The Skeleton Keys: A Large Scale Analysis of Credential Leakage in Mini-apps

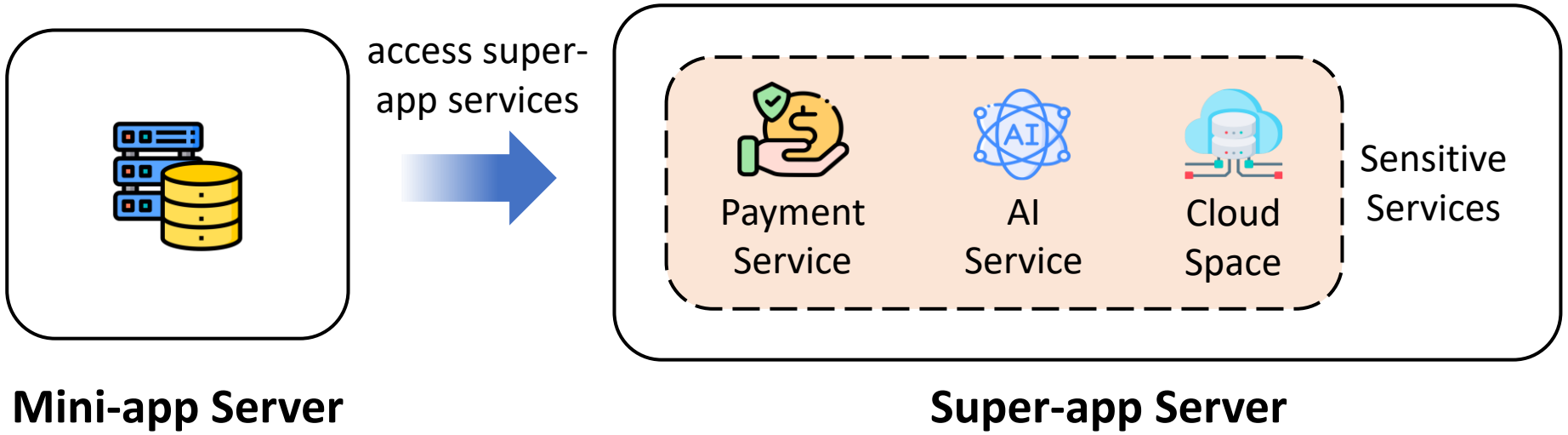
Yizhe Shi, Zhemin Yang, Kangwei Zhong,

Guangliang Yang, Yifan Yang, and Min Yang

Fudan University

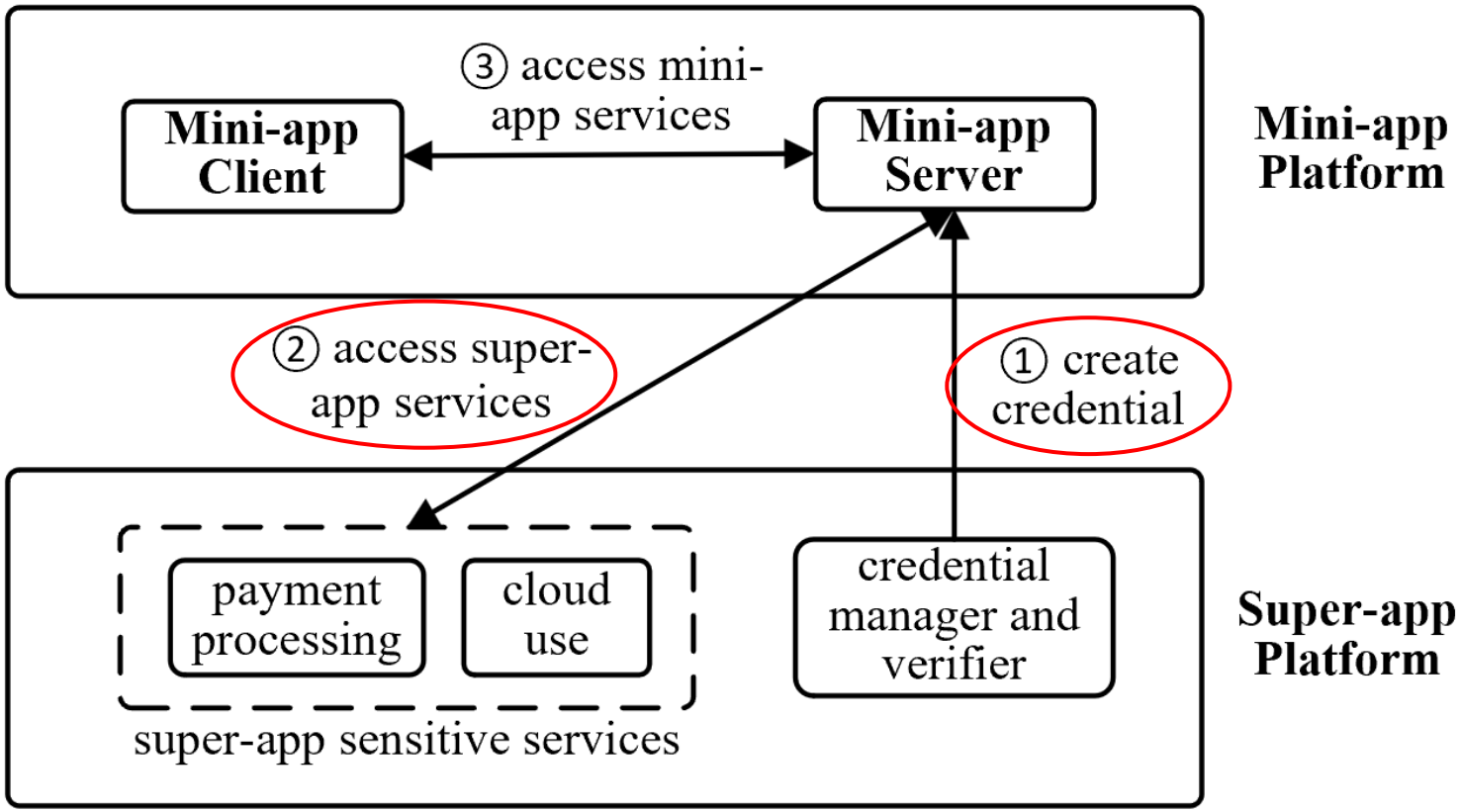
App-in-App Ecosystem

- Super-app
  - A mobile app with rich functionalities, often delegate sensitive services to mini-apps
