Secure Use of USB Storage Devices in Mission-Critical Networks Using Authentication and Cryptographic Controls

Sandeep Tiwari
Scientist/Engineer, NRSC, ISRO
sandeep_tiwari@nrsc.gov.in

Krishna Kishore S V S R
Scientist/Engineer-SG; NRSC, ISRO
kishore_svsr@nrsc.gov.in

Adarsh Sharma
Apprentice, NRSC
adarsh4939@gmail.com

doi: https://doi.org/10.37285/bsp.SACAD2025.50

***Abstract*:** This paper proposes a secure framework for the controlled use of USB storage devices in mission networks. The system ensures that only pre-authenticated USB devices are allowed access, and enforces strict encryption/decryption and signature verification policies for file transfer. By leveraging device enumeration, serial number validation, AES encryption, digital signatures, and network-bound decryption mechanisms, the frame- work minimizes risks of data leakage, unauthorized access, and malware intrusion.

Keywords:

Device authentication, Encryption and decryption, AES encryption, Digital signatures, File transfer security, Serial number validation, Network-bound decryption, Malware prevention, Data leakage protection

I. Introduction:

The widespread use of USB storage devices has introduced both convenience and significant security challenges, particularly within mission-critical networks where data integrity and confidentiality are paramount. While USB drives offer a portable and efficient method of transferring data, they are also a known vector for malware, data ex- filtration, and unauthorized access [1], [2]. In environments like military, research, and space organizations where sensitive information is regularly handled such vulnerabilities are unacceptable. Traditional access control mechanisms often fall short in safeguarding against these risks, especially when users are allowed unrestricted read/write and execution access to USB devices. The risk intensifies when USB drives are moved between secure and insecure environments, potentially bypassing perimeter defences. To address these concerns, this paper proposes a controlled and secure method for using USB storage devices within mission networks. The proposed framework introduces a multi-layered security approach that combines:

- **Device Authentication:** Only pre-approved USB drives, identified through unique serial numbers and make identifiers, are permitted to mount on end-user systems [3].
- **Permission Management**: Authenticated drives are mounted with restricted permissions of read/write ac- cess is allowed, but execution of any files is strictly disabled.
- **File Integrity and Signature Verification**: All files on the USB device are verified against a secure digital signature schema. Unsigned or tampered files are automatically deleted. Centralized database is maintained to validate the integrity of files.
- **Encryption and Decryption Control:** Data transfer to and from the USB device is secured using AES-128 encryption. Decryption is only possible when the USB drive is connected to an authenticated server within a specific trusted network [4].
- **Centralized Key Management:** A secured central database maintains the mapping of USB devices and their encryption keys, ensuring that decryption can occur only under controlled network conditions.

By implementing these measures, the framework not only restricts USB access to trusted devices but also ensures that data cannot be read, written, or executed unless all security checks pass. This layered approach significantly reduces the threat surface associated with USB storage and supports secure data movement without compromising mission integrity.

This paper details the design, implementation, and benefits of this system, along with its role in strengthening operational security within high-assurance network environments.

II. System Architecture

The proposed system architecture focuses on secure and controlled usage of USB storage devices within a mission network environment. It integrates hardware-level device authentication, cryptographic file control, and network- bound data access restrictions to enforce a comprehensive USB usage policy.

A. USB Authentication

When a USB device is inserted into the system, the operating system triggers a process called USB enumeration.
During this process, critical metadata such as Vendor ID (VID), Product ID (PID), and Serial Number is extracted [5]. These identifiers are used to uniquely identify and authenticate the device. A centralized USB Authorization Database maintains a whitelist of all approved USB devices, each entry containing:

- Serial number
- Manufacturer (Make)
- Device Type

Upon detection, the inserted USB's serial number and make are cross-checked against the database:

- If the device is authorized: It is mounted automatically with read and write permission only
- If not: the mounting process is aborted and an alert can be raised

For authenticated USB devices, mounting is done with:

- Read/Write Access
- USB mount point is only visible to root user
- Execution permission disabled

This policy ensures that even if a malicious executable is present on the USB, it cannot be run from within the network.

B. *File Encryption and Verification Mechanism*

1) Uploading Data to USB (Secure Write Path):

- Users copy files into a designated Input Folder on the centralized authenticated server [6].
- A background service encrypts each file using AES-128 encryption. The key used is retrieved from the central database based on the USB's unique identifier [7].
- Each file is digitally signed using a pre-defined signature scheme (e.g. RSA signature) and file digital signature will be kept in database.
- The encrypted, signed file is then moved to the mounted USB storage. Crontab is configured for moving file from designated Input Folder to USB drive.

