International Journal of Applied Mathematics
Volume 38 No. 6s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

RUBBER DUCKY BASED FAST INTERACTIVE CLOUD IAC MIGRATION

Prevesh Kumar Bishnoi [1], Dr. Dharmender Kumar[2], Dr. Prateek Bhanti[3]

[1] Computer Science and Engineering, School of Engineering and Technology Mody University of Science and Technology; [2]GJUST Hisar; [3]Computer Science and Engineering, School of Engineering and Technology Mody University of Science and Technology

Abstract*:*

Originally designed for HID-based penetration testing, Rubber Ducky may be automated between local systems and AWS, Azure, or GCP by DOS, Linux shell scripts. In this research, different rubber duckies are used for the inter-cloud migration process. Using its keystroke injection capabilities, Rubber Ducky can run predefined scripts for reliable data transfer and migration, hence reducing manual participation. This research creates various rubber duckies for multi-cloud migration, catering to different services. This type of arrangement and technique will execute rapid, consistent, safe migration processes. Finally, a framework for the migration between different cloud players is developed and tested. This automated approach of command execution provides consistency and precision, therefore preventing human errors in crucial cloud operations. It is simplifying repeated tasks such configuration updates, file transfers, and authentication help efficiency as well. This method connects the modern cloud systems and traditional computing environments. Good scripting enables businesses to have a simplified and reasonably cost approach to cloud data management. In last it is concluded that the rubber ducky command firing speed is multiple times faster than the human typing speed. In the future the migration process can be performed just by plugging in the rubber ducky or by wifi.

KeyWords – ATtiny85, AWS, Azure, Cloud, GCP, IaC, Migration, Pi-Pico, Rubber Ducky

1. Introduction

In the field of IT automation, cloud migration is an essential process allowing businesses to migrate their workload from on-site infrastructure to cloud environments as AWS, Azure, or GCP. IaC may give a good solution for the automated seamless inter-cloud migration. Historically, cloud migration demands for complex scripting, API integration, and manual interventions, therefore lengthening the process time and raising the risk of human errors. DOS shell programs can efficiently automate cloud transfer tasks by creatively using a Rubber Ducky USB as a Human Interface Device (HID).

Rubber Ducky, a keystroke injection tool, replicas a keyboard to supply pre-programmed payloads when hooked onto a computer. Every cloud provides an access of CLI to fire the command. But this system has many limitations, like slow typing and firing of commands, human error and correcting again, repeating of commands make the system slow for migration. HID devices were originally meant for ethical hacking and penetration testing; the Rubber

Received: August 06, 2025

International Journal of Applied Mathematics
Volume 38 No. 6s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Ducky can also be utilized for reasonable automated tasks, including cloud migration. Programming Rubber Ducky to run DOS/LINUX shell scripts so the IT departments and other users run migration between local systems and cloud platforms with simplicity.

There are many benefits from this:

1. Hand command typing is eliminated by automation, therefore reducing errors.
2. Portability: One can carry and apply it over different systems with little planning needed.
3. Efficiency: Runs predefined scripts fast without involving direct users.

Automating credential exposure inside scripts helps to restrict it. It works with many cloud services (AWS, Azure, GCP) without depending on GUI-based tools, thus proving cross-platform portability. Working with Rubber Ducky for cloud migration calls for scripting a series of keystrokes containing DOS-based commands for data transfer, cloud service configuration, and validation of migration outcomes. The existing technique for the migration is manual; specific tools dependent on specific clouds and Terraform are doing the migration in their own way. But rubber ducky-based migration is not for a specific cloud and fast in respect of firing the command. The process comprises the following phases:

1. Prepare the payload by writing a Rubber Ducky script (.txt) that includes DOS commands for cloud transfer.
2. Duck Encoder will assist you to translate the script into an appropriate inject bin file.
3. Making advantage of the Rubber Ducky To follow migration directions, plug the device into a target system.

Monitoring and validation can help you to confirm that data has been migrated successfully to or from the cloud.

