Runtime Verification for Trustworthy Secure Shell Deployment

Axel Curmi, Christian Colombo, and Mark Vella

University of Malta, Malta

Cryptographic protocol concerns

- Standard practice to establish provable security guarantees of a protocol (e.g., Dolev-Yao model)
- However, the execution of the protocol might still be insecure
  - Protocol implementation might not adhere to the protocol specifications
    - Incorrect implementation of protocol steps
    - Missing checks
  - Malware might interfere with the execution of security-critical code

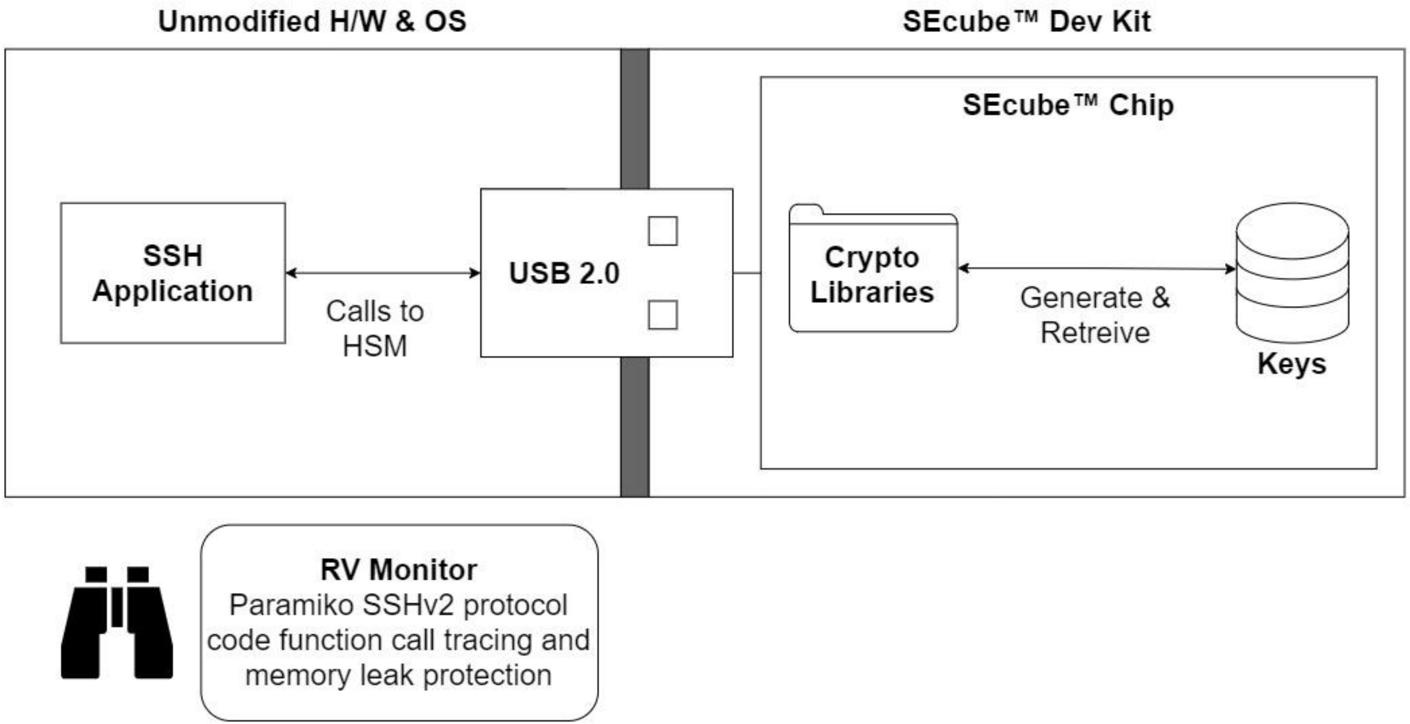
Secure Shell (SSH)

- Adapted as an internet standard in 2006
- Complements existing work on the TLS 1.3 protocol
- Widely **popular** in the industry
- Some features:
    - Remote login and command execution
    - Virtual private networks (VPNs)
    - Port forwarding
    - Secure file transfer

Trusted Execution Environments

- **Isolates** security-critical code from untrusted code (potentially compromised by malware)
- In practice, TEEs are implemented as **CPU modes** offering non-addressable, reserved, and encrypted memory pages
  - Intel SGX
  - AMD SEV/SME
  - ARM TrustZone

The proposed solution: RV-TEE

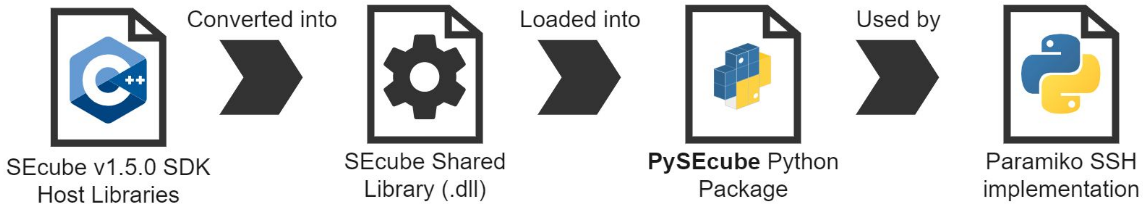


Protocol implementation

- The chosen SSH implementation is **Paramiko**, having as of April 2021:
  - 11.4K dependant repositories
  - 898 dependant packages (Docker Python SDK, Ansible and Apache Airflow)
- Uses ***cryptography*** Python library for cryptographic operations
  - Based on the OpenSSL implementation

Utilisation of HSM in protocol implementation

- The protocol implementation source code is modified such that:
    - The HSM is utilised for cryptographic operations, rather than ***cryptography***
    - Ephemeral cryptographic keys are stored on the HSM, rather than kept in memory
- Changes involving cryptographic operations:
    - **Hashing** during **key exchange** and **key derivation** (*SHA256*)
    - **Encryption** and **decryption** of protocol messages (*AES256*)
    - **Message authentication code** generation (*HMACSHA256*)



Property derivation and specification

- 17 properties were systematically derived from RFCs
  - Focused on RFC standard keywords, e.g. “**MUST”** and “**SHOULD”**
- Focused on the client side of the protocol
  - Weaker security
  - SEcube device used is aimed towards end-users

Property Category	Number of Properties
Temporal	11
Point assertion	5
Real-time	1

Example temporal property

"When a **KEXDH_REPLY** message is received from the server, the client **must verify** the **public host key** with the **signature of the hash obtained**"

Violation of the property leads to:

- Vulnerability to active man-in-the-middle attacks
- Client-to-attacker and attacker-to-server connections are established

High-level RV deployment

- Assert the executed protocol steps conform with the specification
- Instrumentation
  - **Monkey-patching** used for instrumenting Python code
  - **In-line hooking** used for compiled code
- Properties are manually modelled into RV monitors using **LARVA**
  - Automata-based approach
- Initial offline RV configuration
  - Instrumentation is limited to function call tracing
  - Events are **replayed** and **monitored** after the execution of the protocol

Low-level RV deployment

- Monitors data flows across the trust boundary
- Detect malicious interactions with the HSM
- Assert sensitive data is not leaked from the HSM

Future work

- **Analyse** the impact of the introduced **overheads**, in the context of SSH
- Deploy online RV consisting of
  - **Synchronous** monitors for **basic protocol sequence** properties
  - **Asynchronous** monitors for **memory-based** and **processor-intensive** properties
- Experiment with other software and hardware configurations