IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 27 March 2025, accepted 21 April 2025, date of publication 23 April 2025, date of current version 2 May 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3563826

RESEARCH ARTICLE

Unveiling Samsung Quantum Galaxy: Securing Smartphones With Quantum and Post-Quantum Cryptography

OMAR ALIBRAHIM
Information Science Department, Kuwait University, Safat 13060, Kuwait
e-mail: omar.alibrahim@ku.edu.kw

This work was supported by Kuwait Foundation for the Advancement of Sciences (KFAS) of Kuwait under Grant PS24-18EO-01.

ABSTRACT Quantum technologies have emerged as vital tools for enhancing mobile communication security. The Samsung Galaxy series now features a quantum random number generator (QRNG) that offers true randomness, strengthening cryptographic protection. Despite its potential, QRNG integration remains largely unexplored in practical applications. This paper investigates the QRNG chip's integration and functionality in the Samsung Quantum Galaxy smartphone through digital forensics and reverse engineering. We identified a lack of effective QRNG utilization in existing applications and addressed this gap by developing a secure instant messaging and VoIP application that combines QRNG with post-quantum cryptography (PQC). Our solution demonstrates that leveraging QRNG-generated randomness alongside PQC significantly improves security against emerging quantum threats, establishing a foundation for enhanced mobile data protection.

INDEX TERMS Quantum random number generator (QRNG), post-quantum cryptography (PQC), reverse engineering, digital forensics, secure mobile communications, mobile application security.

I. INTRODUCTION

Quantum technologies have gained traction across various industries, particularly for enhancing security measures through true randomness and quantum-resistant cryptography. The Samsung Quantum Galaxy smartphone, with its embedded quantum random number generator (QRNG) chip manufactured by ID Quantique [1], [2], [3], has emerged as a cutting-edge example for integrating quantum technologies into consumer electronics. However, despite its potential, the real-world application of QRNG capabilities remains underexplored.

In this research, we dissected the Samsung Quantum Galaxy model to uncover how the QRNG chip functions within the device. Through digital forensics of the phone's file system and reverse engineering of its binaries and Android applications, we discovered the mechanisms used to invoke the chip's random byte generation capabilities.

The associate editor coordinating the review of this manuscript and approving it for publication was Yuan Gao.

Specifically, we identified a designated Unix device file for the QRNG and determined how it could be called programmatically. Following this discovery, we conducted entropy tests to evaluate the QRNG's randomness quality and compare its performance against that of other random number generators available on the device.

After identifying these technical processes, we scanned the existing apps for meaningful use cases of QRNGs. Surprisingly, we found no killer apps that exploited this unique hardware for security enhancements, leaving a significant untapped opportunity. To address this gap, we developed a secure instant messaging and VoIP mobile application that harnesses the power of the QRNG chip. In addition to the QRNG, we incorporated post-quantum cryptography (PQC) to ensure resistance against future quantum threats, creating a dual-layered security solution. This paper details the steps we took in reverse engineering the device, the insights we gained, and the development process of our secure communication application. Our work not only highlights the current underutilization of quantum technologies in mobile security but also presents a solution that demonstrates the potential of quantum and post-quantum cryptography in safeguarding mobile communications.

The paper begins by discussing quantum and post-quantum technologies, emphasizing their importance in cryptography and secure communications (Section II). It then examines the Quantum Galaxy series, particularly the incorporation of QRNG technology for enhanced security (Section III).

In Section IV, we outline our investigation methodology and describe the purpose of the research, the hypotheses that guided our study, and the structured phases of our methodology, including the discovery, research, and development phases. Section V details our forensic analysis of the phone's filesystem, focusing on QRNG-related artifacts. Section VI assesses the entropy of the phone's random number generators and compares the QRNG with conventional methods. Reverse engineering of QRNG-enabled apps is explored in Section VII, and Section VIII presents the development of a secure instant messaging and VoIP app utilizing the QRNG and post-quantum cryptography. In Section IX, we revisit our hypotheses and discuss the results and conclusions based on our findings. The paper concludes by summarizing the outcomes and their significance for future secure mobile communication (Section X).

II. BACKGROUND

In recent years, the convergence of quantum technologies with cryptographic security has drawn significant attention, particularly for mobile communications. Quantum random number generators (QRNGs) and post-quantum cryptography (PQC) have emerged as key innovations that address the growing need for robust security measures in the face of evolving quantum threats. This section reviews the relevant literature on QRNG technologies, post-quantum cryptography, and secure mobile communications for applications.

A. QUANTUM RANDOM NUMBER GENERATORS (QRNGS)

Quantum random number generators (QRNGs) leverage the inherent unpredictability of quantum phenomena to generate true randomness, which is critical for secure communication and cryptographic applications. Mannalatha, Mishra, and Pathak [4] provided a comprehensive classification of QRNGs, highlighting their reliance on quantum principles, such as superposition and entanglement, which ensure unpredictability. They categorized QRNGs based on their physical mechanisms, such as photon detection and quantum vacuum fluctuations. Photon-based methods, as detailed by Ma et al. [5], have become a popular approach that uses phenomena such as photon arrival times to ensure randomness.

The key advantage of QRNGs over conventional pseudorandom number generators (PRNGs) is their fundamental randomness, making them ideal for applications requiring high security [4], [5] [6]. Despite their strengths, QRNGs are challenging. Hurley-Smith and Hernandez-Castro [7] examined the potential biases in QRNGs, revealing that imperfections in quantum systems can introduce statistical biases, undermining their randomness.

Nie et al. [8] addressed these challenges by proposing measurement–device-independent QRNGs, which aim to eliminate trust assumptions about the devices used to generate randomness. Other studies, such as those by Acerbi et al. [9] and Tisa et al. [10], have focused on practical advancements and proposed integrated QRNGs using photon-counting detectors and CMOS technology to achieve high-speed random number generation suitable for real-world applications [9], [10] [11]. Recent studies have introduced new methods to enhance QRNGs. For example, Pelofske [12] examined programmable quantum annealers as a novel source of randomness, whereas Ash-Saki et al. [13] proposed using machine learning to improve the reliability of QRNGs by correcting biases. These innovations aim to make QRNGs more accessible and scalable while maintaining the high levels of randomness required for secure systems.

B. POST-QUANTUM CRYPTOGRAPHY (PQC)

With the evolution of quantum computing, the requirement for cryptographic system resistance to quantum computing attacks has become increasingly critical. Unlike conventional cryptosystems, such as the Rivest–Shamir–Adleman (RSA) and elliptic curve cryptography (ECC) cryptosystems, which will be threatened in the near future, post-quantum cryptography (PQC) aims to resist the computational power of quantum computers. As part of its post-quantum cryptography standardization, the National Institute of Standards and Technology (NIST) has approved several algorithms for post-quantum standardization. CRYSTALS-Kyber, outlined in FIPS 203 [14], is a module-lattice-based key encapsulation mechanism that provides secure encryption, even against quantum computing attacks. Similarly, CRYSTALS-Dilithium, outlined in FIPS 204 [15], offers a lattice-based digital signature scheme that ensures robust authentication and integrity.

In addition, FIPS 205 [16] introduced SPHINCS+ [17], a stateless hash-based digital signature algorithm. SPHINCS+ operates on hash functions and Merkle trees, providing strong quantum-resistant signatures, although it generates larger signatures than lattice-based schemes do. Together, these standards form the foundation for quantum-secure cryptography, supporting the future of digital security in a quantum computing era [17], [18] [19].

C. INDUSTRY DEVELOPMENT OF QUANTUM CRYPTOGRAPHIC TECHNOLOGIES

The development of quantum technologies is rapidly transforming industries, particularly cybersecurity [20], [21] [22], [23] [24]. With the advent of quantum computing, conventional encryption methods are at risk, propelling companies to innovate quantum-safe solutions to protect data and communication. ID Quantique, a leader in quantum cryptography, has integrated quantum technologies into commercial products, such as Quantum Galaxy series, showing how quantum-based security can be embedded in consumer devices [2]. This is a significant step in the industry, where quantum-safe features are becoming more accessible to the public. In addition to consumer devices, ID Quantique's Clavis XG quantum key distribution (QKD) [20] system showcases enterprise-grade efforts to safeguard critical infrastructure via QKD technology. Such developments reflect a broader industry trend in which quantum technologies are not only theoretical but also actively being implemented in future-proof cybersecurity across various sectors.

Moreover, advancements in quantum random number generators (QRNGs) have enhanced the quality of cryptographic keys, thereby addressing one of the industry's key challenges: the generation of true randomness. Companies such as ID Quantique, Quantum eMotion, QuintessenceLabs, and KETS Quantum are at the forefront of this innovation and are developing QRNG devices that significantly improve the entropy in cryptographic systems [3], [21] [22], [23]. These QRNGs are essential for ensuring that encryption methods are robust enough to withstand quantum threats while also meeting the requirements of high-speed communication networks. Thus, the industry is moving toward integrating QRNGs with quantum-safe key delivery solutions, such as Quantum Xchange's Phio TX system, which supports seamless and secure transitions to quantum-safe cryptography [24]. This convergence of QRNG and QKD technologies signals the industry's commitment to building resilient cryptographic infrastructure that can endure the rise of quantum computing.

D. SECURE COMMUNICATION APPLICATIONS IN SMARTPHONES

End-to-end encryption is a hallmark of secure communication applications. Apps such as WhatsApp [25], Signal [26], and Telegram [27] use various cryptographic techniques to secure communication. Signal protocols, for instance, employ a combination of elliptic curve cryptography (ECC) and Diffie–Hellman key exchange, both of which are vulnerable to cryptanalysis by quantum computers [28]. Although the literature on secure communication apps is vast, most existing solutions rely on conventional cryptography, which, as highlighted in multiple studies, will likely be broken by quantum computing attacks in the future.

The integration of quantum random number generators (QRNGs) and post-quantum cryptography (PQC) in these apps remains an underexplored area, though theoretical work suggests that these technologies can provide enhanced security and future-proofing against quantum threats [29]. Recent research has increasingly emphasized the need to develop cryptographic systems that are resistant to quantum computing attacks. For example, Zhang et al. [30] introduced a reference frame-independent quantum key distribution (QKD) server system that can be integrated with mobile technologies, demonstrating the potential for on-chip QKD in secure communications.