  - e.g., WeChat, TikTok, Alipay, Baidu
- Mini-app
  - Runs within super-apps, offering a native app-like experience



Credential System

- **Credential-based access control** to safeguard sensitive services



Credential System

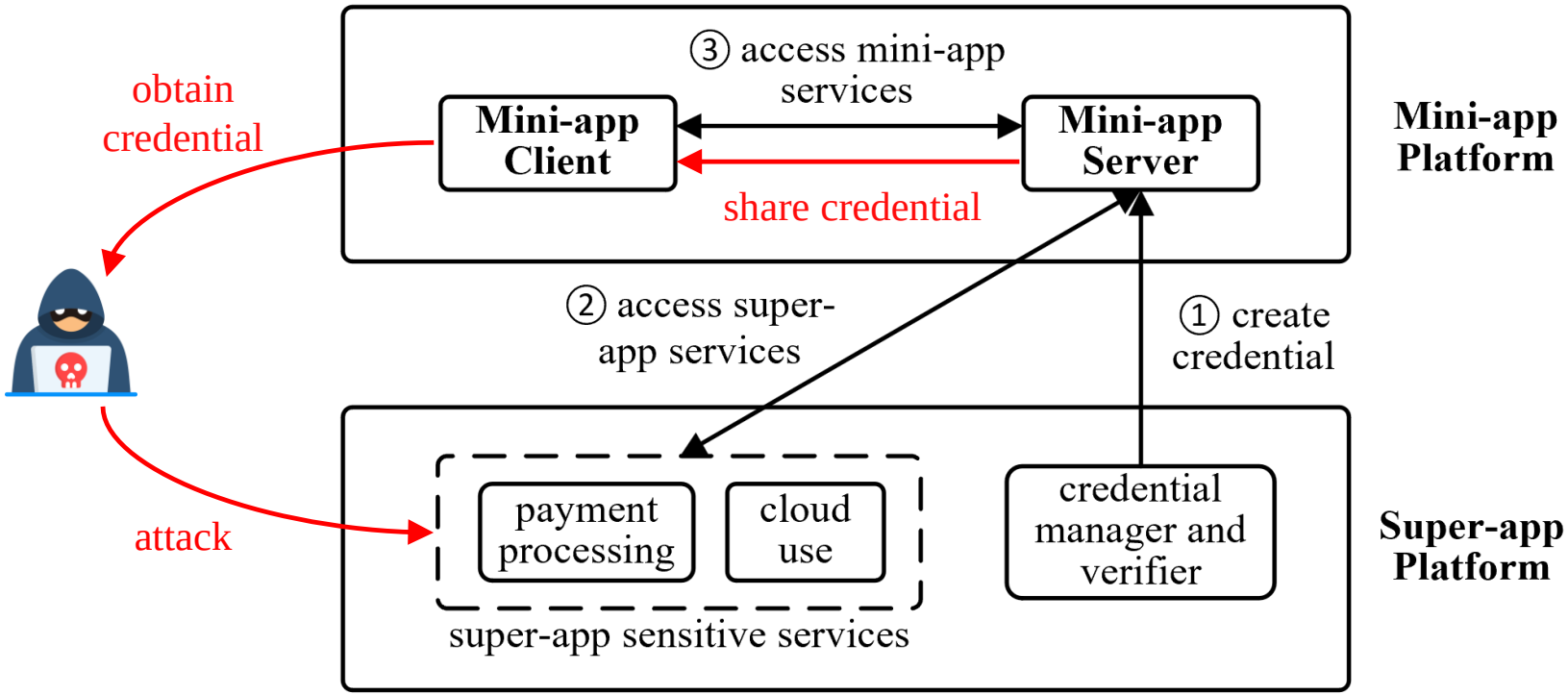
- **Credential-based access control** to safeguard sensitive services