- Key Cloud Migration Actions using DOS Shell Scripts
- Upload and download files by using AWS CLI (AWS S3, AWS EC2 create-instance).
- Azure: Create an Azure VM, provision the Azure VM, and automatically upload storage blobs.
- Sync local files with Google Cloud Storage (gsutil cp, gcloud compute instances generates).
- PowerShell and DOS scripts enable hybrid movement of work between cloud providers.

By adding Rubber Ducky with DOS/shell scripting into cloud migration processes, IT teams can reduce human effort and thereby speed up their activities, and it will be the best solution for the Inter-Cloud migration. Perfect data flows across on-site infrastructure and cloud platforms, including AWS, Azure, and GCP. This inventive approach provides. As cloud computing grows, automating technologies like Rubber Ducky will support security, scalability, and efficiency in planning for cloud migration. In this research two types of rubber ducky, one is Arduino ATtiny85 DigiSpark and the other Raspberry Pi Pico, are used.

2. Review

A detailed review of about the uses of RubberDucky was made by the research of many different articles. Bad USB assaults are a major concern for high-security systems in the always-changing terrain of cybersecurity hazards. By proactive keyboard speed monitoring,

Received: August 06, 2025

International Journal of Applied Mathematics
Volume 38 No. 6s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

this study presents a fresh défense mechanism to improve systems against these advanced threats. By means of four different modes—regular, paranoid, sly, and log-only—Python programs highlight the adaptability of the system and hence offer a flexible defence against different degrees of evil intent. Inspired by the methods of well-known Bad USB gadgets such as Rubber Ducky, our solution uses fast execution characteristics to pre-emptively find and fix dubious keyboard patterns. Comprehensive testing provides a strong defense that dynamically adjusts to fit real-world settings and exceeds the complexity of Bad USB attacks, so guaranteeing the operation of the system. The approach proposed in this work underlines the possibility of the system to increase cybersecurity resilience by means of an improved strategy to prevent Bad USB assaults and so set the basis for next proactive cybersecurity actions [1]. The effort needed for hostile hackers to breach a system has increased as cybersecurity rules and standards enforced by business entities expand. Usually, a hacker's efforts are directed on getting past intrusion policies and firewalls. Running keystroke commands as though a user were directly entering them lets a basic USB device disguising a keyboard circumvent security. This attack aims at people, the weakest area of security. This work presents Ducky-Detector, a heuristic-based tool based on typing patterns detecting hostile USB devices. Reaching almost perfect accuracy, zero false positives, and lowest processing overhead, the utility essentially splits between automated scripts and human input. We modelled real attack situations by testing Ducky-Detector against several commercial and free antivirus solutions with different payloads. On Linux machines, it creates only a 0.9% overhead, so guaranteeing security without compromising performance [2]. As reliance on information security increases, hardware interface protection must get better. Designed by Hak5, the USB Rubber Ducky uses flaws in order to replicate objects to carry illegal activity. The seriousness of this issue is shown by Offensive Security's robust USB attack tools, which enable sophisticated target computer interception and remote control. An intelligent system was built to solve this and evaluate connected peripheral devices. Combining study of user behaviour, program call speed, and exploitation tendencies with a deep learning neural network helps the system to identify maybe dangerous USB devices. Once found, the machine can switch off the hacked USB port on its own automatically. Furthermore, provided by the system is real-time security monitoring and remote interaction made feasible by Telegram API [3].