Mattsson et al. [31] discussed the potential of quantum technology to revolutionize the security of mobile networks. The authors argue that quantum computers can render current encryption algorithms obsolete, but quantum technology can also enable the development of more secure encryption algorithms that are resistant to even the most powerful quantum computing attacks. More recently, Hoque, Aydeger, and Zeydan [32] explored the intersection of post-quantum cryptography and quantum key distribution (QKD) in mobile networks, further highlighting the need for sustainable cryptographic architectures. However, real-world implementations of QRNGs combined with PQC in mobile applications remain largely experimental. For this reason, this research seeks to contribute to the field by presenting new insights and proposing potential pathways for integrating quantum-based security measures into mobile communication applications.

E. QRNGS AND PQC IN PRACTICE: A GAP IN EXISTING APPLICATIONS

To the best of our knowledge, no study has yet demonstrated the effective utilization of QRNGs in mainstream mobile applications despite the introduction of smartphones with QRNG capabilities [6], [33]. Similarly, PQC remains largely experimental in mobile applications owing to performance overheads and the absence of standardized algorithms [18], [34].

The existing literature reveals a rich body of research on QRNGs and PQC as separate fields, but the integration of these technologies in mobile applications is limited. Furthermore, reverse engineering of QRNG-enabled smartphones remains underexplored. This research builds on foundational research of quantum cryptography, post-quantum algorithms, and secure communication applications and contributes new insights into the potential of QRNG-enabled smartphones. By reverse engineering Quantum Galaxy series, this research uncovers the underutilization of QRNGs in existing apps and proposes a solution that leverages both QRNGs and PQC to create a secure communication platform for the future.

III. OVERVIEW OF THE QUANTUM GALAXY SERIES

The Quantum Galaxy series represents a groundbreaking advancement in mobile security by integrating quantum-based cryptographic features to address modern cybersecurity challenges (Figure 1). Launched in 2020 with the release of Quantum Galaxy 1, this series has marked Samsung's commitment to incorporating advanced security technologies into consumer smartphones. Each successive model has introduced improved capabilities: Quantum Galaxy 2 was released in 2021, followed by Quantum Galaxy 3 in 2022, Quantum Galaxy 4 in 2023, and Quantum Galaxy 5 in 2024. The upcoming Quantum Galaxy 6, set for release by the end of 2025, will further solidify the device's position as a leader

FIGURE 1. Samsung Quantum Galaxy smartphone series with an integrated QRNG chip manufactured by ID Quantique for enhanced cryptographic security and true random number generation. (Courtesy of Samsung Inc [2]).

in secure mobile technology by improving its cryptographic features.

A. HARDWARE AND SOFTWARE FEATURES

In addition to its focus on security, the Quantum Galaxy series has consistently delivered cutting-edge hardware and software upgrades. Each iteration has been equipped with high-performance processors, OLED displays, 5G connectivity, and long-lasting battery life to ensure a smooth and responsive user experience. For example, Quantum Galaxy 6 will introduce a 144 Hz refresh rate for its display, offering a fluid visual experience for gaming and media consumption. The series has also consistently improved its camera systems with advanced image processing and multi-lens setups that enhance photography and videography performance. These nonsecurity features make Quantum Galaxy phones secure, versatile, and highly functional for everyday users.

B. STRATEGIC PARTNERSHIPS

A key element of Quantum Galaxy's development is Samsung's strategic partnership with ID Quantique and SK Telecom [2]. The ID Quantique's quantum random number generator (QRNG) chip, first embedded in Quantum Galaxy 1, is the cornerstone of the security infrastructure of the series. This chip uses quantum phenomena to generate true random numbers, ensuring superior entropy and cryptographic strength for secure communication. SK Telecom's involvement has helped expand the phone's user base, particularly in South Korea, where secure communication technologies are in high demand.

C. QUANTUM-BASED SECURITY FEATURES

The Quantum Galaxy series is designed with a strong focus on security that is centered on the integration of the QRNG chip, which significantly enhances its cryptographic capabilities. By generating true random numbers based on quantum phenomena, the QRNG strengthens the encryption processes and improves the unpredictability of secure communications, making the device especially valuable for applications requiring high levels of data security. This includes secure communication, authentication, and encryption-key generation.

D. ADOPTION AND MARKET IMPACT

These security features make the Quantum Galaxy series ideal for sectors that require stringent data protection, such as government institutions, financial organizations, and enterprises handling sensitive information. Quantum Galaxy phones are at the forefront of secure communications technology, with quantum-based encryption methods that can resist modern cryptographic attacks. The adoption of these devices has been steadily increasing in regions where cryptographic security and post-quantum resilience are prioritized, particularly South Korea and parts of Europe.

IV. INVESTIGATION METHODOLOGY

The purpose of this study was to assess the Samsung Quantum Galaxy smartphone as a representative example of a QRNG-enabled computing device. Our research investigated the integration, performance, and utilization of QRNG chips within smartphones. The study further explored the potential of QRNGs as a security enhancement in mobile applications through practical implementation in a secure communication system.

Our research was guided by the following hypotheses:

- **Hypothesis 1**: The QRNG component is accessible and callable at the operating system level in the Samsung Quantum Galaxy smartphone.
- **Hypothesis 2**: The embedded QRNG produces truly random numbers and outperforms other random number generators available on the device.
- **Hypothesis 3**: Apps on the smartphone heavily utilize the QRNG chip to generate cryptographic keys and nonces in encryption processes for data and network security.

Our research methodology consists of three phases: a discovery phase, a research phase, and a development phase. Each phase addresses specific objectives to assess the implementation, performance and practical use of QRNGs in mobile security applications.

A. DISCOVERY PHASE

The first phase aimed to investigate the presence, definition, and accessibility of the QRNG chip within Samsung Quantum Galaxy smartphones. This was essential for understanding the system-level integration of the QRNG chip and determining how developers or system processes might access it programmatically. The goal was to confirm whether the QRNG was implemented as a device file and to identify key technical characteristics that define its behavior.

1) FILE SYSTEM ANALYSIS (DIGITAL FORENSICS)

To explore the phone's file system, we used Android Debug Bridge (ADB) to analyze system directories. Our investigation targeted system files, configuration scripts, and device file structures to identify QRNG-related artifacts. This analysis led to the discovery of **`/dev/qrandom`**, which was confirmed to be the designated Unix device file for QRNGs.

2) ROOTING

To gain deeper access for system-level inspection and QRNG analysis, we rooted the Samsung Quantum Galaxy smartphone. The rooting process was performed using tools such as Magisk for systemless rooting and Odin, Samsung's official flashing tool. Root access allowed us to bypass default security restrictions, enabling the inspection of protected directories and configuration files. With elevated privileges, we identified additional QRNG-related processes, analyzed **`/dev/qrandom`** permissions, and accessed data generated by QRNG-integrated applications.

Rooting was critical for confirming the integration of the QRNG within the phone's cryptographic infrastructure and for retrieving data required for subsequent entropy testing and application analysis.

B. RESEARCH PHASE

The research phase sought to evaluate the QRNG chip's performance, assess its randomness quality, and investigate its utilization in existing mobile applications. This phase involved entropy testing, application threat hunting, and reverse engineering.

1) ENTROPY TESTING

To evaluate the quality of QRNG-generated randomness, we performed entropy analysis using three established test suites:

- **NIST SP800-90b**: To assess the entropy source's robustness and unpredictability.
- **NIST SP800-22**: To evaluate statistical randomness properties and distribution patterns.
- **DieHarder**: To conduct in-depth statistical tests for randomness evaluation.

We collected data samples from random number generators on the device, specifically **`/dev/qrandom`**, **`/dev/random`**, and **`/dev/urandom`**, for comparative analysis. The data samples were evaluated using compression tests, collision detection, and other tests to measure entropy strength. The results were analyzed to determine whether the QRNG significantly outperformed traditional random number generators present in the phone's operating system.

2) QRNG APP HUNTING

To investigate the presence of QRNG-utilizing applications, we performed a comprehensive scan of the phone's file system for APK files referencing **`/dev/qrandom`**. Using ADB commands, we searched system directories, pre-installed application storage locations, and user-installed app directories.

3) REVERSE ENGINEERING

To gain deeper insights into how identified apps attempted to interact with the QRNG, we conducted reverse engineering using Androguard and JADX, which decompile Android APKs into human-readable code.

C. DEVELOPMENT PHASE

Following the discovery that no meaningful applications effectively leveraged the QRNG chip for enhanced security, the development phase aimed to address this gap by developing a secure communication application that harnesses QRNG-generated randomness.

The secure messaging and VoIP application we developed integrates QRNG-generated keys for encryption alongside post-quantum cryptography (PQC) algorithms. This dual-layered security approach was designed to increase resistance against both conventional and quantum-based computing attacks.

The application includes the following features:

- **QRNG key generation**: The app retrieves random data directly from **`/dev/qrandom`** to seed cryptographic keys used for encryption.
- **PQC integration**: The developed app incorporates PQC algorithms to protect against quantum computing threats.
- **End-to-end encryption**: The app ensures that only the communicating users can read the messages, with no third-party access.
- **Private relay server**: The app routes encrypted messages through a secure, anonymizing server to protect user identity and metadata.

To address concerns about data privacy, our application is designed to prioritize data sovereignty. Unlike conventional messaging platforms that route messages through foreign third-party servers, our solution ensures that user data remains under the control of organizations or users within their legal jurisdiction and addresses privacy concerns for politically sensitive users, enterprises, or government institutions.

V. DIGITAL FORENSICS OF QUANTUM GALAXY's FILE SYSTEM

In this section, we investigate the Quantum Galaxy file system to understand how the quantum random number generator (QRNG) chip is identified and utilized by the phone's operating system. Our goal is to determine how the smartphone's operating system interacts with the QRNG, whether it can be called programmatically, and how it can be leveraged by apps on the device. By examining the underlying file system, we seek to uncover how the QRNG is integrated into cryptographic functions, how apps may invoke it, and how developers can build applications that

```
$ cat qrng.rc
on boot
  chmod 0644 /dev/qrandom
  chown system system /sys/class/misc/qrandom/status/
      diag_status
  chown system system /sys/class/misc/qrandom/status/
      selftest_status
```

LISTING 1. Setting QRNG device permissions and ownership on boot.

effectively utilize the QRNG for enhanced security. This analysis provides insights into the technical pathways for accessing and utilizing QRNG chips within a broader mobile ecosystem.