	Credential Type	WeChat	Baidu	Alipay	Tiktok	Line	VK	WeCom	QQ	Feishu	JINRI toutiao	Watermelon	Pipixia	Kuaishou	Taobao	Cainiao	Koubei	Jingdong	Xiaohongshu	Paytm	UnionPay	DingTalk
Root Credential	Mini-app Root Credential	✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓
	Corporation Root Credential							✓														✓
Access Credential	Mini-app Access Credential	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓
	Corporation Access Credential							✓		✓												✓
	Mini-app Group Access Key						✓															
Cryptographic Credential	Server-to-server Session Key	✓	✓		✓			✓	✓	✓	✓	✓	✓	✓	✓	✓			✓			
	Data Encryption Key	✓		✓													✓			✓	✓	

21 popular super-app platforms delegate sensitive services to mini-app servers with 64 critical credentials

Credential Leakage

- **Proper Practice:** Use credentials in the mini-app server side
- **Credential Leakage:** Improperly share credentials with mini-app clients



Research Status

- Credential leakage in the open repositories or mobile applications
  - **Hard-coded and well-structured** credentials
  - Mostly based on fixed patterns or regular expressions

Hard-coded Credential

```
var awsCred = new AWS.Credentials({
    accessKeyId: “AKIA***PLE”,
    secretAccessKey: “wJalr***Key”,
});
const s3 = new AWS.S3({...})
```

Dynamically Retrieved Credential

```
wx.request({
    url: "https://***/getopenid",
    data: {code: login_code, ...},
    success(a) => {
        var credential = a.key;
    }
})
```

matched

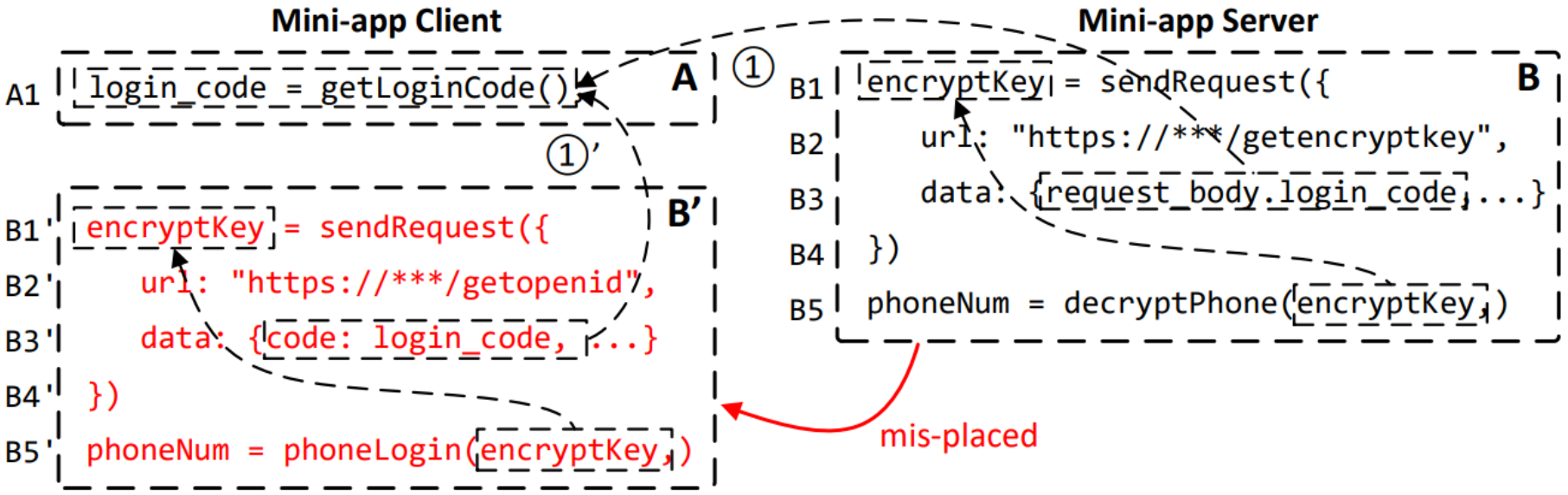
```
- pattern:
    name: AWS Access Key
    regex: "(A3T[A-Z0-9]|AKIA|AGPA|AIDA|AROA|AIPA|ANPA|ANVA|ASIA)[A-Z0-9]{16}"
    confidence: high
- pattern:
    name: AWS Secret Key
    regex: "(?i)aws(.{0,20})?(?-i)['\\\"][0-9a-zA-Z\\/+]{40}['\\\"]"
    confidence: high
- pattern:
    name: AWS MWS key
    regex: "amzn\\.mws\\.[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}"
    confidence: high
```

