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

UCSB

NDSS'17
02-28-2017

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

```java
// 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

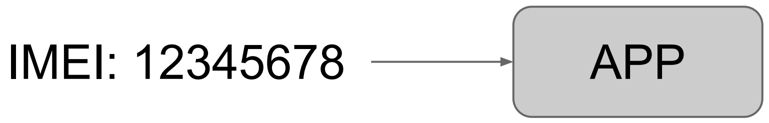
```
// get Android ID using the Java Reflection API
String aid = class.getDeclaredMethod("getAndroidId",
             Context.class).invoke(context);
MessageDigest sha1 = getInstance("SHA-1"); // hash
sha1.update(aid.getBytes());
byte[] digest = sha1.digest();

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

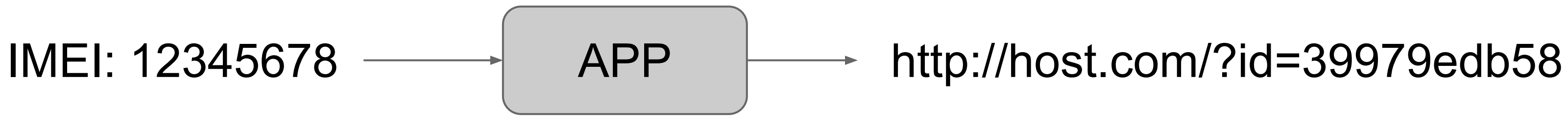
Our Approach

- 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

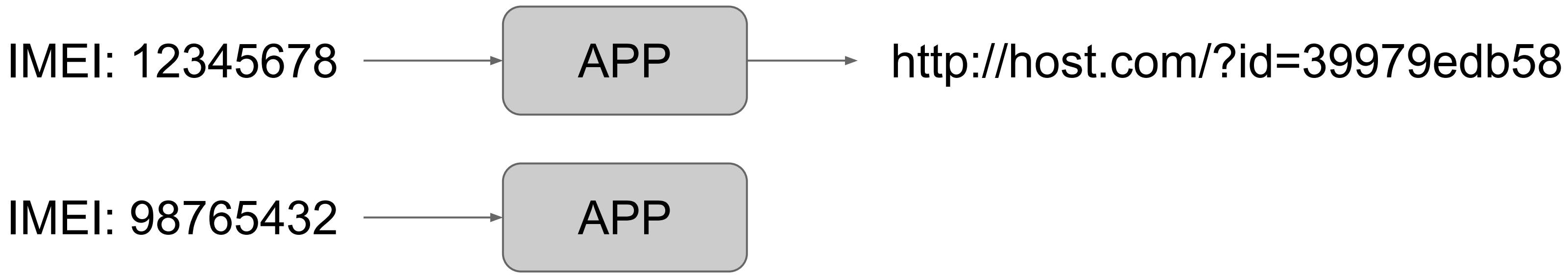

IMEI: 12345678

APP

http://host.com/?id=39979edb58

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

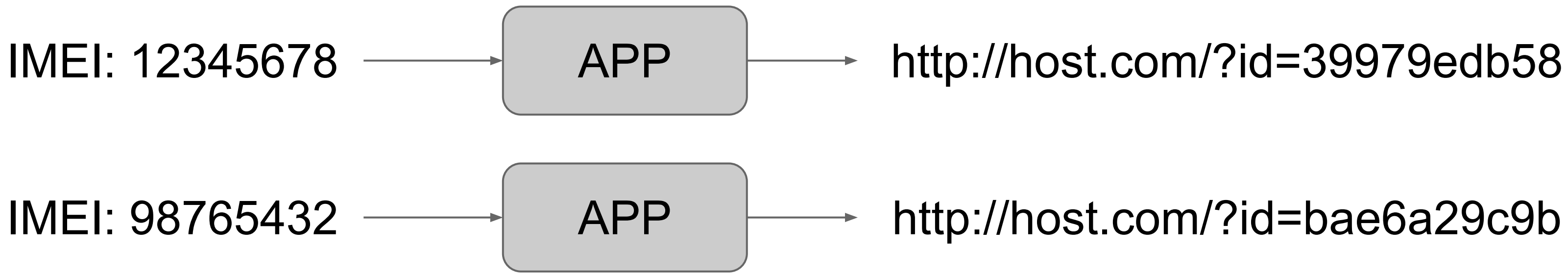
http://host.com/?id=39979edb58