A. LOCATING THE QRNG UNIX DEVICE FILE

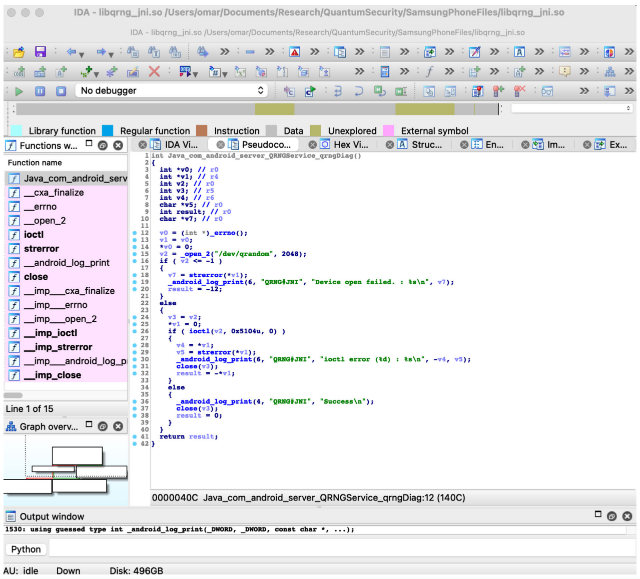
To understand how the QRNG chip is integrated at the operating system level of the Samsung Quantum Galaxy smartphone and how it can be accessed programmatically, we accessed the file system and performed an in-depth forensic analysis, which included listing devices connected via Android Debug Bridge (ADB), executing shell commands on the phone, and identifying QRNG-related files. The QRNG chip in the smartphone may operate at a low level within the device, and exploring the file system helps to identify the location of key system files or device drivers that handle QRNG operations. Consequently, this determines how random numbers are generated and where they are stored or accessed.

By locating the appropriate device file or system call, we can directly interact with the QRNG and bypass higher-level APIs when necessary. This allows the development of custom applications that utilize quantum random number generation for cryptographic operations, key generation, and other secure processes. In our analysis, we identified the QRNG library and the associated **`/dev/qrandom`** Unix device files. This was confirmed by discovering an initialization script, **`qrng.rc`**, which, as the name suggests, is related to the quantum number generator. Examination of its content revealed that it configures QRNG access permissions on boot, specifically for diagnostics, as shown in Listing 1.

The name of the QRNG device file was confirmed by filtering the list of files and directories that were marked as irremovable. As shown in Figure 2, by reverse engineering these files (e.g., using IDAPro [35]), which include native libraries, we verified the use of **`/dev/qrandom`**. The same string was identified in some of the pre-installed Android applications on the smartphone.

B. GENERATING BYTES FROM THE EMBEDDED QRNG

We successfully extracted random data from **`/dev/qrandom`**, thereby confirming its functionality and effectiveness. As depicted in Listing 2, this was performed using the **`dd`** command in the shell, **`dd if=/dev/qrandom of=output.bin bs=1024 count=1`**, which reads 1024 bytes of random data from the QRNG device file and writes it to a file named **`output.bin`** in the SD card directory. Building on this, we developed a test



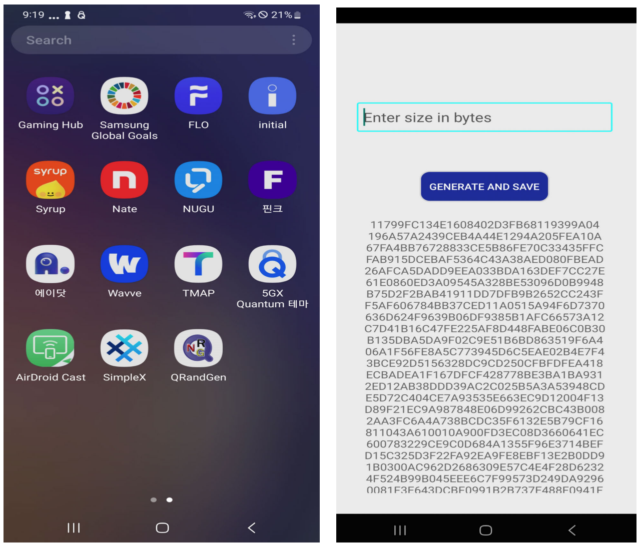
```
int Java_com_android_server_QRNGService_qrngDiag()
{
  int *v0; // r0
  int *v1; // r4
  int v2; // r0
  int v3; // r5
  int v4; // r6
  char *v5; // r0
  int result; // r0
  char *v7; // r0

  v0 = (int *)_errno();
  v1 = v0;
  *v0 = 0;
  v2 = _open_2("/dev/qrandom", 2048);
  if ( v2 <= -1 )
  {
    v7 = strerror(*v1);
    _android_log_print(6, "QRNG#JNI", "Device open failed. : %s\n", v7);
    result = -12;
  }
  else
  {
    v3 = v2;
    *v1 = 0;
    if ( ioctl(v2, 0x5104u, 0) )
    {
      v4 = *v1;
      v5 = strerror(*v1);
      _android_log_print(6, "QRNG#JNI", "ioctl error (%d) : %s\n", -v4, v5);
      close(v3);
      result = -*v1;
    }
    else
    {
      _android_log_print(4, "QRNG#JNI", "Success\n");
      close(v3);
      result = 0;
    }
  }
  return result;
}
```

```
0000040C Java_com_android_server_QRNGService_qrngDiag:12 (140C)
```

```
1530: using guessed type int _android_log_print(_DWORD, _DWORD, const char *, ...);
```

FIGURE 2. Samsung Quantum Phone's QRNG library, which was reverse engineered using IDA Pro, reveals a multithreaded character device `/dev/qrandom`, showcasing the integration of quantum technology.



```
11799FC134E1608402D3FB68119399A04
196A57A2439CEB4A44E1294A205FEA10A
67FA4BB76728833CE5B86FE70C33435FFC
FAB915DCEBAF5364C43A38AED080FBEAD
26AFCA5DADD9EEA033BDA163DEF7CC27E
61E0860ED3A09545A328BE53096D0B9948
B75D2F2BAB41911DD7DFB9B2652CC243F
F5AF606784BB37CED11A0515A94F6D7370
636D624F9639B06DF9385B1AFC66573A12
C7D41B16C47FE225AF8D448FABE06C0B30
B135DBA5DA9F02C9E51B6BD863519F6A4
06A1F56FE8A5C773945D6C5EAE02B4E7F4
3BCE92D5156328DC9CD250CFBFDFEA418
ECBADEA1F167DFCF428778BE3BA1BA931
2ED12AB38DDD39AC2C025B5A3A53948CD
E5D72C404CE7A93535E663EC9D12004F13
D89F21EC9A987848E06D99262CBC43B008
2AA3FC6A4A738BCDC35F6132E5B79CF16
811043A610010A900FD3EC08D3660641EC
600783229CE9C0D684A1355F96E3714BEF
D15C325D3F22FA92EA9FE8EBF13E2B0DD9
1B0300AC962D2686309E57C4E4F28D6232
4F524B99B045EEE6C7F99573D249DA9296
0081F3F643DCBF0991B2B737F488F0941F
```

FIGURE 3. Development of a custom Android application named *QRandGen* that utilizes the on-chip QRNG within the Samsung Quantum phone to generate random numbers.

Android application with a simple user interface to execute this command on the phone's QRNG chip, allowing for easy interaction and validation of the QRNG capabilities (Figure 3).

VI. ENTROPY ANALYSIS OF QUANTUM GALAXY's RANDOM NUMBER GENERATORS

After confirming the name of the QRNG device file (**`/dev/qrandom`**) and the generation of random bytes, the next step is to evaluate the quality of the generated randomness using standardized tests, such as those from the NIST standards [36], [37] and DieHarder [38] test suites.

```
a54x:/sdcard \$ dd if=/dev/qrandom of=output.bin bs=1024 count
     =1

a54x:/sdcard\$ xxd output.bin

00000000: 0b2b 41de a924 1d16 fde2 ce93 eb86 9621   .+A..\$
     .........!
00000010: 519f 18e3 45a2 36bd 261a b175 66de 1742    Q...E
     .6.\&..uf..B
00000020: d3ea b691 b9cf a329 ea4a 5cba 75e3 804d    .......).J
     .u..M
00000030: 8587 6d36 3899 b665 5983 a384 eef4 b0c3    ..m68..eY
     .......
00000040: 6918 1bbe 4252 dc07 9250 96d0 3022 9866    i...BR...P
     ..0.f
00000050: 7a5b 698f e55e 9a6a f5de 1008 2159 1e1c    z\[i..\^.j
     ....!Y..
00000060: 375c defd e5de f081 9c7b 12e3 a982 ba6c
     7.......\\.....l
00000070: 82d7 dd48 5a0e 728f 2c5a 337f 5845 3aad    ...HZ.r.,
     Z3.XE:.
 [.....]
```

LISTING 2. Process of generating random bytes from a Samsung Quantum Galaxy phone using its QRNG chip.

These tests were conducted in a comparative study with other random number generators available on the phone, including **/dev/urandom**, a nonblocking pseudorandom number generator that provides randomness, even with low entropy, and **/dev/random**, a blocking generator that outputs high-entropy randomness directly from the phone operating system entropy pool.

A. COLLECTING TEST SAMPLES FROM QUANTUM GALAXY

Listing 3 depicts a code snippet for a script that we developed to extract random data from multiple device files, including **/dev/urandom**, **/dev/random**, and **/dev/qrandom**, using Android Debug Bridge (ADB) commands. The process begins with the **pull_random_data_from_device()** function, which connects to the Android (i.e. Quantum Galaxy) device, copies the random data from the specified device file using the **dd** command, and stores the output on the SD card of the phone. The random data are then pulled from the Android device to our analysis computer using the **adb pull** command, and the file is subsequently removed from the phone's SD card to maintain cleanliness.

Once the samples were collected for each random number generator, they were analyzed using the NIST SP800-90b [37], NIST SP800-22 [36], and DieHarder test suites [38]. Specifically, NIST SP800-90b assesses the quality of the entropy source in random number generators (RNGs) to guarantee that it performs secure and unpredictable randomization. In contrast, DieHarder provides a more comprehensive and rigorous suite of tests to conduct a more in-depth analysis of RNG performance, whereas NIST SP800-22 concentrates on evaluating the statistical variance of the RNG's output using a set of standard tests.

For our assessment, we collected 100 1 MB samples from each random number generator during our ADB session with the phone. We then wrote a Python script utilizing the **sp800_90b** and **nistrng** packages that implemented the NIST SP800-90b and SP800-22 test suites to perform statistical randomness tests on the binary data. The binary data were read from each sample file and converted into a sequence of 0s and 1s, and the sequence was subsequently converted into a **NumPy** array for further analysis.

```
def pull_random_data_from_device(device_name, output_file_name)
    :
  root_command = f'su -c "dd if={device_name} of=/sdcard/{
      output_file_name} bs=1M count=1"'
  run_adb_command(f'adb shell "{root_command}"')
  run_adb_command(f"adb pull /sdcard/{output_file_name} ./{
      device_name.split('/')[-1]}/")
  run_adb_command(f'adb shell "rm /sdcard/{output_file_name}"')
  time.sleep(2)
```

LISTING 3. Entropy test samples 1 MB in size from each random number generator available on a Samsung Quantum Galaxy smartphone.

