The not-so-silent type

Vulnerabilities across keyboard apps reveal keystrokes to network eavesdroppers

By Jeffrey Knockel, Mona Wang, and Zoë Reichert

APRIL 23, 2024
RESEARCH REPORT #175

munk school OF GLOBAL AFFAIRS & PUBLIC POLICY
UNIVERSITY OF TORONTO
THE CITIZEN LAB

Copyright

About the Citizen Lab, Munk School of Global Affairs & Public Policy, University of Toronto

The Citizen Lab is an interdisciplinary laboratory based at the Munk School of Global Affairs & Public Policy, University of Toronto, focusing on research, development, and high-level strategic policy and legal engagement at the intersection of information and communication technologies, human rights, and global security.

We use a "mixed methods" approach to research that combines methods from political science, law, computer science, and area studies. Our research includes investigating digital espionage against civil society, documenting Internet filtering and other technologies and practices that impact freedom of expression online, analyzing privacy, security, and information controls of popular applications, and examining transparency and accountability mechanisms relevant to the relationship between corporations and state agencies regarding personal data and other surveillance activities.

Acknowledgements

We would like to thank Jedidiah Crandall, Jakub Dalek, Pellaeon Lin, and Sarah Scheffler for their guidance and review of this report. Research for this project was supervised by Ron Deibert.

Suggested Citation

Jeffrey Knockel, Mona Wang, and Zoë Reichert. "The not-so-silent type: Vulnerabilities across keyboard apps reveal keystrokes to network eavesdroppers," *Citizen Lab Report No. 175*, University of Toronto, April 2024. Available at: https://citizenlab.ca/2024/04/vulnerabilities-across-keyboard-apps-reveal-keystrokes-to-network-eavesdroppers/.

Contents

We urge users to install the latest updates to their keyboard apps and that they keep their mobile operating systems up to date. We also recommend that at-risk users consider switching from a cloud-based keyboard app to one that operates entirely on-device.

Key findings

- **We analyzed the security of cloud-based pinyin keyboard apps from nine vendors — Baidu, Honor, Huawei, iFlytek, OPPO, Samsung, Tencent, Vivo, and Xiaomi — and examined their transmission of users' keystrokes for vulnerabilities.**
- **Our analysis revealed critical vulnerabilities in keyboard apps from eight out of the nine vendors in which we could exploit that vulnerability to completely reveal the contents of users' keystrokes in transit. Most of the vulnerable apps can be exploited by an entirely passive network eavesdropper.**
- **Combining the vulnerabilities discovered in this and our previous report analyzing Sogou's keyboard apps, we estimate that up to one billion users are affected by these vulnerabilities. Given the scope of these vulnerabilities, the sensitivity of what users type on their devices, the ease with which these vulnerabilities may have been discovered, and that the Five Eyes have previously exploited similar vulnerabilities in Chinese apps for surveillance, it is possible that such users' keystrokes may have also been under mass surveillance.**
- **We reported these vulnerabilities to all nine vendors. Most vendors responded, took the issue seriously, and fixed the reported vulnerabilities, although some keyboard apps remain vulnerable.**
- **We conclude our report by summarizing our recommendations to various stakeholders to attempt to reduce future harm from apps which might feature similar vulnerabilities.**

1. Introduction

Typing logographic languages such as Chinese is more difficult than typing alphabetic languages, where each letter can be represented by one key. There is no way to fit the tens of thousands of Chinese characters that exist onto a single keyboard. Despite this obvious challenge, technologies have developed which make typing in Chinese possible. To enable the input of Chinese characters, a writer will generally use a keyboard app with an "Input Method Editor" (IME). IMEs offer a variety of approaches to inputting Chinese characters, including via handwriting, voice, and optical character recognition (OCR). One popular phonetic input method is Zhuyin, and shape or stroke-based input methods such as Cangjie or Wubi are commonly used as well. However, used by nearly 76% of mainland Chinese keyboard users, the most popular way of typing in Chinese is the pinyin method, which is based on the pinyin romanization of Chinese characters.

All of the keyboard apps we analyze in this report fall into the category of input method editors (IMEs) that offer pinyin input. These keyboard apps are particularly interesting because they have grown to accommodate the challenge of allowing users to type Chinese characters quickly and easily. While many keyboard apps operate locally, solely within a user's device, IME-based keyboard apps often have cloud features which enhance their functionality. Because of the complexities of predicting which characters a user may want to type next, especially in logographic languages like Chinese, IMEs often offer "cloud-based" prediction services which reach out over the network. Enabling "cloud-based" features in these apps means that longer strings of syllables that users type will be transmitted to servers elsewhere. As many have previously pointed out, "cloud-based" keyboards and input methods can function as vectors for surveillance and essentially behave as keyloggers. While the content of what users type is traveling from their device to the cloud, it is additionally vulnerable to network attackers if not properly secured. This report is **not** about how operators of cloud-based IMEs read users' keystrokes, which is a phenomenon that has already been extensively studied and documented. This report is primarily concerned with the issue of protecting this sensitive data from network eavesdroppers.

In this report, we analyze the security of cloud-based pinyin keyboard apps from nine vendors: Baidu, Honor, Huawei, iFlytek, OPPO, Samsung, Tencent, Vivo, and Xiaomi. We examined these apps' transmission of users' keystrokes for vulnerabilities. Our analysis revealed critical vulnerabilities in keyboard apps from eight out of the nine vendors — all but Huawei — in which we could exploit that vulnerability to completely reveal the contents of users' keystrokes in transit.

Between this report and our Sogou report, we estimate that close to one billion users are affected by this class of vulnerabilities. Sogou, Baidu, and iFlytek IMEs alone comprise over 95% of the market share for third-party IMEs in China, which are used by around a billion people. In addition to the users of third party keyboard apps, we found that the default keyboards on devices from three manufacturers (Honor, OPPO, and Xiaomi) were also vulnerable to our attacks. Devices from Samsung and Vivo also bundled a vulnerable keyboard, but it was not used by default. In 2023, Honor, OPPO, and Xiaomi alone comprised nearly 50% of the smartphone market in China.

Having the capability to read what users type on their devices is of interest to a number of actors — including government intelligence agencies that operate globally — because it may encompass exceptionally sensitive information about users and their contacts including financial information, login credentials such as usernames or passwords, and messages that are otherwise end-to-end encrypted. Given the known capabilities of state actors, and that Five Eyes agencies have previously exploited similar vulnerabilities in Chinese apps for the express purpose of mass surveillance, it is possible that we were not the first to discover these vulnerabilities and that they have previously been exploited on a mass scale for surveillance purposes.

We reported these issues to all eight of the vendors in whose keyboards we found vulnerabilities. Most vendors responded, took the issue seriously, and fixed the reported vulnerabilities, although some keyboard apps remain vulnerable. Users should keep their apps and operating systems up to date. We recommend that they consider switching from a cloud-based keyboard app to one that operates entirely on-device if they are concerned about these privacy issues.

The remainder of this report is structured as follows. In the "Related work" section, we outline previous security and privacy research that has been conducted on IME apps and past research which relates to issues of encryption in the Chinese app ecosystem. In "Methodology", we describe the reverse engineering tools and techniques we used to analyze the above apps. In the "Findings" section, we explain the vulnerabilities we discovered in each app and (where applicable) how we exploited these vulnerabilities. In "Coordinated disclosure", we discuss how we reported the vulnerabilities we found to the companies and their responses to our outreach. Finally, in "Discussion", we reflect on the impact of the vulnerabilities we discovered, how they came to be, and ways that we can avoid similar problems in the future. We provide recommendations to all stakeholders in this systemic privacy and security failure, including users, IME and keyboard developers, operating systems, mobile device manufacturers, app store operators, International standards bodies, and security researchers.

2. Related work

There has been much work analyzing East Asian apps for their security and privacy properties. As examples from outside of China, researchers studied LINE, a Japanese-developed app, and KakaoTalk, a South Korean-developed app, finding that they have faults in their end-to-end encryption implementations. When it comes to Chinese software, the Citizen Lab has previously revealed privacy and security issues in several Chinese web browsers, and identified vulnerabilities in the Zoom video conferencing platform and the MY2022 Olympics app. Unfortunately, even developers of extremely popular apps often overlook implementing proper security measures and protecting user privacy.

Some work has been concerned specifically with the privacy issues with cloud-based keyboard apps. As the technology powering keyboard apps became more popular and sophisticated, awareness of the potential security risks associated with these apps grew. Two main areas of concern have received the most attention from security researchers when it comes to cloud-based keyboard apps: whether user data is secure *in the cloud servers* and whether it is secure *in transit* as it moves from the user's device to a cloud server.

Some researchers have expressed concern over companies handling sensitive keystroke data and have made attempts to ameliorate the risk of the cloud server being able to record what you typed. In 2013, the Japanese government published concerns it had with privacy regarding the Baidu IME, particularly the cloud input function. Researchers have also been concerned with surveillance via other "cloud-based" IMEs, like iFlytek's voice input. While there has been a push to develop privacy-aware cloud-based IMEs that would keep user data secret, they are not widely used. While it is concerning what companies might do with user keystroke data, our research pertains to the security of user keystroke data before it even reaches cloud servers and who else other than the cloud operator may be able to read it.

Other research has studied the leakage of sensitive information when user keystroke data is in transit between a user's device to a remote cloud server. If not properly encrypted, data can be intercepted and collected by network eavesdroppers. In 2015 security researchers proposed and evaluated a system to identify keystroke leakages in IME traffic, revealing that at least one IME was transmitting sensitive data without encrypting it at all. Another investigation in the same year showed that the most popular IME, Sogou, was sending users' device identifiers in the clear. In our 2023 report we exposed Sogou falling short once more, finding that Sogou allowed network eavesdroppers to read what users were typing—as they typed—in any application. All of these discoveries point to developers of these applications overlooking the importance of transport security to protect user data from network attackers.

While previous work studying the security of keystroke network data in transit investigates single keyboard apps at a time, our report is the first to holistically evaluate the network security of the cloud-based keyboard app landscape in China.

3. Methodology

We analyzed the Android and, if present, the iOS and Windows versions of keyboard apps from the following keyboard app vendors: Tencent, Baidu, iFlytek, Samsung, Huawei, Xiaomi, OPPO, Vivo, and Honor. The first three — Tencent, Baidu, and iFlytek — are software developers of keyboard apps whereas the remaining six — Samsung, Huawei, Xiaomi, OPPO, Vivo, and Honor — are mobile device manufacturers who either developed their own keyboard apps or include one or more of the other three developers' keyboard apps preinstalled on their devices. We selected these nine vendors because we identified them as having integrated cloud recommendation functionality into their products and because they are popularly used. To procure the versions we analyzed, between August and November, 2023, we downloaded the latest versions of them from their product websites, the Apple App Store, or, in the case of the apps developed or bundled by mobile device manufacturers, by procuring a mobile device that has the app preinstalled on the ROM. In the case that we obtained the app as pre-installed on a mobile device, we ensured that the device's apps and operating system were fully updated before beginning analysis of its apps. The devices we obtained were intended for the mainland Chinese market, and, when device manufacturers had two editions of their device, a Chinese edition and a global edition, we analyzed the Chinese edition.

To better understand whether these vendors' keyboard apps securely implemented their cloud recommendation functionality, we analyzed them to determine whether they sufficiently encrypted users' typed keystrokes. To do so, we used both static and dynamic analysis methods. We used jadx to decompile and statically analyze Dalvik bytecode and IDA Pro to decompile and statically analyze native machine code. We used frida to dynamically analyze the Android and iOS versions and IDA Pro to dynamically analyze the Windows version. Finally, we used Wireshark and mitmproxy to perform network traffic capture and analysis.