This investigation also looks at frame-up attacks, in which a Rubber Ducky USB sets false evidence on a PC of a target. The same platforms were used for experiments combining human and machine interactions. Forensic photo analysis revealed little artifacts left by programmable USB devices that let either an automated or a human script be recognized. This research draws attention to the risk of miscarriages of justice in case hostile automation cannot be distinguished from human activity by digital forensic investigators [4]. Social engineering is still a main cause of USB-based assaults. This page addresses several techniques penetration testers apply to exploit human vulnerabilities like phishing emails, credential theft, and payload execution via Rubber Ducky. Set, the Social Engineering Toolkit, helps attackers create complex attacks grounded on dishonesty. If security experts are to balance these risks, they have to implement multi-layered authentication rules. Python is still a valuable tool for penetration testing; however, it is imperative to protect scripting environments against unapproved running-

Received: August 06, 2025

International Journal of Applied Mathematics
Volume 38 No. 6s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

through. It also covers reverse shell payloads executed by PowerShell-based Rubber Ducky assaults compromising Windows operating systems. [5]

Web browsers save sensitive credentials more and more, so attackers employ Bad USB methods to get usernames and passwords. Running a Rubber Ducky configured with Chrome Pass and Password Fox lets attackers access stored credentials in Google Chrome and Mozilla Firefox. Given this research indicates credential theft via Bad USB may be performed in under 14 seconds, tight USB security restrictions are very necessary. [6] Frequently used in both the military and commercial sectors, radio frequency (RF) exposure from handheld transmitter-receiver devices presents still another major security concern. This work presents calculations indicating possible radiation hazards exceeding ANSI C95.1-1982 safety standards arising from small spiral (rubber ducky) antennas. Research results underline the need of appropriate handling techniques for using radio transceivers near the human body [7]. Further models verify that amateur radio operators utilizing small spiral antennas could be fairly highly exposed to electromagnetic energy, maybe beyond safety criteria. Users should be warned not to hold transceivers near their faces during transmission. One should either keep a transmitter four or five inches away or use external antennas to reduce health hazards. [8]

This paper presents a thorough investigation of Bad USB risks together with a keystroke monitoring-based security system driven by artificial intelligence increasing detection accuracy. The method improves cybersecurity resilience by means of deep learning and behavioral analysis to identify automated attacks. Biometric-based authentication methods will be investigated in future work in order to lower false positives and raise attack detection accuracy. By means of proactive cybersecurity solutions, businesses can reduce USB-based risks and protect their vital equipment against recently arising cyberattacks.

Finally, by the review it can be concluded that use of rubber ducky may be beneficial for the cloud and multi-cloud migration in respect of time, efficiency, etc.

3. Proposed Design

For this aim two distinct variants are used. One is PiPico; second comes Arduino Attiny85. The little and reasonably priced microcontroller ATtiny85 Arduino-based Rubber Ducky can perform specified keystroke injection attacks, acting as a USB HID (Human Interface Device). Usually running on the Digispark ATTiny85, the Arduino IDE is used to program. Its limited storage and processing capability make it ideal for simple automation chores like opening a command prompt and running rudimentary scripts. Since the target device is not network connected, physical access to it controls everything. Conversely, the more evolved replacement using the RP2040 chip and built-in WiFi capability is the Raspberry Pi Pico W Rubber Ducky. This is a very effective tool for penetration testing since it permits remote payload execution and data exfiltration over a network. Pico W supports MicroPython or CircuitPython unlike ATtiny 85, thereby allowing more sophisticated programming and automation. Though it requires more development knowledge and is substantially larger, its ability to run wirelessly makes it a better candidate for advanced cybersecurity uses. A small comparison of three modes of command fire as in following Table 3.1

Received: August 06, 2025

International Journal of Applied Mathematics
Volume 38 No. 6s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Table 3.1 Comparison Of Keystroke-Based Injection Devices

Feature	Human Typist	ATtiny85 HID	Raspberry Pi Pico W HID
Keystroke Speed	3-12 KPS (Keystrokes Per Second)	~100+ KPS	~100+ KPS
Words Per Minute (WPM)	40-80 WPM	1000+ WPM (equivalent)	1000+ WPM (equivalent)
Keystroke Delay	~100-300 ms per key	~5-10 ms per key	~5-10 ms per key
Typing Accuracy	~95-99% (humans make errors)	100% (pre-programmed)	100% (pre-programmed)
Remote Execution	No	No	Yes (via WiFi)
Automation Capability	Limited (manual input)	Fully automated (pre-programmed script)	Fully automated (pre-programmed + remote control)

Here we used cloud-to-local system and inter-cloud migration, focusing on the storage like AWS S3, Azure Blob, and GCP Storage.