B. RUNNING STATISTICAL AND ENTROPY TESTS

The **sp800_90b** package implements several functions that are part of the NIST SP800-90b standard, which assesses the quality of number generators, particularly in cryptographic systems. This standard specifies various tests to evaluate the entropy and unpredictability of binary data. The **h_initial()** function estimates the *initial test*, which measures the randomness of a dataset. The **h_collision()** function runs the *collection test* to detect repeated patterns, whereas the **h_compression()** function uses the *compression test* to determine whether the data can be significantly reduced, suggesting a lack of randomness. The **h_bitstring_markov()** function applies a *Markov test*, which checks for bit-to-bit dependencies. Similarly, the **h_bitstring_lz78y()** function performs the *LZ78Y test*, which is a compression-based analysis of the randomness. Finally, the **h_bitstring_lag_prediction()** function conducts a *lag prediction test* to evaluate whether bits can be predicted from past sequences. These tests are part of the comprehensive framework of NIST SP800-90b for ensuring the robustness of number generators in secure applications.

On the other hand, **run_all_ battery()** function of the **nistrng** package is employed to run the entire suite of randomness tests on this binary sequence, specifically using **SP800_22R1A_BATTERY**, which represents the set of tests defined by NIST SP800-22. Similarly, the **run_all_battery()** function supports multiple statistical tests, such as the frequency test, run test, and linear complexity test, and reports whether each test passes or fails based on the analyzed data. The results are stored in a list of tuples, which includes the test name and result *(Pass/Fail)*.

To run the Dieharder test suite, we installed Brown's tool [38] via the package manager in Ubuntu along with its dependencies. Because we wanted to test the randomness of each sample file, we used the **dieharder -a -g 201 -f /path/to/sample/file.bin** command, where the **-a** option runs all the available tests and **-g 201** specifies that Dieharder should read from a file. Dieharder outputs a table with results for each test, including a p-value and an assessment of whether the data passed, was weak, or failed the randomness test*(Pass/Weak/Fail)*.

TABLE 1. NIST SP800-90b entropy tests for samsung quantum galaxy RNGs.

Test	/dev/urandom	/dev/random	/dev/qrandom
Initial Entropy	7.138 (pass)	7.124 (pass)	7.140 (pass)
Collision Test	1.000 (pass)	1.000 (pass)	1.000 (pass)
Compress Test	0.854 (pass)	0.852 (pass)	0.851 (pass)
Markov Test	0.999 (pass)	0.999 (pass)	0.990 (pass)
LZ78Y Test	0.997 (pass)	0.998 (pass)	0.988 (pass)
Lag Pred. Test	0.998 (pass)	0.999 (pass)	0.999 (pass)

TABLE 2. Dieharder test results (pass/weak/fail) for samsung quantum galaxy RNGs.

Test	/dev/urandom	/dev/random	/dev/qrandom
3dsphere	98/2/0	98/2/0	98/2/0
birthdays	88/11/1	89/10/1	87/12/1
count_1s	62/20/18	52/30/18	20/21/59
dna	12/43/45	23/36/41	2/10/88
parking	92/8/0	92/8/0	87/12/1
rank_6x8	5/22/73	8/16/76	7/22/71
runs	87/53/60	64/69/67	77/58/65
sums	84/16/0	89/11/0	90/10/0
bitdist	500/175/525	519/151/530	12/38/1150
kstest	40/35/25	33/37/30	1/5/94
min_dist	0/34/366	0/31/369	1/35/364
permute	66/76/258	68/85/247	49/61/290
monobit	18/31/51	25/25/50	0/0/100
sts_runs	21/24/55	23/26/51	0/0/100

C. COMPLIANCE AND COMPARATIVE RESULTS

The NIST SP 800-90b test results presented in Table 1 show that all three number generators **/dev/urandom**, **/dev/random**, and **/dev/qrandom** produce highly random outputs, but there are some subtle differences. Device **/dev/qrandom** has the highest initial entropy value of 7.1396, slightly outperforming **/dev/urandom** (7.1384) and **/dev/random** (7.1237), indicating that it generates data with slightly more unpredictability. However, in tests such as the Markov test, the **/dev/qrandom** scores were slightly lower, at 0.99, compared to the near-perfect 0.9994 score obtained for both **/dev/urandom** and **/dev/random**, suggesting that it might exhibit minor patterns in short-term sequences.

The compression test scores were also similar, with **/dev/qrandom** achieving a score of 0.8511, which is marginally lower than those of **/dev/urandom** (0.8535) and **/dev/random** (0.8522), indicating that the data from **/dev/qrandom** are slightly more compressible. In the *LZ78Y test*, **/dev/qrandom** achieved a score of 0.988, while **/dev/urandom** and **/dev/random** performed better, achieving scores of 0.9966 and 0.9977, respectively, again indicating that **/dev/qrandom** might have slightly less randomness in terms of the data structure. Despite these small differences, all three generators pass key tests such as the collision test (scores of 1 across the board) and the lag prediction test (with scores of 0.9987 for both **/dev/random** and **/dev/qrandom**), proving their robustness for cryptographic use.

Table 2 presents the entropy test results for number generators of the Samsung Quantum Galaxy series using the Dieharder standard. The results reveal mixed performance across three sources: **/dev/urandom**, **/dev/random**, and **/dev/qrandom**. Some tests, such as **diehard_3dsphere** and **diehard_parking_lot**, show consistently high pass rates across all generators, indicating reliable performance in these areas. However, tests such as **diehard_birthdays** exhibit minor variability, with all three sources showing similar results but with some weak and failed outcomes. A significant area of concern is the **diehard_dna** test, in which **/dev/qrandom** displays a notably high failure rate, suggesting potential weaknesses in handling DNA-like sequence patterns. Bitstring-related tests, including **diehard_count_1s_str** and **rgb_bitdist**, highlight a stark difference in performance, with **/dev/qrandom** showing significantly higher failure rates (e.g., 12/38/1150 for **rgb_bitdist**), indicating challenges in maintaining randomness in the bit distribution. Moreover, **/dev/qrandom** completely fails the **sts_monobit** and **sts_runs** tests (0/0/100), suggesting issues with the uniformity and randomness of the sequences. Meanwhile, tests such as **diehard_rank**_6 $\times$ 8 and **rgb_minimum_distance** showed high failure rates across all generators, indicating common challenges in these assessments. Overall, while **/dev/urandom** and **/dev/random** perform better across various tests, the entropy quality of **/dev/qrandom** appears less robust, particularly when handling certain randomness evaluations. Furthermore, several tests that all generators failed, such as the extended "dab" tests (**0dab_filltree**, **0dab_filltree2**, **dab_dct**) and the original Dieharder tests, including **diehard_operm5**, **diehard_opso**, **diehard_oqso**, **diehard_count_1s_byt**, and **diehard_craps**, are omitted from the table, reflecting their inability to pass these more rigorous evaluations.

Table 3 compares the performances of three random number generators (**/dev/urandom**, **/dev/random**, and **/dev/qrandom**) across various randomness tests using the NIST SP800-22 standard. The results indicate the ratio of the number of test passes to the number of failures. For the binary matrix rank test, all three generators performed similarly, with **/dev/qrandom** performing slightly better, with a score of 99 percent, indicating strong rank characteristics among them. In the FrequencyWithinBlock test, a significant discrepancy is evident, as **/dev/qrandom** achieves only 13 passes compared to its 87 failures, whereas **/dev/urandom** and **/dev/random** have high pass rates of 98 and 97 percent, respectively, suggesting potential issues with **/dev/qrandom** in maintaining a uniform frequency distribution. The longest run of ones test results show good

performance for all three generators, with pass rates above 97 percent, indicating that their runs are generally in line with the randomness expectations. However, in the Monobit test, **`/dev/qrandom`** struggles, passing only 2 out of 100 tests, unlike **`/dev/urandom`** and **`/dev/random`**, which demonstrate nearly perfect performance and achieve pass rates of 99 and 100%, respectively. In the nonoverlapping template matching test, all generators show comparable performance, although not ideal, with **`/dev/qrandom`** trailing slightly and achieving a rate of 68 percent compared to the other two generators, which achieve a rate of 70 percent.
The random excursion variant results are also similar, with **`/dev/random`** slightly leading with a 63 percent pass rate, followed by **`/dev/urandom`** (60 percent) and **`/dev/qrandom`** (58 percent), indicating moderate difficulties for all generators in this area. A stark contrast appears in the runs test, where **`/dev/urandom`** and **`/dev/random`** perform excellently, with pass rates of 99 and 100, respectively, while **`/dev/qrandom`** fails in all of its seven tests. Notably, all three generators failed the approximate entropy, cumulative sums, discrete Fourier transform, linear complexity, Maurer's universal, overlapping template matching, random excursion and serial tests, indicating challenges across these more complex random measures. Overall, the results suggest that while **`/dev/urandom`** and **`/dev/random`** maintain consistent performance in most tests, **`/dev/qrandom`** exhibits weaknesses, particularly in maintaining balance and uniformity in output, warranting further investigation of its randomness properties.

TABLE 3. NIST test results (pass/fail) for samsung quantum galaxy RNGs using the NIST SP800-22 standard (pass/fail).

Test	/dev/urandom	/dev/random	/dev/qrandom
Binary Matrix	97/3	98/2	99/1
Freq. Block	98/2	97/3	13/87
Longest Run	99/1	97/3	98/2
Monobit	99/1	100/0	2/98
Non-Overlap Template	70/30	70/30	68/32
Random Excur. Variant	60/40	63/37	58/42
Runs	99/1	100/0	0/7

The vendor's confirmation that the IDQ205C2 QRNG chip embedded in the Samsung Quantum Galaxy passes the NIST 800-90b test suite, though not the DieHarder or NIST 800-22 test suites, aligns with our own entropy and experimental findings. Our results, presented in the tables, demonstrate that while the chip generates high-quality entropy sufficient for NIST 800-90b compliance, it lacks the robustness to pass broader statistical randomness tests. This limitation stems from the absence of post-processing, which could be integrated into future models to enhance their performance in more diverse tests. For reference, the technical validation of the IDQ205C2's (i.e. Quantum Galaxy QRNG chip) entropy generation was certified via NIST 900-80b, and the certificate can be viewed [39].