To prepare for our dynamic analysis of each keyboard app, after installing it, we enabled the pinyin input if it was not already enabled. The keyboards we analyzed generally prompted users to enable cloud functionality after installation or on first use. In such cases, we answered such prompts in the affirmative or otherwise enabled cloud functionality through the mobile device's or app's settings.

In our analysis, we assume a fairly conservative threat model. For most of our attacks, we assume a *passive* network eavesdropper that monitors network packets that are sent from a user's keyboard app to a keyboard app's cloud server. In one of our attacks, specifically against apps using Tencent's Sogou API, we allow the adversary to be *active* in a limited way in that the adversary may additionally transmit network traffic to the cloud server but does not necessarily have to be a machine-in-the-middle (MITM) or spoof messages from the user in a layer 3 sense. In all of our attacks, the adversary also has access to a copy of the client software, but the server is a black box.

We note that, as neither Apple's nor Google's keyboard apps have a feature to transmit keystrokes to cloud servers for cloud-based recommendations, we did (and could) not analyze these keyboards for the security of this feature. However, we observed that none of the mobile devices that we analyzed included Google's keyboard, Gboard, preinstalled, either. This finding likely results from Google's exit from China reportedly due to the company's failure to comply with China's pervasive censorship requirements.

4. Findings

Among the nine vendors whose apps we analyzed, we found that there was only one vendor, Huawei, in whose apps we could not find any security issues regarding the transmission of users' keystrokes. For each of the remaining eight vendors, in at least one of their apps, we discovered a vulnerability in which keystrokes could be completely revealed by a passive network eavesdropper (see Table 1 for details).

The ease with which the keystrokes in these apps could be revealed varied. In one app, Samsung Keyboard, we found that the app performed no encryption whatsoever. Some apps appeared to internally use Sogou's cloud functionality and were vulnerable to an attack which we previously published. Most vulnerable apps failed to use asymmetric cryptography and mistakenly relied solely on home-rolled symmetric encryption to protect users' keystrokes.

The remainder of this section details further analysis of the apps we analyzed from each vendor and, when present, their vulnerabilities.

✘✘	working exploit created to decrypt transmitted keystrokes for both **active and passive** eavesdroppers
✘	working exploit created to decrypt transmitted keystrokes for an **active** eavesdropper
!	weaknesses present in cryptography implementation
✔	no known issues
N/A	product not offered or not present on device analyzed

Legend

Keyboard developer	Android				iOS	Windows
Tencent[†]	✘				N/A	✘
Baidu	!				!	✘✘
iFlytek	✘✘				✔	✔
	Pre-installed keyboard developer					
Device manufacturer	Own	Sogou	Baidu	iFlytek		
Samsung	✘✘	✔[*]	✘✘	N/A	N/A	N/A
Huawei	✔[*]	✔	N/A	N/A	N/A	N/A
Xiaomi	N/A	✘[*]	✘✘	✘✘	N/A	N/A
OPPO	N/A	✘	✘✘[*]	N/A	N/A	N/A
Vivo	✔[*]	✘	N/A	N/A	N/A	N/A
Honor	N/A	N/A	✘✘[*]	N/A	N/A	N/A

[*] Default keyboard app on our test device.

[†] Both QQ Pinyin and Sogou IME are developed by Tencent; in this report we analyzed QQ Pinyin and found the same issues as we had in Sogou IME.

Table 1: Summary of vulnerabilities discovered in popular keyboards and in keyboards pre-installed on popular phones.

4.1. Tencent

We have previously analyzed one Tencent keyboard app, Sogou, in a previous report. We were motivated by our previous findings analyzing Sogou to analyze another Tencent keyboard app, QQ Pinyin. We analyzed QQ Pinyin on Android and Windows. We found that the Android version (8.6.3) and Windows version (6.6.6304.400) of this software communicated to similar cloud servers as Sogou and contained the same vulnerabilities to those which we previously reported in Sogou IME (see Table 2 for details).

Platform	File/Package Name	Version analyzed	Secure?
Android	com.tencent.qqpinyin	8.6.3	✘
Windows	QQPinyin_Setup_6.6.6304.400.exe	6.6.6304.400	✘

Table 2: The versions of QQ Pinyin that we analyzed.

4.2. Baidu

We analyzed Baidu IME for Windows, Android, and iOS. We found that Baidu IME for Windows includes a vulnerability which allows network eavesdroppers to decrypt network transmissions. This means third parties can obtain sensitive personal information including what users have typed. We also found privacy and security weaknesses in the encryption used by the Android and iOS versions of Baidu IME (see Table 3 for details).

Platform	File/Package Name	Version analyzed	Secure?	Protocol
Windows	BaiduPinyinSetup_ 6.0.3.44.exe	6.0.3.44	✘✘	BAIDUv3.1
Android	com.baidu.input	11.7.19.9	!	BAIDUv4.0
iOS	com.baidu.inputMethod	11.7.20	!	BAIDUv4.0

Table 3: The versions of Baidu IME that we analyzed.

The Android version transmitted keystrokes information via UDP packets to `udpolimeok.baidu.com` and that the Windows and iOS versions transmitted keystrokes to `udpolimenew.baidu.com`. The two mobile versions that we analyzed, namely the Android and iOS versions, transmitted these keystrokes according to a stronger protocol, whose payload begins with the bytes 0x04 0x00. The Windows version transmitted these keystrokes according to a weaker protocol, whose UDP payload begins with the bytes 0x03 0x01. We henceforth refer to these protocols as the BAIDUv4.0 and BAIDUv3.1 protocols, respectively. In the remainder of this section we detail multiple weaknesses in the BAIDUv4.0 protocol

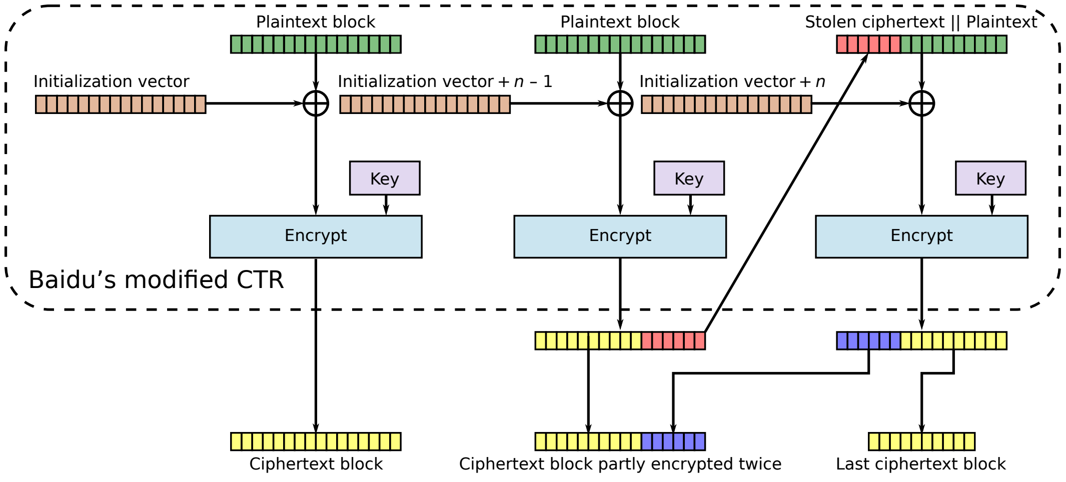


Figure 1: Illustration of BCTR mode encryption scheme used by Baidu IME on Android and iOS. Adapted from this figure.

used by the Android and iOS versions and explain how a network eavesdropper can decrypt the contents of keystrokes transmitted by the BAIDUv3.1 protocol.

4.2.1. Weaknesses in BAIDUv4.0 protocol

To encrypt keystroke information, the BAIDUv4.0 protocol uses elliptic-curve Diffie-Hellman and a pinned server public key (pk_s) to establish a shared secret key for use in a modified version of AES.

Upon opening the keyboard, before the first outgoing BAIDUv4.0 protocol message is sent, the application randomly generates a client Curve25519 public-private key pair, which we will call (pk_c, sk_c). Then, a Diffie-Hellman shared secret k is generated using sk_c and a pinned public key pk_s. To send a message with plaintext P, the application reuses the first 16 bytes of pk_c as the initialization vector (IV) for symmetric encryption, and k is used as the symmetric encryption key. The resulting symmetric encryption of P is then sent along with pk_c to the server. The server can then obtain the same Diffie-Hellman shared secret k from pk_c and sk_s, the private key corresponding to pk_s, to decrypt the ciphertext.

The BAIDUv4.0 protocol symmetrically encrypts data using a modified version of AES, which symbols in the code indicate Baidu has called *AESv3*. Compared to ordinary AES, AESv3 has a built-in cipher mode and padding. AESv3's built-in cipher mode mixes bytes differently and uses a modified counter (CTR) mode which we call Baidu CTR (BCTR) mode, illustrated in Figure 1.

Generally speaking, any CTR cipher mode involves combining an initialization vector v with the value i of some counter, whose combination we shall notate as $v + i$. Most commonly, the counter value used for block i is simply i, i.e., it begins at zero and increments for each subsequent block, and AESv3's implementation follows this convention. There is no standard way to compute $v + i$ in CTR mode, but the way that BCTR combines v and i is by adding i to the left-most 32-bits of v, interpreting this portion of v and i in little-endian byte order. If the sum overflows, then no carrying is performed on bytes to the right of this 32-bit value. The implementation details we have thus far described do not significantly deviate from a typical CTR implementation. However, where BCTR mode differs from ordinary CTR mode is in how the value $v + i$ is used during encryption. In ordinary CTR mode, to encrypt block i with key k, you would compute

$$\text{plain}_i \text{ XOR encrypt}(v + i, k).$$

In BCTR mode, to encrypt block i, you compute

$$\text{encrypt}(\text{plain}_i \text{ XOR } (v + i), k).$$

As we will see later, this deviation will have implications for the security of the algorithm.

While ordinarily CTR mode does not require the final block length to be a multiple of the cipher's block size (in the case of AES, 16 bytes), due to Baidu's modifications, BCTR mode no longer automatically possesses this property but rather achieves it by employing ciphertext stealing. If the final block length n is less than 16, AESv3's implementation encrypts the final 16 byte block by taking the last (16 - n) bytes of the penultimate ciphertext block and prepending them to the n bytes of the ultimate plaintext block. The encryption of the resultant block fills the last (16 - n) bytes of the penultimate ciphertext block and the n bytes of the final ciphertext block. Note, however, that this practice only works when the plaintext consists of at least two blocks. Therefore, if there exists only one plaintext block, then AESv3 right-zero-pads that block to be 16 bytes.