patterns of credentials

Missed but Significant Problem : Dynamically Leaked Credentials

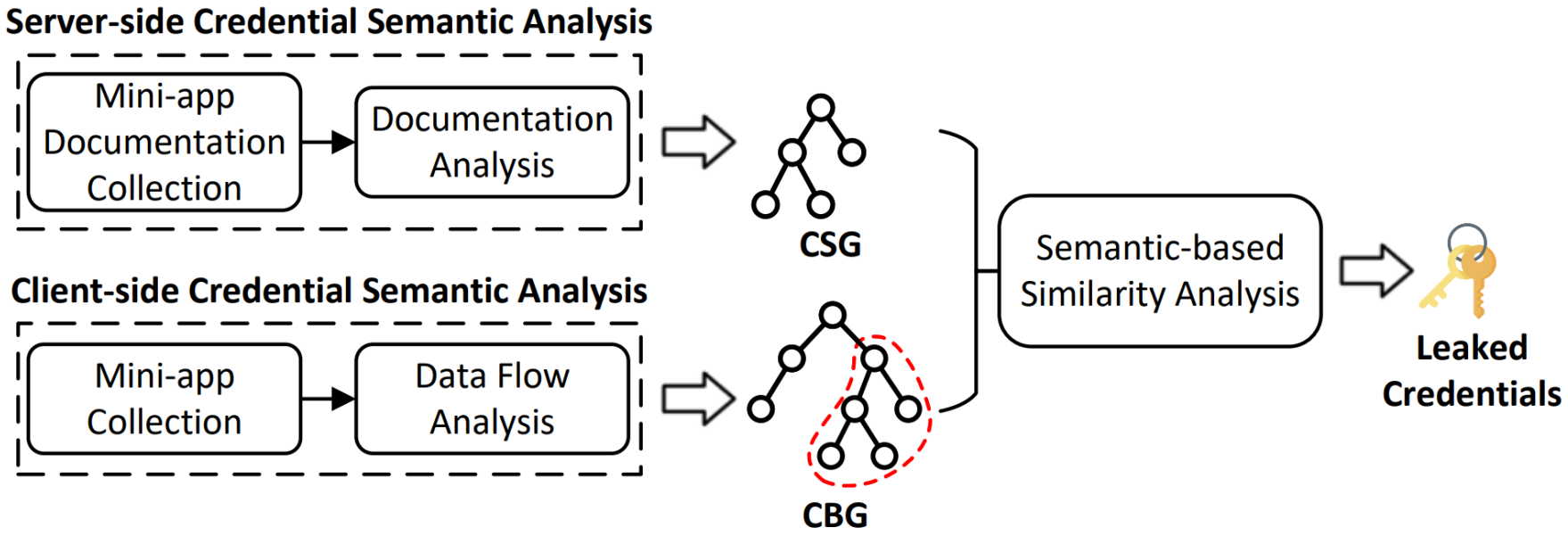
Our Insights

- **Main Insight:** During the **credential migration**, the credential-use behaviors still exhibit similar patterns