3.1. Design and Assumption: For the purpose of modeling and comparing the time required for manual command-line execution versus Rubber Ducky script automation in inter-cloud migration tasks, it is assumed that the average typing speed for an experienced technical user lies between 40 and 52 words per minute, where one word is defined as five characters. This equates to approximately 200 to 260 characters per minute, or roughly 3.33 to 4.33 characters per second. Each migration command is considered to have an average length of 80 characters, which is typical for cloud storage transfer commands in AWS S3, Azure Blob, and Google Cloud Storage involving flags and parameters. Manual typing is subject to syntactic errors, and therefore an additional 10 seconds per command is allocated for error correction. Furthermore, cognitive delay and context switching between commands are considered, with an average overhead of 15 seconds per command to account for recalling commands, validating parameters, or reviewing terminal output. On the other hand, the Rubber Ducky device is assumed to execute pre-stored commands with negligible human delay, injecting keystrokes at hardware speed with an average execution time of 0.2 seconds per line. Each migration direction is expected to require three commands—typically authentication, transfer, and verification—and the study considers six migration directions: AWS to Azure, AWS to GCP, Azure to AWS, Azure to GCP, GCP to Azure, and GCP to AWS.

3.1.1 Manual Command Execution Time

$T_h = (C / R_h) + E_h + O$

Received: August 06, 2025

International Journal of Applied Mathematics
Volume 38 No. 6s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Where: C = Length of a command (characters), R_h = Human typing rate (characters/sec), E_h = Error-correction time (sec), O = Context-switching time (sec).

$T_{h,dir} = L \times T_h$

Where: L = Number of commands per migration direction, T_h = Manual execution time for one command (sec).

$T_{h,total} = N \times T_{h,dir}$

Where: N = Number of migration directions, T_h,dir = Manual execution time for one migration direction (sec).

3.1.2 Rubber Ducky Command Execution Time

$T_{r,dir} = L \times T_{line\_ducky}$

Where: L = Number of commands per migration direction, T_line_ducky = Average time per command line injected by Rubber Ducky (sec/line).

$T_{r,total} = N \times T_{r,dir}$

Where: N = Number of migration directions, T_r,dir = Rubber Ducky execution time for one migration direction (sec).

3.1.3 Relative Speedup Factor

Speedup = $T_{h,total} / T_{r,total}$

Where: T_h,total = Manual execution time for all directions (sec), T_r,total = Rubber Ducky execution time for all directions (sec).

3.2. **Sequential Diagrams:** Although the conversion process from AWS S3, Azure Blob Storage, and GCP Cloud Storage takes data from one local system, numerous techniques assure safe and effective data transport. The migration service helps the user initiate the migration. This service serves as a middleman between the authentication, data retrieval, and storage among several cloud providers. Every cloud platform runs on a safe authentication method whereby the migration tool signs in using suitable credentials or access tokens. Originally the migration tool for AWS S3 is connecting and authenticating using AWS credentials. Figure 3.2(a) is a sequential diagram for S3 migration from AWS to a local system. It gets the list of accessible buckets once approved and chooses the migration target bucket. It runs one at a time, choosing from the bucket a list of kept items. Everything downloaded from AWS S3 finds a home on the local PC. The migration program terminates the relationship with AWS S3 and makes sure every object has been effectively kept after the download.

Azure Blob Storage is also taken under consideration and the sequence diagram is in Figure 3.2(b). Using Azure credentials helps to link the migration tool to cloud storage. It gathers the easily accessible containers and chooses the one appropriate for migration. From a list within the chosen container, the service downloads blobs one after the other locally. Every copy of a file is examined to guarantee none of data loss happens. Once every blob has been effectively migrated, the link to Azure Blob Storage closes.