IMEI: 98765432

APP

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

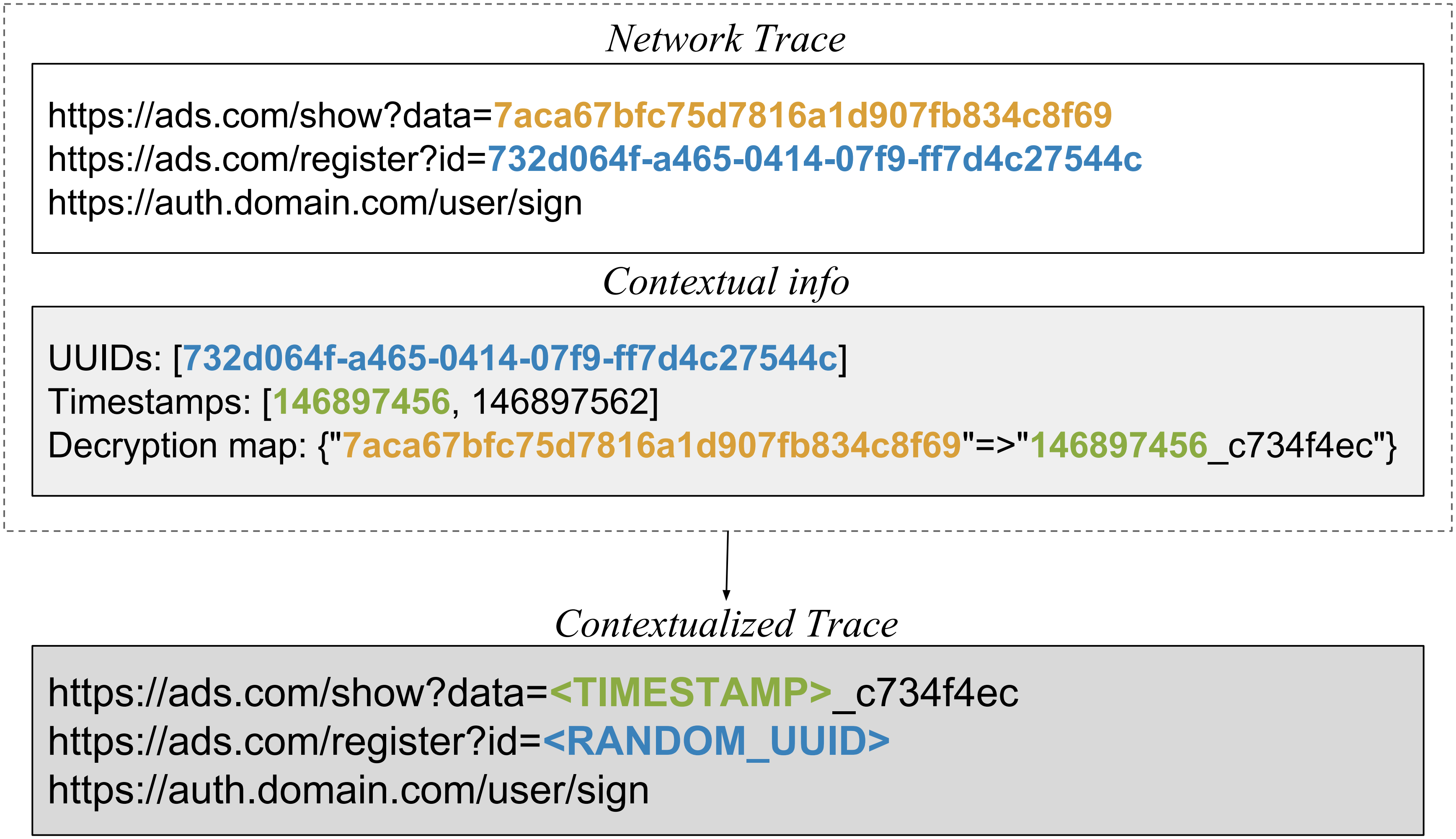
Encryption

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

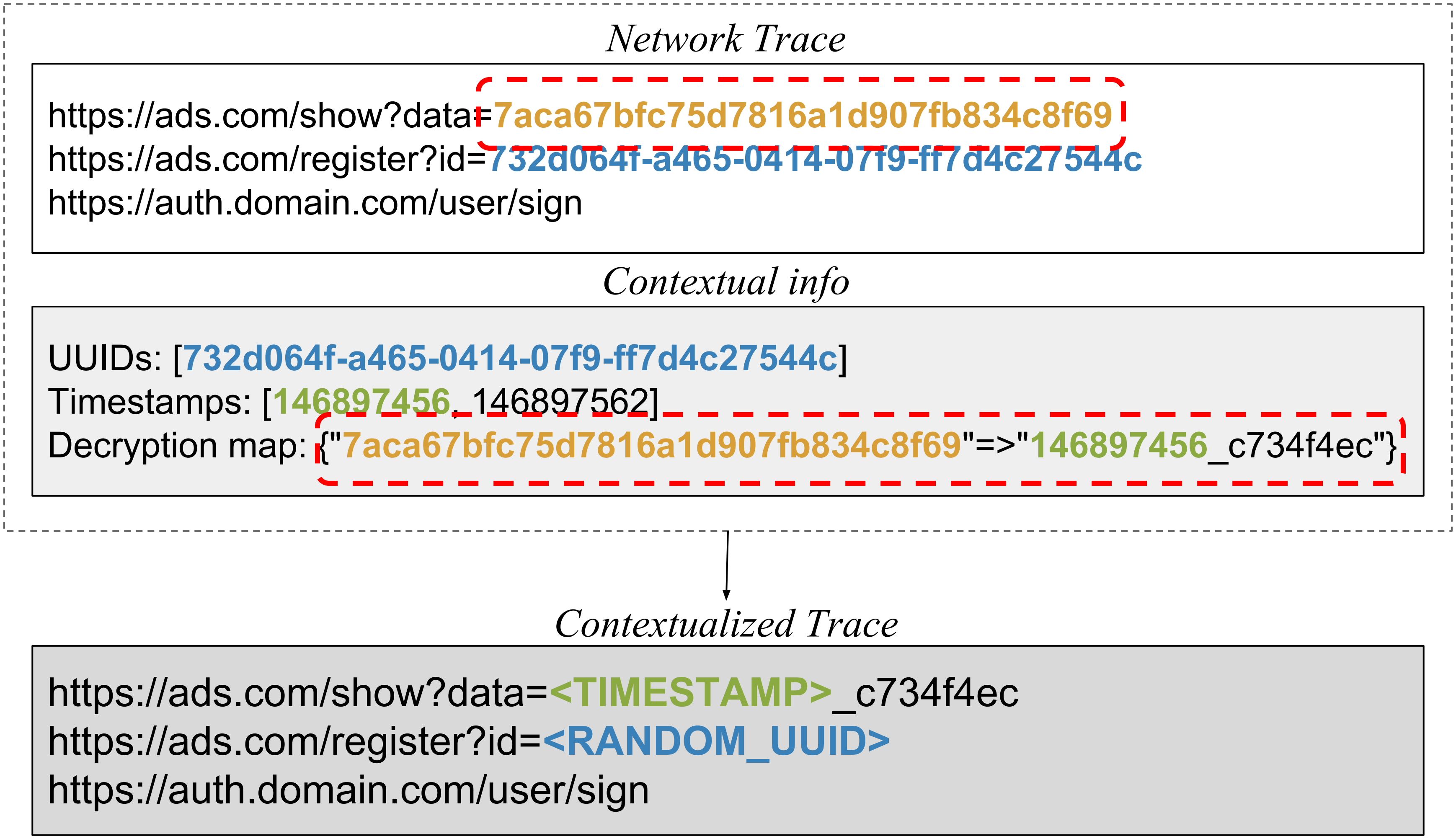
Contextual info

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

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

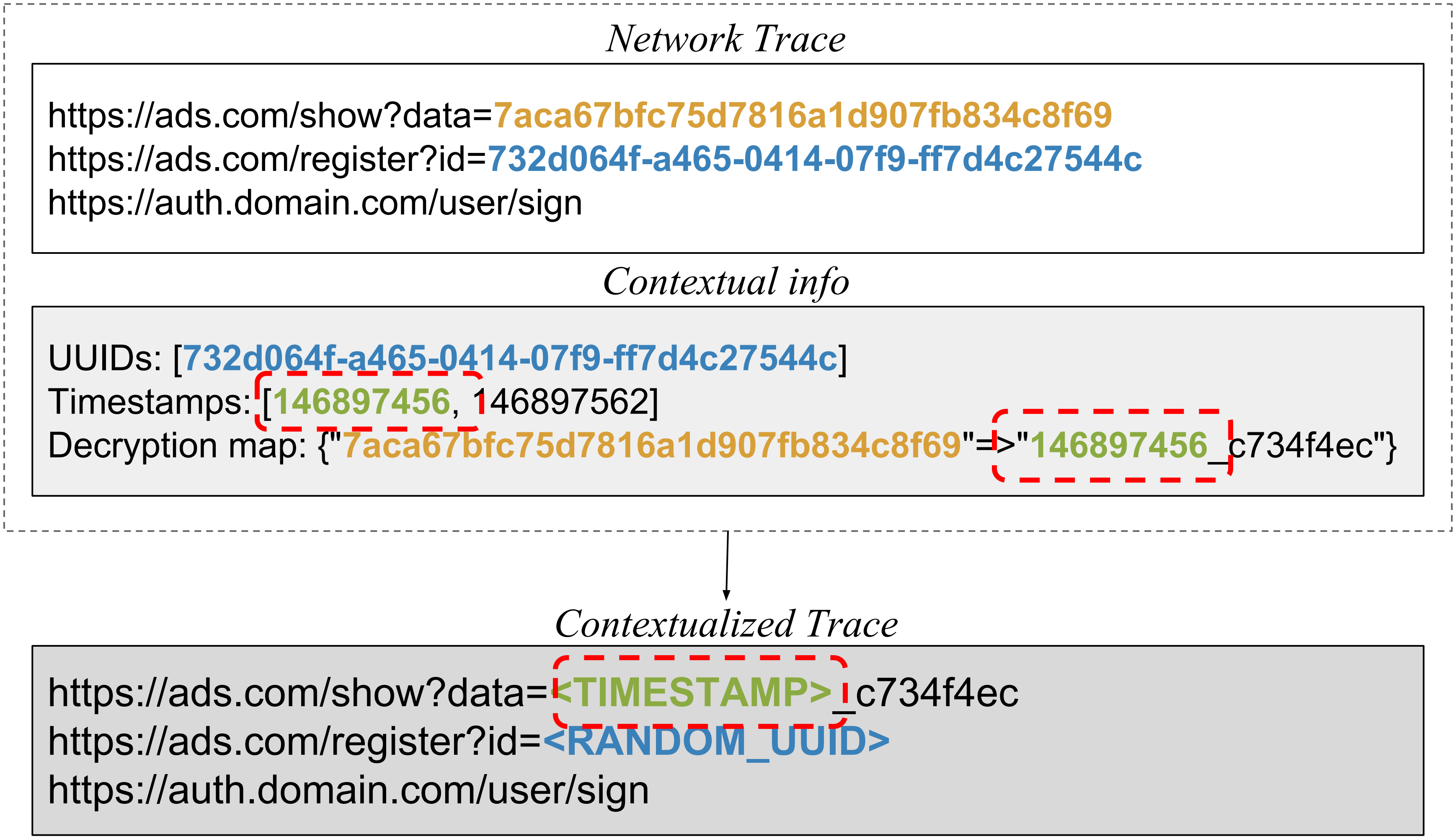
Contextual info