Privacy issues with key and IV re-use. Since the IV and key are both directly derived from the client key pair, the IV and key are reused until the application generates a new key pair. This only happens when the application restarts, such as when the user restarts the mobile device, the user switches to a different keyboard and back, or the keyboard app is evicted from memory. From our testing, we have

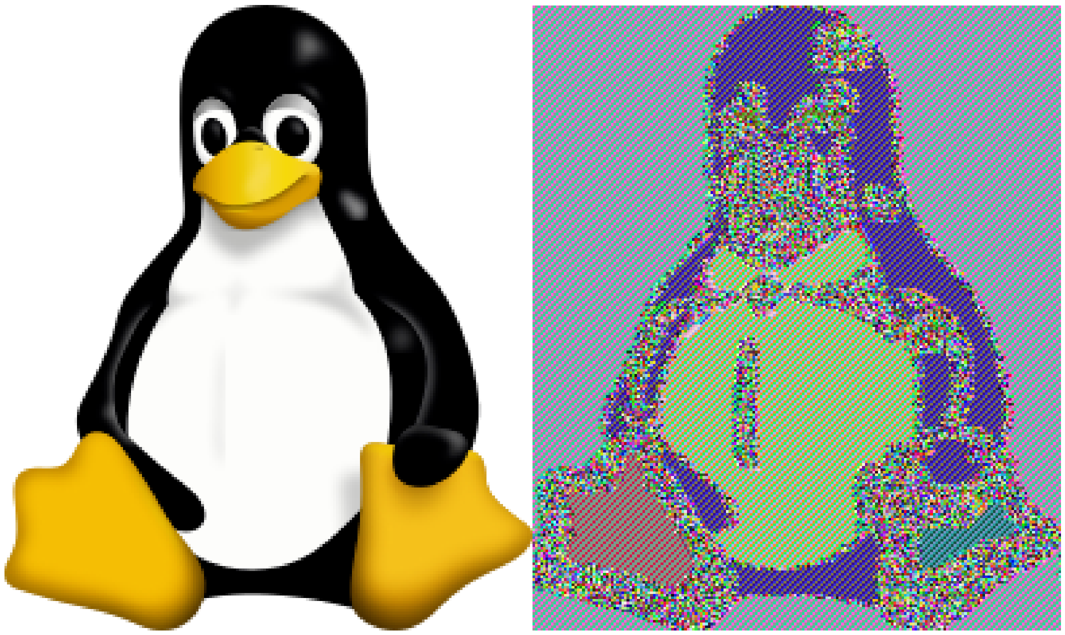


Figure 2: When a bitmap image (left) is encrypted in ECB mode, patterns in the image are still visible in the ciphertext (right). Adapted from these figures.

observed the same key and IV in use for over 24 hours. There are various issues that arise from key and IV reuse.

Re-using the same IV and key means that the same inputs will encrypt to the same encrypted ciphertext. Additionally, due to the way the block cipher is constructed, if blocks in the same positions of the plaintexts are the same, they will encrypt to the same ciphertext blocks. As an example, if the second block of two plaintexts are the same, the second block of the corresponding ciphertexts will be the same.

Weakness in cipher mode. The electronic codebook (ECB) cipher mode is notorious for having the undesirable property that equivalent plaintext blocks encrypt to equivalent ciphertext blocks, allowing patterns in the plaintext to be revealed in the ciphertext (see Figure 2 for an illustration).

While BCTR mode used by Baidu does not as flagrantly reveal patterns to the same extent as ECB mode, there do exist circumstances in which patterns in the plaintext can still be revealed in the ciphertext. Specifically, there exist circumstances in which there exists a counter-like pattern in the plaintext which can be revealed by the ciphertext (see Figure 3 for an example). These circumstances are possible due to the fact that (IV + *i*) is XORed with each plaintext block *i* and then encrypted, unlike ordinary CTR mode which encrypts (IV + *i*) and XORs it with the plaintext.

Block	Plaintext																Ciphertext															
0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	e2	d4	00	1c	c6	5d	80	33	0c	b9	48	7d	d5	27	72	7a
1	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	e2	d4	00	1c	c6	5d	80	33	0c	b9	48	7d	d5	27	72	7a

Figure 3: When encrypted with the randomly generated key `<96 66 08 d1 6f 80 82 86 a7 b7 da 43 96 ee d1 a2>` and IV `<48 5b 54 92 0c 80 a6 20 29 6f 95 e5 c5 6a 3d e2>` using Baidu's modified CTR mode, the above plaintext blocks in positions 0 and 1 encrypt to the same ciphertext.

Thus, when using BCTR mode, if the plaintext exhibits similar counting patterns as (IV + *i*), then for multiple blocks the value ((IV + *i*) XOR plaintext block *i*) may be equivalent and thus encrypt to an equivalent ciphertext.

More generally, BCTR mode fails to provide the cryptographic property of diffusion. Specifically, if an algorithm provides diffusion, then, when we change a single bit of the plaintext, we expect half of the bits of the ciphertext to change. However, the example in Figure 3 illustrates a case where changing a single bit of the plaintext caused zero bits of the ciphertext to change, a clear violation of the expectations of this property. The property of diffusion is vital in secure cryptographic algorithms so that patterns in the plaintext are not visible as patterns in the ciphertext.

Other privacy and security weaknesses. There are other weaknesses in the custom encryption protocol designed by Baidu IME that are not consistent with the expected standards for a modern encryption protocol used by hundreds of millions of devices.

Forward secrecy issues with static Diffie-Hellman. The use of a pinned static server key means that the cipher is not forward secret, a property of other modern network encryption ciphers like TLS. If the server key is ever revealed, any past message where the shared secret was generated with that key can be successfully decrypted.

Lack of message integrity. There are no cryptographically secure message integrity checks, which means that a network attacker may freely modify the ciphertext. There is a CRC32 checksum calculated and included with the plaintext data, but a CRC32 checksum does not provide cryptographic integrity, as it is easy to generate CRC32 checksum collisions. Therefore, modifying the ciphertext may be possible. In combination with the issue concerning key and IV reuse, this protocol may be vulnerable to a swapped block attack.

```python
def derive_fixed_key():
    key = []
    x = 0
    for i in range(16):
        key.append((~i ^ ((i + 11) * (x >> (i & 3)))) & 0xff)
        x += 1937
    return bytes(key)
```

Figure 4: Python code equivalent to the code that the BAIDUv3.1 protocol uses to derive its fixed key. The function takes no input and derives the same key on every invocation.

4.2.2. Vulnerability in BAIDUv3.1 protocol

The BAIDUv3.1 protocol is weaker than the BAIDUv4.0 protocol and contains a critical vulnerability that allows an eavesdropper to decrypt any messages encrypted with it. The protocol in the versions of Baidu's keyboard apps that we analyzed encrypts keystrokes using a modified version of AES which we call *AESv2*, as we believe it to be the predecessor cipher to Baidu's AESv3. When a keyboard app uses the BAIDUv3.1 protocol with the AESv2 cipher, we say that it uses the *BAIDUv3.1+AESv2* scheme. Normally, AES when used with a 128-bit key performs 10 rounds of encryption on each block. However, we found that AESv2 uses only 9 rounds but is otherwise equivalent to AES encryption with a 128-bit key.

The BAIDUv3.1+AESv2 scheme encrypts keystrokes using AESv2 in the following manner. First, a key is derived according to a fixed function (see Figure 4). Note that the function takes no input nor references any external state and thus always generates the same static key

$$k_f = \text{<}\texttt{ff 9e d5 48 07 5a 10 e4 ef 06 c7 2e a7 a2 f2 36}\text{>}.$$

To encrypt a protobuf-serialized message, the BAIDUv3.1 protocol first snappy-compresses it, forming a compressed buffer. The 32-bit, little-endian length of this compressed message is then prepended to the compressed buffer, forming the plaintext. A randomly generated 128-bit key k_m is used to encrypt the plaintext using AESv2 in ECB mode. The resulting ciphertext is stored in bytes 44 until the end of the final UDP payload. Key k_f is used to encrypt k_m using AESv2 in ECB mode. The resulting ciphertext is stored in bytes 28 until 44 of the final UDP payload.

```
[...]
2 {
  1: "nihaocanyoureadthis"
  5: 3407918
}
3 {
  1: 107
  2: 10
  5: 1
}
4 {
  1: "1133d4c64afbf1feda85d3c497dd6164|0"
  2: "wn1||0"
  3: "6.0.3.44"
  4: "notepad.exe"
}
[...]
```

Figure 5: Excerpt of decrypted information, including what we had typed ("nihao-canyoureadthis") and the app into which it was typed ("notepad.exe").

We found that these encrypted protobuf serializations include our typed keystrokes as well as the name of the application into which we were typing them (see Figure 5).

A vulnerability exists in the BAIDUv3.1+AESv2 scheme that allows a network eavesdropper to decrypt the contents of these messages. Since AES is a symmetric encryption algorithm, the same key used to encrypt a message can also be used to decrypt it. Since k_f is fixed, any network eavesdropper with knowledge of k_f, such as from performing the same analysis of the app as we performed, can decrypt k_m and thus can decrypt the plaintext contents of each message encrypted in the manner described above. As we found that users' keystrokes and the names of the applications they were using were sent in these messages, a network eavesdropper who is eavesdropping on a user's network traffic can observe what that user is typing and into which application they are typing it by taking advantage of this vulnerability.

4.3. iFlytek

We analyzed iFlytek (also called xùnfēi from the pinyin of 讯飞) IME on Android, iOS, and Windows. We found that iFlytek IME for Android includes a vulnerability which allows network eavesdroppers to recover the plaintext of insufficiently encrypted network transmissions, revealing sensitive information including what users have typed (see Table 4 for details).

Platform	File/Package Name	Version analyzed	Secure?
Android	com.iflytek.inputmethod	12.1.10	✗✗
iOS	com.iflytek.inputime	12.1.3338	✓
Windows	iFlyIME_Setup_3.0.1734.exe	3.0.1734	✓

Table 4: The versions of Xunfei IME analyzed.

The Android version of iFlytek IME encrypts the payload of each HTTP request sent to `pinyin.voicecloud.cn` with the following algorithm. Let *s* be the current time in seconds since the Unix epoch at the time of the request. For each request, an 8-byte encryption key is then derived by first performing the following computation:

$$x = (s \% \text{ 0x5F5E100}) \text{ ^ } \text{0x1001111}$$

The 8-byte key *k* is then derived from *x* as the lowest 8 ASCII-encoded digits of *x*, left-padded with leading zeroes if necessary, in big-endian order. In Python, the above can be summarized by the following expression:

$$k = \text{b'\%08u'} \% ((s \% \text{ 0x5F5E100}) \text{ ^ } \text{0x1001111})$$

The payload of the request is then padded with PKCS#7 padding and then encrypted with DES using key *k* in ECB mode. The value *s* is transmitted in the HTTP request in the clear as a GET parameter named “time”.

Since DES is a symmetric encryption algorithm, the same key used to encrypt a message can also be used to decrypt it. Since *k* can be easily derived from *s* and since *s* is transmitted in the clear in every HTTP request encrypted by *k*, any network eavesdropper can easily decrypt the contents of each HTTP request encrypted in the manner described above. (Since *s* is simply the time in single second resolution, it also stands to reason that a network eavesdropper would have general knowledge of *s* in any case.)

```
1: 0
2: 0
3: 49
4: "xxxxx"
5: 0
7 {
  1: "app_id"
  2: "100IME"
}
7 {
  1: "uid"
  2: "230817031752396418"
}
7 {
  1: "cli_ver"
  2: "12.1.14983"
}
7 {
  1: "net_type"
  2: "wifi"
}
7 {
  1: "OS"
  2: "android"
}
8: 8
```

Figure 6: Decrypted information revealing what we had typed (“xxxxx”).

We found that users’ keystrokes were transmitted in a protobuf serialization and encrypted in this manner (see Figure 6). Therefore, a network eavesdropper who is eavesdropping on a user’s network traffic can observe what that user is typing by taking advantage of this vulnerability.

