea\x37\xfb\xfa\xc0\xcc\x47\x46\xce\x01
             \x25\x0a\x82\x5b\x6b\x38

0x48aeb6f0 = 0x4432cd80.doFinal(0x485af740)
*0x485af740: "Content"
*0x48aeb6f0: \xf5\xff\x0a\xab\xf0\x5b\xd9\xd5\x6a\x0f
             \x6c\xda\x30\xaf\xf1\x3a
```

Decryption map

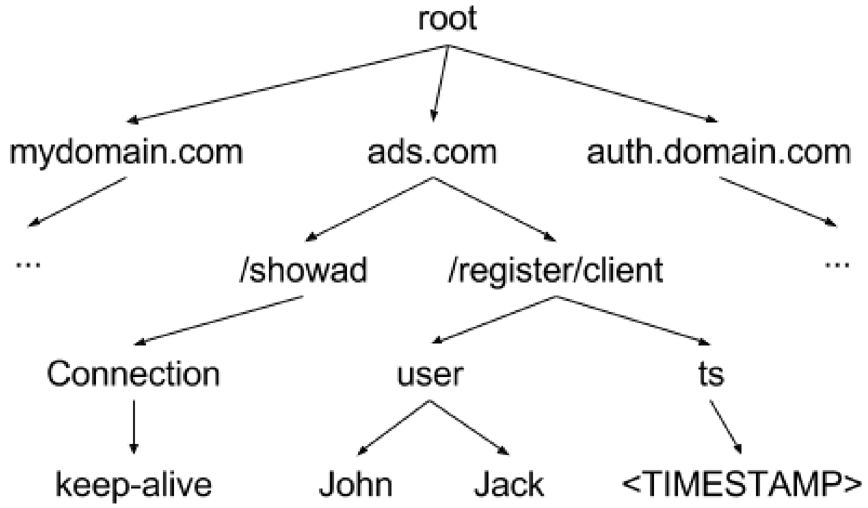
```
\xea\x37\xfb\xfa\xc0\xcc\x47\x46
\xce\x01\x25\x0a\x82\x5b\x6b\x38

\xf5\xff\x0a\xab\xf0\x5b\xd9\xd5
\x6a\x0f\x6c\xda\x30\xaf\xf1\x3a
```

"Plaintext|Content"

A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Network Behavior Summary

- Model the network behavior summary using a tree-based data structure
- The tree has four layers: domain names, paths, and key-value pairs (HTTP queries and HTTP headers)
- Parse known data structures (e.g., JSON) according to the HTTP Content-Type (e.g., application/json)



A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Differential Analysis

- For each HTTP flow from the final run, navigate the tree and check if each field of the flow is part of the tree
- If not, compare the new field with the fields in the same position in the tree (e.g., same domain, path, and key)
  - compare with the most similar field
- Decode known encodings (i.e., Base64, URLencode)
- Align using Needleman-Wunsch algorithm
- Generate a regexp (merges gaps, replaces them with *)
- Obtain differences extracting substrings that match the wildcards of the regular expression from a field

A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Risk Analysis

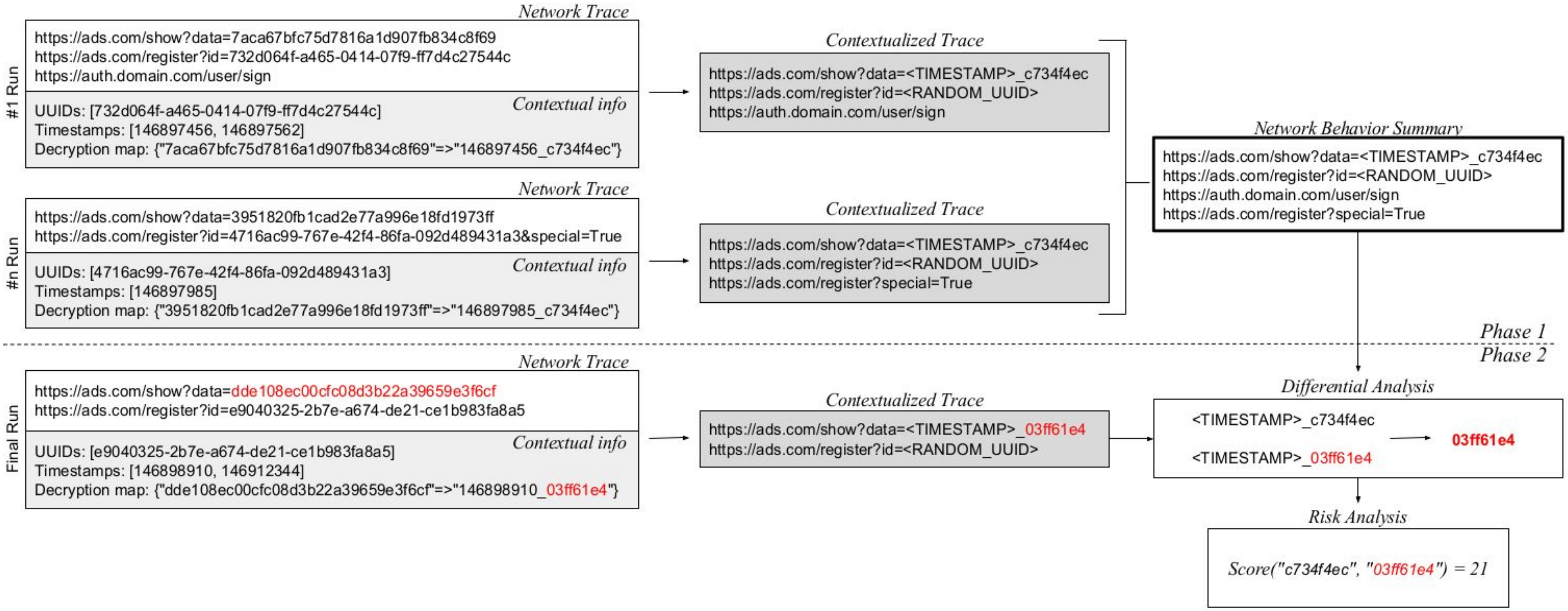
- A = app
- D = differences detected analyzing A
- F = all the fields in the tree of A's network behavior
- S_A= overall score that quantifies how many bits the app is leaking

$$distance(x,y) = \begin{cases} Hamming(x,y) & \text{if } len(x) = len(y) \\ Levenshtein(x,y) * 8 & \text{otherwise} \end{cases}$$