It is crucial that any postprocessing improvements are hardware-based, as software postprocessing could be vulnerable to reverse engineering, making it easier to bypass. By embedding postprocessing within the QRNG hardware, future models can ensure stronger tamper-resistant randomness. This approach would allow the QRNG to meet a wider range of cryptographic standards, providing both high security and reliability in critical applications.

VII. HUNTING AND REVERSE ENGINEERING OF QRNG-ENABLED APPS

After confirming the Unix device file for the quantum random number generator (QRNG) on the smartphone, learning how to extract binary random data from it, and validating the quality of the generated bytes using entropy tests, the next step was to identify applications that utilize the QRNG capabilities on the smartphone. To accomplish this, we conducted online research on the vendor's website, app stores, and other Internet sources to locate the source code or executables for these QRNG-enabled apps. In addition, we scripted a program to directly search for these applications on a mobile device. For this purpose, the phone was rooted to obtain higher privileges in the file system, thereby facilitating a comprehensive search for pre-installed apps. After locating references to the QRNG Unix device file **(`/dev/qrandom`)**, we reverse engineered the apps to analyze their use of the QRNG in the application logic.

A. PRECURSORY ONLINE SEARCH

A precursory analysis of all pre-installed drawer apps was conducted to understand how the phone's quantum capabilities were leveraged. To commence the search for QRNG-enabled apps, we had to circumvent regional restrictions to access specific applications. Furthermore, we also had to register for a Samsung account to obtain the required applications and themes. We started our online search by visiting Samsung's official website and developer portal to find documentation, SDKs, and APIs relevant to the QRNG capabilities of the Quantum Galaxy series. Following this, we conducted a search of the Google Play Store and Galaxy Store for pertinent applications using specific keywords. We then investigated open-source platforms such as GitHub for projects that reference the QRNG or **`/dev/qrandom`** and interact with developer communities on forums such as Stack Overflow and Reddit. We then searched for QRNG use by examining the APKMirror and APKPure websites for pertinent APKs. Finally, we reviewed academic papers and security blogs, including XDA Developers, to identify additional QRNG applications and use cases.

B. ROOTING PROCESS

After completing our online search for QRNG-enabled apps, the next step required the rooting of Samsung Galaxy to gain elevated privileges for in-depth system access and analysis of QRNG usage. We needed to root the phone to gain system-level access to the QRNG functionalities and pre-installed apps. This process involves unlocking the bootloader, a necessary step that removes the manufacturer's security restrictions but also wipes the phone's SD card and application data. Nevertheless, this step enabled us to install custom firmware and gain deeper access to phones.

To achieve this, we used a tool on Android called Magisk [40], which allowed for a *systemless* root and provided the capability to modify the system without permanently changing critical system files. This was ideal for our research, as it allowed us to explore the phone's internals without disrupting normal operation. After patching the phone's boot image with Magisk, we used Odin [41], a Samsung flashing tool, to install the rooted image. This step carries risks because flashing system files can potentially brick the device if an error occurs; however, luckily, the process was completed without any issues. Without root access, we could not fully access the file system or inspect QRNG-related files such as **`/dev/qrandom`** and related apps to see how they used QRNGs. Rooting opened the opportunity to explore the phone's cryptographic and security functionalities, which was essential for our research.

C. SEARCHING FOR QRNG-ENABLED APPS

After successfully rooting one of the phones, we developed a Python script to scan and identify applications on the phone's file system via embedded QRNG capabilities. To achieve this, we implemented a custom heuristic method that searches for downloads and analyzes APK files from a Quantum Galaxy device using ADB commands. The script downloads the apps to a local machine (analysis computer), which then searches for specific strings within the downloaded APK files using Androguard [42], a powerful open-source tool for reverse engineering Android applications. Androguard allows us to decompile and analyze APK files, making it useful for identifying QRNG usage. To manage permissions and storage constraints on the device, the script temporarily copies the apps to the phone's **`/sdcard`**, downloads them to the local machine, and then deletes the temporary copies from **`/sdcard`**. Additionally, for easier readability and organization, the downloaded apps (i.e., APK files) were automatically renamed to their respective package names.

Listing 4 depicts the **`download_apk()`** function, which facilitates the process of copying an APK file from an Android device to a local machine using the ADB and root privileges. As shown, it begins by accepting the **`source_path`** of the APK on the device and a **`local_directory`** on the local machine where the APK will be saved. First, the function copies the APK from its original location to the device's **`/sdcard`** directory using root privileges, as this location provides easier access for file transfers.

Once copied, the APK is pulled from the **`/sdcard`** to the local machine using the **`adb pull`** command. If successful, the APK is saved to the specified directory, and the function attempts to extract its package name to rename the

```python
#Download the APK from the device to the local machine
def download_apk(source_path, local_directory):

    # Step 1: Copying the APK file to the /sdcard directory
      using root privileges
    temp_filename = "{os.path.basename(source_path)}"
    temp_path = f"/sdcard/{temp_filename}"
    copy_command = f"cp {source_path} {temp_path}"
    copy_output = adb_command(copy_command)
    if "error" in copy_output.lower():
        print(f"Error copying file '{source_path}' to /sdcard:
      {copy_output}")
        return

    # Step 2: Pulling the APK from the /sdcard directory to the
      local machine
    pull_command = f"adb pull {temp_path} {local_directory}"
    result = subprocess.run(pull_command.split(),
      capture_output=True, text=True)
    if result.returncode != 0:
        print(f"Error pulling file '{temp_path}': {result.
      stderr}")
    else:
        print(f"Pulled and saved as {temp_filename}")

    # Step 3: Extracting the package name, and renaming the APK
    package_name = get_package_name(local_directory)
    if package_name:
        renamed_path = os.path.join(os.path.dirname(
      local_directory), f"{package_name}.apk")
        os.rename(local_directory, renamed_path)
        print(f"Renamed APK to {renamed_path}")
    else:
        print(f"Failed to extract package name for {
      package_name}")

    # Step 4: Deleting the APK from the /sdcard directory
    delete_command = f"rm {temp_path}"
    delete_output = adb_command(delete_command)
    if "error" in delete_output.lower():
        print(f"Error deleting file '{temp_path}' from /sdcard:
       {delete_output}")
    else:
        print(f"Deleted temporary file {temp_path}")
```

LISTING 4. Custom `download_apk()` function for copying an APK from the mobile device to a local machine using root access and ADB with automatic renaming and cleanup.

file for clearer identification. Finally, the function deletes the temporary APK file from the **`/sdcard`** directory on the mobile device to maintain cleanliness. This function automates the APK extraction process, making it especially useful for app analysis or reverse engineering and ensuring proper file organization and access control through root privileges.

As shown in Listing 5, the **`search_apk_local()`** function was designed to search for a specific string within the DEX files of a locally saved app. It uses two parameters, **`apk_path`**, which is the path to the APK file, and **`search_string`**, which is the string to search for in the APK's DEX files. The function first uses the **`apk.APK`** class (from the AndroGuard library) to open the APK and extract its DEX files. It then iterates over each DEX file, decodes them while ignoring errors, and searches for the specified string in the decoded content. If the string is found, it prints a success message and returns **`True`**; otherwise, it returns **`False`** after completing the search. The function also includes error handling for cases where the APK file is missing and prints an error message if a **`FileNotFoundError`** message appears. This function is useful for reverse engineering or analyzing APK files to find specific code patterns or libraries, such as QRNG-related functions or API calls within an app's executable content.

```
 def search_apk_local(apk_path, search_string):
try:
  a = apk.APK(apk_path)
  dex_files = a.get_all_dex()
  found = False
  for dex_file in dex_files:
    if search_string in
       dex_file.decode(errors='ignore'):
        print(f"String found in APK")
        found = True
```

LISTING 5. **Custom function `search_apk_local()` to search for a specified string within the DEX files of a locally saved APK, which is useful for reverse engineering and application analysis.**

After executing the scripts, we successfully retrieved all APKs stored on the phone's file system, including pre-installed applications developed by Samsung or their partner SK Telecom, as well as applications obtained from Google Play. Among the 546 applications that were extracted, seven used the QRNG functionality, as shown in the Listing below.

```
$ md5sum *
852a4eadf3b1003861c064dbbd42eace  com_elevenst.apk
887c7a45bdb5d2ee2c2a7eb6755a5a56  com_elevenst_skpay.apk
f6d8d852e74aa648b9a6d403e374c450  com_skt_t_smart_charge.apk
4df4ac40ad721ff882f6b270b52a6461  com_sktelecom_myinitial.apk
5af1e196a489a1e8f66dd4ad36ac6369  com_sktelecom_tid.apk
928b8fdd947c7c8f58180fa4acfb83cb  com_tms.apk
```

LISTING 6. **A list of QRNG-enabled apps extracted from Samsung Galaxy smartphones and their respective MD5 hashes.**

After conducting an exhaustive file system search on the phone to identify QRNG-enabled apps, we used an automated text search to detect the presence of the Unix device file `/dev/qrandom` in the string table of the DEX files within each APK. The goal is to understand the context in which the QRNG is utilized within these apps. After detecting the presence of the device file string, we reverse engineered each app found in the above list to read the code that referenced the device file.

To reverse engineer Android applications, we used Jadx, a comprehensive decompiler that converts APK files into readable source code, making it easier to inspect QRNG-related applications. Fortunately, we did not need to analyze the native binaries employed by APKs. However, if it was necessary to examine the phone's QRNG-related binaries, we would have used IDA Pro [35], a powerful disassembler and debugger that allows for this in-depth analysis.

Listings 7 and 8 demonstrate the different uses of the QRNG in applications. In Listing 7, the code snippet checks for the presence of the Unix file device `/dev/qrandom`, primarily for fingerprinting purposes. If a device is detected, the function returns 'true' and includes this result as an HTTP request parameter to be sent to a remote server. By contrast, Listing 8 presents a more advanced application of the QRNG, where a random seed is generated to create public and private key pairs within a `Runnable` interface. However, the implementation is incomplete. Although key pairs are generated, they are neither stored nor utilized after their creation.

```
public static boolean m15567a() {
    boolean z10;
    if (!f14444b) {
        File file = new File("/dev/qrandom");
        f14443a = file;
        if (file.exists()) {
            Log.d("QRNG-LIB", "Successfully found the
 relevant node: /dev/qrandom");
            z10 = true;
        } else {
            Log.e("QRNG-LIB", "Can't find the relevant node
 : /dev/qrandom");
            z10 = false;
        }
        f14444b = z10;
    }
    return f14444b;
}
```

LISTING 7. **Android app using a simple Boolean if statement to determine whether the QRNG Unix device is present.**

As depicted in the code in Listing 8, although the `keyPairGenerator.generateKeyPair()` method is invoked, the resulting `KeyPair` object is not assigned to a variable or passed for further use and is devoid of the usefulness of the `KeyGeneratorRunnable` class.