This process ensures that:

- Files are encrypted before leaving the host.
- Integrity and origin of files can be verified upon later access.
- Unauthorized users cannot create or copy files directly to the USB in plaintext.

2). Reading Data from USB (Secure Read Path): When the USB is inserted into an authenticated centralized server

- Each file is checked for a valid digital signature kept in centralized database.
- Files with invalid signatures are deleted to prevent tampering or injection attacks.
- Valid files are decrypted using the appropriate key from the database.
- Decrypted files are copied to a Secure Output Folder with read-only access.

This flow ensures that only files which have passed integrity checks and originated from secure sources are made avail- able to users.

C. *Centralized Database and Key Management*

The central Security Database is hosted inside the mission network and is inaccessible to unauthorized users. It maintains:

- A list of authorized USB device serial numbers and their metadata.
- A Device-Key Map, linking each USB to a unique AES encryption/decryption key.
- A record of latest file signatures.

This database is accessed only by background services running on authenticated machines.

Key features include:

- No direct user access to encryption keys
- *Network-bound decryption*: Decryption operations only succeed when the system is connected to the mission network and can authenticate to the database.
- Automatic logging of all file transfers for auditing [8]. This design ensures that even if an encrypted USB is physically stolen or plugged into an external device, its contents remain inaccessible.

D. *System Workflow Overview*

Writing in USB drive [see Fig. 1]

1) Device Insertion → USB enumeration → Serial check → Mount (if approved)
2) File Upload → Place in input folder → Encrypt & digital sign → Write to USB

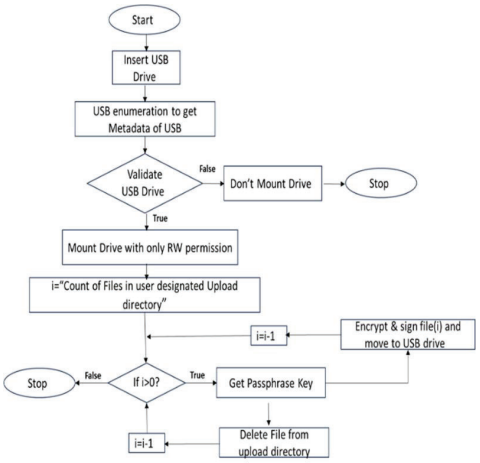

Fig 1. Write Path to USB

Reading from USB drive [see Fig. 2]

3) File Access → Read from USB → Signature check → Decrypt → Copy to output folder.
4) Database Role → Manages whitelist, keys, & signature rules → Accessible only within mission network.

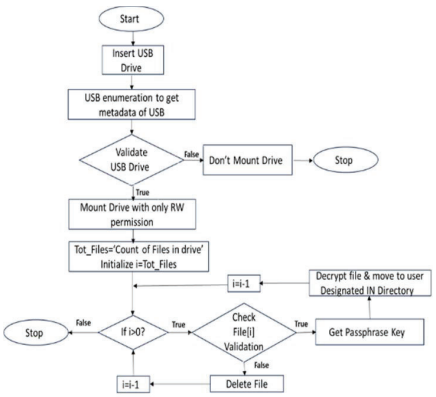

Fig2. Read Path from USB

III. Implementation Details

The secure USB control framework has been implemented on Red Hat Enterprise Linux (RHEL) using native tools, shell scripting, and database integration. The system components are designed to work together to authenticate USB drives, securely handle file

transfers, and enforce network-bound access restrictions. This section details the tools, commands, and workflows used in the implementation.

A. ***USB Device Authentication***

1) *Device Identification and Enumeration:* Upon USB insertion, the system uses the dmesg command to parse kernel messages and extract the USB device's Make and Serial Number.

```
root# dmesg | grep -i "usb"
```

Listing 1. USB device check via dmesg

A script extracts the required information using grep, gawk, or regex parsing [9].

2) ***Validation Against MySQL Reference Table:***

A MySQL database table named (e.g., usb_auth) stores the following attributes:

- Serial Number
- Make
- PassPhrase Key (AES-128 encryption key)
- File name
- Encrypted File digital signature

Access to the MySQL database is restricted to:

- A specific internal network subnet
- Authenticated credentials, known only to the background service.