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

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

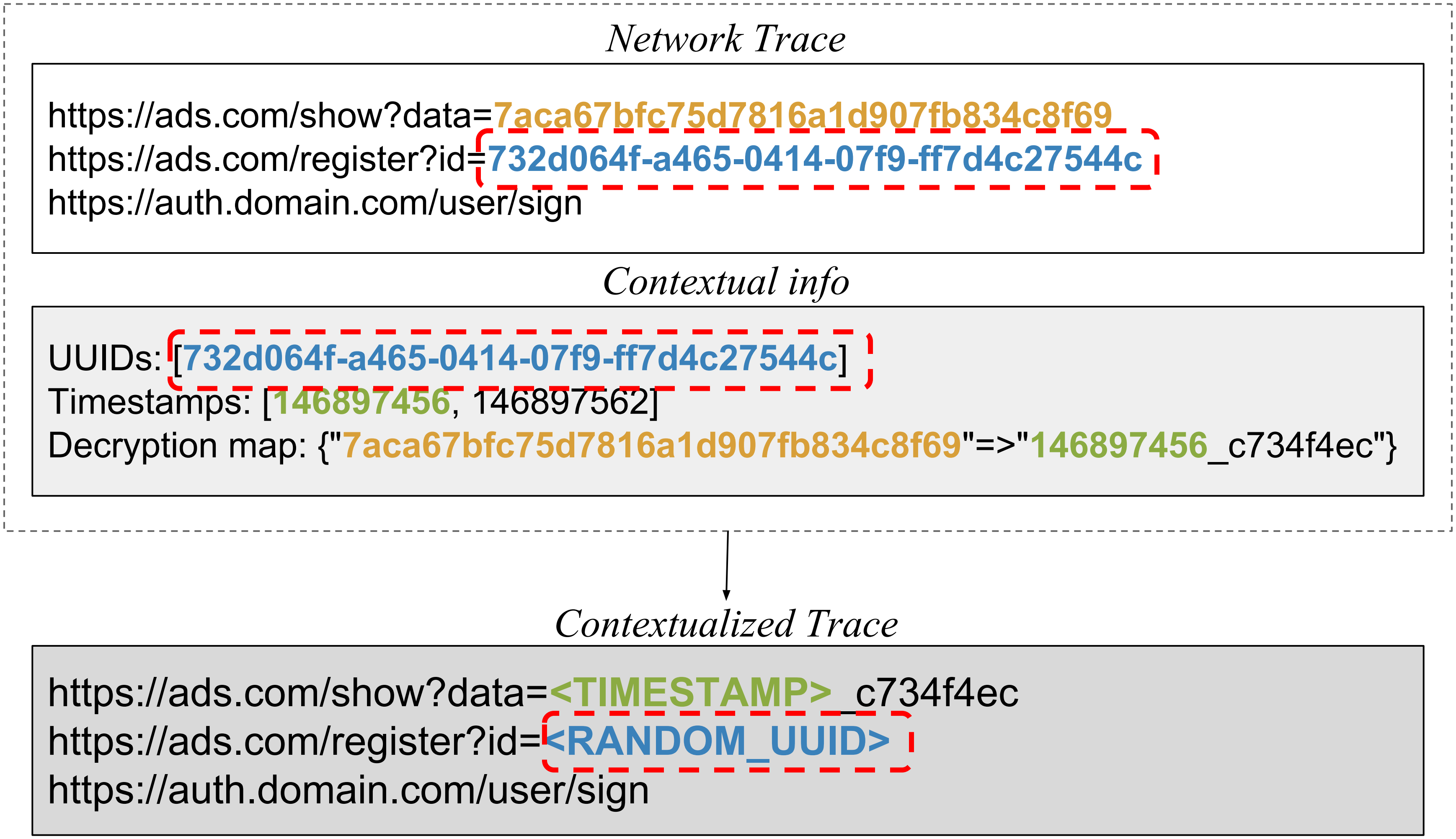
Contextual info

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

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

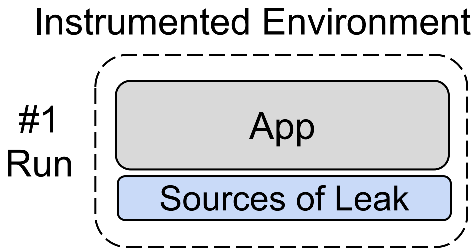
Contextual info

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

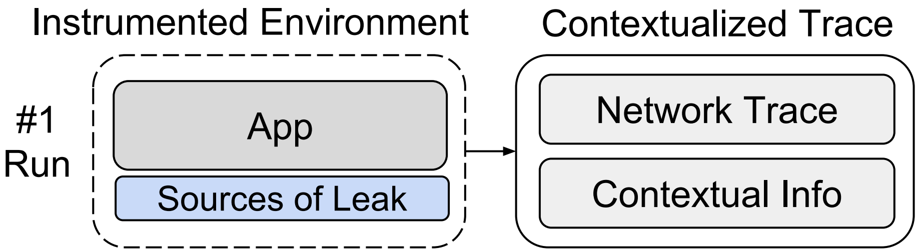
Contextualized Trace