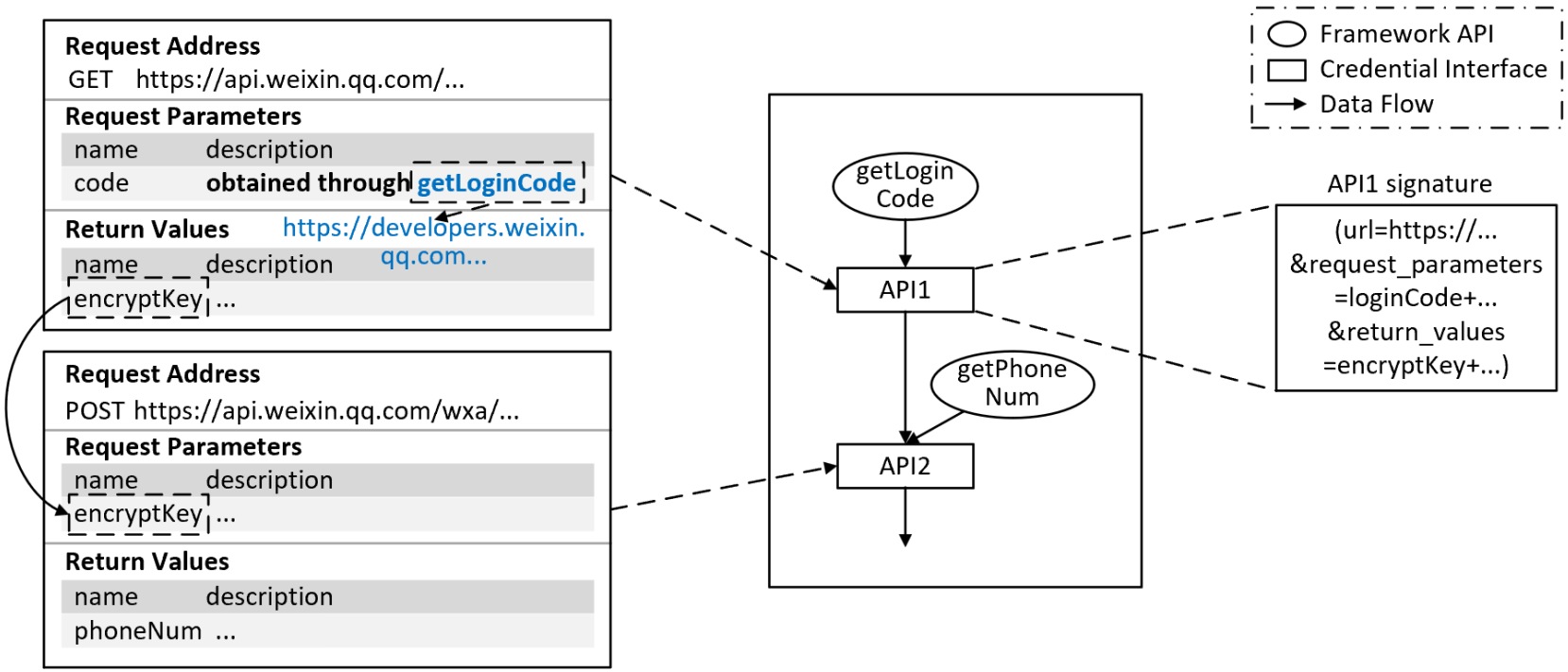
Architecture

- **KeyMagnet**
  - Phase #1: Server-side Credential Semantic Analysis
  - Phase #2: Client-side Credential Semantic Analysis
  - Phase #3: Semantic-based Similarity Analysis



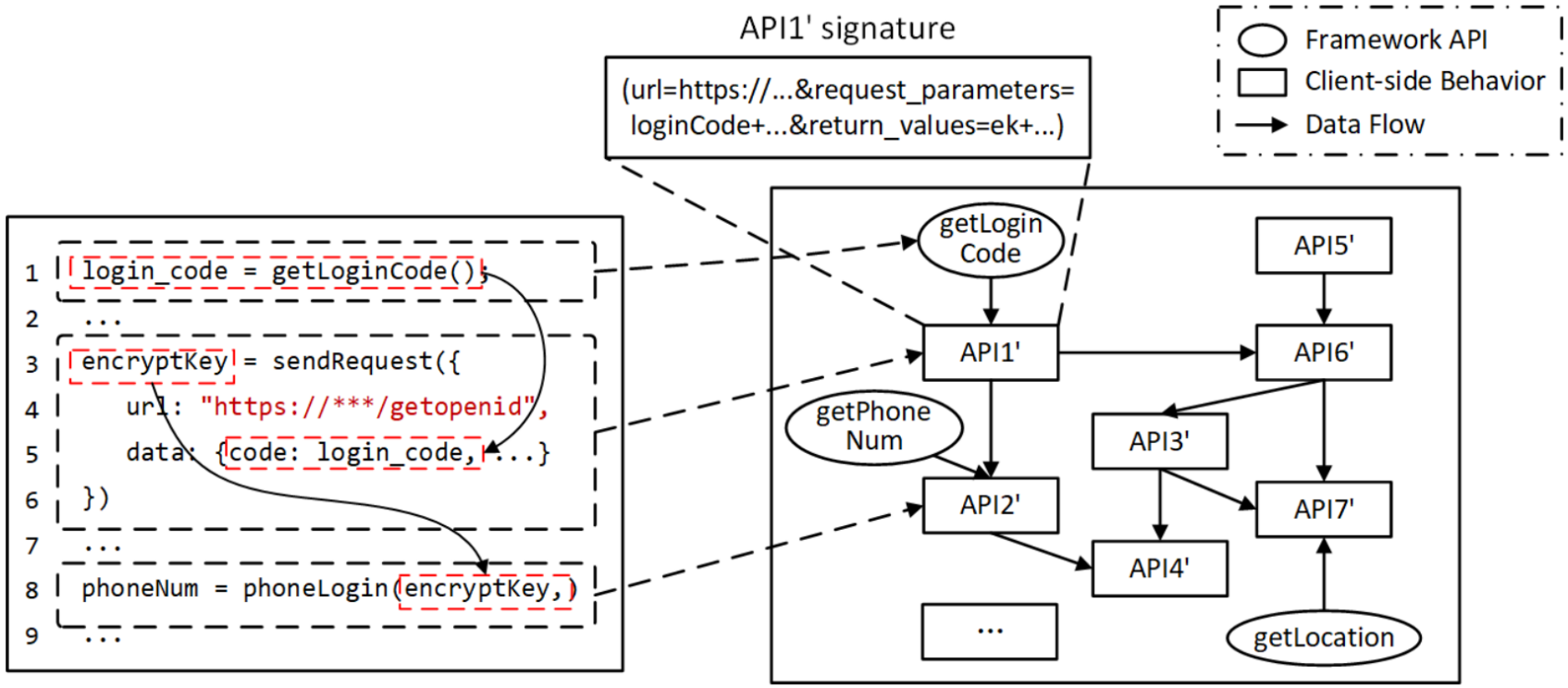
Server-side Credential Semantic Analysis