The USB validation script connects to the database using a secure CLI and queries the table for a match [10].

B. ***Disk Detection and Mounting***

Once validated, the USB drive is identified using the fdisk command:

```
root# fdisk -l | grep -i "Disk /dev/sd"
```

Listing 2. Identifies the plugged USB drive

This helps determine the block device (e.g., /dev/sdb1) assigned by the OS. The USB is then mounted with noexec flag to disable execution of binaries:

```
root# mount -o rw,noexec /dev/sdb1 /mnt
    /secure_usb
```

Listing 3. Mounts the identified USB drive with read/write and with no binary execution permission.

C. ***File Encryption and Decryption***

The GNU Privacy Guard (gpg) package is used for AES- 128 symmetric encryption and decryption of files [11].

1) *Encryption Workflow (USB Write path):*

- User places file in designated input folder (e.g. /data/IN/)
- Background script retrieves encryption key from MySQL (based on Serial Number)

```
root# gpg --decrypt --passphrase
    "$KEY" -o output_file.txt
    encrypted_file.gpg
```

- File is encrypted with gpg tool:

```
root# gpg --symmetric --cipher-algo
    AES128 -- passphrase "$KEY" -o
    encrypted_file.gpg input_file.txt
```

Listing 4. Encrypts the user's .txt input file with symmetric AES-128 encryption, uses the passphrase stored in $KEY and writes encrypted .gpg output file.

- Encrypted file will be digitally signed, and moved to USB mount path (/mnt/secure_usb/). Digital signature and file name is stored in MySql database server.

2) *Decryption Workflow (USB Read Path):*

- Inserted USB is authenticated
- Files on USB are scanned:
  - Signature verified
  - Invalid files deleted
- Verified files are decrypted using stored key:

Listing 5. Decrypts the .gpg encrypted file using earlier stored "$KEY" passphrase and outputs file.

- Decrypted files are placed in a read-only output direc- tory (e.g., /data/output/)

D. ***Automation and Monitoring***

The system uses crontab to run background monitor- ing and processing scripts at regular intervals. A polling mechanism checks for :

- New USB insertions
- Input folder activity
- File processing tasks

Example crontab entry (runs every minute):

```
* * * * * /usr/local/bin/usb_monitor.sh
    >> /var/ log/usb_system.log 2>&1
```

Listing 6. Cron job running usb_monitor.sh every minute, logging output and errors to /var/log/usb_system.log

This ensures timely response to USB events without requiring user intervention [12].

E. ***Activity Logging***

All critical events included are given as follow:

- USB insertions
- Authorization attempts
- Encryption/decryption actions
- Signature failures

Above mentioned events are logged into a separate MySQL audit table, which includes:

- Timestamp
- Device Serial Number
- Result (Success/failure)

This facilitates auditing, accountability, and forensic analysis in case of security incidents [13].

F. ***Security Measures***

- MySQL access is restricted to secure network IPs using host-based firewall rules.
- Database credentials are stored securely in a read-only configuration file with root-only permissions.
- USBs are mounted using noexec, nosuid and nodev flags for maximum protection.

This implementation demonstrates how a secure USB framework can be built using standard Linux utilities, cryptographic tools, and controlled automation—making it both cost-effective and robust for mission-critical environments [14], [9].

IV. SECURITY FEATURES

The proposed USB control system integrates multiple layers of security to address the risks associated with USB device usage in mission-critical networks. Each component of the architecture plays a specific role in ensuring confidentiality, integrity, and controlled access to sensitive data.

A. *Device-Level Access Control*

Only USB devices that are pre-approved and listed in the central MySQL database (based on serial number and make) are allowed to mount. This whitelist approach ensures:

- Unauthorized or rogue USB drives are rejected.
- Only verified hardware can access or exchange data.

This helps to prevent malware injection and data theft via unapproved USB drives [9], [15].

B. *Execution Restriction on Mounted Drives*

Even for authenticated USB drives, mounting is per- formed with the noexec flag. This disables execution of any binaries or scripts from the USB, effectively neutralizing malicious payloads that may have been inadvertently introduced [14], [15].

Stops propagation of auto-run malware or exploits via executable files.