$$S_A = \sum_{\forall d \in D} \min_{\forall f \in F} distance(d, f)$$

A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Agrigento: Analysis Example



#1 Run

Network Trace

https://ads.com/show?data=7aca67bfc75d7816a1d907fb834c8f69
https://ads.com/register?id=732d064f-a465-0414-07f9-ff7d4c27544c
https://auth.domain.com/user/sign

Contextual info

UUIDs: [732d064f-a465-0414-07f9-ff7d4c27544c]
Timestamps: [146897456, 146897562]
Decryption map: {"7aca67bfc75d7816a1d907fb834c8f69"=>"146897456_c734f4ec"}

Contextualized Trace

https://ads.com/show?data=<TIMESTAMP>_c734f4ec
https://ads.com/register?id=<RANDOM_UUID>
https://auth.domain.com/user/sign

#n Run

Network Trace

https://ads.com/show?data=3951820fb1cad2e77a996e18fd1973ff
https://ads.com/register?id=4716ac99-767e-42f4-86fa-092d489431a3&special=True

Contextual info

UUIDs: [4716ac99-767e-42f4-86fa-092d489431a3]
Timestamps: [146897985]
Decryption map: {"3951820fb1cad2e77a996e18fd1973ff"=>"146897985_c734f4ec"}

Contextualized Trace

https://ads.com/show?data=<TIMESTAMP>_c734f4ec
https://ads.com/register?id=<RANDOM_UUID>
https://ads.com/register?special=True

Network Behavior Summary

https://ads.com/show?data=<TIMESTAMP>_c734f4ec
https://ads.com/register?id=<RANDOM_UUID>
https://auth.domain.com/user/sign
https://ads.com/register?special=True

Phase 1

Phase 2

Final Run

Network Trace

https://ads.com/show?data=dde108ec00cfc08d3b22a39659e3f6cf
https://ads.com/register?id=e9040325-2b7e-a674-de21-ce1b983fa8a5

Contextual info

UUIDs: [e9040325-2b7e-a674-de21-ce1b983fa8a5]
Timestamps: [146898910, 146912344]
Decryption map: {"dde108ec00cfc08d3b22a39659e3f6cf"=>"146898910_03ff61e4"}

Contextualized Trace

https://ads.com/show?data=<TIMESTAMP>_03ff61e4
https://ads.com/register?id=<RANDOM_UUID>

Differential Analysis

<TIMESTAMP>_c734f4ec
<TIMESTAMP>_03ff61e4
→ **03ff61e4**

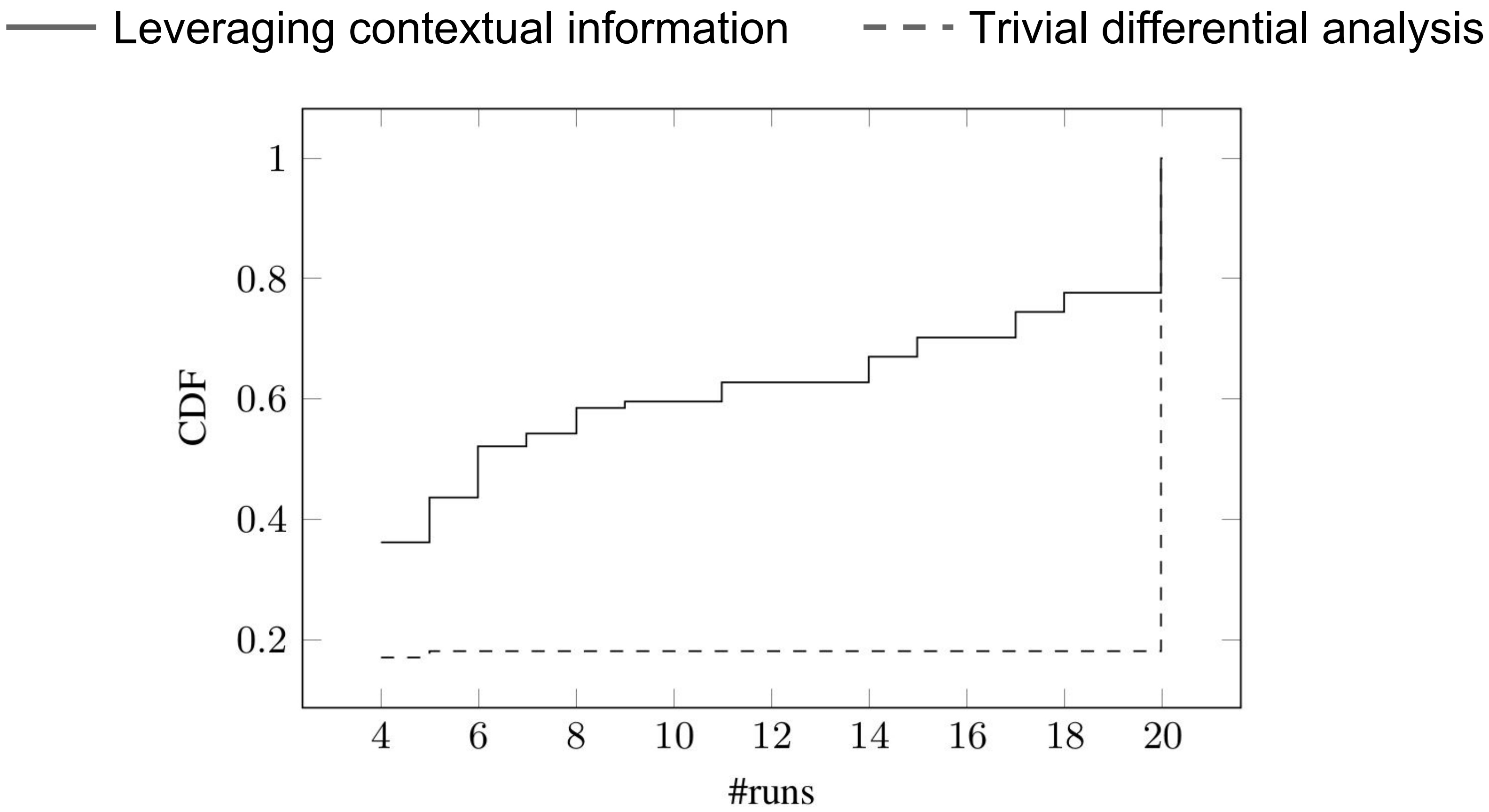
Risk Analysis