Received: August 06, 2025

International Journal of Applied Mathematics
Volume 38 No. 6s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

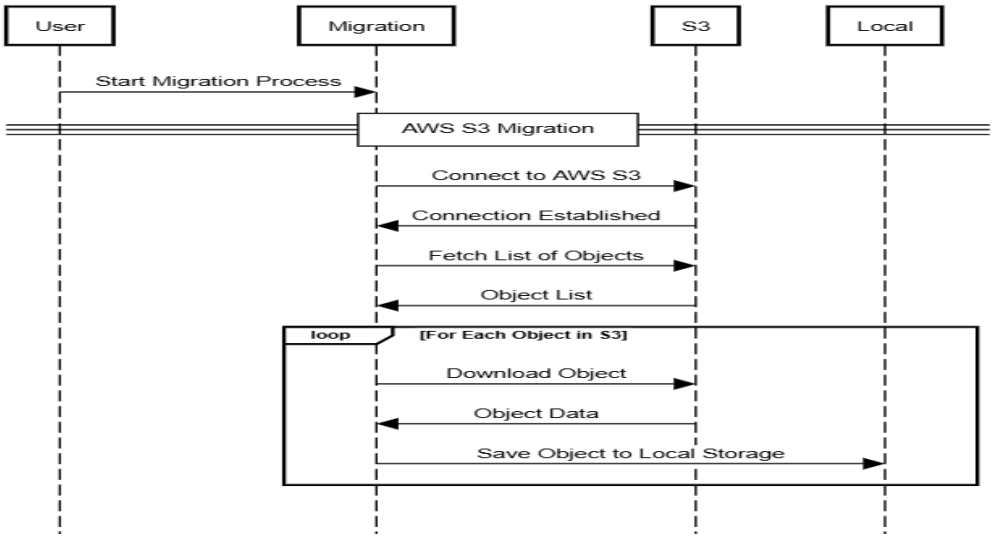
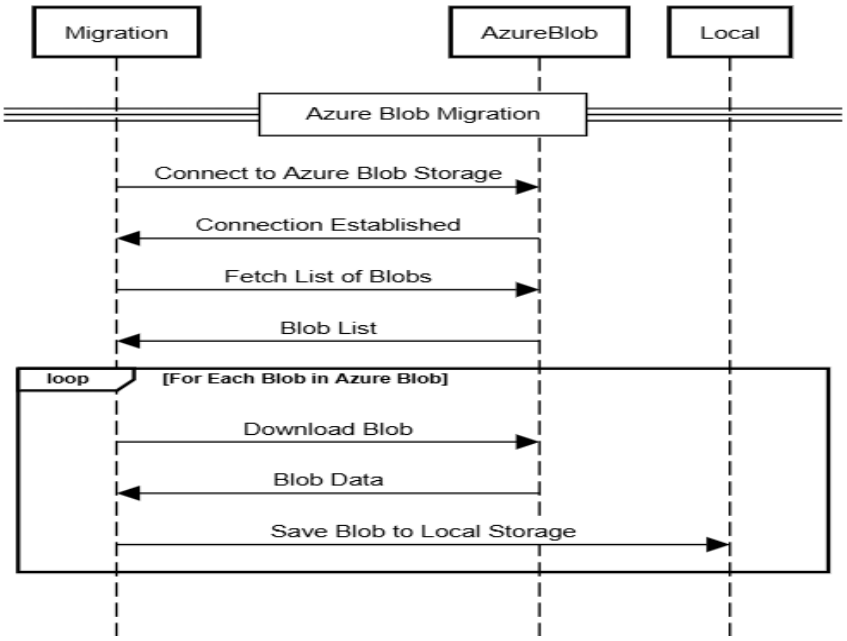


Figure 3.2(a)

The move towards GCP Cloud Storage also fits the identical pattern with the sequence diagram in Figure 3.2(c). The service initially authenticating using GCP credentials generates a list of accessible buckets. After selecting, it shows every item retained in the target bucket. Everything then downloads and finds place on the local storage system. Once finished, the migration project guarantees that every file has been kept exactly before linking to GCP Cloud Storage.