Finally, the DES encryption algorithm is an older encryption algorithm with known weaknesses, and the ECB block cipher mode is a simplistic and problematic cipher mode. The use of each of these technologies is problematic in itself and opens the Android version of iFlytek IME’s communications to additional attacks.

4.4. Samsung

We analyzed Samsung Keyboard on Android as well as the versions of Sogou IME and Baidu IME that Samsung bundled with our test device, an SM-T220 tablet running ROM version T220CHN4CWF4. We found that Samsung Keyboard for Android and Samsung's bundled version Baidu IME includes a vulnerability that allows network eavesdroppers to recover the plaintext of insufficiently encrypted network transmissions, revealing sensitive information including what users have typed (see Table 5 for details).

Application name	Package name	Version analyzed	Secure?
Samsung Keyboard	com.samsung.android.honeyboard	5.6.10.26	✗✗
百度输入法 (Baidu IME)	com.baidu.input	8.5.20.4	✗✗
搜狗输入法三星版 (Sogou IME Samsung Version)	com.sohu.inputmethod.sogou.samsung	10.32.38.202307281642	✓

Table 5: The keyboards analyzed on the Samsung OneUI 5.1 platform.

4.4.1. Samsung Keyboard (com.samsung.android.honeyboard)

We found that when using Samsung Keyboard on the Chinese edition of a Samsung device and when Pinyin is chosen as Samsung Keyboard's input language, Samsung Keyboard transmits keystroke data to the following URL in the clear via HTTP POST:

```
http://shouji.sogou.com/web_ime/mobile_pb.php?durtot=33
9&h=8f2bc112-bbec-3f96-86ca-652e98316ad8&r=android_oe
m_samsung_open&v=8.13.10038.413173&s=&e=&i=&fc=0&base=
dW5rbm93biswLjArMC4w&ext_ver=0
```

The keystroke data is contained in the request's HTTP payload in a protobuf serialization (see Figure 7).

```
1 {
  1: "8f2bc112-bbec-3f96-86ca-652e98316ad8"
  2: "android_oem_samsung_open"
  3: "8.13.10038.413173"
  4: "999"
  5: 1
  7: 2
}
2 {
  1: "\351\000"
  2: "\372\213"
}
4: "com.tencent.mobileqq"
7: "nihaocanyoureadthis"
16: 10
17 {
  3 {
    1: 1
    2: 5
  }
  5: 1
  9: 1
}
18: ""
19 {
  1: "0"
  4: "339"
}
```

Figure 7: Protobuf message transmitted after typing "nihaocanyoureadthis".

The device on which we were testing was fully updated on the date of testing (October 7, 2023) in that it had all OS updates applied and had all updates from the Samsung Galaxy Store applied.

Since Samsung Keyboard transmits keystroke data via plain, unencrypted HTTP and since there is no encryption applied at any other layer, a network eavesdropper who is monitoring a Samsung Keyboard user's network traffic can easily observe that user's keystrokes if that user is using the Chinese edition of the ROM with the Pinyin input language selected.

When using the global edition of the ROM or when using a non-Pinyin input language, we did not observe the Samsung keyboard communicating with cloud servers.

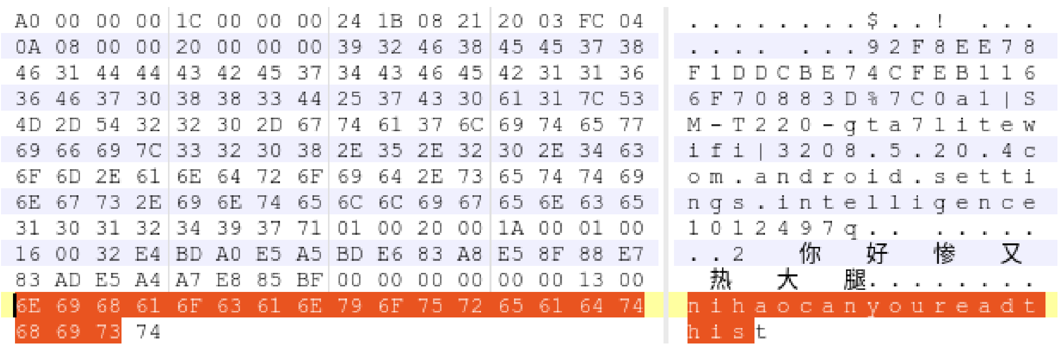
4.4.2. 百度输入法 (Baidu IME)

We found that the version of Baidu IME bundled with our Samsung test device transmitted keystroke information via UDP packets to `udpolimenew.baidu.com`. This version of Baidu IME used the BAIDUv3.1 protocol that we describe in the Baidu section earlier but with a different cipher and compression algorithm as indicated in each transmission's header. In the remainder of this section we explain how a network eavesdropper can, just like with AESv2, decrypt the contents of messages encrypted using a scheme we call *BAIDUv3.1+AESv1* (see Table 6).

Protocol	Scheme	Cipher	Mode	Cipher versus AES
BAIDUv3.1	BAIDUv3.1+AESv1	AESv1	ECB	Additional permutations
	BAIDUv3.1+AESv2	AESv2	ECB	Missing round
BAIDUv4.0	BAIDUv4.0+AESv3	AESv3	BCTR	Uses home-rolled cipher mode

Table 6: Summary of ciphers used across different Baidu protocols.

Samsung's bundled version of Baidu IME encrypts keystrokes using a modified version of AES which we name *AESv1*, as we believe it to be the predecessor to Baidu's AESv2. When encrypting, AESv1's key expansion is like that of standard AES, except, on each but the first subkey, the order of the subkey's bytes are additionally permuted. Furthermore, on the encryption of each block, the bytes of the block are additionally permuted in two locations, once near the beginning of the block's encryption immediately after the block has been XOR'd by the first subkey and again near the end of the block's encryption immediately before S-box substitution. Aside from complicating our analysis, we are not aware of these modifications altering the security properties of AES, and we have developed an implementation of this algorithm to both encrypt and decrypt messages given a plaintext or ciphertext and a key.



```
A0 00 00 00 1C 00 00 00 24 1B 08 21 20 03 FC 04   . . . . . . . . $ . . !   . . .
0A 08 00 00 20 00 00 00 39 32 46 38 45 45 37 38   . . . .   . . . 9 2 F 8 E E 7 8
46 31 44 44 43 42 45 37 34 43 46 45 42 31 31 36   F 1 D D C B E 7 4 C F E B 1 1 6
36 46 37 30 38 38 33 44 25 37 43 30 61 31 7C 53   6 F 7 0 8 8 3 D % 7 C 0 a 1 | S
4D 2D 54 32 32 30 2D 67 74 61 37 6C 69 74 65 77   M - T 2 2 0 - g t a 7 l i t e w
69 66 69 7C 33 32 30 38 2E 35 2E 32 30 2E 34 63   i f i | 3 2 0 8 . 5 . 2 0 . 4 c
6F 6D 2E 61 6E 64 72 6F 69 64 2E 73 65 74 74 69   o m . a n d r o i d . s e t t i
6E 67 73 2E 69 6E 74 65 6C 6C 69 67 65 6E 63 65   n g s . i n t e l l i g e n c e
31 30 31 32 34 39 37 71 01 00 20 00 1A 00 01 00   1 0 1 2 4 9 7 q . .   . . . . .
16 00 32 E4 BD A0 E5 A5 BD E6 83 A8 E5 8F 88 E7   . . 2   你    好    惨    又
83 AD E5 A4 A7 E8 85 BF 00 00 00 00 00 00 13 00     热    大    腿. . . . . . . .
6E 69 68 61 6F 63 61 6E 79 6F 75 72 65 61 64 74   n i h a o c a n y o u r e a d t
68 69 73 74                                       h i s t
```

```
0: [800,
    1276,
    10,
    0,
    "92F8EE78F1DDCBE74CFEB1166F70883D%7C0",
    "a1|SM-T220-gta7litewifi|320",
    "8.5.20.4",
    "com.android.settings.intelligence",
    "1012497q",
    "",
    "2你好惨又热大腿",
    ""],
1: [0, "", "nihaocanyoureadthis"]
```

Figure 8: The decrypted and decompressed payload, revealing what we had typed ("nihaocanyoureadthis", highlighted) and the app into which it was typed ("com.android.settings.intelligence"); on top is a hex dump of, when decrypted and decompressed, the resulting proprietary binary blob, and below it is our understanding of how to parse it.

Samsung's bundled version of Baidu IME encrypts keystrokes by applying AESv1 in electronic codebook (ECB) mode in the following manner. First, the app uses the fixed 128-bit key,

$$k_f = \texttt{<ff 9e d5 48 07 5a 10 e4 ef 06 c7 2e a7 a2 f2 36>},$$

to encrypt another, generated, key, k_m. The fixed key k_f is the same key the BAIDUv3.1 protocol uses for AESv2 (see Figure 4). The encryption of k_m is stored in bytes 64 until 80 of each UDP packet's payload. The key k_m is then used to encrypt the remainder of a zlib-compressed message payload, which is stored at byte 80 until the end of the UDP payload. We found that the encrypted payload included, in a binary container format which we did not recognize, our typed keystrokes as well as the name of the application into which we were typing them (see Figure 8).

A vulnerability exists in the BAIDUv3.1+AESv1 scheme that allows a network eavesdropper to decrypt the contents of these messages. Since AES, including AESv1, is a symmetric encryption algorithm, the same key used to encrypt a message can also be used to decrypt it. Since k_f is hard-coded, any network eavesdropper with knowledge of k_f can decrypt k_m and thus decrypt the plaintext contents of each message encrypted in the manner described above. As we found that users' keystrokes and the names of the applications they were using were sent in these messages, a network eavesdropper who is eavesdropping on a user's network traffic can observe what that user is typing and into which application they are typing it by taking advantage of this vulnerability.

Additionally, in the version of Baidu Input Method distributed by Samsung, we found that key k_m was not securely generated using a secure pseudorandom number generator (secure PRNG). Instead, it was seeded using a custom-designed PRNG that we believe to have poor security properties, and, instead of using a high entropy seed, the PRNG generating k_m was seeded using the message plaintext. However, even without these weaknesses in the generation of k_m, the protocol is already completely insecure to network eavesdroppers as described in the previous paragraphs.

4.5. Huawei

We analyzed the keyboards preinstalled on our Huawei Mate 50 Pro test device. We found no vulnerabilities in the manner of transmission of users' keystrokes in the versions of Huawei's keyboard apps that we analyzed (see Table 7 for details). Specifically, Huawei used TLS to encrypt keystrokes in each version that we analyzed.

Application name	Package Name	Version analyzed	Secure?
搜狗输入法 (Sogou IME)	com.sohu.inputmethod.sogou	11.31	✓
小艺输入法 (Celia IME)	com.huawei.ohos.inputmethod	1.0.19.333	✓

Table 7: The versions of the Huawei keyboard apps analyzed on HarmonyOS 4.0.0.

4.6. Xiaomi

We analyzed the keyboards preinstalled on our Xiaomi Mi 11 test device. We found that they all include vulnerabilities that allow network eavesdroppers to decrypt network transmissions from the keyboards (see Table 8 for details). This means that network eavesdroppers can obtain sensitive personal information, including what users have typed.

Application name	Package Name	Version analyzed	Secure?
百度输入法小米版 (Baidu IME Xiaomi Version)	com.baidu.input_mi	10.6.120.480	✘✘
搜狗输入法小米版 (Sogou IME Xiaomi Version)	com.sohu.inputmethod. sogou.xiaomi	10.32.21. 202210221903	✘
讯飞输入法小米版 (iFlytek IME Xiaomi Version)	com.iflytek. inputmethod.miui	8.1.8014	✘✘

Table 8: The versions of the Xiaomi keyboard apps analyzed on MIUI 14.03.31.