The details of each QRNG-enabled app are provided in Table 4. As shown, these apps were pre-installed on the Samsung Quantum Galaxy as part of SK Telecom's partnership subscription. They utilized the phone's embedded QRNG capabilities to protect services such as transactions, identity verification, and telecom services. Given its high entropy, the QRNG is used mostly to seed public and private key generation to set up an authentication service when a user is registered for the first time or as a nonce to prevent replay attacks during an authentication process.

VIII. QUANTUM-BASED INSTANT MESSAGING AND VoIP APP

Investigating how existing apps utilize the QRNG functionality on Samsung Quantum Galaxy smartphones and then reverse engineering those apps, reveal that the embedded QRNG is not used meaningfully. While many applications refer to the Unix device file `/dev/qrandom`, the QRNG is often underutilized or not fully implemented. For example, in one case, a random seed is generated using the QRNG but is discarded without being used. In another case, the app generates a QRNG seed to initialize a public–private key pair, but the key pair is neither stored nor used for cryptographic operations, that is, the key pair object is not assigned to a variable or passed for further use.

A. NO KILLER APPLICATIONS

Frequently, the QRNG device file is referenced in an unreachable code or in functions that invoke a stub after

TABLE 4. QRNG-enabled application package list and descriptions.

Package	Icon	Description	QRNG Usage
com.elevenst.*		11th shopping platform for browsing, buying, order tracking, and notifications.	QRNG seed generation for biometric authorization public key pairs.
com.skt.t_smart_charge		SK Telecom EV charging management app: station locations, reservations, monitoring, and payments.	QRNG implementation check and placeholder for future service.
com.sktelecom.*		SK Telecom services: telecom, payments, media, smart home, and security.	QRNG for TID login, user seed generation, and authentication.
com.tms		T membership app for SK Telecom discounts, savings, and subscription benefits.	QRNG initialization for biometric authorization key pairs.

```
    if (C3965a.m6100a()) {
        FileInputStream fileInputStream = new FileInputStream("
      /dev/qrandom");
        BufferedInputStream bufferedInputStream = new
      BufferedInputStream(fileInputStream, 32);
        bufferedInputStream.read(bArr);
        bufferedInputStream.close();
        fileInputStream.close();
        secureRandom = new SecureRandom(bArr);
        new KeyGeneratorRunnable(c5316e2, secureRandom,
      runnableC5314c).start();
    } else {
        throw new IOException("QRNG device node is not enabled.
      ");
    }
```

LISTING 8. QRNG used to seed the generation of a public-private cryptographic key pair for the Android keystore.

random bytes are read from the chip. Sometimes, mobile apps simply confirm the existence of a QRNG without invoking any reads, after which they transfer the result to a remote server as a fingerprinting parameter. There may be several reasons why embedded QRNG chips have not been fully utilized by both pre-installed and third-party applications. One potential explanation is that developers may be unaware of their functionality, or that comprehensive documentation is not available. Regrettably, developers have yet to produce killer applications that fully harness their capabilities.

As there is currently no "killer app" that truly utilizes the QRNG for smartphones, we suggest the creation of a mobile app that achieves quantum-based instant messaging and VoIP application. This application leverages the QRNG to produce cryptographically secure keys for end-to-end encryption, thereby guaranteeing that user communication is highly secure. The application offers an unparalleled level of security for both voice and messaging communication by incorporating the QRNG into the key exchange and encryption processes, surpassing the capabilities of conventional pseudorandom number generators. The merit of developing such an application is that it has the potential to showcase the practical application of quantum cryptography in everyday communications, thereby addressing the current gap and exhibiting the real-world potential of the QRNG on smartphones.

B. MOTIVATION FOR THE USE CASE

The motivation for developing quantum-based instant messaging and VoIP applications arises from the significant privacy and security risks associated with existing social media and messaging platforms, which are often unsuitable for managing sensitive data. While these platforms are effective for public engagement and content distribution, they frequently lack the robust confidentiality and encryption required for secure internal communication within organizations.

Many social media applications collect substantial user metadata, such as phone numbers, message timestamps, and communication patterns, on their servers, and they may also retain undelivered messages. Despite security and privacy assurances by various social media platforms, these data usually remain vulnerable to breaches and may be handed over to authorities under legal pressure (e.g. [43]). These concerns underscore the need for a more secure alternative, which could be provided by anonymous instant messaging and VoIP apps utilizing quantum random number generator (QRNG) technology and a post-quantum cipher suite for enhanced cryptographic security, along with a private relay server to prevent third-party surveillance.

Our proposed solution addresses critical security issues that plague existing social media and messaging apps, offering QRNG-based post-quantum instant messaging and VoIP mobile apps to overcome these challenges.

1) *Data sovereignty*: Many popular apps store and transmit data through servers located in foreign jurisdictions, rendering user information susceptible to government requests, data breaches, and surveillance by third parties. WhatsApp, for example, collects extensive metadata, such as phone numbers, message timestamps, and communication patterns, on its centralized servers, which may be handed over to authorities under pressure. In contrast, our solution ensures that data are securely stored and transmitted within local jurisdictions, guaranteeing that sensitive information is protected and fully compliant with local laws. This is particularly vital for politically sensitive

users or organizations with strict data sovereignty requirements.

2) *Future-proof security*: While current encryption standards in apps, such as Telegram, provide some level of security, they remain vulnerable to the upcoming threats posed by quantum computing. Quantum computers can break the cryptographic algorithms on which these platforms rely, exposing user communications to interception. Our solution incorporates post-quantum cryptography (PQC), which is specifically designed to resist quantum computing attacks, ensuring that communications remain secure not only today but also in the future as quantum computing becomes more widespread.
3) *True randomness*: Existing messaging platforms generate encryption keys using traditional number generators, which may not provide the highest level of randomness, potentially weakening encryption. Our application uses quantum random number generators (QRNGs) to create encryption keys with true randomness, significantly enhancing the strength and unpredictability of the keys. This renders our encryption far more resistant to brute-force attacks and other cryptographic exploits.
4) *End-to-end encryption*: Although apps such as WhatsApp and Telegram offer end-to-end encryption, they still collect metadata and may not fully prevent third-party surveillance. Our app provides complete end-to-end encryption not only for messaging but also for secure document sharing and VoIP communications. This ensures that even if communication is intercepted, it remains inaccessible to third parties. This is especially crucial for politically exposed persons (PEPs) and organizations handling sensitive data, as it prevents both metadata collection and content interception.

C. USE CASE DESCRIPTION

Specifically, our use case involves integrating a Samsung Quantum Galaxy smartphone equipped with ID Quantique's QRNG chip, with a SimpleX Chat mobile application [44] for secure instant messaging and VoIP calls. Phones generate cryptographic keys using true quantum randomness, which strengthens encryption, especially when combined with post-quantum cryptography, to defend against future quantum computing threats. Organizations must deploy a private relay server to manage encrypted communications and ensure that data stay off public servers.

The server should be configured to handle secure key exchanges via the secure messaging protocol (SMP) and to support secure file transfers through the extended file transfer protocol (XFTP), both encrypted with QRNG-generated keys. In addition, the TOR [45] network is used to anonymize the communications. APIs retrieve cryptographic keys from QRNG chips for use in SimpleX Chat encryption modules and secure all messages and calls in an end-to-end manner. The system is further fortified with post-quantum encryption algorithms, such as SPHINCS+ [17] and NTRU-Encrypt [46], ensuring long-term security against quantum computing attacks, thus proofing the organization's future communication infrastructure.

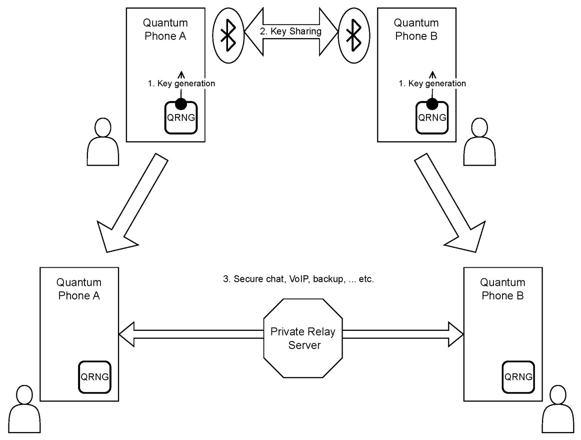


FIGURE 4. Comprehensive solution for mobile communication security and privacy, enabling true random generation of cryptographic keys and secure initialization of post-quantum cryptography with end-to-end encryption and private relay servers.

Figure 4 shows a comprehensive security solution for mobile users by integrating the SimpleX Chat platform with Samsung Galaxy phones. The embedded QRNG chip generates a truly random cryptographic key, which is essential for secure communication. This process is initiated by an integration program running on the phone's operating system. The QRNG chip is accessed via a multithreaded device named `/dev/qrandom` to acquire random bytes for key generation. Each phone involved in the communication performed this initial step independently. In the subsequent step, a cryptographic key is shared with the relevant parties. This key exchange is facilitated through an out-of-band channel to prevent interception during transmission on the main channel and ensure secure key exchange using a separate, trusted channel. Users may physically meet to scan a QR code on each other's phones, or, alternatively, keys can be exchanged via proximity wireless transfers, such as Bluetooth low energy (BLE) [47], infrared, or similar technologies. This secure key exchange ensures that only the intended parties possess the cryptographic keys that are necessary for communication.

D. CONFIGURING A PRIVATE RELAY SERVER

The final step involves establishing a remote communication channel after secure key exchange. As users move to different locations, their phones use a preconfigured relay server to create a secure communication channel. As an intermediary, this server encrypts and decrypts all messages using previously exchanged cryptographic keys. The integration of QRNG technology, secure key exchange methods, and

```
$ sudo smp-server start

SMP server v6.0.3
Fingerprint: ISdc4X3rxl3m46QGr90IIi4__bgoc-_S6Q40v_w9YPE=
Server address: smp://ISdc4X3rxl3m46QGr90IIi4__bgoc-
      _S6Q40v_w9YPE=@privateserver.dyndns.org

Store log: /var/opt/simplex/smp-server-store.log
Listening on port 5223 (TLS)...
expiring messages after 21 days
not expiring inactive clients
creating new queues allowed

[INFO 2024-09-03 09:15:16 +0300 src/Simplex/Messaging/Server/
      Env/STM.hs:227] restoring queues from file /var/opt/
      simplex/smp-server-store.log
[INFO 2024-09-03 09:15:16 +0300 src/Simplex/Messaging/Server.hs
      :1554] restoring messages from file /var/opt/simplex/smp-
      server-messages.log
[INFO 2024-09-03 09:15:16 +0300 src/Simplex/Messaging/Server.hs
      :1564] messages restored
[INFO 2024-09-03 09:15:16 +0300 src/Simplex/Messaging/Server.hs
      :1598] restoring server stats from file /var/opt/simplex/
      smp-server-stats.log
[INFO 2024-09-03 09:15:16 +0300 src/Simplex/Messaging/Server.hs
      :1606] server stats restored

server stats log enabled: /var/opt/simplex/smp-server-stats.
      daily.log
[INFO 2024-09-03 09:15:16 +0300 src/Simplex/Messaging/Transport
      /Server.hs:173] binding to [::]:5223
```

LISTING 9. Setting up and managing a private relay server for the SimpleX Chat mobile app by routing messages through a dedicated server to enhance user privacy and data security.

reliable relay servers provides a robust solution for secure mobile communication.