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

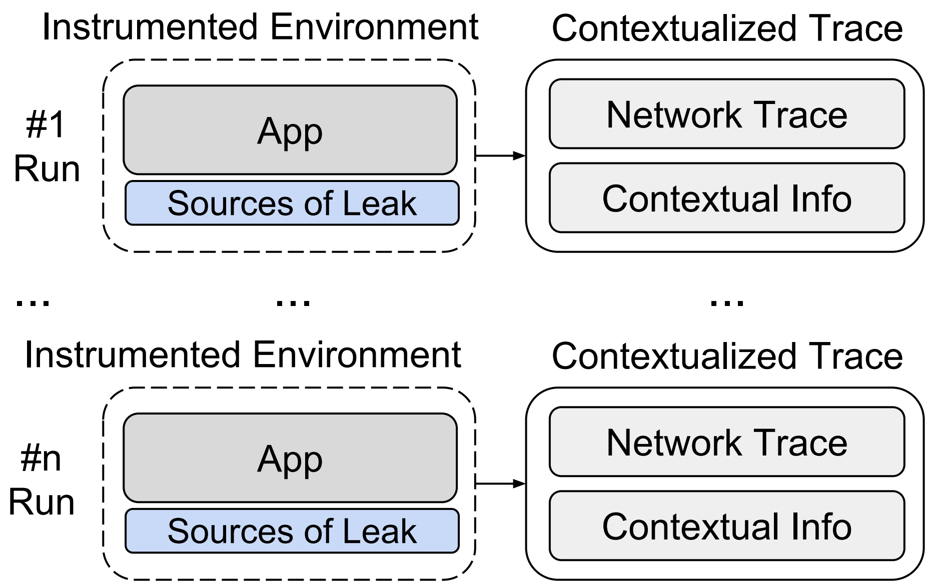
Agrigento: High-level Overview



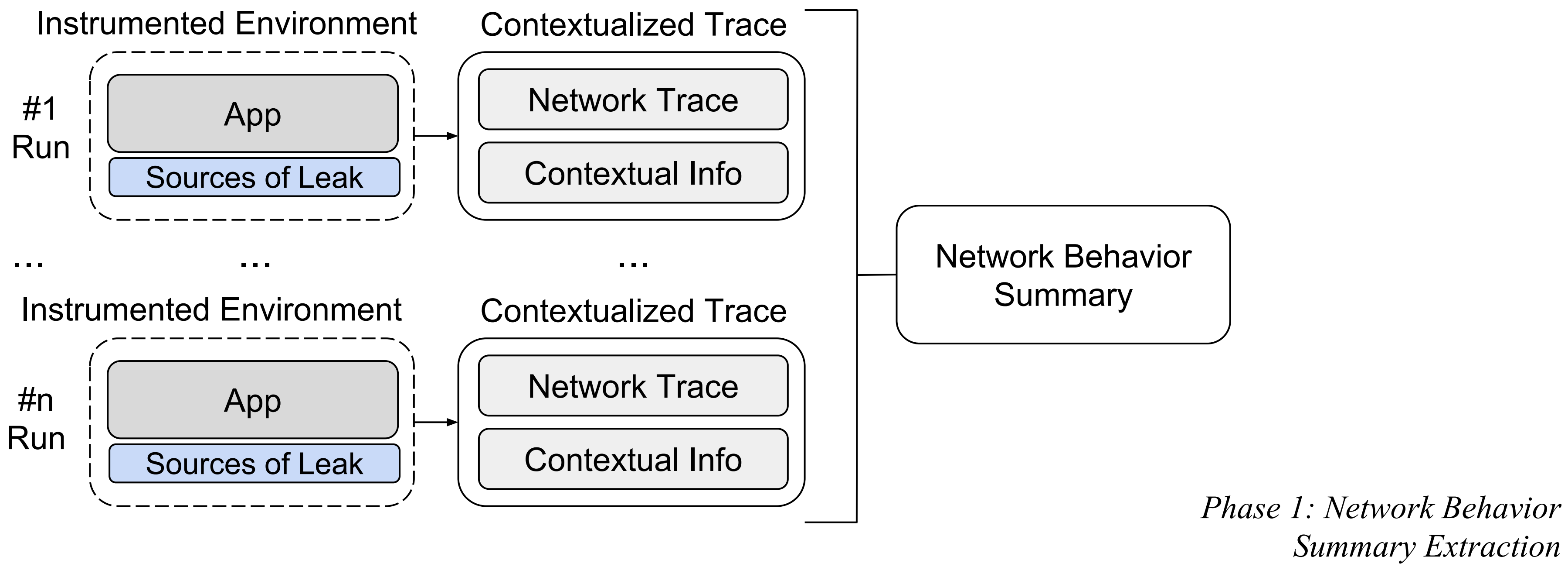
Agrigento: High-level Overview



Agrigento: High-level Overview

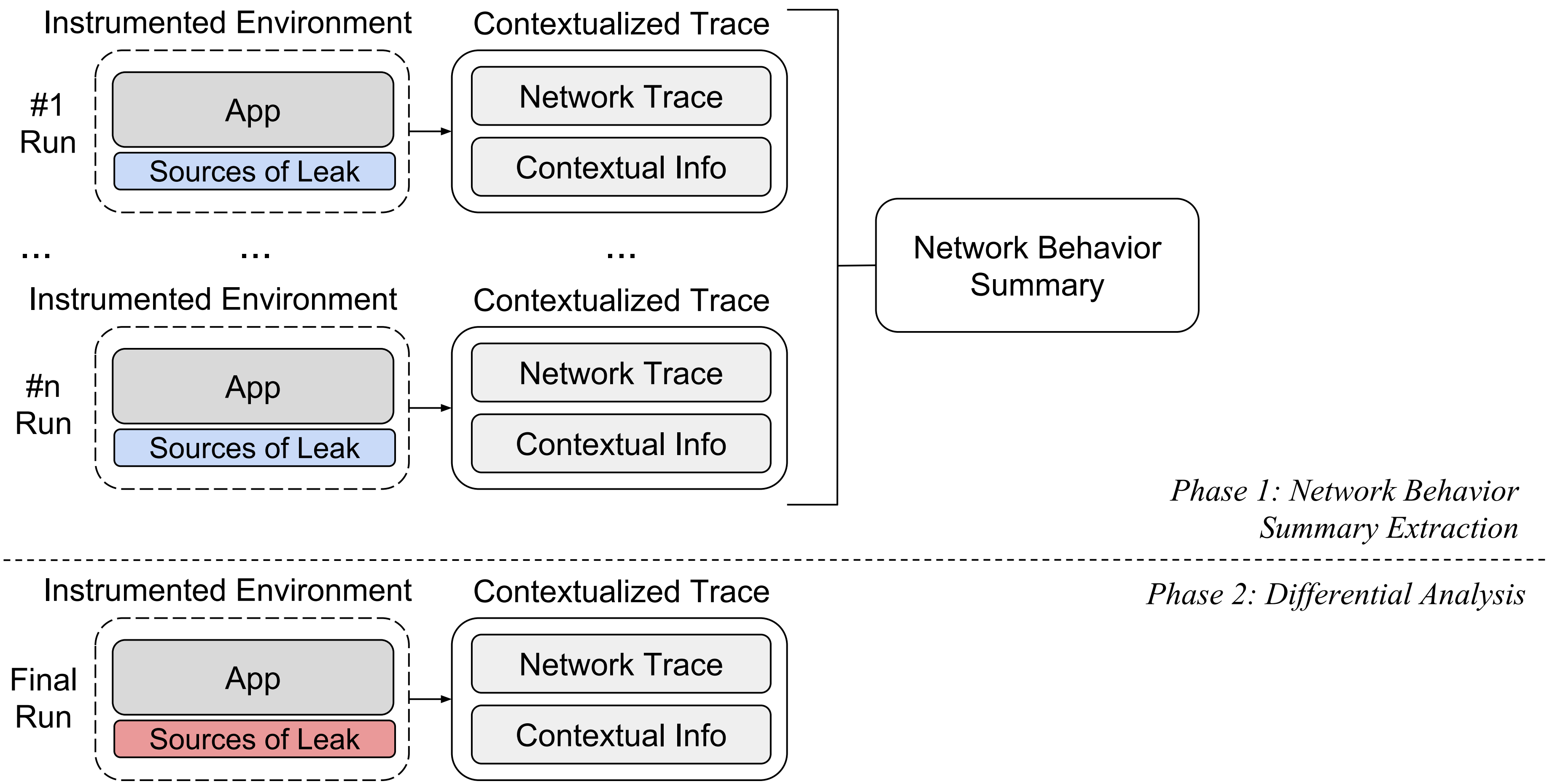


Agrigento: High-level Overview



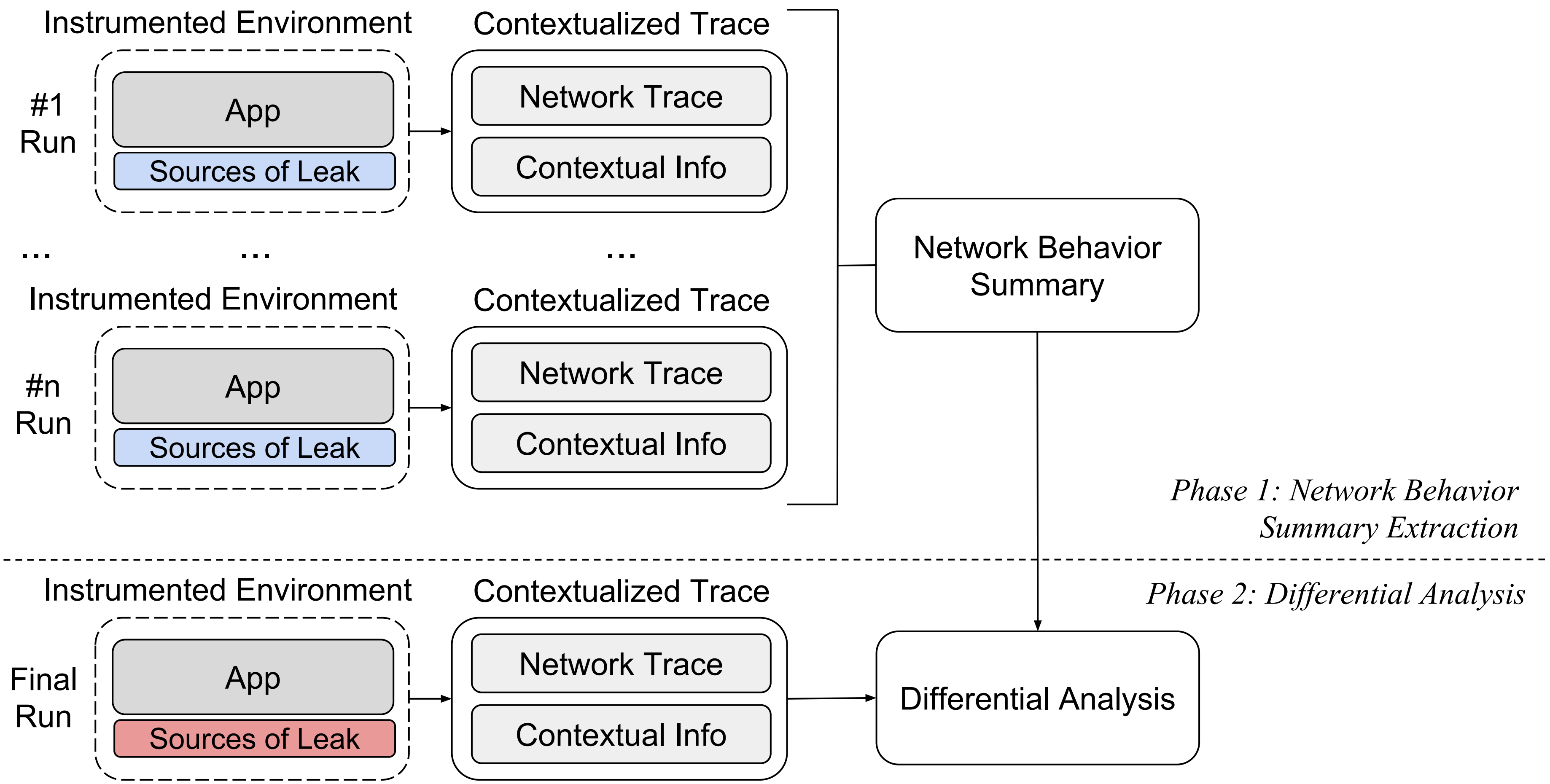
A. Contininella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Agrigento: High-level Overview