The migration method ensures perfect operation by means of logging and error control all around. Any problems or inadequate downloads set retry processes meant to preserve data integrity in action. Noting that all files from AWS S3, Azure Blob Storage, and GCP Cloud Storage are now kept locally, the migration tool alerts the user at the end of the process, thereby certifying their successful completion.

Figure 3.2(b)



Received: August 06, 2025

International Journal of Applied Mathematics
Volume 38 No. 6s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

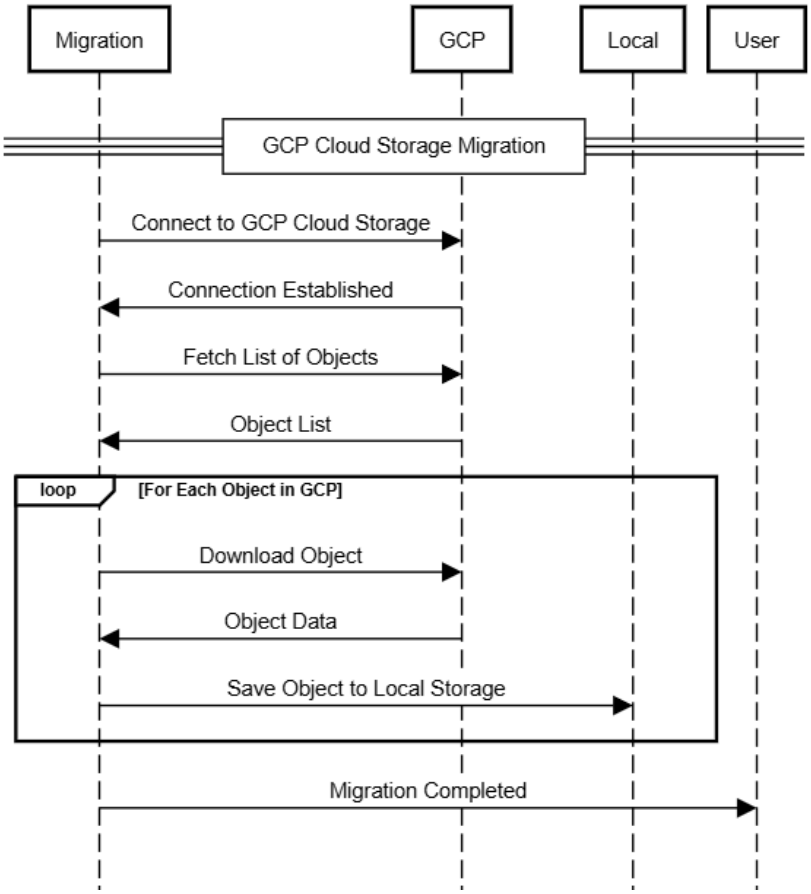


Figure 3.2(c)

4. Results

4.1 Results of testing by Payload Script for S3 Bucket, Azure Blob Storage and GCP Cloud Storage:

Here's a look at how long it takes to move data between Google Cloud Storage, Azure Blob, and AWS S3 in six different ways. I compared doing it manually versus using a pre-made Rubber Ducky script.

Firing the commands by hand, like typing in CLI commands with all the parameters, checking them, and running them, takes about 45 to 65 seconds per person. The time changes because the commands differ in length and how tricky they are. Moving info to Azure Blob seems to take a bit longer (50–65 seconds) compared to AWS and GCP (45–60 seconds).

With Rubber Ducky scripts, things happen fast—only 8 to 13 seconds! That's because the commands go right to the hardware without needing to type anything. The times vary a little depending on the command length and how fast the keystrokes are injected.

In short, moving stuff manually takes around 55 seconds, while a Rubber Ducky does it in just 11 seconds. That makes the command part about five times faster, not counting the data transfer time.