In this section we detail vulnerabilities in three different keyboard apps included with MIUI 14.0.31 in which users' keystrokes can be, if necessary, decrypted, and read by network eavesdroppers.

4.6.1. 百度输入法小米版 (Baidu IME Xiaomi Version)

We found that Xiaomi's Baidu-based keyboard app encrypts keystrokes using the BAIDUv3.1+AESv2 scheme which we detailed previously. When the app's messages are decrypted and deserialized, we found that they include our typed keystrokes as well as the name of the application into which we were typing them (see Figure 9).

Like we explained previously a vulnerability exists in the BAIDUv3.1+AESv2 scheme that allows a network eavesdropper to decrypt the contents of these messages. As we found that users' keystrokes and the names of the applications they were using were sent in these messages, a network eavesdropper who is eavesdropping

```
[...]
2 {
  1: "nihaonihaoqqwerty"
}
3 {
  1: 53
  2: 10
  3: 1080
  4: 2166
  5: 5
}
4 {
  1: "DC0F75E6809F0FAAB46EDE2F2D6302ED%7CVAPBN4NOH"
  2: "p-a1-3-66|2211133C|720"
  3: "10.6.120.480"
  4: "com.miui.notes"
  5: "1000228c"
  6: "\346\242\205\345\267\236"
}
[...]
```

Figure 9: Excerpt of decrypted information, including what we had typed ("nihaonihaoqqwerty") and the application into which it was typed ("com.miui.notes").

on a user's network traffic can observe what that user is typing and into which application they are typing it by taking advantage of this vulnerability.

4.6.2. 搜狗输入法小米版 (Sogou IME Xiaomi Version)

The Sogou-based keyboard app is subject to a vulnerability which we have already publicly disclosed in Sogou IME (搜狗输入法) in which a network eavesdropper can decrypt and recover users' transmitted keystrokes. Please see the corresponding details in this report for full details. Tencent responded by securing Sogou IME transmissions using TLS, but we found that Xiaomi's Sogou-based keyboard had not been fixed.

4.6.3. 讯飞输入法小米版 (iFlytek IME Xiaomi Version)

Similar to iFlytek's own IME for Android, we found that Xiaomi's iFlytek keyboard app used the same faulty encryption. We found that users' keystrokes were sent to `pinyin.voicecloud.cn` and encrypted in this manner.

```
{"p":{"m":53,"f":0,"l":0},"i":"nihaoniba"}
```

Figure 10: Excerpt of decrypted information, including what we had typed (“nihaoniba”).

Therefore, a network eavesdropper who is eavesdropping on a user’s network traffic can observe what that user is typing by taking advantage of this vulnerability (see Figure 10).

4.7. OPPO

We analyzed the keyboard apps preinstalled on our OPPO OnePlus Ace test device. We found that they all include vulnerabilities that allow network eavesdroppers to decrypt network transmissions from the keyboards (see Table 9 for details). This means that network eavesdroppers can obtain sensitive personal information, including what users have typed.

Application name	Package Name	Version analyzed	Secure?
百度输入法定制版 (Baidu IME Custom Version)	com.baidu.input_oppo	8.5.30.503	✗✗
搜狗输入法定制版 (Sogou IME Custom Version)	com.sohu. inputmethod.sogouoem	8.32.0322. 2305171502	✗

Table 9: The versions of the OPPO keyboard apps analyzed on ColorOS 13.1.

In this section we detail vulnerabilities in two different keyboard apps included with MIUI 14.0.31 in which users’ keystrokes can be, if necessary, decrypted, and read by network eavesdroppers.

4.7.1. 百度输入法定制版 (Baidu IME Custom Version)

We found that OPPO’s Baidu-based keyboard app encrypts keystrokes using the BAIDUv3.1+AESv2 scheme which we detailed previously. When the app’s messages are decrypted and deserialized, we found that they include our typed keystrokes as well as the name of the application into which we were typing them (see Figure 11).

```
[...]
2 {
  1: "nihaonihao"
}
3 {
  1: 28
  2: 10
  3: 1240
  4: 2662
  5: 5
}
4 {
  1: "47148455BDAEBA8A253ACBCC1CA40B1B%7CV7JTLNPID"
  2: "p-a1-5-105|PHK110|720"
  3: "8.5.30.503"
  4: "com.android.mms"
  5: "1021078a"
}
[...]
```

Figure 11: Excerpt of decrypted information, including what we had typed ("nihaonihao") and the application into which it was typed ("com.android.mms").

Like we explained previously a vulnerability exists in the BAIDUv3.1+AESv2 scheme that allows a network eavesdropper to decrypt the contents of these messages. As we found that users' keystrokes and the names of the applications they were using were sent in these messages, a network eavesdropper who is eavesdropping on a user's network traffic can observe what that user is typing and into which application they are typing it by taking advantage of this vulnerability.

4.7.2. 搜狗输入法定制版 (Sogou IME Custom Version)

The Sogou-based keyboard app is subject to a vulnerability which we have already publicly disclosed in Sogou IME (搜狗输入法) in which a network eavesdropper can decrypt and recover users' transmitted keystrokes. Please see the corresponding details in this report for full details. Tencent responded by securing Sogou IME transmissions using TLS, but we found that OPPO's Sogou-based keyboard had not been fixed.

4.8. Vivo

We analyzed the keyboard apps preinstalled on our Vivo Y78+ test device. We found that the Sogou-based one includes vulnerabilities that allow network eavesdroppers to decrypt network transmissions from the keyboards (see Table 10 for details). This means that network eavesdroppers can obtain sensitive personal information, including what users have typed.

Keyboard name	Package Name	Version analyzed	Secure?
搜狗输入法定制版 (Sogou IME Custom Version)	com.sohu.inputmethod.sogou.vivo	10.32.13023.2305191843	✘
Jovi输入法 (Jovi IME)	com.vivo.ai.ime	2.6.1.2305231	✔

Table 10: The versions of the Vivo keyboard apps analyzed on origin OS 3.

The Sogou-based keyboard app is subject to a vulnerability which we have already publicly disclosed in Sogou IME (搜狗输入法) in which a network eavesdropper can decrypt and recover users' transmitted keystrokes. Please see the corresponding details in this report for full details. Tencent responded by securing Sogou IME transmissions using TLS, but we found that Vivo's Sogou-based keyboard had not been fixed.

4.9. Honor

We analyzed the keyboard apps preinstalled on our Honor Play7T test device. We found that the Baidu-based one includes vulnerabilities that allow network eavesdroppers to decrypt network transmissions from the keyboards (see Table 11 for details). This means that network eavesdroppers can obtain sensitive personal information, including what users have typed.

```
[...]
2 {
  1: "nihaonihaonihaoq"
  5: 6422639
}
3 {
  1: 91
  2: 10
  3: 720
  4: 1552
  5: 5
}
4 {
  1: "A49AD3D3789A136975C2B28201753F03%7C0"
  2: "p-a1-5-115|RKY-AN10|720"
  3: "8.2.501.1"
  4: "com.hihonor.mms"
  5: "1023233d"
  7: "A00-TWGTFEV5OFZ7WZ2AFN5TCDE4BPNO7XRZ-BVEZBI4D"
}
[...]
```

Figure 12: Excerpt of decrypted information, including what we had typed ("nihaonihaonihaoq") and the application into which it was typed ("com.hihonor.mms").

Application name	Package Name	Version analyzed	Secure?
百度输入法荣耀版 (Baidu IME Honor Version)	com.baidu.input_hihonor	8.2.501.1	✘✘

Table 11: The versions of the Honor keyboard apps analyzed on Magic UI 6.1.0.

We found that Honor's Baidu-based keyboard app encrypts keystrokes using the BAIDUv3.1+AESv2 scheme which we detailed previously. When the app's messages are decrypted and deserialized, we found that they include our typed keystrokes as well as the name of the application into which we were typing them (see Figure 12).

Like we explained previously a vulnerability exists in the BAIDUv3.1+AESv2 scheme that allows a network eavesdropper to decrypt the contents of these messages. As we found that users' keystrokes and the names of the applications they were using were sent in these messages, a network eavesdropper who is eavesdropping on a user's network traffic can observe what that user is typing and into which application they are typing it by taking advantage of this vulnerability.

As of April 1, 2024, "Baidu IME Honor Version", the default IME on the Honor device we tested, is still vulnerable to passive decryption. We also discovered that on our Play7T device, there was no way to update "Baidu IME Honor Version" through the device's app store. In responding to our disclosures, Honor asked us to disclose to Baidu and that it was Baidu's responsibility to patch this issue.

5. Other affected keyboard apps

Given our limited resources to analyze apps, we were not able to analyze every cloud-based keyboard app available. Nevertheless, given that these vulnerabilities appeared to affect APIs that were used by multiple apps, we wanted to approximate the total number of apps affected by these vulnerabilities.

We began by searching VirusTotal, a database of software and other files that have been uploaded for automated virus scanning, for Android apps which reference the string "get.sogou.com", the API endpoint used by Sogou IME, as these apps may require additional investigation to determine whether they are vulnerable. Excluding apps that we analyzed above, this search yielded the following apps:

- com.sohu.sohuvideo
- com.tencent.docs
- com.sogou.reader.free
- com.sohu.inputmethod.sogou.samsung
- com.sogou.text
- com.sogou.novel
- com.sogo.appmall
- com.blank_app
- com.sohu.inputmethod.sogou.nubia
- com.sogou.androidtool
- com.sohu.inputmethod.sogou.meizu
- com.sohu.inputmethod.sogou.zte
- sogou.mobile.explorer.hmct
- sogou.mobile.explorer
- com.sogou.translatorpen

- com.sec.android.inputmethod.beta
- com.sohu.inputmethod.sogou.meitu
- com.sec.android.inputmethod
- sogou.mobile.explorer.online
- com.sohu.sohuvideo.meizu
- com.sohu.inputmethod.sogou.oem
- com.sogou.map.android.maps
- sogou.llq.online
- com.sohu.inputmethod.sogou.coolpad
- com.sohu.inputmethod.sogou.chuizi
- com.sogou.toptennews
- com.sogou.recmaster
- com.meizu.flyme.input

We have not analyzed these apps and thus cannot conclude that they are necessarily vulnerable, or even keyboard apps, but we provide this list to help reveal the possible scope of the vulnerabilities that we discovered. When we disclosed this list to Tencent, Tencent requested an additional three months to fix the vulnerabilities before we publicly disclosed this list, suggesting credence to the idea that apps in this list are largely vulnerable. Similarly, after excluding apps that we had already analyzed, the following are other Android apps which reference the strings "udpolimenew.baidu.com" or "udpolimeok.baidu.com", the API endpoints used by Baidu Input Method:

- com.adamrocker.android.input.simeji
- com.facemoji.lite.xiaomi.gp
- com.facemoji.lite.xiaomi
- com.preff.kb.xm
- com.facemoji.lite.transsion
- com.txthinking.brook
- com.facemoji.lite.vivo
- com.baidu.input_huawei
- com.baidu.input_vivo
- com.baidu.input_oem
- com.preff.kb.op
- com.txthinking.shiliew
- mark.via.gp
- com.qinggan.app.windlink

- com.baidu.mapauto

These findings suggest that a large ecosystem of apps may be affected by the vulnerabilities that we discovered in this report.