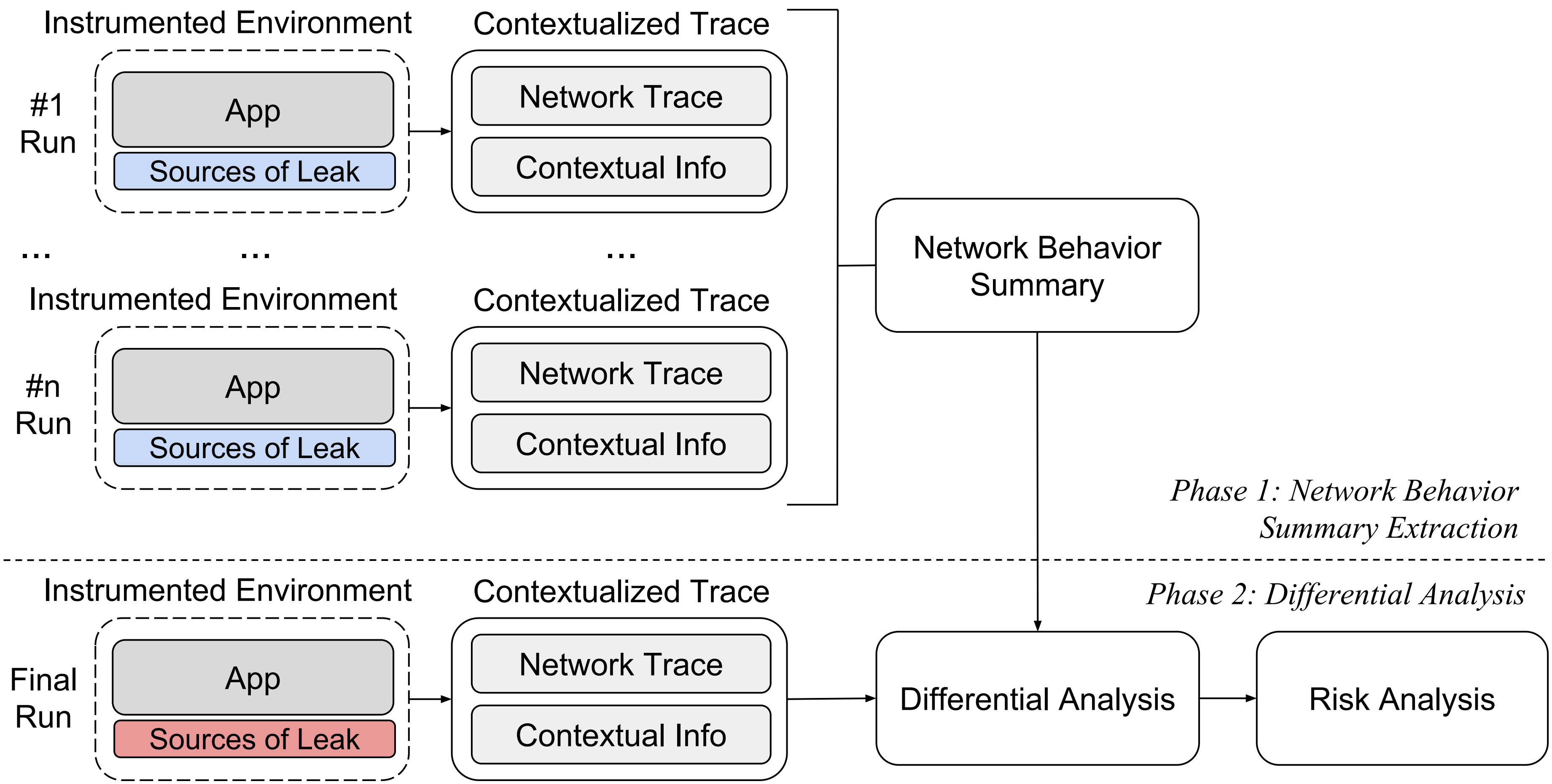


A. Continella et al. *Obfuscation-Resilient Privacy Leak Detection for Mobile Apps Through Differential Analysis*

Agrigento: High-level Overview

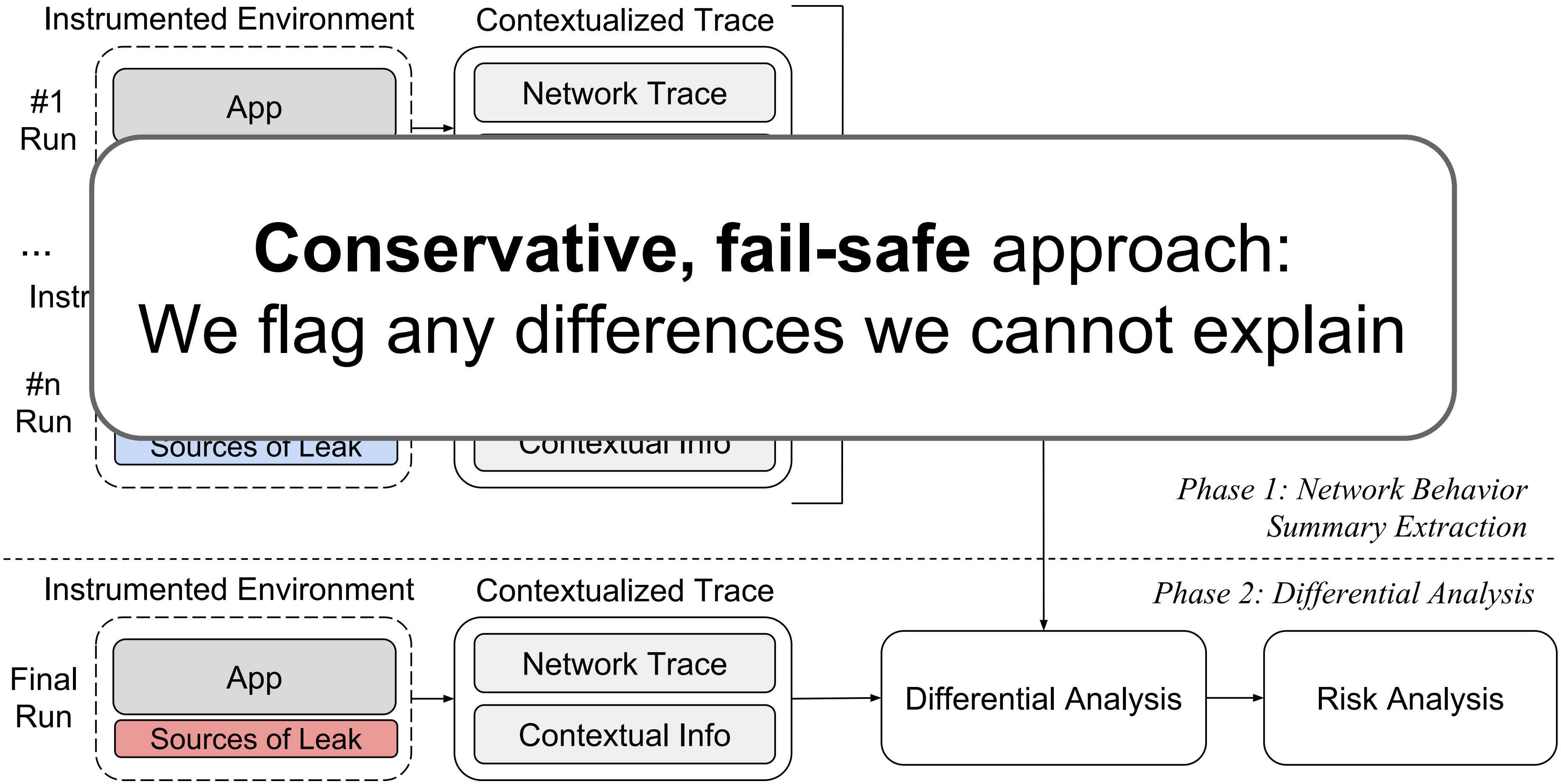


Agrigento: High-level Overview



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

Agrigento: High-level Overview

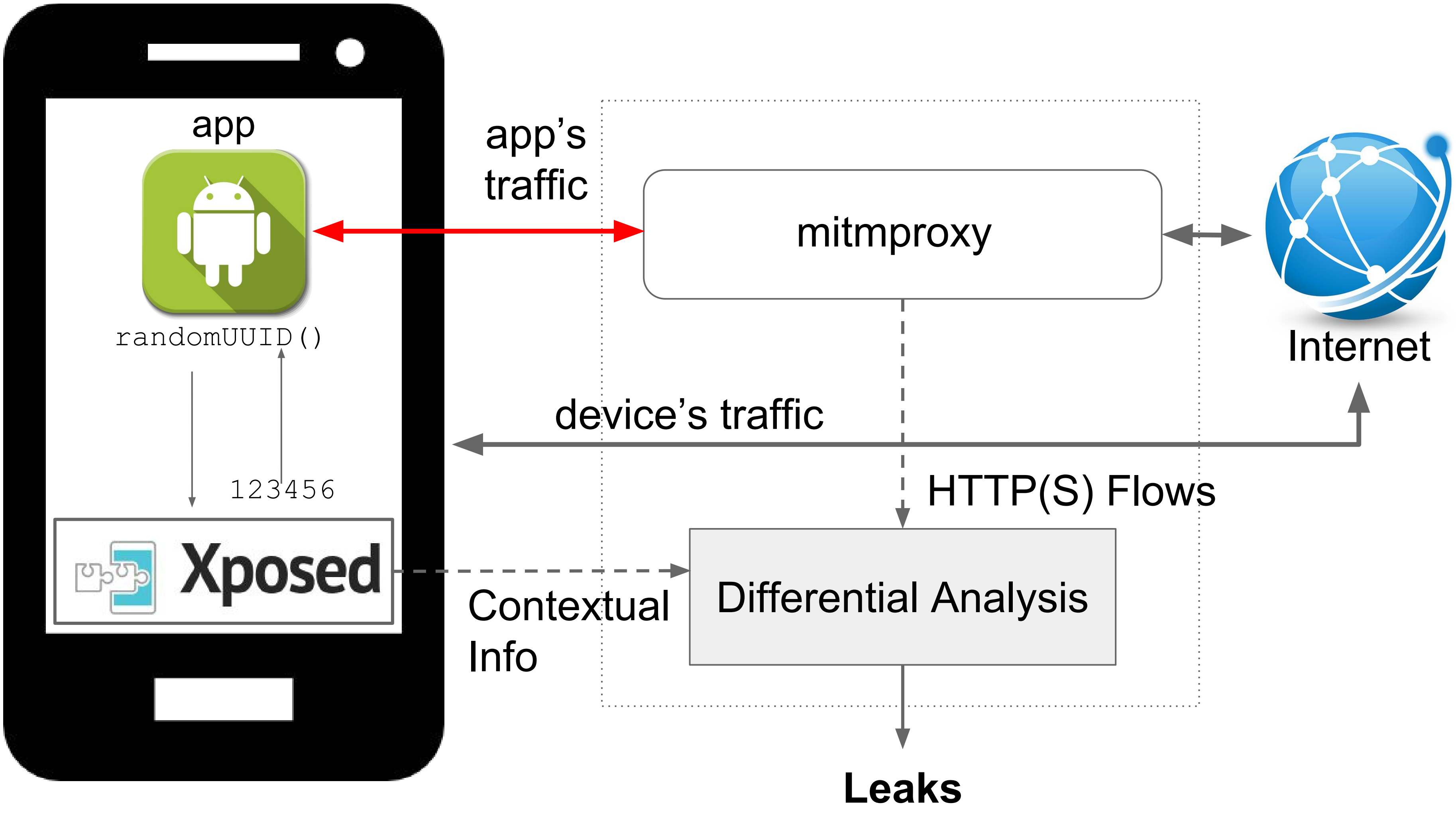


Conservative, fail-safe approach:
We flag any differences we cannot explain

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

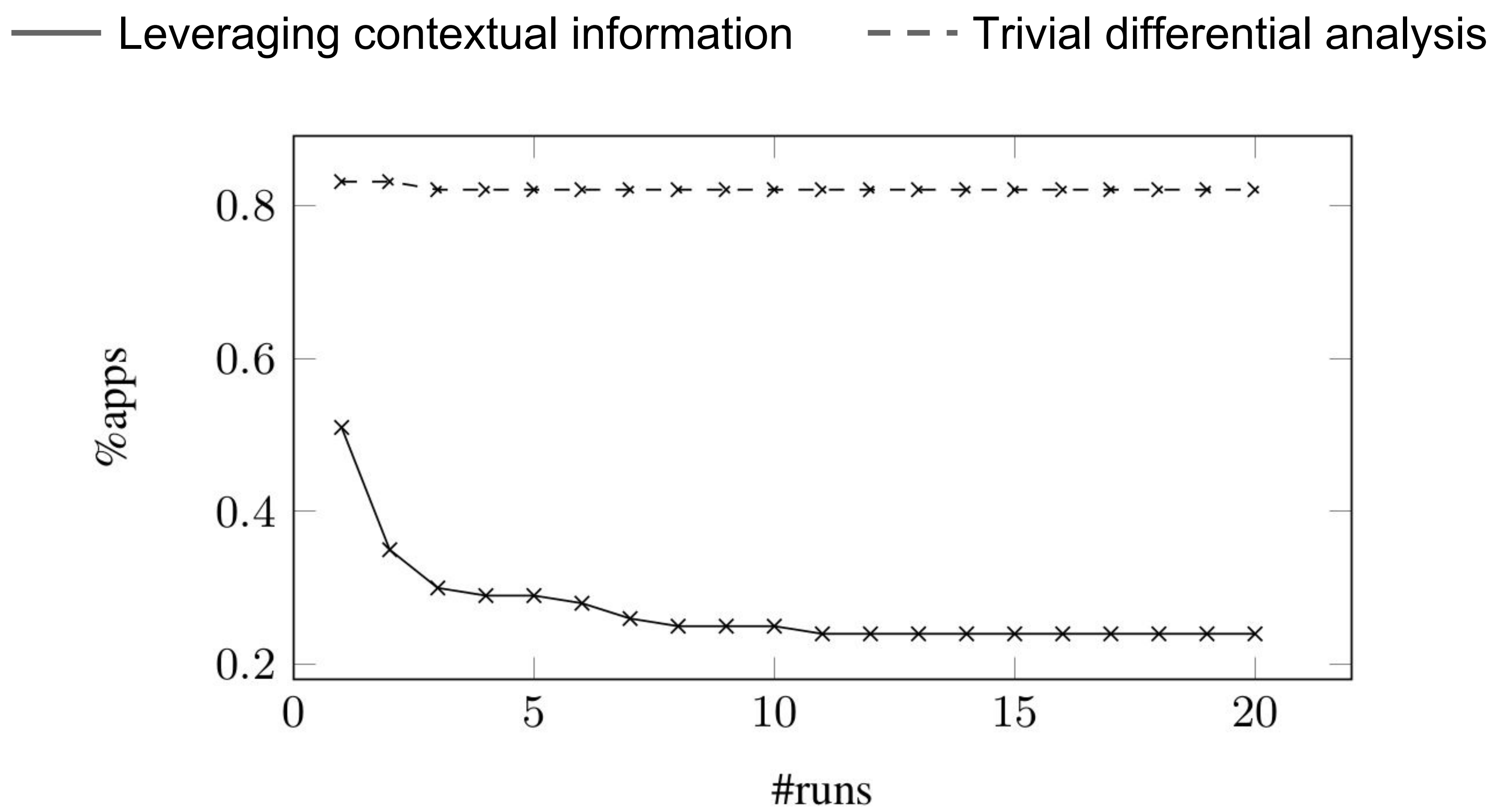


Experimental Setup & Datasets

- 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

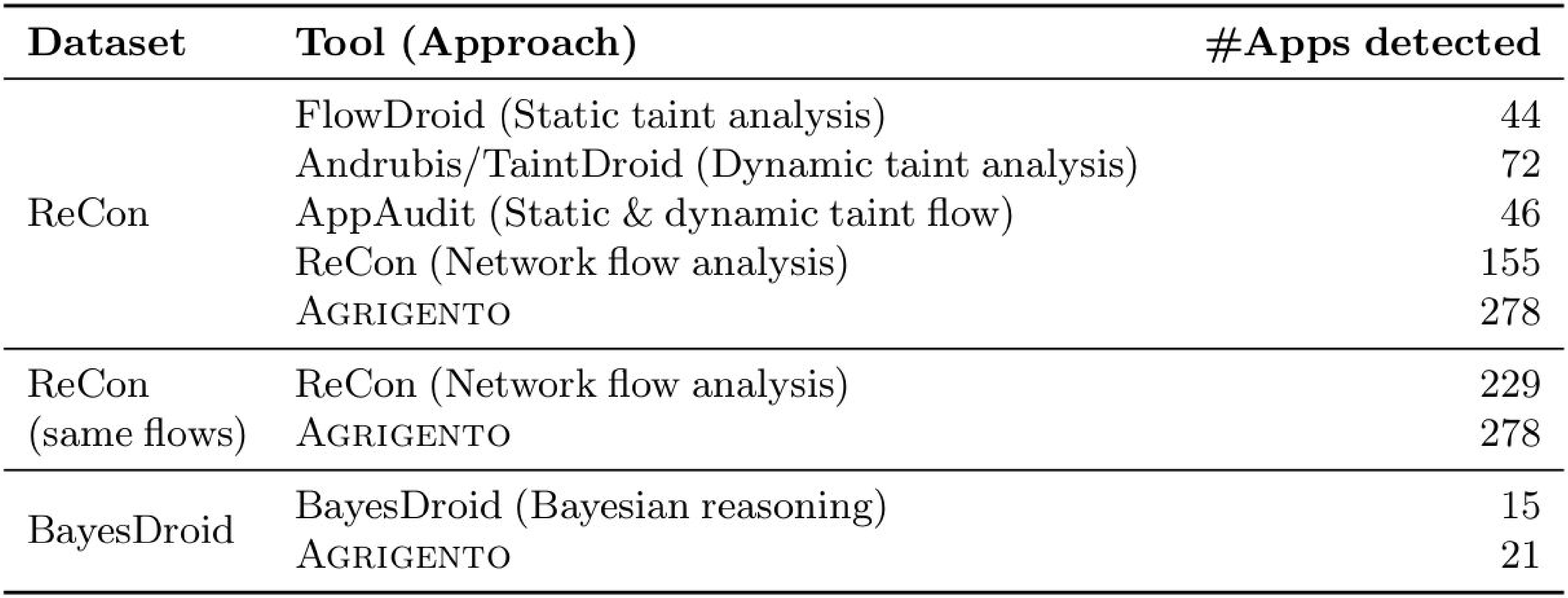
Non-Determinism in Network Traffic

- 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

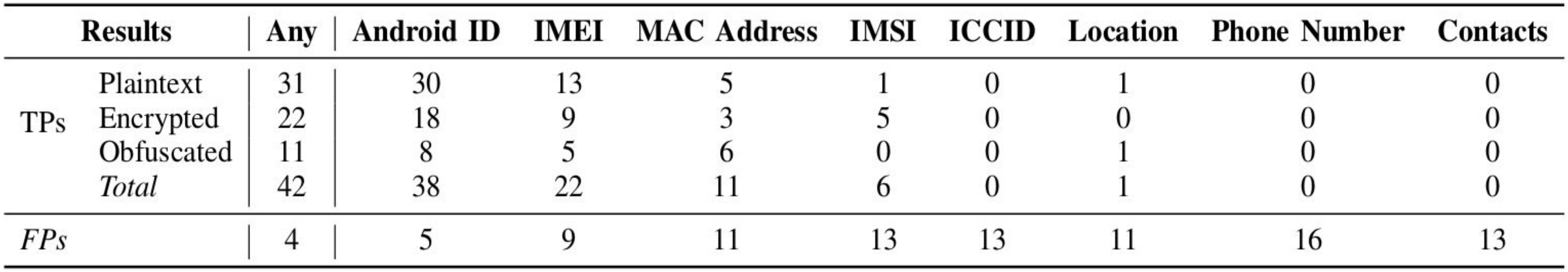


Dataset	Tool (Approach)	#Apps detected