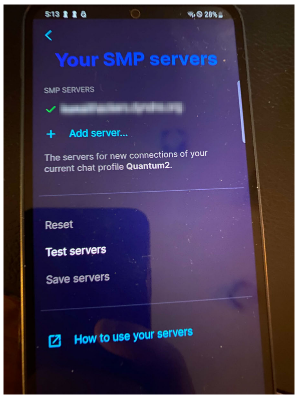
Listing 9 illustrates a private relay server run for the SimpleX Chat mobile app. As illustrated, the setup involves configuring the server to manage encrypted communications and secure file transfers. After selecting a robust server, the SimpleX Chat server software was installed and configured to authenticate connections from authorized clients, i.e. Quantum Galaxy phones, using QRNG-generated cryptographic keys.

For secure messaging, the SMP is enabled, and encryption algorithms, preferably post-quantum cryptographic algorithms such as SPHINCS+ [17] or NTRUEncrypt [46], are configured to ensure end-to-end encryption for all message exchanges. Similarly, XFTP was set up to handle secure file transfers by encrypting files with QRNG-generated keys before transmission. The server is also configured to manage secure storage for files during transfers, ensuring that all the data remain protected. Optionally, the TOR network [45] can be integrated to anonymize the server by adding an additional layer of privacy by routing communication through various layers of public proxies.

Once the system is configured, testing is conducted to ensure that the server securely handles both the messaging and file transfers. Finally, the SimpleX Chat application was deployed on client devices, and the server was maintained to ensure continued security and performance.

Figure 5 exhibits the configuration of a private relay server on the SimpleX Chat mobile application. The relay server acts as an intermediary for encrypted communication, ensuring secure message routing between clients. It demonstrates the server's role in anonymizing communication and maintaining user privacy by encrypting all transferred data using QRNG-generated keys.

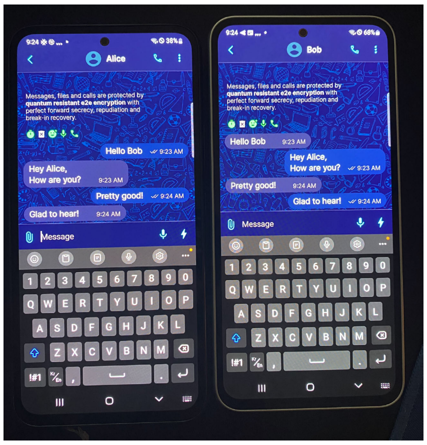
Figure 6 illustrates the initiation of a secure instant messaging session between two users, Alice and Bob, with the SimpleX Chat application on Samsung Quantum Galaxy devices. As shown, the QRNG-based key generation process enabled the generation of safeguarded cryptographic keys for establishing an end-to-end encryption connection that guarantees the confidentiality and integrity of exchanged messages.



[redacted]

The servers for new connections of your current chat profile **Quantum2**.

FIGURE 5. Configuring a private relay server on the SimpleX chat end-user mobile application.



Messages, files and calls are protected by **quantum resistant e2e encryption** with perfect forward secrecy, repudiation and break-in recovery.

Hello Bob 9:23 AM
Hey Alice, How are you? 9:23 AM
Pretty good! 9:24 AM
Glad to hear! 9:24 AM

FIGURE 6. Initializing a private instant messaging chat between alice and bob using the SimpleX chat app on samsung quantum galaxy smartphones.

E. IMPLEMENTING THE QRNG-SEEDED CRYPTOGRAPHIC KEY

Listings 10 and 11 demonstrate how to leverage the quantum random number generator (QRNG) functionality on Samsung Quantum Galaxy smartphones to generate cryptographic keys for SimpleX Chat. The `QRNGKeyGeneratorSpi` class is a custom implementation of the `KeyGeneratorSpi` class

```
class QRNGKeyGeneratorSpi : KeyGeneratorSpi() {
 override fun engineGenerateKey(): SecretKey {
   val keyBytes = ByteArray(keySize / 8)
     try {
    FileInputStream("/dev/qrandom").use { fis ->
        if (fis.read(keyBytes) != keyBytes.size) {
          throw IOException("Failed to read enough random bytes
      ")
        }
         }
   } catch (e: IOException) {
     throw RuntimeException("Error reading from /dev/qrandom", e
      )
   }
   return SecretKeySpec(keyBytes, algorithm)
   }
}
```

LISTING 10. Defining a wrapper class for the service provider interface (SPI) to enable secure cryptographic key generation with the QRNG. The QRNG key generation is integrated into the SimpleX Chat mobile app.

```
private fun createSecretKey(alias: String): SecretKey? {
if (keyStore.containsAlias(alias))
  return getSecretKey(alias)
  // Register the custom provider
  val provider = QRNGProvider()
  Security.addProvider(provider)
  val keyGenerator: KeyGenerator = KeyGenerator.getInstance(
      KEY_ALGORITHM, provider.name)
  keyGenerator.init(
    KeyGenParameterSpec.Builder(alias, KeyProperties.
      PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT)
       .setBlockModes(BLOCK_MODE)
       .setKeySize(KEYSIZE)
  .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
       .build()
  )
  return keyGenerator.generateKey()
}
```

LISTING 11. Function used to create a secret key using a custom QRNG provider for encryption and decryption.

and reads random bytes directly from the **`/dev/qrandom`** device file. This ensures that the generated key is based on high-entropy quantum-generated random numbers, which are much more secure than those of traditional pseudo-random number generators. **`QRNGProvider`** registers **`QRNGKeyGeneratorSpi`** for specific algorithms, such as AES, allowing the custom key generator to be used in cryptographic operations through the Java cryptography architecture (JCA).

The **`createSecretKey()`** method, located in the **`Cryptor`** Kotlin class, defines a function that creates a secret key by using a custom QRNG-based provider. If a key already exists for a given alias in the Android keystore, it retrieves that key; otherwise, it registers **`QRNGProvider`**, which integrates the QRNG into the key generation process. This ensures that any generated key uses the true quantum randomness. Together, these components enable secure key generation by incorporating the QRNG into the cryptographic workflow, ensuring that all cryptographic operations, such as encryption and decryption, are based on truly random keys. This approach significantly enhances the security of SimpleX Chat, where privacy and secure communication are essential. By leveraging the QRNG, the generated keys are resistant to cryptographic attacks, including those that may arise from quantum computing advancements.

IX. DISCUSSION

This section discusses the study's findings and analyzes the results in relation to the three proposed hypotheses. The results highlight both the potential and limitations of the QRNG chip integrated within the Samsung Quantum Galaxy smartphone, along with insights into its practical application in mobile security solutions.

A. HYPOTHESIS 1 RESULTS

Our investigation confirmed that the QRNG is implemented as a dedicated device file (**`/dev/qrandom`**) that is accessible at the operating system level. This clear system-level integration simplifies programmatic access and provides developers with a straightforward method to leverage true randomness in their applications. The presence of **`/dev/qrandom`** was further validated through the **`qrng.rc`** system configuration file, which assigns appropriate permissions and ensures the device file is initialized at boot.

Despite this accessible interface, the lack of comprehensive documentation appears to have hindered meaningful adoption. Developers may be unaware of the QRNG's presence, its capabilities, or the correct methods for integrating it into security-critical functions.

Conclusion: The QRNG was integrated as a device file is *confirmed*, but improved guidance and resources for developers are necessary to encourage effective utilization.

B. HYPOTHESIS 2 RESULTS

The entropy testing results showed that the QRNG demonstrated strong performance in accordance with NIST SP800-90b standards, confirming its ability to produce high-entropy random data. This compliance indicates that the QRNG effectively meets security requirements for cryptographic randomness when evaluated against NIST's entropy-focused criteria.

However, the QRNG produced inconsistent results in broader randomness evaluations such as NIST SP800-22 and DieHarder. The QRNG struggled in key statistical tests, including the monobit test and run test, which revealed patterns that reduced its unpredictability. These weaknesses suggest that while the QRNG can generate strong entropy, its current implementation may lack sufficient postprocessing mechanisms to refine output quality. Without improved postprocessing, the QRNG performance may fall short in applications that demand consistently robust randomness.

Conclusion: The second hypothesis is *partially rejected*. Although the QRNG chip produces strong entropy under specific conditions, its broader performance is inconsistent. Postprocessing enhancements are recommended to improve reliability in diverse randomness tests.

C. HYPOTHESIS 3 RESULTS

Our exploration of existing applications revealed that the QRNG is largely underutilized. Although several APKs contained references to **`/dev/qrandom`**, these references were frequently incomplete, inactive, or used only for device fingerprinting rather than secure key generation.

No meaningful integration of the QRNG in security-critical functions such as encryption or secure communications was identified.

The limited utilization reflects two key issues:

1) *Lack of awareness*: Developers may be unaware of the QRNG's presence or its potential benefits.
2) *Limited documentation*: Insufficient guidance on effective QRNG integration likely contributed to incomplete implementations or abandoned development efforts.

To address this gap, we developed a secure messaging and VoIP application that combines QRNG-generated randomness with post-quantum cryptography (PQC) to demonstrate a practical use case. The proposed application leverages the QRNG for secure key generation while ensuring data sovereignty by hosting encrypted communications within a trusted local infrastructure. This design mitigates the risks associated with third-party data storage and aligns with data protection regulations.

Conclusion: The third hypothesis is *rejected*. Existing applications do not meaningfully utilize the QRNG chip for secure key generation or cryptographic operations. The proposed messaging and VoIP app demonstrates a practical way to leverage QRNGs for enhanced security and privacy.