6. Coordinated disclosure

We reported the vulnerabilities that we discovered to each vendor in accordance with our vulnerability disclosure policy. All companies except Baidu, Vivo, and Xiaomi responded to our disclosures. Baidu fixed the most serious issues we reported to them shortly after our disclosure, but Baidu has yet to fix all issues that we reported to them. The mobile device manufacturers whose preinstalled keyboard apps we analyzed fixed issues in their apps except for their Baidu apps, which either only had the most serious issues addressed or, in the case of Honor, did not address any issues (see Table 12 for details). Regarding QQ Pinyin, Tencent indicated that “with the exception of end-of-life products, we aim to finalize the upgrade for all active products to transmit EncryptWall requests via HTTPS by the conclusion of Q1 [2024]”, but, as of April 1, 2024, we have not seen any fixes to this product. Tencent may consider QQ Pinyin end-of-life as it has not received updates since 2020, although we note that it is still available for download. For timelines and full correspondence of our disclosures to each vendor, please see the Appendix.

✘✘	working exploit created to decrypt transmitted keystrokes for both **active and passive** eavesdroppers
✘	working exploit created to decrypt transmitted keystrokes for an **active** eavesdropper
!	weaknesses present in cryptography implementation
✓	no known issues or all known issues fixed
N/A	product not offered or not present on device analyzed

Legend.

Keyboard developer	Android				iOS	Windows
Tencent[†]	✗				N/A	✗
Baidu	!				!	!
iFlytek	✓				✓	✓
	Pre-installed keyboard developer					
Device manufacturer	Own	Sogou	Baidu	iFlytek		
Samsung	✓	✓[*]	!	N/A	N/A	N/A
Huawei	✓[*]	✓	N/A	N/A	N/A	N/A
Xiaomi	N/A	✓[*]	!	✓	N/A	N/A
OPPO	N/A	✓	![*]	N/A	N/A	N/A
Vivo	✓[*]	✓	N/A	N/A	N/A	N/A
Honor	N/A	N/A	✗ ✗[*]	N/A	N/A	N/A

[*] Default keyboard app on our test device.

[†] Both QQ Pinyin and Sogou IME are developed by Tencent; in this report we analyzed QQ Pinyin and found the same issues as we had in Sogou IME.

Table 12: Status of vulnerabilities after disclosure as of April 1, 2024.

To summarize, we no longer have working exploits against any products except Honor's keyboard app and Tencent's QQ Pinyin. Baidu's keyboard apps on other devices continue to contain weaknesses in their cryptography which we are unable to exploit at this time to fully decrypt users' keystrokes in transit.

6.1. Barriers to users receiving security updates

Users can receive updates to their keyboard apps on their phones' app stores, and such updates typically install in the background without user intervention. In our testing, updating keyboard apps was typically performed without friction. However, in some cases, a user may need to also ensure that they have fully updated their operating system before they will receive the fixes to our reported vulnerabilities for their keyboard app through the app store. In the case of the Honor device we tested, there was no update mechanism for the default keyboard used by the operating system through the app store. Honor devices bundled with a vulnerable version of the keyboard will remain vulnerable to passive decryption. In the case of the Samsung Galaxy Store, we found that on our device a user must sign in with a Samsung account before receiving security updates to their keyboard app. In the case the user does not have a Samsung account, then they must create one. We believe that installing important security updates should be frictionless, and we recommend that Samsung and app stores in general not require the registration of a user account before receiving important security updates.

We also learned from communication with Samsung's security team that our test device had been artificially stuck on an older version of Baidu IME (version 8.5.20.4) compared to the one in the Samsung Galaxy Store. This is because, although the test device was using a Chinese ROM, we were prevented from receiving updates to Baidu IME because the app was geographically unavailable in Canada, where we were testing from. Samsung addressed this issue by adding Baidu's keyboard app to the global market. Generally speaking, we recommend that Samsung and other app stores do not geoblock security updates to apps that are already installed.

6.2. Language barriers in responsible disclosures

We suspect that a language barrier may have prevented iFlytek from responding to our initial disclosure in English. After we did not receive a response for one month, we re-sent the same disclosure e-mail, but with a subject line and one-sentence summary in simplified Chinese. iFlytek responded within three days of this second email and promptly fixed the issues we noted. All future disclosure emails to the Chinese mobile device manufacturers were then written with Chinese subject lines and a short summary in Chinese. Though obvious in hindsight, we encourage security researchers to consider if the company to which they are disclosing uses a different language than the researcher. We suggest submitting vulnerability disclosures, at the very least, with short summaries and email subject lines in the official language of the company's jurisdiction to prevent similar delays as we may have encountered in disclosure timelines.

7. Limitations

In this report we detail vulnerabilities relating to the security of the transmission of users' keystrokes in multiple keyboard apps. In this work we did not perform a full audit of any app or make any attempt to exhaustively find every security vulnerability in any software. Our report concerns analyzing keyboard apps for a class of vulnerabilities that we discovered, and the absence of our reporting of other vulnerabilities should not be considered evidence of their absence.

8. Discussion

In this section we discuss the impact of the vulnerabilities that we found, speculate as to the factors that gave rise to them, and conclude by introducing possible ways to systemically prevent such vulnerabilities from arising in the future.

8.1. Impact of these vulnerabilities

The scope of these severe vulnerabilities cannot be overstated: until this and our previous Sogou report, the majority of Chinese mobile users' keystrokes were decryptable by network adversaries. The keyboards we studied comprise over 95% of the third-party IME market share, which is estimated to be over 780 million users by marketing agencies. In addition, the three phone manufacturers which pre-installed and by default used vulnerable keyboard apps comprise nearly 50% of China's smartphone market.

The vulnerabilities that we discovered would be inevitably discovered by anyone who thinks to look for them. Furthermore, the vulnerabilities do not require technological sophistication to exploit. With the exception of the vulnerability affecting many Sogou-based keyboard apps that we previously discovered, all of the vulnerabilities that we covered in this report can be exploited entirely passively without sending any additional network traffic. This also means any existing logs of network data sent by these keyboards can be decrypted in the future. As such, we might wonder, are these vulnerabilities actively under mass exploitation?

While many governments may possess sophisticated mass surveillance capabilities, the Snowden revelations gave us unique insight into the capabilities of the United States National Security Agency (NSA) and more broadly the Five

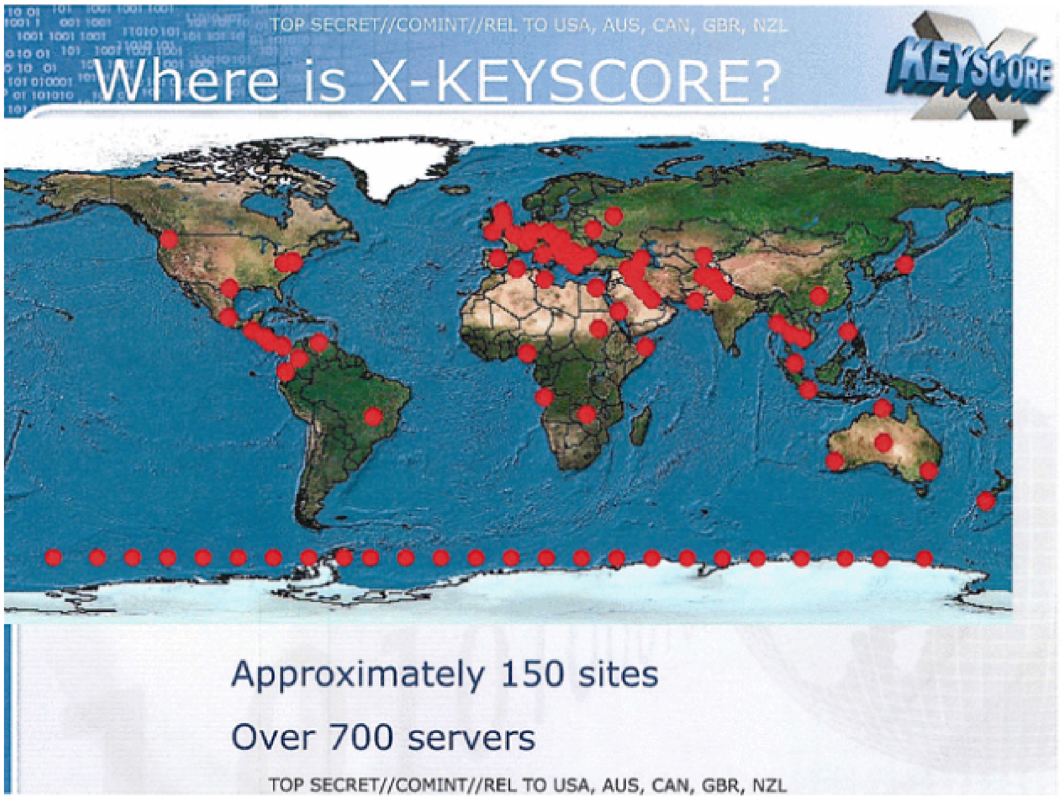
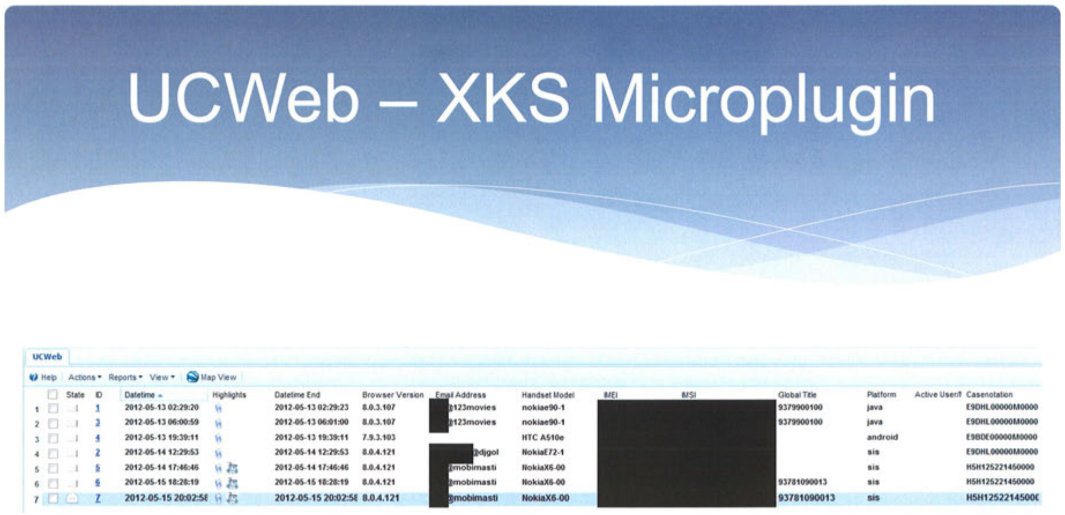

TOP SECRET//COMINT//REL TO USA, AUS, CAN, GBR, NZL
Where is X-KEYSCORE?
KEYSCORE
Approximately 150 sites
Over 700 servers
TOP SECRET//COMINT//REL TO USA, AUS, CAN, GBR, NZL

Figure 13: Locations of XKEYSCORE servers as described in a 2008 NSA slide deck.