ReCon	FlowDroid (Static taint analysis)	44
	Andrubis/TaintDroid (Dynamic taint analysis)	72
	AppAudit (Static & dynamic taint flow)	46
	ReCon (Network flow analysis)	155
	AGRIGENTO	278
ReCon (same flows)	ReCon (Network flow analysis)	229
	AGRIGENTO	278
BayesDroid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

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

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
30247a3f363026663d333539347a3132383826 6c603d687c76
7225316325306625306661 6f6e74656e762f6a732c74637062
6f7926636f652466723f6a7476702531612732662532666 16d
6d2e65616f656b69726b75732672706978 67636e6177302 66a
683d6561643761373231643135 3c65613a31386e6760656330
373636393634343363266d64643f6561633336303b64336a39
35316663303666636137326 1363a61616335636761266d6673
3f353b32306d38323061323064 3b653464393438 3a31663636
623b3232376761612661 6d65613d31393333313333313331 31
333133312661743d6365656e765f6f6f6a696c6d26617e3f76
727771746665666 76e6665722b6d6f606b6c652733632b392e
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=3330342673f3a3930
643667663b33383831303d343526613f2d363830247a3f363026663d333539
347a3132383826 6c603d687c7672253163253066253066616f6e74656e762f
6a732c746370626f7926636f652466723f6a74767025316127326625326661
6d6d2e65616f656b69726b757326727069786763 6e617730266a683d656164
376137323164313 53c65613a31386e676065633037363639363434 3363266d
64643f65616333363 03b64336a393531666330366663613732613 63a616163
3563676126 6d66733f353b32306d383230613230643b653464393438 3a3166
3636623b323237676161 26616d65613d3139333333133333133313133313331
2661743d6365656e765f6f6f6a696c6d26617e3f7672777174666566 676e66
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