- **Insight:** Developer documentation offers the hints of credential-use semantics in the server side
- **Credential-use Semantic Graph:** Represent the API-level credential semantics



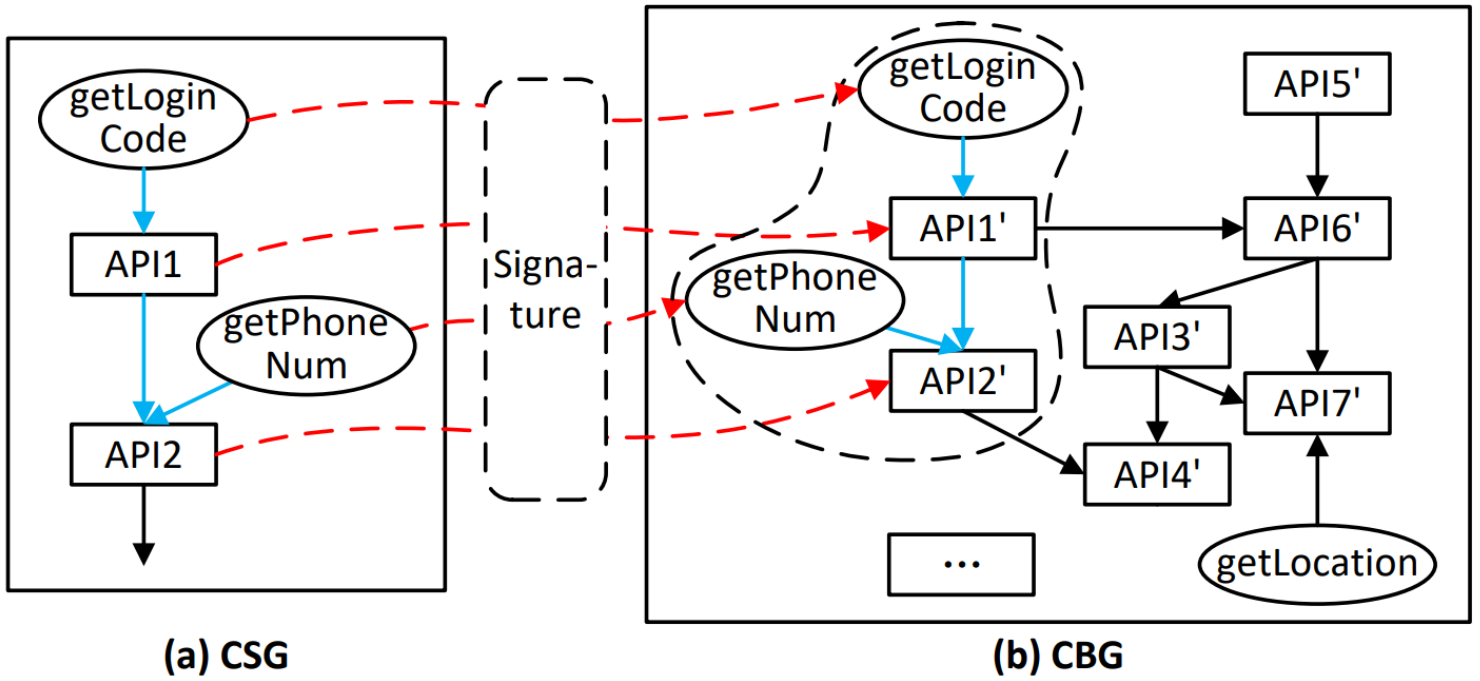
Client-side Credential Semantic Analysis

- **Insight:** Client-side network behaviors encompass credential-use semantics
- **Client-side Behavior Graph:** Represent the mini-app client-side semantics



Semantic-based Similarity Analysis

- **Challenge:** Semantic gap between client- and server-side semantics
- **Approach:** Prove the semantic isomorphism between the CSG and CBG



Evaluation - Performance

- Performance of KeyMagnet
  - Ground Truth : Randomly sampled 500 mini-apps that are identified as vulnerable and non-vulnerable
  - Precision: 95.04% / Recall: 85.56%

Super-app	TP	FP	TN	FN	Precision	Recall	F1-score
WeChat	466	34	389	111	93.20%	80.76%	86.54%
Baidu	483	17	488	12	96.60%	97.58%	97.09%
Alipay	478	22	430	70	95.60%	87.23%	91.22%
TikTok	476	24	363	137	95.20%	77.65%	85.53%
Line	110	8	490	10	93.22%	91.67%	92.44%
VK	0	0	425	0	-	-	-
Overall	2013	105	2585	340	95.04%	85.56%	90.05%