Eyes. The revelations disclosed, among other programs, an NSA program called XKEYSCORE for collecting and searching Internet data in realtime across the globe (see Figure 13). Leaked slides describing the program specifically reveal only a few examples of XKEYSCORE plugins. However, one was a plugin that was written by a Five Eyes team to take advantage of vulnerabilities in the cryptography of Chinese-developed UC Browser to enable the Five Eyes to collect device identifiers, SIM card identifiers, and account information pertaining to UC Browser users (see Figure 14 for an illustration).

The similarity of the vulnerability exploited by this XKEYSCORE plugin and the vulnerabilities described in this report are uncanny, as they are all vulnerabilities in the encryption of sensitive data transmissions in software predominantly used by Chinese users. Given the known capabilities of XKEYSCORE, we surmise that the Five Eyes would have the capability to globally surveil the keystrokes of all of the keyboard apps that we analyzed with the exception of Sogou and the apps licensing its software. This single exception exists because Sogou cannot be monitored passively and would require sending packets to Sogou servers. Such communications would be measurable at Sogou's servers and at other vantage points,



UCWeb – XKS Microplugin

UCWeb

Help | Actions ▾ | Reports ▾ | View ▾ | Map View

	State	ID	Datetime	Highlights	Datetime End	Browser Version	Email Address	Handset Model	IMEI	IMSI	Global Title	Platform	Active User?	Casenotation
1		1	2012-05-13 02:29:20		2012-05-13 02:29:23	8.0.3.107	[redacted]@123movies	nokiae90-1	[redacted]	[redacted]	9379900100	java		E9DHL00000M0000
2		3	2012-05-13 06:00:59		2012-05-13 06:01:00	8.0.3.107	[redacted]@123movies	nokiae90-1	[redacted]	[redacted]	9379900100	java		E9DHL00000M0000
3		4	2012-05-13 19:39:11		2012-05-13 19:39:11	7.9.3.103		HTC A510e	[redacted]	[redacted]		android		E9BDE00000M0000
4		2	2012-05-14 12:29:53		2012-05-14 12:29:53	8.0.4.121	[redacted]@djgol	NokiaE72-1	[redacted]	[redacted]		sis		E9DHL00000M0000
5		5	2012-05-14 17:46:46		2012-05-14 17:46:46	8.0.4.121	[redacted]@mobimasti	NokiaX6-00	[redacted]	[redacted]		sis		H5H12522145000
6		6	2012-05-15 18:28:19		2012-05-15 18:28:19	8.0.4.121	[redacted]@mobimasti	NokiaX6-00	[redacted]	[redacted]	93781090013	sis		H5H12522145000
7		7	2012-05-15 20:02:58		2012-05-15 20:02:58	8.0.4.121	[redacted]@mobimasti	NokiaX6-00	[redacted]	[redacted]	93781090013	sis		H5H1252214500

Figure 14: The dashboard of an XKEYSCORE plugin used to monitor for transmissions of sensitive data insufficiently encrypted by UC Browser as described in a 2012 Five Eyes slide deck.

potentially revealing the Five Eyes's target(s) of surveillance to Sogou or Chinese network operators. Therefore, targets of outdated Sogou software would be undesirable victims of mass surveillance, even if such non-passive measurements were within the known capabilities of XKEYSCORE or other Five Eyes programs.

Given the enormous intelligence value of knowing what users are typing, we can conclude that not only do the NSA and more broadly the Five Eyes have the capabilities to mass exploit the vulnerabilities we found but also the strong motivation to exploit them. If the Five Eyes' capabilities are an accurate reflection of the capabilities and motivations of other governments, then we can assume that many other governments are also capable and motivated to mass exploit these vulnerabilities. The only remaining question is whether any government had knowledge of these vulnerabilities. If they did not have such knowledge before our original report analyzing Sogou, they may have acquired after it in the same way that our original research inspired us to look at similar keyboard apps for analogous vulnerabilities. Unfortunately, short of future government leaks, we may never know if or to what extent any state actors mass exploited these vulnerabilities.

Even though we disclosed the vulnerabilities to vendors, some vendors failed to fix the issues that we reported. Moreover, users of devices which are out of support or that otherwise no longer receive updates may continue to be vulnerable. As such, many users of these apps may continue to be under mass surveillance for the foreseeable future.

8.2. How did these vulnerabilities arise

We analyzed a broad sample of Chinese keyboard apps, finding that they are almost universally vulnerable to having their users' keystrokes being decrypted by network eavesdroppers. Yet there is no common library or a single implementation flaw responsible for these vulnerabilities. While some of the keyboard apps did license their code from other companies, our overall findings can only be explained by a large number of developers independently making the same kind of mistake. As such, we might ask, how could such a large number of independent developers almost universally make such a critical mistake?

One attempt to answer this question is to suggest that these were not mistakes at all but deliberate backdoors introduced by the Chinese government. However, this hypothesis is rather weak. First, user keystroke data is already being sent to servers within Chinese legal jurisdiction, and so the Chinese government would have access to such data anyways. Second, the vulnerabilities that we found give the ability not just to the Chinese government to decrypt transmitted keystrokes but to any other actor as well. In an ideal backdoor, the Chinese government would want the desirable property that only they have access to the backdoor. Finally, the Chinese government has made strides to study and improve the data security of apps developed and used in China, attempting to prevent and fix the very sort of vulnerabilities which we discovered. For instance, a 2020 report from CNCERT/CC found that 60 percent of the 50 banking applications that they investigated did not encrypt any user data transmitted over the network, among a litany of other common security issues.

Were Chinese app developers skeptical of using cryptographic standards perceived as "Western"? Countries such as China and Russia have their own encryption standards and ciphers. To our knowledge none of the faulty encryption implementations that we analyzed adhered to any sort of known standard in any country, and each appeared to be home-rolled ciphers. However, it is possible that Asian developers are less inclined to use encryption standards that they fear may contain backdoors such as the potential Dual_EC_DRBG backdoor.

Perhaps Chinese app developers could be skeptical of standards such as SSL/TLS as well. The TLS ecosystem has also only become nearly-universal in the past decade. Especially before broad oversight of certificate authorities became commonplace, there were many valid criticisms of the SSL/TLS ecosystem. In 2011, digital rights organizations EFF and Access Now were both concerned about the certificate authority (CA) infrastructure underpinning SSL/TLS transport encryption. Even today, the vast majority of root certificates trusted by major OSes and browsers are operated by certificate authorities based in the Global North. We also note that all of the IMEs containing vulnerabilities were first released before 2013 and likely had a need for secure network transmission before SSL/TLS became the de-facto standard for strong transport encryption.

Still, it has been a decade since the Snowden leaks demonstrated the global, urgent, and practical need for strong encryption of data-in-transit in 2013, and the TLS ecosystem has largely stabilized, with CA root lists of many major browsers and OSes controlled by voting bodies and certificate transparency deployed. As of 2024, almost 95% of web traffic from users of Firefox in the United States is traveling over HTTPS. In addition, the speed in which both iFlytek and Sogou switched to TLS demonstrates that making the change to standard TLS is not necessarily a time or resource issue. Even if skepticism towards SSL/TLS explains the reluctance to adopt it in the early 2010s, we are not sure why there is much more inertia in the Chinese Internet ecosystem against making the switch to TLS.

Finally, mobile devices and other operating systems are still incapable of guaranteeing the security of data under transmission, despite iOS and Android having introduced restrictions into their APIs. For instance, iOS 9 implemented App Transport Security, a policy placing restrictions on the ability to transmit data without TLS. However, there are two limitations of this technology. First, an app can specify exceptions to this policy in its Info.plist resource. Second, the policy affects high level APIs and leaves communications over lower level socket-based APIs unregulated. Similar to iOS, Android 9 disables cleartext traffic using certain high level APIs by default, but an app may exclude specific domains or avoid the policy by using lower level APIs.

8.3. Can we systemically address these vulnerabilities?

Individually analyzing apps for this class of vulnerabilities and individually reporting issues discovered is limited in the scale of apps that it can fix. First, while we can attempt to manually analyze some of the most popular keyboard apps, we will never be able to analyze every app at large. Second, we might not be able to predict which apps to look at in the first place. For instance, before we analyzed Sogou and the keyboard apps featured in this report, we never would have expected that their network transmissions would be so easily vulnerable to interception. In light of the limitations of the methods that we employed in this report, in the remainder of this section we discuss possibilities for how we might systematically or wholesale address apps which transmit sensitive data over networks without sufficient encryption.

8.3.1. By security researchers paying more attention to the Chinese Internet

There appears to be a general failure of researchers to analyze Chinese apps and the Chinese Internet ecosystem at large, despite its size and influence. The Google Play Store and Apple App Store ecosystems, for instance, are commonly studied by privacy researchers, but many Chinese app stores are overlooked, despite that many popular Chinese apps have more users than their counterparts on the Google Play Store. While the vulnerabilities that we discovered were not all trivial to find and many took substantial analysis to attack, most would have been inevitably discovered by any researcher analyzing these apps for data security. A researcher studying network traffic from users of Chinese devices could also have identified strange, non-standard traffic.

8.3.2. By using app store enforcement

One might call on app stores to enforce the use of sufficient encryption to protect sensitive data in transit. App stores already have a number of rules that they enforce through a combination of automated and manual review. Calling on app stores to enforce sufficient encryption of in-transit sensitive data is tempting given the resources of the companies operating the app stores. However, failing any other innovation, the same scaling issues that apply to other researchers studying these apps will apply to those working for these companies.

8.3.3. By using device permission models

On Android devices, installing *any* keyboard, regardless of whether or how it communicates with servers over the Internet, brings up a pop-up with the following text:

> This input method may be able to collect all the text you type, including personal data like passwords and credit card numbers.

The wording of these warning messages is overbroad and does not necessarily help users distinguish between keyboards that transmit keystrokes over the network, keyboards that transmit keystrokes insecurely (using something other than standard TLS) over the network, and keyboards that do not transmit any data at all.

iOS devices, on the other hand, sandbox their keyboards by default. There is a "Full Access" or "open access" permission that must be explicitly granted to keyboards before they have network access, among other privileges. Without this permission, third-party keyboards cannot transmit network data. We recommend Android also adopt a more fine-grained permission model for keyboards.

Furthermore, the vulnerable apps that we studied transmit data using low level socket APIs versus higher level APIs that require the usage of TLS or HTTPS. One might desire that separate system calls be designed for TLS or HTTPS traffic in addition to the lower level socket system calls so that devices could implement an UNSAFE_INTERNET permission that would be required for apps to use the lower level system calls while still allowing TLS-encrypted traffic for apps that do not have this permission.

While this approach may have some merit, it also has certain drawbacks. It makes sense for situations where apps are untrustworthy and the operating system is completely trustworthy, but there are common situations where the operating system could be not as or even less trustworthy than apps that it is running. One common case would be a user who is running an up-to-date app on an out of date operating system, possibly because the user's device is no longer receiving operating system updates. In such a case, the app's implementation of TLS is more likely to be secure than that of the operating system. Furthermore, a user's operating system may be compromised by malware or otherwise be untrustworthy in itself. Introducing a TLS system call would centralize the encryption of all sensitive data and grant the operating system easy visibility into all unencrypted data. In any case, innovating in areas of encryption is an important right of application developers, and it may not make sense to stifle apps like Signal because of their use of end-to-end or other novel encryption by requiring them to obtain an UNSAFE_INTERNET permission.