C. *Enforced Encryption with Key Isolation*

All files copied to USB drives are encrypted using AES-128, and keys are never exposed to end users. Keys are stored in a centralized database accessible only by internal systems over a secure, trusted network.
Even if a USB is lost or stolen, encrypted data remains inaccessible without both the correct key and network context [6], [16].

D. *Digital Signature Verification*

Each file written to a USB is signed using a predefined secure signature process. Upon re-insertion, the system validates the signature before decrypting or granting access. Tampered or unsigned files are deleted immediately.

It protects against unauthorized file injection or tampering during transit [13], [17].

E. Network-Bound Decryption

Decryption of USB files is only permitted when:

- The device is authenticated.
- The host system is connected to the secured internal network.
- The system can access the central key database.

This helps to prevent off-network data access, even if both the USB and software are compromised [9], [16].

V. CONCLUSION

The use of USB storage devices in mission-critical environments presents a unique challenge: balancing operational convenience with stringent security requirements. This paper has presented a comprehensive solution that allows controlled, secure use of USB devices within a mission network by leveraging device-level authentication, encrypted file transfer, signature validation, and network-bound decryption.

Through the use of standard Linux tools such as dmesg, fdisk, gpg, and crontab, combined with a centralized MySQL database, the framework ensures that only authorized USB devices can interact with the system, and only in a tightly controlled manner. All data transfers to USB are encrypted using AES-128, and decryption is permitted only under secure network conditions with validated credentials. Unauthorized devices and unsigned files are automatically rejected, and all actions are logged for accountability.

By isolating encryption keys from users, enforcing file integrity verification, and implementing strict access controls, the proposed solution significantly reduces the risk of data leakage, malware introduction, and misuse of portable storage in sensitive environments. This system can be adopted in military, space, intelligence, and other domains where information assurance is non-negotiable.

Ultimately, the framework demonstrates that with careful architectural planning and the right combination of open-source tools, high-assurance data transfer using USB devices is both achievable and sustainable in modern mission networks.

References:

[1] D. V. Pham, A. Syed, and M. N. Halgamuge, “Universal serial bus based software attacks and protection solutions,” Digital Investigation, vol 7, 2011.

[2] F. Griscioli and M. Pizzonia, “Usbcaptchain: Preventing (un)con- ventional attacks from promiscuously used usb devices in indus- trial control systems,” *arXiv preprint arXiv:1810.05005*,, 2018

[3] Y. Wang and F.-H. Hsu, “Usbips framework: Protect- ing hosts from malicious usb peripherals,” *arXiv preprint arXiv:2409.12850, 2024.*

[4] Y. Yu, “Encryption technology for computer network data security protection,” *Security and Communication Networks*, vol. 2022, p. 1789222, 2022.

[5] Microsoft, "Standard usb identifiers - windows drivers," 2024.
[6] W. Stallings, *Cryptography and Network Security: Principles and Practice*. Pearson, 2017.
[7] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtain- ing digital signatures and public-key cryptosystems," *Communi- cations of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.
[8] W. E. Burr, D. Dodson, W. Polk, and S. Aggarwal, "Nist special publication 800-57 part 1 revision 3: Recommendation for key management," 2011.
[9] Vasilomanolakis, F. Kargl, and S. Wurster, "Usb security: On the effectiveness of device authentication," in *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pp. 1–12, 2015.
[10] Smith and A. Johnson, "Securing database access in mission- critical systems," *Journal of Information Security*, vol. 10, 2019.
[11] D. J. Bernstein and T. Lange, "The gnu privacy guard (gnupg)"
[12] J. Turnbull, *The Pro Linux System Administrator*. New York, NY: Apress, 2014.
[13] M. Bashir *et al.*, "Log-based intrusion detection: A survey," *Jour- nal of Network and Computer Applications*, vol. 74, pp. 97–113, 2017.
[14] J. V. Wheeler and M. Messier, *Secure Programming Cookbook for C and C++: Recipes for Cryptography, Authentication, Input Validation & More*. O'Reilly Media, 2012.
[15] W. Zhang and et al., "A survey on usb security threats and countermeasures," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 3, pp. 2417–2441, 2019.
[16] L. Wang *et al.*, "Secure data storage and access control for cloud-based usb storage," *Journal of Network and Computer Applications*, vol. 109, pp. 87–98, 2018.
[17] Y. Chen *et al.*, "Digital signature schemes: A survey," *Journal of Information Security and Applications*, vol. 53, p. 102540, 2020.