Evaluation - Landscape

- Statistics of Credential Leakage
  - 84,491 credential leakage issues in 54,728 mini-apps

Super -app	Root Credential		Access Credential		Crypto Credential	
	#app	%total	#app	%total	#app	%total
WeChat	22207	10.89%	20987	10.29%	**23421**	**11.48%**
Baidu	517	0.60%	336	0.39%	**1085**	**1.26%**
Alipay	3929	4.24%	**5916**	**6.38%**	3092	3.33%
TikTok	268	1.78%	**1889**	**12.56%**	726	4.83%
Line	**69**	**1.73%**	49	1.23%	0	0
VK	0	0	0	0	0	0
Overall	26990	6.71%	**29177**	**7.25%**	28324	7.04%

Observation

- Credential Leakage Crossing Apps
  - Cross-superapp credential leakage
  - Cross-miniapp credential leakage
- Template-based Leakage
- Leakage Scenarios

Security Hazards

Service	WeChat	Baidu	Alipay	Tiktok	Line	VK
Mini-app Login	✓	✓	✓	✓	✓	✓
UserInfo Retrival	✓			✓		✓
Data Analysis Service	✓	✓		✓		✓
Customer Service	✓	✓	✓	✓		
Message Service	✓	✓	✓	✓	✓	✓
Cloud Development	✓					
Payment Service	✓		✓	✓	✓	
AI Service	✓					
Logistics Service	✓	✓				
Shopping Service	✓		✓	✓		
Promotion Service		✓		✓		✓
Live Streaming Service	✓			✓		
Bio-authentication	✓		✓			
Content Security Check	✓	✓		✓		✓
Short Link Generation	✓	✓	✓	✓		✓

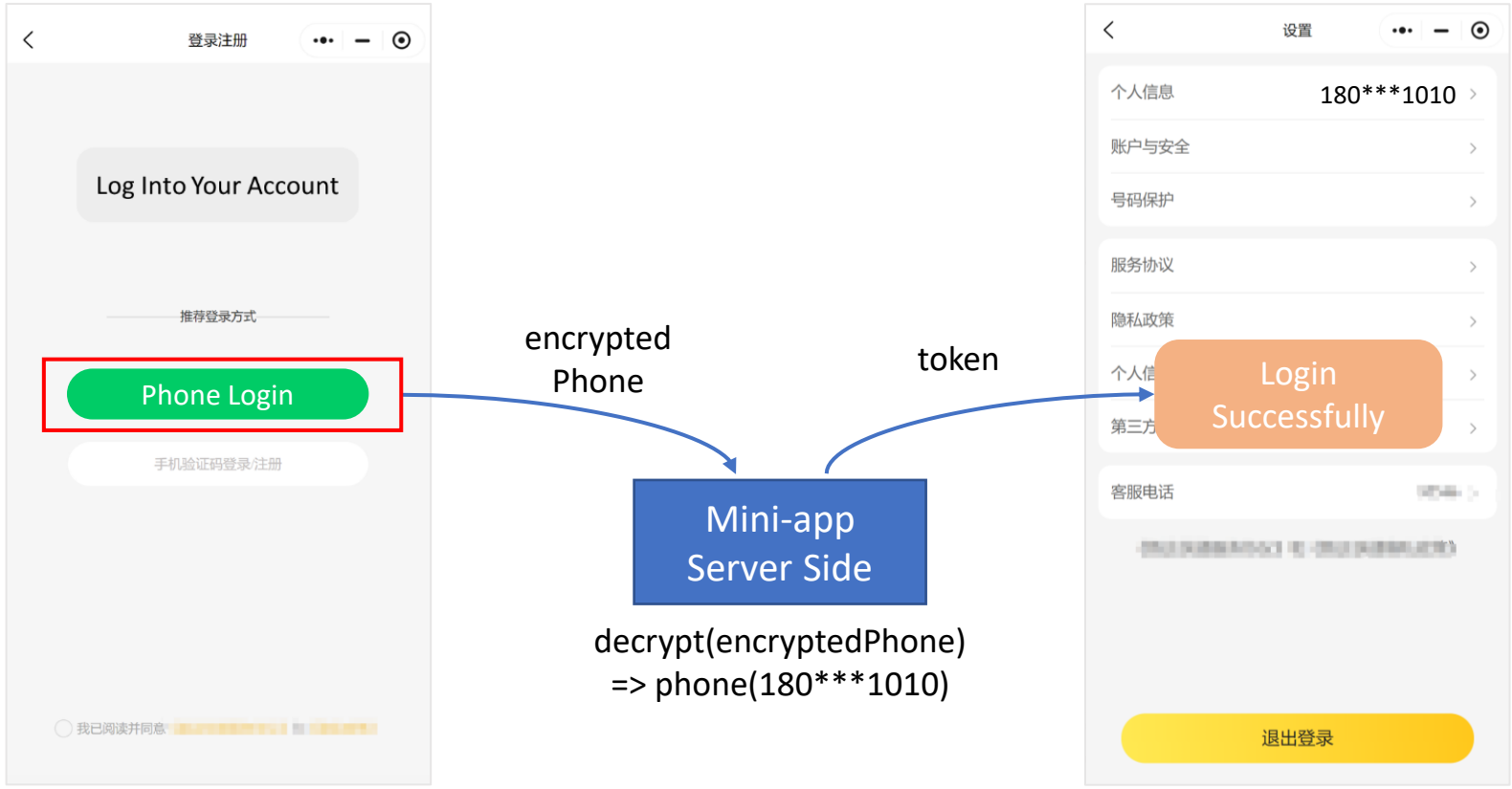
Functionality of Credentials in Mini-apps