Received: August 06, 2025

International Journal of Applied Mathematics
Volume 38 No. 6s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

Table 4.1 Performance comparison across platforms and methods

Storage Service	Manual Command (Console)	ATtiny85 (Keystroke Injection)	Pi Pico W (Payload Execution)
AWS S3	47.5	8.5	8.5
Azure Blob Storage	52.5	10.0	10.0
GCP Cloud Storage	50.0	9.0	9.0

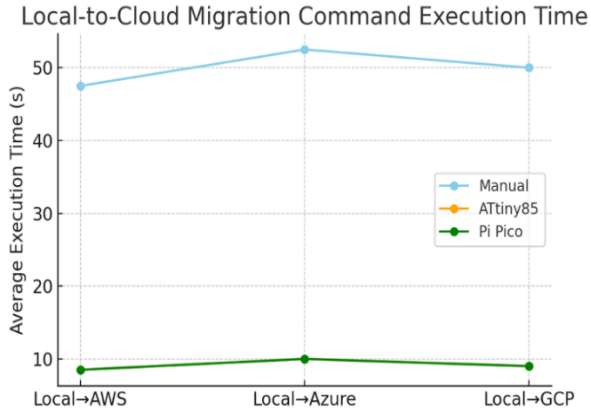


Figure 4.1 Cloud Storage to Local Migration Time Comparison

In Figure 4.1 and Table 4.1, it displays about the execution time comparison for local system to cloud storage migrations targeting AWS S3, Azure Blob, and GCP Storage. Manual migration times range from 47.5 to 52.5 seconds, while ATtiny85 and Pi Pico achieve 8.5 to 10 seconds. The pattern results the inter-cloud migration results for storage, confirming that hardware-based automation substantially reduces the time required to issue repetitive, parameter-rich commands. This efficiency gain is consistent across all local-to-cloud scenarios examined.

4.2 Intercloudmigration AWS S3, Azure Blob and GCP Storage:

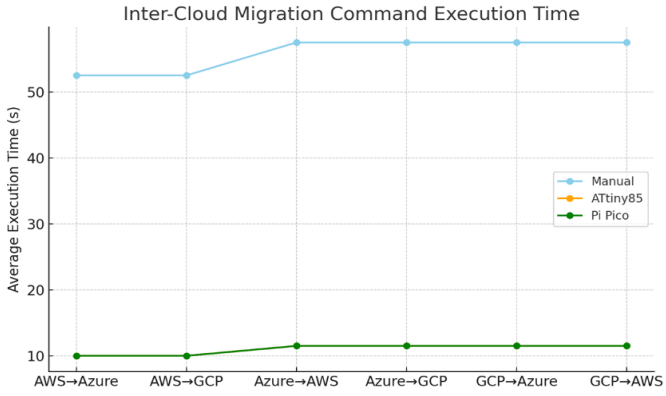


Figure 4.2 Cloud-To-Cloud Migration Time Comparison

Received: August 06, 2025

International Journal of Applied Mathematics
Volume 38 No. 6s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

In Figure 4.2 and Table 4.2, the execution time comparison for inter-cloud migrations between AWS S3, Azure Blob, and GCP Storage. Finally, the results indicate manual execution consistently taking 45–65 seconds, while both ATtiny85 and Pi Pico automation reduce this to 8–13 seconds across all routes. Finally, it demonstrates a clear advantage of scripted hardware injection over human typing for multi-cloud CLI tasks. The close match between ATtiny85 and Pi Pico values indicates that keystroke injection speed is the primary limiting factor rather than hardware differences. Overall, automation yields roughly a five-fold improvement in the command-firing phase.