X. CONCLUSION

This research highlights the underutilization of QRNG technology in existing Samsung Quantum Galaxy applications and proposes a secure instant messaging and VoIP solution that integrates QRNG-generated randomness with PQC. By combining these technologies, our application enhances mobile security and provides resistance to quantum computing attacks. While the QRNG chip demonstrated strong entropy performance in NIST SP800-90b tests, it underperformed in DieHarder and NIST SP800-22 evaluations, suggesting the need for improved postprocessing methods. Furthermore, our investigation revealed that no existing *killer app* effectively utilizes the QRNG's capabilities. Our work underscores the importance of QRNG integration in practical security applications and lays the groundwork for future developments in secure quantum-based mobile communications.

ACKNOWLEDGMENT

The author thank Richard Schöbel from ID Quantique for facilitating the purchase of the Samsung Quantum Galaxy smartphone through their partners, enabling them to conduct this research effectively. They also thank Naser Alibrahim and Majid Alqabandi for their invaluable assistance in rooting the smartphone, and conducting an online precursory search for QRNG-enabled apps.

REFERENCES

[1] ID Quantique. (2020). *Quantum-safe Solutions: Securing the Future Today*. [Online]. Available: https://www.idquantique.com/quantum-safe-security/
[2] ID Quantique. (2023). *SK Telecom and Samsung Unveil the Galaxy Quantum 4*. [Online]. Available: https://www.idquantique.com/sk-telecom-and-samsung-unveil-the-galaxy-quantum-4/
[3] ID Quantique. (May 2024). *Quantis QRNG Chips*. [Online]. Available: https://www.idquantique.com/random-number-generation/products/quantis-qrng-chips/
[4] V. Mannalatha, S. Mishra, and A. Pathak, "A comprehensive review of quantum random number generators: Concepts, classification and the origin of randomness," *Quantum Inf. Process.*, vol. 22, no. 12, p. 439, Dec. 2023.
[5] X. Ma, X. Yuan, Z. Cao, B. Qi, and Z. Zhang, "Quantum random number generation," *NPJ Quantum Inf.*, vol. 2, no. 1, p. 16021, Jun. 2016.
[6] M. Herrero-Collantes and J. C. Garcia-Escartin, "Quantum random number generators," *Rev. Mod. Phys.*, vol. 89, no. 1, Feb. 2017, Art. no. 015004.
[7] D. Hurley-Smith and J. Hernandez-Castro, "Quantum leap and crash: Searching and finding bias in quantum random number generators," *ACM Trans. Privacy Secur.*, vol. 23, no. 3, p. 16, 2020.
[8] Y.-Q. Nie, J.-Y. Guan, H. Zhou, Q. Zhang, X. Ma, J. Zhang, and J.-W. Pan, "Experimental measurement-device-independent quantum random-number generation," *Phys. Rev. A, Gen. Phys.*, vol. 94, no. 6, Dec. 2016, Art. no. 060301.
[9] F. Acerbi, N. Massari, L. Gasparini, A. Tomasi, N. Zorzi, G. Fontana, L. Pavesi, and A. Gola, "Structures and methods for fully-integrated quantum random number generators," *IEEE J. Sel. Topics Quantum Electron.*, vol. 26, no. 3, pp. 1–8, May 2020.
[10] S. Tisa, F. Villa, A. Giudice, G. Simmerle, and F. Zappa, "High-speed quantum random number generation using CMOS photon counting detectors," *IEEE J. Sel. Topics Quantum Electron.*, vol. 21, no. 3, pp. 23–29, May 2015.
[11] Y.-Q. Nie, H.-F. Zhang, Z. Zhang, J. Wang, X. Ma, J. Zhang, and J.-W. Pan, "Practical and fast quantum random number generation based on photon arrival time relative to external reference," *Appl. Phys. Lett.*, vol. 104, no. 5, Feb. 2014, Art. no. 051110.
[12] E. Pelofske, "Analysis of a programmable quantum annealer as a random number generator," *IEEE Trans. Inf. Forensics Security*, vol. 19, pp. 3636–3643, 2024.
[13] A. Ash-Saki, M. Alam, and S. Ghosh, "Improving reliability of quantum true random number generator using machine learning," in *Proc. 21st Int. Symp. Quality Electron. Design (ISQED)*, Santa Clara, CA, USA, Mar. 2020, pp. 273–279.
[14] *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard*, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, 2024.
[15] *FIPS 204: Module-Lattice-Based Digital Signature Standard*, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, 2024.
[16] *FIPS 205: Stateless Hash-Based Digital Signature Standard*, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, 2024.
[17] D. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe. (2019). *Sphincs+: Submission to the NIST Post-Quantum Cryptography Standardization Project*. [Online]. Available: https://sphincs.org
[18] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehle, "CRYSTALS–kyber: A CCA-secure module-lattice-based KEM," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Apr. 2018, pp. 353–367.
[19] L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-dilithium: A lattice-based digital signature scheme," *IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 2018, no. 1, pp. 238–268, 2018.
[20] ID Quantique. (May 2024). *Clavis XG QKD System*. [Online]. Available: https://www.idquantique.com/quantum-safe-security/products/clavis-xg-qkd-system/
[21] KETS Quantum. (Oct. 2024). *Quantum Random Number Generator*. [Online]. Available: https://kets-quantum.com/quantum-random-number-generator/
[22] Quantum eMotion Corp. (Oct. 2024). *QRNG2: High-Speed Quantum Entropy*. [Online]. Available: https://www.quantumemotion.com/qrng2
[23] QuintessenceLabs. (Oct. 2024). *qStream Quantum Random Number Generator (QRNG) Overview*. [Online]. Available: https://info.quintessencelabs.com/
[24] Quantum Xchange. (2024). *Phio TX: Crypto-Agile, Quantum-Safe Key Delivery*. [Online]. Available: https://quantumxc.com/phio-tx/
[25] (Oct. 2024). *Whatsapp*. [Online]. Available: https://www.whatsapp.com

[26] Signal Messenger. (Oct. 2024). *Signal: Private Messenger*. [Online]. Available: https://signal.org

[27] B. Cogliati, J. Ethan, and A. Jha, "Subverting Telegram's end-to-end encryption," *IACR Trans. Symmetric Cryptol.*, vol. 2023, no. 1, pp. 5–40, Mar. 2023.

[28] M. Marlinspike. (2013). *Advanced Cryptographic Ratcheting*. signal Blog. [Online]. Available: https://signal.org/blog/advanced-ratcheting

[29] M. Mosca, "Cybersecurity in an era with quantum computers: Will we be ready?" *IEEE Secur. Privacy*, vol. 16, no. 5, pp. 38–41, Sep. 2018.

[30] P. Zhang, K. Aungskunsiri, E. Martín-López, J. Wabnig, M. Lobino, R. W. Nock, J. Munns, D. Bonneau, P. Jiang, H. W. Li, A. Laing, J. G. Rarity, A. O. Niskanen, M. G. Thompson, and J. L. O'Brien, "Reference-frame-independent quantum-key-distribution server with a telecom tether for an on-chip client," *Phys. Rev. Lett.*, vol. 112, no. 13, Apr. 2014, Art. no. 130501.

[31] J. P. Mattsson, B. Smeets, and E. Thormarker, "Quantum technology and its impact on security in mobile networks," *Ericsson Technol. Rev.s*, vol. 2021, no. 12, pp. 2–12, Dec. 2021.

[32] M. Hoque, M. Hossain, and F. Sultana, "Machine learning-assisted quantum random number generation," in *Proc. IEEE Int. Conf. Commun., Comput., Signal Process. (ICCCSP)*, Jul. 2024, pp. 1–5.

[33] D. Frauchiger, R. Renner, and M. Troyer, "True randomness from realistic quantum devices," 2013, *arXiv:1311.4547*.

[34] C. Peikert, "A decade of lattice cryptography," *Found. Trends® Theor. Comput. Sci.*, vol. 10, no. 4, pp. 283–424, 2016.

[35] Hex-Rays. (Oct. 2024). *IDA Pro: Interactive Disassembler. Version 9.0*. [Online]. Available: https://hex-rays.com/ida-pro

[36] *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications (NIST Special Publication 800-22 Revision 1a)*, Nat. Inst. Standards Technol., Gaithersbur, MD, USA, 2010.

[37] *Recommendation for the Entropy Sources Used for Random Bit Generation (NIST Special Publication 800-90b)*, Nat. Inst. Standards Technol., Gaithersbur, MD, USA, 2018.

[38] R. Brown. (Oct. 2004). *Dieharder: A Random Number Test Suite*. [Online]. Available: https://webhome.phy.duke.edu/~rgb/General/dieharder.php

[39] National Institute of Standards and Technology. (Jun. 2017). *Entropy Validation Certificate: Id Quantique Sa, IDQ250C2. Certificate, no. 63*. [Online]. Available: https://csrc.nist.gov/projects/cryptographic-module-validation-program/entropy-validations/certificate/63

[40] Topjohnwu. (Oct. 2024). *Magisk*. [Online]. Available: https://github.com/topjohnwu/Magisk

[41] Samsung Odin. (Oct. 2024). *Samsung Odin Flash Tool*. [Online]. Available: https://www.samsungodin.com

[42] A. Desnos and G. Gueguen. (Oct. 2024). *Androguard Project*. [Online]. Available: https://github.com/androguard/androguard

[43] CNN. (Sep. 2024). *Telegram Ceo Durov's Arrest and User Data Changes Raise Privacy Concerns*. [Online]. Available: https://edition.cnn.com/2024/09/23/tech/telegram-ceo-durov-arrest-user-data-changes/index.html

[44] (Oct. 2024). *Simplex Chat*. [Online]. Available: https://simplex.chat/

[45] The Tor Project. (Oct. 2024). *Tor: Anonymity Online*. [Online]. Available: https://www.torproject.org

[46] J. Hoffstein, J. Pipher, and J. H. Silverman, "NTRU: A ring-based public key cryptosystem," in *Algorithmic Number Theory*. Berlin, Germany: Springer, 1998, pp. 267–288.

[47] Bluetooth Special Interest Group. (2021). *Bluetooth Core Specification Version 5.3*. [Online]. Available: https://www.bluetooth.com/specifications/bluetooth-core-specification/

OMAR ALIBRAHIM received the master's degree from Rice University, in 2007, and the Doctor of Philosophy degree in computer science from Southern Methodist University, in 2012. He is currently an Assistant Professor with Kuwait University with extensive experience in cybersecurity research and development. He previously worked with several cybersecurity firms in U.S. as a Cybersecurity Researcher and a Consultant, gaining valuable industry experience. He leads multiple private-sector research and development initiatives in cybersecurity, both in the State of Kuwait and U.S. His research interests include malware analysis, reverse engineering, and cyber threat intelligence.