security hazards

- Account Hijacking
- Payment Deception
- Phishing Attack
- Sensitive Information Theft
- Mini-app Function Manipulation
- . . .

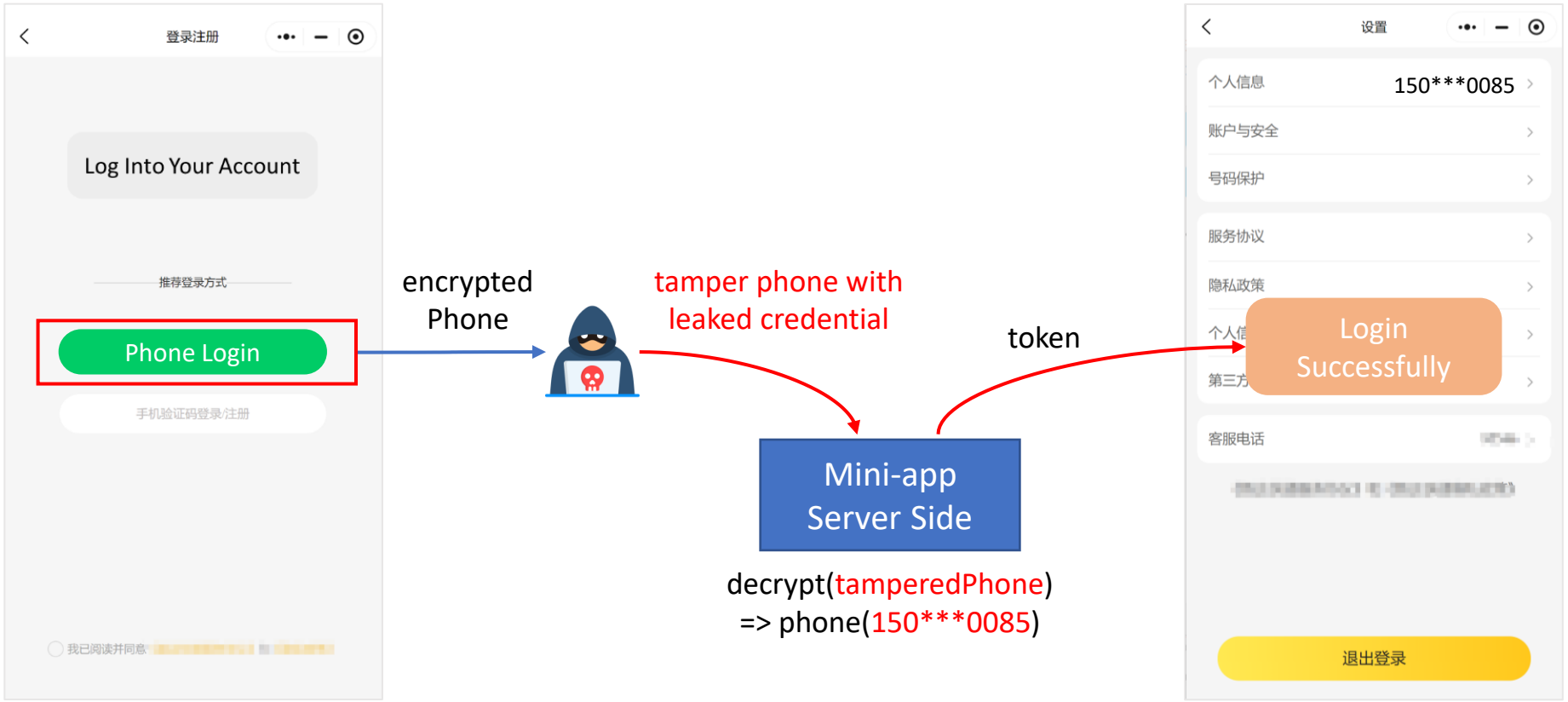
Security Hazards

- Account Hijacking



Security Hazards

- Account Hijacking



Summary

- Our work is the first to systematically study the app-in-app credential system and unveil its security implications
- We propose a novel approach, called KeyMagnet, to detect the credential leakage in mini-apps
- We have evaluated KeyMagnet with 413,775 mini-apps and have identified 84,491 credential leaks. We analyze the root causes of the prevalent leakage and propose corresponding mitigation strategies

Thank You!