Table 4.2 Inter-cloud migration time comparison for different methods and services

Migration Path	Manual Command (Console)	ATtiny85 (Keystroke Injection)	Pi Pico W (Payload Execution)
AWS S3 → Azure Blob	45–60	8–12	8–12
AWS S3 → GCP Storage	45–60	8–12	8–12
Azure Blob → AWS S3	50–65	10–13	10–13
Azure Blob → GCP	50–65	10–13	10–13
GCP → Azure Blob	50–65	10–13	10–13
GCP → AWS S3	50–65	10–13	10–13

5. Conclusion

This study found that using simple hardware automation tools like Rubber Ducky (with ATtiny85 or Pi Pico) can really speed up cloud storage migrations, whether you're moving between clouds or from local storage to the cloud. Doing things by hand took about 50–55 seconds, but automation got it done in just 8–12 seconds—that's like saving 80% of the time and making things five times faster! The ATtiny85 and Pi Pico performed about the same, which suggests the delay in injecting keystrokes is what slows things down, not the device itself. This kind of automation is super useful in situations where you're dealing with multiple clouds or a mix of cloud and local setups, where you need to run commands often and correctly. The relative speed fluctuations among all cloud providers—AWS S3, Azure Blob, and GCP Storage—remain consistent; Pi Pico W noticeably surpasses both hand execution and ATtiny85. Although handwork is suitable for small-scale projects, Pi Pico W is finally the best choice for automation; ATtiny85 is the least efficient for such projects. In the future the same research can be repeated for the integration of Rubber-Ducky with IaC to get more throughput.

ACKNOWLEDGMENT

The authors want to thank their organizations, GJUST Hisar and Mody University of Science and Technology, for providing the best environment for research and development.

Received: August 06, 2025

Volume 38 No. 6s, 2025
ISSN: 1311-1728 (printed version); ISSN: 1314-8060 (on-line version)

REFERENCES

[1]. N. T. Arun Jothi, S. Anu, K. Harsha, and R. Devi Priya, "USB Rubber Ducky Hunter: A Proactive Defense Against Malicious USB Attacks," in Proc. 2024 IEEE International Conference on Intelligent Systems for Cybersecurity (ISCS), pp. 1–6, 2024, doi: 10.1109/ISCS61804.2024.10581045.

[2]. L. Arora, N. Thakur, and S. K. Yadav, "USB Rubber Ducky Detection by using Heuristic Rules," in Proc. 2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), pp. 156–160, 2021, doi: 10.1109/ICCCIS51004.2021.9397064.

[3]. A. Tyutyunnik and A. Lazarev, "Intelligent System for Preventing Rubber Ducky Attacks Using Deep Learning Neural Networks," in Proc. 2021 International Russian Automation Conference (RusAutoCon), pp. 471–476, 2021, doi: 10.1109/RusAutoCon52004.2021.9537355.

[4]. D. O. Lawal, D. W. Gresty, D. E. Gan, and L. Hewitt, "Have You Been Framed and Can You Prove It?," in Proc. 2021 IEEE International Conference on Cyber Security and Resilience (CSR), pp. 302–308, 2021, doi: 10.1109/CSR51186.2021.9527963.

[5]. G. Khawaja, "Social Engineering Attacks," in Data and Cybersecurity Handbook, Wiley, 2021, pp. 235–256, doi: 10.1002/9781119671455.ch13.

[6]. A. A. Muslim, A. Budiono, and A. Almaarif, "Implementation and Analysis of USB based Password Stealer Using PowerShell in Google Chrome and Mozilla Firefox," in Proc. 2020 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS), pp. 55–60, 2020, doi: 10.1109/ICIMCIS51567.2020.9354301.

[7]. P. E. E. Thomas Chesworth, "Potentially hazardous near-field electromagnetic emanations from handheld transceivers," in IEEE International Symposium on Electromagnetic Compatibility, pp. 314–319, 1994, doi: 10.1109/ISEMC.1994.363887.

[8]. E. T. Chesworth, "Near field energy densities of hand-held transceivers," in IEEE Transactions on Electromagnetic Compatibility, vol. 31, no. 1, pp. 24–28, Feb. 1989, doi: 10.1109/15.16666.

Received: August 06, 2025