One might alternatively desire for apps at large to not be able to access the Internet at all. Instead of an UNSAFE_INTERNET permission, what about introducing an INTERNET permission to govern all Internet socket access, similar to the "Full Access" permission which iOS already applies to keyboard apps? Android devices in fact already have such a permission that apps must request to use Internet (AF_INET) sockets, but it is not a permission that is exposed to ordinary users either in the Google Play Store or through any stock Android user interface, and it is automatically granted when installing an app. Unfortunately, given all of the interprocess communication (IPC) vehicles on modern smart devices, restricting Internet socket access may not guarantee that the app could not communicate over the Internet (e.g., through Google Play services). GrapheneOS, an open source Android-based operating system, implements a NETWORK permission. However, denying this permission can lead to surprising results where apps can still communicate with the Internet via IPC with other apps. As such, we recommend that both the developers of Android and iOS work toward a meaningful INTERNET permission that would adequately inform users of whether an app communicates over the Internet.

8.3.4. By international standards bodies better engaging with Chinese developers

We encourage International standards bodies like the IETF to continue to engage and outreach Chinese Internet companies and engineers in good faith to further reduce friction in cross-linguistic knowledge transfer. The presence of these similar but independent vulnerabilities demonstrate that there is a friction in the transfer and implementation of knowledge between the English-speaking cryptography community and the Chinese cryptography community. For instance, Schneier's Law or the oft-repeated mantra "don't roll your own crypto" may be common knowledge to cryptographers trained in English, but perhaps lost in translation. A lag across linguistic boundaries means that general information like the recent stabilization of TLS and webPKI infrastructure may travel more slowly, and updating encryption software to reflect new information may lag even further behind. One other possible example of this phenomenon is that, according to Firefox Telemetry, up until 2020, the Japanese Internet ecosystem also significantly lagged behind the global average in HTTPS adoption.

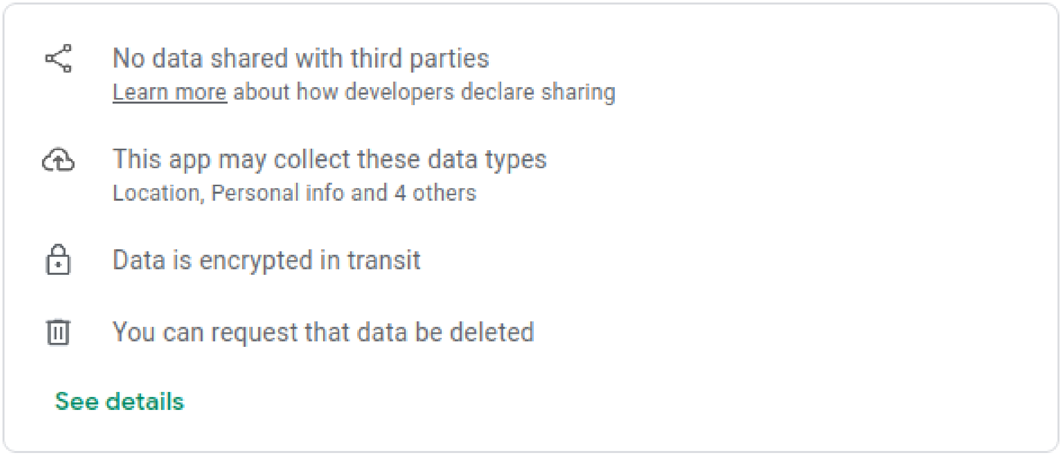
Although protocols put out by IETF and other International standards bodies can be far from bulletproof, these bodies can still help facilitate international communication about the current state-of-the-art in protocol encryption. The burden of cross-linguistic and cross-cultural exchange on technical standards falls on global standards bodies. Western media outlets and researchers tend to uniformly attribute the actions and participation of private Chinese companies within standards bodies to government actors seeking sovereignty over Internet standards. While skepticism may be warranted in certain cases, there is also research that challenges a simplistic and overbroad narrative. As a single data point, we note that we did not find these issues in Huawei’s keyboards, whose employees are often noted as especially active participants in IETF standard-setting.

8.3.5. By using automated static or dynamic analysis

There has been a failure of automated tools to detect insecure traffic at large. Longitudinal TLS telemetry has largely been focused on web-based perspectives (i.e., how many domains support TLS or how many web connections are encrypted by TLS?), and the mobile perspective is often overlooked, despite the increasing dominance of mobile traffic globally. Although there are some research projects that survey TLS usage in Android mobile apps at scale, there is no public longitudinal data from these projects (i.e., they are run as one-off studies), and many focus on the Google Play’s Android ecosystem, thereby excluding the Chinese mobile Internet. There is perhaps a need for public longitudinal TLS telemetry for popular mobile applications globally, via automated static or dynamic analysis at scale.

8.3.6. By using attestations in app stores

Another way for users to gain visibility into the security and privacy properties of their apps is through the use of developer attestations, such as the ones that appear in data safety sections in many popular app stores. Both the Apple App Store and the Google Play Store collect and display such attestations to varying extents, including attestations as to what data an app collects (if any) and with whom it is shared (if anyone). Additionally, the Play Store allows developers the


No data shared with third parties
Learn more about how developers declare sharing

This app may collect these data types
Location, Personal info and 4 others

Data is encrypted in transit

You can request that data be deleted

See details

Figure 15: An example of an attestation for Microsoft SwiftKey.

opportunity to attest to performing "encryption in transit" (see Figure 15 for an example). These attestations allow users to clearly see what security and privacy properties an app's developer claims it to have and, like privacy policies, they provide means of redress if violated.

We wanted to evaluate whether the apps that we analyzed lived up to their attestations concerning their encryption in the app stores in which they are available. Among the apps that we analyzed, only Baidu IME was available in the Play Store. At the time of this writing, it does not attest to its data being encrypted in transit. Although other apps that we analyzed were available in Apple's App Store, to our knowledge, this store does not display an attestation for whether the app encrypts data in transit. As such, across both the Google Play and the Apple App stores, attestations were insufficient for compelling the keyboard apps' developers to implement proper encryption or in providing users any opportunity for redress.

In light of the above findings, we believe that users would benefit from the following recommendations: (1) that app store operators require developers to attest to whether or not an app encrypts data in transit, (2) that app store operators display not only when developers attest to all data being encrypted in transit but also display a warning when they fail to, and (3) that app store operators require apps in certain sensitive categories, such as keyboard apps, to either positively attest to encrypting all data in transit or to attest to not transmitting any data at all.

Since most of the apps that we found perform some type of encryption, even if it were wholly inadequate, one might wonder if attesting that data is merely "encrypted" is enough, since the data arguably did have some manner of encryption applied to it during transit. The Play Store provides some guidance on this topic. Under the question — “How should I encrypt data in transit?” — the documentation notes: “You should follow best industry standards to safely encrypt your app’s data in transit. Common encryption protocols include TLS (Transport Layer Security) and HTTPS.”

Another issue with attestations is that they provide no guarantee that an app behaves as its developers attest, as developers can, after all, make false attestations. While we wish that attestations could guarantee that an app sufficiently implements proper cryptography to the same extent that a permission system can guarantee an app does not use a microphone, false attestations provide an opportunity for redress. For instance, apps which are found to violate attestations would be subject to removal from app stores. Furthermore, apps which violate attestations could be subject to fines by regulatory bodies such as the FTC. Finally, apps which violate the attestation could be liable to civil suits.

While the apps we analyzed were predominantly available from Chinese app stores, we equally recommend that Chinese app stores adopt these recommendations in addition to the Apple App Store and the Google Play Store. Moreover, while this report focuses on the problem of poor encryption practices as it applies to Chinese apps, the problem to varying extents applies to apps of all other provenances.

9. Summary of recommendations

We conclude our report by summarizing our recommendations to multiple stakeholders.

Recommendations to security researchers

- Researchers should analyze more apps from the East Asian app ecosystem and from other popular ecosystems which may be outside of their own locale.
- Researchers should develop better static and dynamic analysis techniques to recognize the types of vulnerabilities that we discovered in this report at scale.

- Researchers submitting vulnerability disclosures to a company should include short summaries and email subject lines in the official language of the company's jurisdiction.

Recommendations to international standards bodies

- International standards bodies should continue to engage with security engineers from Chinese Internet companies.

Recommendations to app store operators

- App stores should not require account registration as a condition to receive security updates.
- App stores should not geoblock security updates.
- App stores should allow developers to attest to all data being transmitted with encryption, similar to the ability in the Google Play Store.
- App stores should display not only when developers attest to all data being encrypted in transit but also display a warning when they fail to.
- App stores should require apps in certain sensitive categories, such as keyboard apps, to either positively attest to encrypting all data in transit or to attest to not transmitting any data at all.

Recommendations to keyboard app developers

- Use well-tested and standard encryption protocols, like TLS or QUIC.
- Make every attempt to provide features on-device without requiring transmitting sensitive data to cloud servers.

Recommendations to mobile operating system developers

- Android should implement sandboxing by default for keyboard apps, similar to iOS, that prevents a keyboard from transmitting network traffic among other activities until a user grants the app full access.
- The developers of Android and iOS should work toward a meaningful INTERNET permission that would adequately inform users of whether *any* app communicates over the Internet.

Recommendations to device manufacturers

- Conduct security audits of third-party keyboards that you intend to preinstall by default on your operating systems.

Recommendations to users

- Users of Honor’s pre-installed keyboard or users of QQ pinyin should switch keyboards immediately.
- Users of any Sogou, Baidu, or iFlytek keyboard, including the versions that are bundled or pre-installed on operating systems, should ensure their keyboards and operating systems are up-to-date.
- Users of any Baidu IME keyboard should consider switching to a different keyboard or disabling the “cloud-based” feature.
- Users with privacy concerns should not enable “cloud-based” features on their keyboards or IMEs or should switch to a keyboard that does not offer “cloud-based” prediction.
- iOS users with privacy concerns should not enable “Full Access” for their keyboards or IMEs.

A. Known affected software

We recommend that all users keep their operating systems and apps, including keyboard apps, up to date. If you use any of the following software, we especially recommend you update to the most recent version of your OS and application. As of April 1, 2024, *the following software has fixes available:*

Separately installed, third-party keyboards

- Sogou IME / 搜狗输入法 for Android and Windows
- Baidu IME / 百度输入法 for Windows (this software has only been partially fixed, see below)
- iFlytek IME / 讯飞输入法 for Android

Pre-installed on Samsung devices with Chinese edition ROM

- Samsung Keyboard
- Baidu IME / 百度输入法

Pre-installed on Xiaomi devices with Chinese edition ROM

- Sogou IME Xiaomi Version / 搜狗输入法小米版
- iFlytek IME Xiaomi Version / 讯飞输入法小米版

Pre-installed on OPPO devices with Chinese edition ROM

- Sogou IME Custom Version / 搜狗输入法定制版

Pre-installed on Vivo devices with Chinese edition ROM

- Sogou IME Custom Version / 搜狗输入法定制版

The following software does not use TLS and may still contain weaknesses:

Separately installed, third-party keyboards

- Baidu IME / 百度输入法 for Android, Windows, and iOS

Pre-installed on Xiaomi devices with Chinese edition ROM

- Baidu IME Xiaomi Version / 百度输入法小米版

Pre-installed on OPPO devices with Chinese edition ROM

- Baidu IME Custom Version / 百度输入法定制版

The following software has not been fixed and is easily exploitable, and we suggest that users switch to another keyboard entirely:

Separately installed, third-party keyboards

- QQ Pinyin IME / QQ拼音输入法 for Android and Windows

Pre-installed on Honor devices with Chinese edition ROM

- Baidu IME Honor Version / 百度输入法荣耀版

B. Disclosure timelines

For the disclosure timelines, please see here.