Score("c734f4ec", "03ff61e4") = 21

A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Non-Determinism in Network Traffic

- Top 100 Google Play apps from the ReCon dataset
- CDF of the number of runs required for convergence (for *K* = 3)



A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Comparison with ReCon

- ReCon authors run their tool on the network traffic dumps we collected during our analysis
- ReCon flagged 229 apps
- Agrigento correctly detected all the apps identified by ReCon, and, in addition, it detected **49** apps that ReCon did not flag
- Manual verification **32** did indeed leak at least one source of private information
- **5** apps false positives
- We cannot classify the remaining 12 cases

Comparison with BayesDroid

- 54 apps from BayesDroid (from 2013)
    - 10 apps did not work properly
- BayesDroid flagged **15** of the 54 apps
- Agrigento flagged **21** apps, 10 of the 15 apps detected by BayesDroid
    - we manually looked at the network traces of the remaining 5 apps and we did not see any leak
- **11 new** apps detected.
    - Apps used encryption and/or obfuscation

Case Study: InMobi

```
http://i.w.inmobi.com/showad.asm?u-id-map=iB
7WTkCLJvNsaEQakKKXFhk8ZEIZlnL0jqbbYexcBAXYHH
4wSKyCDWVfp+q+FeLFTQV6jS2Xg97liEzDkw+XNTghe9
ekNyMnjypmgiu7xBS1TcwZmFxYOjJkgPOzkI9j2lryBa
LlAJBSDkEqZeMVvcjcNkx+Ps6SaTRzBbYf8UY=&u-key
-ver=2198564
```

Case Study: InMobi

1. Hash Android ID
2. **XOR** hash with a **randomly** generated key
3. Base64 && ~> JSON
4. Encrypt JSON with RSA, using an hardcoded pub key
5. Base64 the JSON

```
http://i.w.inmobi.com/showad.asm?u-id-map=iB7WTkCLJvNsaEQa
kKKXFhk8ZEIZlnL0jqbbYexcBAXYHH4wSKyCDWVfp+q+FeLFTQV6jS2Xg9
7liEzDkw+XNTghe9ekNyMnjypmgiu7xBS1TcwZmFxYOjJkgPOzkI9j2lry
BaLlAJBSDkEqZeMVvcjcNkx+Ps6SaTRzBbYf8UY=&u-key-ver=2198564
```

A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*